# 5Gtango

## D2.2 Architecture Design

<table>
<thead>
<tr>
<th>Project Acronym</th>
<th>5GTANGO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Title</td>
<td>5G Development and Validation Platform for Global Industry-Specific Network Services and Apps</td>
</tr>
<tr>
<td>Project Number</td>
<td>761493 (co-funded by the European Commission through Horizon 2020)</td>
</tr>
<tr>
<td>Instrument</td>
<td>Collaborative Innovation Action</td>
</tr>
<tr>
<td>Start Date</td>
<td>01/06/2017</td>
</tr>
<tr>
<td>Duration</td>
<td>30 months</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>D2.2 Architecture Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workpackage</td>
<td>WP2</td>
</tr>
<tr>
<td>Due Date</td>
<td>M6</td>
</tr>
<tr>
<td>Submission Date</td>
<td>30/11/2017</td>
</tr>
<tr>
<td>Version</td>
<td>0.1</td>
</tr>
<tr>
<td>Status</td>
<td>To be approved by the EC</td>
</tr>
<tr>
<td>Editor</td>
<td>Manuel Peuster, Stefan Schneider (UPB)</td>
</tr>
<tr>
<td>Contributors</td>
<td>Carlos Parada, José Bonnet (ALT), Felipe Vicens (ATOS), Ricard Vilalta, Raul Muñoz, Ricardo Martínez, Ramon Casellas (CTTC), Chris Woods, Peter Twamley (HWIRL), George Xilouris, Stavros Kolometos (NCSR), Peer Hasselmeyer (NEC), Huu Trung Thieu, Laurent Roulet (NBLF), Theodore Zahariadis, Antón Roman (QUO), Panos Trakadas, Panos Karkazis (SYN), Eleni Fotopoulou, Anastasios Zafeiropoulos (UBITECH), Manuel Peuster, Stefan Schneider (UPB), Dimosthenis Kyriazis, Marios Touloupou, Maria Siapera, Konstantinos Douloudis, Evgenia Kapassa (UPRC), Wouter Tavernier, Thomas Soenen (IMEC)</td>
</tr>
<tr>
<td>Reviewer(s)</td>
<td>Panos Trakadas (SYN), Yash Shekawat (NURO)</td>
</tr>
</tbody>
</table>

**Keywords:**
- Architecture
- V&V
- Catalogue
- SDK
- Service Platform
## Deliverable Type

<table>
<thead>
<tr>
<th>R</th>
<th>Document</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEM</td>
<td>Demonstrator, pilot, prototype</td>
<td></td>
</tr>
<tr>
<td>DEC</td>
<td>Websites, patent filings, videos, etc.</td>
<td></td>
</tr>
<tr>
<td>OTHER</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Dissemination Level

<table>
<thead>
<tr>
<th>PU</th>
<th>Public</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO</td>
<td>Confidential, only for members of the consortium (including the Commission Services)</td>
<td></td>
</tr>
</tbody>
</table>

**Disclaimer:**

This document has been produced in the context of the 5GTANGO Project. The research leading to these results has received funding from the European Community’s 5G-PPP under grant agreement n° 761493.

All information in this document is provided “as is” and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors’ view.
Executive Summary:

This deliverable D2.2 describes the initial overall 5GTANGO architecture, which is based on the pilot requirements in D2.1 [11] and the first V&V description in D3.1 [12]. The architecture and involved components also take the state-of-the-art and main related solutions and systems into account and, specifically, build upon 5G-PPP phase one project SONATA [1].

As main contributions, D2.2 provides a high-level overview of the initial 5GTANGO architecture and describes the novel service packages, which allow the compatibility with different service platforms with a single package through a novel layering concept. It also provides an overview of the workflow of 5GTANGO’s three main components and a detailed characterization of their structure. These main components are the software development kit (SDK), the verification and validation (V&V) platform, and the service platform. The SDK supports developers in creating network services, in generating the corresponding descriptors, and in packaging and testing the services. The V&V platform allows the users to thoroughly test the functionality and performance of the submitted network services. After testing, the V&V platform issues digitally signed results, which can be used to compare and choose from different network services. The service platform handles the management and orchestration of network services and ensures resource isolation (network slicing). Furthermore, D2.2 provides a high-level description of the interactions between the aforementioned components. D2.2 clearly specifies all of 5GTANGO’s extensions to existing components (e.g., to SONATA) and provides an overview of state-of-the-art and related work.

This architecture design aims at starting and guiding the development of the SDK, V&V, and service platform in WP4, WP3, and WP5, respectively. Details concerning implementation will be described in future deliverables. Insights gained during the agile development process may lead to changes in the described initial architecture, that will be reflected in subsequent deliverables.
## Contents

<table>
<thead>
<tr>
<th>List of Figures</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td>xi</td>
</tr>
</tbody>
</table>

1 Introduction

1.1 Scope of this Document ............................................. 1
1.2 Relationship to 5G-PPP Phase One Project SONATA .................. 2
1.3 Relation to Other 5GTANGO Deliverables .......................... 3
  1.3.1 Relation of D2.2 to Existing Deliverables .................... 3
  1.3.2 Relation of D2.2 to Upcoming Deliverables .................... 3
1.4 Document Structure .................................................. 4

2 Architecture Overview ................................................. 5

2.1 Architectural Principles ............................................ 5
2.2 Overall Architecture ............................................... 6
2.3 Catalogue Deployments .............................................. 7
2.4 Network Service Programming Model and Descriptors ............... 8
  2.4.1 Network Service Programming Model ........................... 9
  2.4.2 Descriptors .................................................... 10
2.5 Network Service Packages .......................................... 13
  2.5.1 SONATA Network Service Package Format ....................... 13
  2.5.2 ETSI VNF Package Format ..................................... 14
  2.5.3 5GTANGO Generic Package Format ............................... 15

3 SDK ............................................................................ 18

3.1 SDK Workflow ................................................................ 19
  3.1.1 Workspace and Project Creation ................................. 19
  3.1.2 Image Creation ..................................................... 19
  3.1.3 Fetch from Catalog ............................................... 19
  3.1.4 Management Component Development ......................... 19
  3.1.5 Descriptor Editing ............................................... 21
  3.1.6 Test Creation ..................................................... 21
  3.1.7 Descriptor Validation ............................................ 21
  3.1.8 Packaging ......................................................... 21
  3.1.9 Emulation ......................................................... 21
  3.1.10 Test Result Package Generation and Analysis .............. 22
3.2 SDK Components ........................................................ 22
  3.2.1 SDK Descriptor Support .......................................... 22
  3.2.2 Packaging Tool .................................................... 23
  3.2.3 Catalogue and Platform Access ................................ 24
  3.2.4 Emulation Platform ................................................ 24
  3.2.5 Test Definition Support ......................................... 27
3.2.6 Monitoring and Profiling Data Analysis ........................................... 31
3.2.7 State Handling Support ................................................................. 32

4 Verification and Validation (V&V) ......................................................... 34
4.1 V&V Overview ..................................................................................... 34
4.1.1 How the V&V Helps in Avoiding Technical Debt ............................... 34
4.1.2 Multiple Test Environments ............................................................. 34
4.1.3 Metadata and Packages and Continuous Testing ............................... 35
4.2 V&V Workflow ..................................................................................... 35
4.2.1 Upload of Network Services/VNFs with Metadata for Testing .......... 35
4.2.2 Manual Triggering of Test Pass ......................................................... 36
4.2.3 Upload of Tests and Metadata .......................................................... 36
4.2.4 Modification of Tests or Test Metadata ............................................. 36
4.2.5 Automatic Test Execution via Modified Event .................................... 36
4.2.6 Obtaining Test Results ................................................................. 39
4.2.7 Tested Package Push/On-boarding .................................................... 39
4.2.8 Automated Package Retest/Push ......................................................... 41
4.3 V&V Components ............................................................................. 42
4.3.1 Component Description ................................................................... 42
4.3.2 Test Repository and Test Result Repository .................................... 43
4.3.3 Test Engine ..................................................................................... 45
4.3.4 Test Platform Manager (TPM) ......................................................... 46
4.3.5 5GTANGO Catalogue in the V&V ................................................... 54

5 Service Platform ................................................................................ 57
5.1 Service Platform Workflows .............................................................. 57
5.1.1 User Management Workflows ......................................................... 57
5.1.2 Network Services and Virtual Network Functions ................. 58
5.1.3 Network Slice Lifecycle ................................................................. 71
5.1.4 Policy Workflow ........................................................................... 76
5.1.5 SLA Workflow ............................................................................... 80
5.2 Service Platform Components ......................................................... 84
5.2.1 Gatekeeper .................................................................................... 84
5.2.2 Catalogues ..................................................................................... 85
5.2.3 MANO Framework ....................................................................... 86
5.2.4 Network Slice Manager ............................................................... 89
5.2.5 Infrastructure Abstraction ............................................................. 91
5.2.6 Baseline Monitoring Framework .................................................... 92
5.2.7 Extensions to Existing Monitoring Framework ...................... 95
5.2.8 Network Service and Function Repositories .................................... 98
5.2.9 Network and Virtualised Infrastructure Management .................. 99
5.2.10 Graphical User Interface ............................................................ 100
5.2.11 Policy Manager .......................................................................... 102
5.2.12 SLA Manager ............................................................................ 103

6 Interactions ...................................................................................... 106
6.1 Catalogue Interactions .................................................................... 106
6.1.1 Catalogue SP Interaction ............................................................... 106
6.1.2 Catalogue SDK Interaction .......................................................... 106
6.1.3 Catalogue V&V Interaction ........................................... 107
6.2 SDK and V&V Interactions ............................................. 107
  6.2.1 Planned Interactions ............................................. 107
  6.2.2 Example Interaction Sequence ................................ 108
6.3 SDK and the Service Platform ...................................... 108
  6.3.1 Planned Interactions ............................................. 108
  6.3.2 Example Interaction Sequence ................................ 110
6.4 V&V and the Service Platform .................................... 110
  6.4.1 Using the Service Platform as a Test Environment ........ 110
6.5 Service Platform interactions with VIMs and WIMs .......... 112
6.6 Recursive Service Platform to Service Platform Interactions . 113

7 Conclusion 115

A State of the Art and Related Work 116
  A.1 EU-founded Collaborative Projects ............................. 116
    A.1.1 SONATA ..................................................... 116
    A.1.2 T-NOVA .................................................... 117
  A.2 Open Source Initiatives .......................................... 119
    A.2.1 Open Source MANO (OSM) .................................. 119
    A.2.2 ONAP ......................................................... 121
    A.2.3 OpenBaton .................................................. 122
  A.3 Commercial Solutions ............................................. 123
    A.3.1 NetCracker AVP .............................................. 123
    A.3.2 NOKIA ....................................................... 124
    A.3.3 AMDOCS ...................................................... 125
    A.3.4 ADVA ......................................................... 125
    A.3.5 CIENA ....................................................... 126
  A.4 Catalogue State of the Art ...................................... 126

B Acronyms 128

C Bibliography 131
List of Figures

2.1 Overall 5GTANGO Architecture .................................................. 7
2.2 Catalogue Deployments ................................................................. 8
2.3 Simplified example of the descriptor hierarchy ................................ 11
2.4 Entity relationship diagram of 5GTANGO descriptors based on SONATA descriptors 12
2.5 ETSI VNF package flavors given in ETSI GS NFV-SOL 004 with TOSCA-Metadata directory (a) and with single YAML-based metadata file (b) [27] ................................................. 14
2.6 5GTANGO network service package structure .................................. 15
3.1 Service Development Kit’s architecture .......................................... 18
3.2 5GTANGO service development kit’s workflow ................................ 20
3.3 SONATA packaging tool and its workflow steps as well as interactions with other SONATA tools [48] ................................................................. 23
3.4 Scope of the emulation platform in the simplified ETSI NFV reference architecture . 25
3.5 Deployment example: Emulation platform that emulates four PoPs and controlled by the SONATA service platform and OSM .................................................. 26
3.6 Emulator used as a locally installed prototyping and test execution platform ...... 27
3.7 SDK test development ................................................................. 28
3.8 Network service test example: Scalable DHCP service with DHCP request generators 30
4.1 Network service/VNF upload ........................................................... 36
4.2 Manual test triggering ................................................................. 37
4.3 Test upload ................................................................................. 37
4.4 Modification of tests or test metadata ............................................. 38
4.5 Test modification and addition ....................................................... 39
4.6 Network service upload ............................................................... 40
4.7 Obtaining test results ................................................................. 40
4.8 Pushing/On-boarding a tested package ......................................... 41
4.9 Functional architecture of the V&V platform .................................. 42
4.10 Test engine architecture .............................................................. 45
4.11 Test States ................................................................................. 47
4.12 Test state query .......................................................................... 48
4.13 Test cancellation .......................................................................... 49
4.14 Layers of the test platform manager ............................................ 49
4.15 Test preparation - shutting down services .................................... 51
4.16 Test preparation - test environment catalogue management ............... 51
4.17 Test preparation - pushing new services ....................................... 52
4.18 Test preparation - starting services ............................................. 52
4.19 Test preparation - configuring data collection ................................. 53
4.20 Test executing ............................................................................. 53
4.21 Test shutdown - shutdown metric collection .................................. 54
4.22 Shutdown running services .......................................................... 55
4.23 Test environment catalogue cleanup ............................................ 55
4.24 V&V and Catalogue interaction .............................................. 56

5.1 Service platform architecture ................................................. 58
5.2 User authentication on the service platform .............................. 59
5.3 Service platform’s user authorization ...................................... 59
5.4 Package on-boarding (pushed into the service platform) .......... 60
5.5 Package on-boarding (pulled from the V&V Platform) .............. 61
5.6 Instantiated service lifecycle ................................................ 62
5.7 Service instantiation (1) ...................................................... 63
5.8 Service instantiation (2) ...................................................... 64
5.9 Instantiated service lifecycle commands .................................. 66
5.10 Service termination (1) ...................................................... 67
5.11 Service termination (2) ...................................................... 68
5.12 Receive alerts from 5GTANGO message bus ............................. 70
5.13 Collect monitoring data from Monitoring Manager API .......... 70
5.14 Request streaming monitoring data through websocket .......... 71
5.15 Slice Lifecycle Management ............................................... 72
5.16 Network Slice Template On-boarding .................................... 73
5.17 Slice instantiation ............................................................ 74
5.18 Slice instance scaling ....................................................... 75
5.19 Slice instance termination .................................................. 75
5.20 Resources and slices overview ............................................. 77
5.21 Policy formulation for a network service (1) ....................... 78
5.22 Policy formulation for a network service (2) ....................... 79
5.23 Policy enforcement for a network service ............................... 80
5.24 Policy expressions formulation during development ............. 81
5.25 SLA template for service provider/infrastructure Provider Workflow .......... 81
5.26 Signed SLA workflow part I - If end-user NOT accept the SLA Template .......... 82
5.27 Signed SLA Workflow Part II(a) - If there is NOT mapping yet .......... 83
5.28 Signed SLA Workflow Part II(b) - If there is mapping .......... 83
5.29 SLA Monitoring Information Workflow ................................. 84
5.30 Gatekeeper’s internal architecture ...................................... 84
5.31 ETSI compliant 5GTANGO MANO framework architecture .... 86
5.32 NetSlice-NS-relation [20] .................................................... 89
5.33 Network slice manager’s internal architecture ....................... 90
5.34 Infrastructure abstraction architecture ................................ 91
5.35 Components of 5GTANGO Monitoring Framework ............... 93
5.36 Architecture diagram of distributed monitoring framework ...... 95
5.37 Diagram depicting 5GTANGO testing process ....................... 96
5.38 5GTANGO runtime monitoring ............................................ 97
5.39 Frequency Analyzer Mechanism ......................................... 98
5.40 WIM overall view ........................................................... 100
5.41 Policy Manager Internal Architecture ................................ 102
5.42 SLA Manager Internal Architecture ................................... 105

6.1 Typical SDK and V&V interactions ........................................ 109
6.2 Typical SDK and SP interactions .......................................... 111

A.1 SONATA’s high-level architecture mapped to the ETSI NFV reference architecture 117
A.2 T-NOVA’s orchestrator architecture (TeNOR) . . . . . . . . . . . . . . . . . . . . . . . . . 118
A.3 Open Source MANO (OSM) architecture to ETSI mapping . . . . . . . . . . . . . . 119
A.4 ONAP architecture overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
A.5 OpenBaton architecture (from [21]) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 122
A.6 NEC/Netcracker’s agile virtualization platform and practice . . . . . . . . . . . . . . 124
List of Tables

4.1 Structure of the test results ................................................. 44
4.2 Test state overview ................................................................. 46
A.1 NEC Netcracker feature overview ......................................... 124
A.2 Nokia CloudBand feature overview ........................................ 125
A.3 Amdocs feature overview ...................................................... 125
A.4 ADVA Ensemble Orchestrator feature overview ......................... 125
A.5 Ciena Blue Planet feature overview ....................................... 126
A.6 State of the art catalogue feature overview ............................. 126
1 Introduction

5G networks will not only improve the radio access but also drastically introduce the flexibility of the network by using software-defined networking (SDN) and network function virtualization (NFV). In NFV, network services consist of multiple interconnected virtual network functions (VNFs), e.g., firewalls, IDS, or DPI. These network functions are implemented in software and can run on standard x86 machines.

This virtualization allows dynamic scaling, i.e., flexibly determining a suitable number of VNFs and allocated resources according to the current load. In doing so, no resources are wasted and operational expenditures (OPEX) are reduced while always ensuring good quality of service. Furthermore, VNFs can easily and quickly be migrated and placed at different nodes in the network, minimizing the resulting delay and required bandwidth. As VNFs are software (as opposed to old hardware NFs), the development and deployment effort can be significantly reduced, leading to shorter time-to-market and lower capital expenditures (CAPEX).

There are various ongoing projects aiming to achieve flexible management and orchestration (MANO) of network services and VNFs. However, current approaches are still not completely mature and only support limited functionality. 5GTANGO builds upon the SONATA project to extend and improve the MANO platform/service platform.

Another challenge of NFV is to support and simplify creation of network service packages and descriptors, even for complex configurations. 5GTANGO tackles this challenge by providing an extensive SDK that supports creation of network service descriptors and packages, even supporting recursive network services. A main novelty of 5GTANGO is the Validation and Verification (V&V) platform. The V&V is a stand-alone platform that allows developers, operators, or 3rd parties to test their VNFs and network services in an automated fashion. In doing so, the V&V provides a flexible set of tests covering various functional and performance tests. The V&V issues the test results as digitally signed certificates. These results provide feedback to the developer, supporting further improvements of a network service. They also confirm the quality of a tested network service to operators or 3rd parties, allowing them to compare different versions and implementations of network services to select the most suitable ones. Finally, the test results also offer valuable insights for the service platform that can be used during placement and orchestration of the network services. The service platform manages and orchestrates submitted network services while abstracting the infrastructure details. It also provides management of different users and isolation of resources (network slicing).

Overall, 5GTANGO will greatly alleviate the burdens of development, deployment, management, and orchestration throughout the lifecycle of network services by introducing a complete DevOps approach.

1.1 Scope of this Document

The goal of this deliverable is to describe the high-level architecture and concepts envisioned for 5GTANGO. The deliverable provides a broad overview of 5GTANGO’s architecture, containing the SDK, the V&V, and the service platform. For each of these components, we describe the workflow, the high-level architecture, and their interactions without going into technical detail. Moreover, we
discuss state-of-the-art related to 5GTANGO.

Some details described here (e.g., APIs or functionality of internal components) might change and evolve throughout the project based on the experience gained during the development of the first prototypes. All changes and updates in the architecture design will be reflected in deliverable D2.3.

1.2 Relationship to 5G-PPP Phase One Project SONATA

The 5GTANGO project is based on the concepts and developments of the 5G-PPP phase one project SONATA [1]. SONATA is developing a NFV framework that provides a programming model and development toolchain for virtualized services, fully integrated with a DevOps-enabled service platform and orchestration system.

All project results are open source and publicly available in a GitHub repository [57] under Apache v2.0 license, a permissive license that guarantees full rights for adoption, modification and distribution. The SONATA project has a 30 month workplan that started in July 2015 and many partners involved in SONATA are involved in 5GTANGO as well. This brings a lot of experience and in-depth knowledge about available SONATA developments, tools, and work flows to the 5GTANGO consortium. The overall SONATA architecture [46] with its two major building blocks SONATA SDK and SONATA service platform that are based on SONATA’s DevOps workflow are described in the following in more detail.

• SONATA’s Network Service SDK: The SDK supports service developers with both a programming model and a set of software tools. The SDK allows developers to define complex services consisting of multiple VNFs. Further the SDK provides a multi-PoP emulation platform for rapid prototyping of VNFs and services as well as primitive validation and profiling solutions. All components of the SONATA SDK are described in SONATA’s deliverables D3.1 - D3.3 [47, 48, 49] in more detail. Many of SONATA SDK components will serve as basis for 5GTANGO’s SDK, for example, the emulation platform or the component to access the service platform. However, 5GTANGO will heavily modify and improve these components to fit the project’s needs as described in Sec. 3. One example for this is the packaging tool which will be extended in order to support 5GTANGO’s novel package format that offers packaging flexibility that goes beyond the SONATA concepts (see Sec. 2.5). Another examples for required modifications are all SDK components that interact with catalogues and service platform. They need to be updated to support new interfaces, interact with the V&V, or allow to deal with extended metadata information generated in the V&V. Finally, a set of new tools will be developed from scratch, to support test development for the V&V as well as advanced data analysis tasks utilizing performance data captured using automated performance tests (see Sec. 3.2.5 and Sec. 3.2.6).

• SONATA’s Service platform: The service platform is the second major component of SONATA. Due to the fully customisable and modular design of its management and orchestration (MANO) framework, the service platform offers customisation opportunities on two levels. First, the service platform operator can modify the platform, e.g., to support a desired business model, by replacing components of the loosely coupled MANO framework (MANO plugins). Second, service developers can influence the orchestration and management functionalities of the platform pertaining to their own services, by bundling small management programs, so-called function- and service-specific managers (FSMs/SSMs), with their services. The design and implementation details of SONATA’s service platform components are described in SONATA’s deliverables D4.1 - D4.3 [50, 51, 52]. Like for the SDK, large
parts of the SONATA service platform will be re-used for 5GTANGO. However, due to the new V&V component, substantial updates of external interfaces, interactions, and work flows are required. This in particular involves novel service platform interfaces that allow the use of the service platform as test execution platform for the V&V (see Sec. 4.3.4). Finally, the 5GTANGO service platform will provide a set of novel components in order to support advanced slicing concepts (see Sec. 5.2.4) as well as policy and SLA management solutions enabling automated service management and assurance based on metadata gathered by the V&V (see Sec. 5.2.11 and Sec. 5.2.12).

- **SONATA’s NFV DevOps Workflow**: The SONATA system is designed for agile development and operation of network services. It enables a DevOps workflow between the SDK tools and the service platform, which allows developers and operators to closely collaborate. In addition to this, SONATA components and service platform itself have been developed in an agile way. To do so, a complex continuous integration and delivery (CI/CD) pipeline was used to simplify the integration of different components as described in SONATA’s deliverables D5.1 - D5.4 [53, 54, 55, 56]. 5GTANGO will exploit the expertise and knowledge generated during this process for its own developments where possible.

Even if we can built on some of the existing components, major extensions, substantial updates, and new developments will be needed for 5GTANGO. More specifically the introduction of V&V creates the need for extensions on existing SDK and service platform implementations, in the following two ways: First, being, from an architecture perspective placed in between SDK and service platform, there is a need of adapting the interfaces and the technologies to communicate with V&V. Secondly, and most importantly, the V&V component brings new functionality that must be supported or provided by the SDK and V&V components in order to create a complete DevOps-based platform. As an example, V&V introduces the concept of metadata collection that is a new requirement for service platform.

### 1.3 Relation to Other 5GTANGO Deliverables

The architecture design described in this deliverable builds upon existing deliverables and is related to other 5GTANGO deliverables as described in the following.

#### 1.3.1 Relation of D2.2 to Existing Deliverables

- **D2.1 (M3) [11]**: Deliverable D2.1 defines the pilots and specifies the functional and non-functional requirements for 5GTANGO and its components. The architecture design of SDK, V&V, and service platform takes these requirements into account.

- **D3.1 (M4) [12]**: Deliverable D3.1 describes the V&V strategy, which is also reflected in the V&V workflow described in D2.2.

#### 1.3.2 Relation of D2.2 to Upcoming Deliverables

- **D3.2 (M10)**: Deliverable D3.2 will present a first prototype of the V&V platform and describe its components and APIs in more (technical) detail than D2.2.

- **D4.1 (M10)**: Deliverable D4.1 will present a first prototype of the SDK toolset and describe its components and APIs in more (technical) detail than D2.2.
• **D5.1 (M10):** Deliverable D5.1 will present a first prototype of the service platform and describe its components and APIs in more (technical) detail than D2.2.

• **D2.3 (M16):** As the project progresses, some architectural decisions or requirements will have to be adapted based on the experience made during development of the first prototypes. D2.3 will reflect this updated architectural design.

### 1.4 Document Structure

The remainder of this deliverable is structured as follows. Sec. 2 provides an overview of the overall architecture and architectural principles. 5GTANGO’s three main components are described in the following three sections. Sec. 3 introduces the workflow, architecture, and components of the SDK, supporting the development, packaging, and uploading of new network services. In sec. 4, we describe the V&V platform that allows for automatic testing of network services. This helps developers achieve high quality of their network services and operators to select from various existing services. Sec. 5 describes 5GTANGO’s service platform, which handles the management and orchestration of network services as well as infrastructure abstraction, slice management, etc.

Sec. 6 describes the interactions of the aforementioned three main components with each other and with the catalogues. In Sec. A, we describe and compare existing state-of-the-art and related work (other EU-funded projects, open source initiatives, and commercial solutions). Finally, sec. 7 concludes this deliverable.
2 Architecture Overview

This deliverable describes the architecture of the software platform to be created in the 5GTANGO project. It breaks down the overall system into smaller components that serve particular purposes. Architectural descriptions can be applied at various levels of granularity of the system. In this document, we are staying at a rather high level of abstraction. Detailed descriptions and further component break-down, for example, down to the class level, are not part of this document. Nevertheless, the overall 5GTANGO system is broken down into components of more manageable size, whose architecture will then be refined by the partners working on and implementing these components in the upcoming deliverables as described in Sec. 1.3.

The architectural descriptions in this document mainly detail the component architecture, i.e., the structure of the system to be developed is described in terms of interacting software artefacts. The descriptions include purpose, functionality and behaviour of the components, as well as the interactions between them. The component descriptions abstract from their actual implementation. Indeed, the implementation of the components can change over time, while the architecture stays the same.

This document focuses on the description of the 5GTANGO platform. It does not detail the software architecture of the pilot applications which are to be developed within the 5GTANGO project to validate and showcase the features of the platform. Nevertheless, an important driver for the architecture of both the platform and the applications is the service model that the 5GTANGO project adopts. This service model is therefore introduced and described in this document as well.

2.1 Architectural Principles

Development of the 5GTANGO platform is based on a number of guiding principles. These principles were selected to ensure evolvability, reusability, maintainability, and stability of the platform. They are described as follows.

- **Simplicity.** Components in the 5GTANGO platform should follow the Unix philosophy of doing only one thing and doing it well [7]. More elaborate functionality emerges from combining multiple components. This guideline is designed to foster maintainability of the developed components. As the components focus on realizing a specific functionality, it is easier to grasp that functionality. Complexity is reduced and the implementation is expected to have fewer bugs than large components with varied functionality would have.

- **Loose coupling.** The principle of loose coupling was selected to ensure the development of a modular system which allows independent development and evolution of the individual components. Aspects of loose coupling include avoiding APIs which exhibit implementation-specific details, such as data structures and functional decomposition. Instead, APIs are designed to resemble the application domain as much as possible. Implementation details, such as the use of a relational database, are to be hidden from users of a component. Loose coupling is designed to allow for the independent evolution of servers and clients. In particular, additional functionality on the server side should not render existing clients unusable or incompatible. Similarly, servers should not be restricted to being used by a specific set of
clients. Rather, servers should be agnostic to its clients and the set of clients and their purpose should be unrestricted and changeable over time. One important result of loose coupling is the associated replaceability of components. If the API is not exposing internal implementation details, server components providing that API can be replaced with functionally equivalent components without the clients noticing. Similarly, clients can be replaced with new ones fulfilling the same or different functions than before without the server noticing or needing to be changed. The software lifecycle of creating, deploying, running, and retiring software components is supported by loose coupling and component independence.

- **Micro-services based.** The 5GTANGO platform is to be implemented by micro-services. Micro-services are small, independently deployable components which are realizing the principle of loose-coupling. They are also fostering the use of the simplicity principle, as they are geared towards small, dedicated components, which are typically functionally specific (“do one thing”) and comprehensive (“do them well”).

- **Support of multiple service platforms.** The 5GTANGO platform should be as much as possible agnostic to the service platform on which it deploys network services. All components developed in the project should therefore be as much as possible independent of the specificities of a particular service platform. As this is not always possible, for example because of the reliance on a particular feature or capability which is present in only one or a few service platforms, some features will only be available on this subset of service platforms. Nevertheless, the fundamental functionalities need to be available on all supported service platforms.

- **Support for physical and virtual infrastructure.** The 5GTANGO platform is aimed at supporting services beyond those offered by traditional network operators, including those from industrial and media companies. To properly support these verticals, the inclusion of physical equipment, e.g. manufacturing machines, sensors, and cameras, is mandatory. Pure software functions supporting services will be deployed in virtual environments, such as the cloud. To properly support the chosen use cases, the 5GTANGO platform also needs to support software functions controlling and operating physical equipment. Placement decisions will need to be based on the attachment points of the physical devices.

### 2.2 Overall Architecture

The 5GTANGO’s platform is split into three large subsystems, which are modelled according to three phases. These phases represent stages in the lifecycle of a service: development, verification & validation, and operation. Although these phases may overlap, they typically happen at different timescales and are performed by different actors:

1. **Phase 1:** Development of a network service, and publishing of that network service in a catalogue

2. **Phase 2:** Testing, validation, and verification of a network service with a V&V platform, and publishing of the results in a catalogue

3. **Phase 3:** Selecting a network service from a catalogue, deployment and operation of that network service using the service platform orchestration capabilities such as placement, monitoring, scaling, etc.
These phases are typically executed by different stakeholders, which may or may not belong to the same entity. The following roles exist and can be mapped to persons, companies, departments, etc.

- **Developer**: Develops network service, tests it, and publishes it.
- **V&V Provider**: Provides testing environment, executes tests, returns signed results, and maintains the results in catalogues.
- **Operator**: Selects existing network service, and deploys it.

A single individual can take on multiple roles. Similarly, any one of these roles can be taken on by multiple actors. For example, it is expected that there will be multiple developers, operators, and V&V providers.

The three phases and their associated subsystems are shown in Fig. 2.1. The subsystems are referred to as SDK (service development kit), V&V (verification and validation platform), and service platform. Details of these components are elaborated in the remainder of this document. Following the loose-coupling principle, the subsystems are independent of each other and can be deployed and used independently.

### 2.3 Catalogue Deployments

In the 5GTANGO architecture multiple catalogue instances can be deployed. The purpose and associated functionality of each instance depends on the stakeholder requirements for the given role; this can include such aspects as control of visibility, and security types of data stored. The catalogue in all the deployments will be based on the same technology, but each instance will utilise only the functions that it requires, thus offering different configurations for each stakeholder.

As shown in Fig. 2.2, we consider the following catalogue deployments:
1. **5GTANGO catalogue in the V&V:** We foresee multiple cases of V&V operators (with specialisations). Each one of those may have its own catalogue deployment.

2. **5GTANGO catalogue in the Service Platform:** Each service platform operator may maintain their own catalogue deployment to host packages of their currently available services. It is evident that each service platform catalogue may be in sync or completely independent from other catalogues.

3. **Public 5GTANGO catalogue:** There may be stand-alone deployments of the catalogue that will be able to host services from multiple V&V and service platform catalogues that third-parties may access.

There may be many V&V providers and service platforms. Each one of those will have its own instance of the 5GTANGO catalogue with all or a subset of its functionality. The catalogues will have interfaces for searching, storing and retrieving VNFs/network services, tests and metadata. The catalogue that is deployed in the V&V platform will contain tests and VNFs/network services (and their metadata) that have been tested and have either passed or failed the tests. The catalogue that is deployed in the service platform will contain only VNFs/network services (and their metadata) that have passed V&V tests. There may also be public catalogues that contain VNFs/network services and their metadata that all the other components can access.

### 2.4 Network Service Programming Model and Descriptors

This section gives an overview of the network service programming model including description approaches for single VNFs and complex network services.
### 2.4.1 Network Service Programming Model

A network service programming model defines the concepts and abstractions that developers build and use in order to define network services consisting of virtualised as well as physical components. A network service involves any service which provides one or multiple interfaces to:

1. Process or adapt network packets (for example: L2 data frames or L3 datagrams).
2. Process or adapt sessions (for example: transport layer sessions, as well as application layer sessions).
3. Process or adapt application-level requests (for example: web requests, remote function calls).

The first two flavours refer to network functionality which is typically associated with middlebox functionality [43] targeting improved network security through firewalls or intrusion detection systems, or improved network performance via, e.g., WAN optimizers or network proxies. The third flavour refers to more traditional cloud services, including web- and database services. The 5GTANGO network service programming model aims to provide a unified model for developing and composing such services in an efficient, unified and robust manner, building further on the foundations of the SONATA programming model and associated platform. The latter, on its turn is solidly based on the model imposed by the ETSI NFV ISG [18]. Below we shortly recapitulate its essential characteristics and terminology:

- **Individual, isolated components processing packets, sessions or request** are referred as **network functions (NF)**. When they are entities in software, which can run on virtualized environments relying on compute or network hypervisors (e.g., Xen [14] or OpenVirtex [13]), we refer to them as VNFs, when they are entities available on special-purpose hardware, we refer to the as physical network functions (PNFs). Individual VNFs might further be decomposed into **Virtual Deployment Units (VDU)**.

- **Management of NFs** is decomposed into two parts:
  - **Functional management** is performed by the **Element Manager**, responsible for fault, configuration, accounting, performance and security management, involving for example the configuration of the cache behaviour of an individual cache NF.
  - **Virtual instance life-cycle management** is performed by the VNF Manager or **Function Specific Manager (FSM)**. This involves the setup, scale-up/down, and tear-down of individual VNFs.

- A network service is a composition of NFs, involving: i) a **Network Connection Topology (NCT)** consisting of links between Connection Points of NFs, and ii) an NF **Forwarding Graph (NF-FG)** describing the forwarding or chaining behaviour between NFs as allowed by the NCT.

- **Management of the virtualized instance of the network service** is performed by the **Service Specific Manager (SSM)**. Analogously to the FSM, SSM functionality is responsible for the setup, maintenance, scale-in/out, as well as tear-down of the virtualized network service instance.

This overall network service programming model concepts have more or less converged in the state of the art, as can be witnessed in the range of network service description formats currently
available. The TOSCA framework [58] has been extended to support NFV services as proposed by ETSI NFV ISG, the OpenStack HEAT [30] has undergone some modifications to provide improved support for network-focused services, several research projects [16, 61, 29] have proposed their form of YAML, JSON, XML-based network service descriptors, or even dedicated programming languages, as well as the ETSI-related open-source OSM project proposed its own YAML-based descriptors [28]. Although the global concepts have converged, the above overview illustrates that no current format is clearly taking a leading role.

The 5GTANGO approach aims to improve this situation by focusing on the following (re-)design principles for the network service description model:

- The description of individual NFs as well as of network services should be equally readable for humans as for machines. This might imply that a two-step generation or conversion process between high-level templates and low-level XML, YAML, JSON-files will be needed.

- Composing services should be easy, robust and predictable, enabling the re-use of services. Recursiveness in the definition and description of services is therefore a strong requirement.

- Existing data models and descriptors should be easily extensible with low-risk of breaking earlier deployment frameworks.

- To cope with the current diversity in network service description formats, it is desirable to adopt a 5GTANGO description model which can be easily converted and transformed to other formants.

- The proposed programming model should maximally allow for agile development, testing - and deployment workflows, as envisioned by the integrated DevTestOps strategy of the project.

### 2.4.2 Descriptors

Network service descriptors (NSDs) and virtual network function descriptors (VNFDs) contain and describe all relevant aspects of network services and constituent VNFs in a structured way. The syntax and structure of these descriptors is clearly defined by an NSD-schema and VNFD-schema, respectively. The schemas ensure that all relevant information is included and can easily be parsed. It also allows automatic syntax checks of new NSDs or VNFDs in a CI-way.

#### 2.4.2.1 Virtual Network Function Descriptor

The VNFD contains basic information such as name, version, vendor, etc. Another core element of each VNFD are the virtual deployment units (VDUs). Each VDU describes and links to a VM image that performs some functionality. The VNFD specifies which VDUs are involved and how they are connected.

Furthermore, the VNFD can describe lifecycle events such as start/stop/scale/…, deployment flavours, and function-specific managers (FSMs). FSM which allow fine-grained management of individual VNFs (e.g., affecting placement).

#### 2.4.2.2 Network Service Descriptor

Network service descriptors specify the involved network functions and how they are connected. The connection of involved VNFs is described by virtual links and the forwarding graph. Similar
to VNFDs, NSDs can contain lifecycle events (start/stop/scale) service-specific managers (SSMs) that allow fine-grained management of the service.

A novelty in 5GTANGO is the introduction of recursive NSDs that allow the description of recursive network services, i.e., including not only VNFs but also other network services. Allowing such recursive references enables faster and easier reuse and extension of existing network services by just reusing and referencing the corresponding NSDs. In doing so, the creation of new NSDs also becomes less error-prone.

Fig. 2.3 shows the hierarchy of descriptors: NSDs can reference other NSDs recursively and/or consist of references to VNFDs. The corresponding VNFs contain VDUs that perform specific tasks. While referenced NSDs or VNFDs are described in separate descriptor files, VDUs are described within the corresponding VNFD.

Fig. 2.4 illustrates the structure of NSDs and VNFDs, including all fields supported by the schema. Required fields are printed in cursive. Primitive types are directly annotated in the diagram.

Network services, virtual network functions, and virtual deployment units each define connection points that allow the communication with the outside world. Typically, there is one connection point for input, one for output, and one for management. These connection points are connected with virtual links (vLinks). Specifically, vLinks of a VNF connect the connection points of the constituting VDUs and vLinks of a network service connect the connection points of the constituting VNFs or network services.

The forwarding graphs of a network service use the constituting VNFs/NS and vLinks to construct forwarding paths. These paths express the order in which the VNFs/NS are traversed. Network services and VNFs can further contain SSMs or FSMs, respectively, and other information as indicated by Fig. 2.4.

2.4.2.3 Vision of descriptors in 5GTANGO

The basic structure of the NSDs and VNFDs is derived and reused from SONATA. In 5GTANGO, we will adjust and extend these descriptors as follows. We already introduced the notion of recursive NSDs to describe recursive network services that reference other network services. In doing so, existing network services can easily be reused and extended by simply referencing the existing NSD and adding VNFs. Otherwise, the old NSD would have to be copied and adjusted manually which is time-consuming and error prone. To enable recursive NSDs, we added an optional section of included network services (similar to included VNFs) and extended or adjusted existing sections and fields. For example, virtual links can now connect connection points of VNFs and network services.
Figure 2.4: Entity relationship diagram of 5GTANGO descriptors based on SONATA descriptors
Furthermore, 5GTANGO will describe monitoring information in separate descriptors rather than within VNFDs (like in Sonata). This separation of concerns helps simplify each descriptor. It also allows the specification of various monitoring descriptors for a single VNFD. This is useful as different metrics might be of interest for different scenarios.

Finally, 5GTANGO aims at closer alignment to the ETSI NFV information model. Renaming relevant fields and identifying gaps between the two information models will increase 5GTANGO’s impact on standardization and chances of adoption.

2.5 Network Service Packages

Network services that consist of multiple network functions are the central concept in the NFV world. After they have been created and tested, they are deployed and operated on a variety of service platforms. To allow the exchange of such services between different entities, domains, and platforms the concept of network service and VNF packages is used. This concept allows to package single VNFs or entire network services into single, easy-to-handle, well-defined, and exchangeable container artifacts.

To leverage the benefits of packages and a consistent package management, current cloud computing environments adopted the concept as well. Evidently, this allows for a simplified life-cycle management and a simplified migration not only of virtualized execution objects (e.g., virtual machines or containers), but complete services that might comprise several virtual instances. The prominent TOSCA Cloud Service Archive (CSAR) [22] for instance, is a container file containing multiple files organized in several subdirectories. Artefacts, such as configuration files, software code, and virtual machine images, in a CSAR may be sealed. That is, all artefact referred to in such a CSAR are actually contained in the CSAR. A CSAR or a selective artefact within a CSAR may be signed. When signing an artefact of a CSAR the digest of this artefact as well as the public key of the entity signing the artefact is included in the CSAR together with a corresponding certificate.

These cloud package concepts are naturally very close to the NFV domain and are already in use by a couple of projects, like SONATA or OSM [1, 17]. However, these early adopters usually use their own package specifications making the on-boarding process to different catalogs or platforms very complicated, today. To address this, ETSI recently started to define and specify a common VNF package format, based on the TOSCA CSAR standard [22] which we explain in Sec. 2.5.2 in more detail. Even though, this package format is a good starting point, some important features are still missing in the specification, e.g., support for complete network services inside a package. The 5GTANGO packaging format will closely follow the ETSI package format and extend it wherever needed as described in Sec. 2.5.3. The overall goal of this work is to create a generic package format that allows to package VNFs and services for different target platforms to simplify V&V and on-boarding procedures as much as possible. Further, we aim to actively contribute the concepts and insights developed in 5GTANGO to the existing package specifications.

2.5.1 SONATA Network Service Package Format

The package format developed by SONATA already follows the basic concepts of the CSAR specification, by using a single manifest file (also called package descriptor) that acts as an index to describe files contained in the package [46, 48]. The main advantage of SONATA packages is that they already allow to package entire network services instead of single network functions only. To do so, the package descriptor points to a single NSD that acts as the main entry point of the service description. Further, does the package descriptor list one or multiple VNFDs that describe
Besides the ability to package network services, SONATA packages can be either shipped as slim or fat packages. Where slim packages do not include large artifacts, like VM disk images, but references to external storage locations, e.g., a download server. This concept also exists in the CSAR approach and is referred to as sealed or un-sealed package artifacts where sealed artifacts have to be included inside a package.

The SONATA package format currently does not support packaging of single VNFs, nor is it fully compatible to the ETSI packaging standard. This is why 5GTANGO plans to extend and improve this packaging format even further.

### 2.5.2 ETSI VNF Package Format

The format for VNF packages is described in *ETSI GS NFV-SOL 004* [27] and is directly based on TOSCA’s CSAR format [22] following TOSCA Simple Profile YAML v.1.1. A CSAR package is an archive file using the ZIP file format and includes metadata information that describe the actual content and structure of the packages. This allows to pack any number of artifacts with arbitrary folder structures into a CSAR package. A CSAR package based on TOSCA Simple Profile YAML 1.1 can exist in two flavors:

1. **CSAR with TOSCA-Metadata directory**: The CSAR contains a TOSCA-Metadata directory, which includes a TOSCA.meta file which provides an entry information for processing a CSAR file.

2. **CSAR zip without TOSCA-Metadata directory**: A package that contains a single YAML file at the root of the archive. The YAML file is a TOSCA definition template and acts as entry definition.

Both flavors are equivalent in terms of flexibility and expressiveness. Examples package structure for each of them are given in Fig. 2.5.
The ETSI VNF package specification foresees that a VNF package contains several additional files in addition to the VNFD and VNF artifacts, like disk images. For example, a change history file, testing files, licensing information, or certification files. All this will be required in 5GTANGO as well. However, the ETSI specification does not foresee packages for network services, yet.

Even though, the ETSI VNF package specification is relatively young and still under development, first MANO projects, like OpenBaton [21] already offer support for it. We assume this trend to continue, since also OSM [17] plans to support the ETSI package format with OSM release four.

2.5.3 5GTANGO Generic Package Format

The main idea of the 5GTANGO package format is to go beyond existing package formats for NFV services by (a) allowing the package not only to contain single VNFs (like ETSI VNF packages) but also to contain entire composed network services. A 5GTANGO package can be a VNF package, a network service only package, or a package that contains both the network service and the VNF it consists of. If a package contains only a VNF, it is used to on-board this single VNF to a catalog (or a service platform) which can then later be referenced by an network service that is on-boarded with another package. (b) A 5GTANGO package allows to package different descriptor formats for the same VNF or network service inside a single package, e.g., SONATA and OSM descriptors describing the same VNF or network service. We call this concept layering. The benefits of this concept is that a developer can ship a service that is compatible to different platforms within a single package. An example for this is the V&V which can then test the network service against different target platforms, e.g., SONATA and OSM. (c) In 5GTANGO, service or VNF packages can be given to a V&V component that will test the contained artifacts and generate test results. These results can themselves be packed inside 5GTANGO packages (test result packages) which are then signed and returned by the V&V platform to ensure integrity. These test result packages need to reference the exact service of VNF for which they executed the tests. To do so, 5GTANGO packages can reference other 5GTANGO packages using unique package identifiers. In addition to this, the signed result packages will store hashes of the tested packages so that a third party can verify the integrity of the referenced package. (d) 5GTANGO package are built on top of the existing ETSI VNF package definition [27] that is built on top of the TOSCA CSAR specification [22]. 5GTANGO extends these formats where needed to implement the new features ((a)-(c)) but tries to stay as close to the original specification as possible. This will help to allow adoption of the 5GTANGO packaging tools by other projects or to contribute to the existing standards.

Fig. 2.6 shows the general concept of the 5GTANGO package format. Starting from the package’s manifest, arbitrary artifacts, like descriptors, can be included in the package or referenced by it. The
figure also shows how different versions of the same artifacts are supported through 5GTANGO’s novel layering concept described in the next sections.

2.5.3.1 Novel Features

VNF and Network Service Package Support

5GTANGO packages support the packaging of entire network services instead of single VNFs only. To implement this, the entry point, specified in the manifest of the package, can either point to a VNFD or NSD. The platform, or any other component will then check the type of the entry point descriptor to understand which package type is processed.

1. VNF package: Entry point to VNFD
2. Network service package: Entry point to NSD

There might also be packages that only contain test results. The entry point of such packages will point to the corresponding test results which allows the identification of the package type. We do not specify these options in more detail now, but leave their specification for the first and second prototype release of 5GTANGO.

Package Layers

5GTANGO packages allow to package different flavors of the same service or VNF inside a single package. The benefit of this is that multiple service specifications for different target platforms can be included in a package, which helps to bypass the fact that still not all MANO systems use a common descriptor format and/or information model. We realize this through a concept called layered service packages as shown in Fig. 2.6. To implement this layering, we allow to tag each artifact referenced from the manifest with an arbitrary number of layer names:

Source: files/images/vnf_firewall_osm.qcow2
Algorithm: SHA-256
Hash: 34adsd7867asdasdlk234asdd13
Layers: etsi.osm

Source: files/images/vnf_firewall_tango.tar.gz
Algorithm: SHA-256
Hash: 34adsd7867asdasdlk234mn233
Layers: eu.5gtango, vim.docker

Using these (optional) layer fields, packaging tools can filter the artifacts inside a package and only present the ones matching the filter to the MANO system or other components. If, for example, the package should be unpacked for a deployment on an OSM platform, only files that are tagged with etsi.osm will be extracted from the package. All packaging tool shall assume that the layer default is always assigned to each artifact, making the Layers keyword an optional field.

Package References

In order to allow packages with test results to point to the corresponding service or VNF packages, the 5GTANGO package format allows to reference external packages. For this, the 5GTANGO format inherits the concept for unique identifiers based on a vendor.name.version triple that was already introduced by SONATA [46]. This concept has turned out to be very useful since it is
human readable. The reference to a package identifier can be encoded inside a URI to also provide additional location information needed to fetch the referenced package, as shown in the following examples:

PkgRef_URI: tgocatalog://cat.5gtango.eu/upb.firewall.0.2.tgo
PkgRef_Algorithm: SHA-256
PkgRef_Hash: 45adzd7867asdasdlk234mn233

PkgRef_URI: ftp://privateserver.com/nec.ids.0.3.tgo
PkgRef_Algorithm: SHA-256
PkgRef_Hash: 11x4zd7867asdasdlk234mn233

PkgRef_URI: local://atos.encoder.0.1.tgo
PkgRef_Algorithm: SHA-256
PkgRef_Hash: 67x4zd7867asdasdlk234mn233

In addition to the actual reference to another package, a hash of the referenced package is stored so that another entity is able to check the integrity of the referenced package.

2.5.3.2 Specification

The specification of the exact package format will be done through schema files as it was done in the SONATA project. The benefit of this is the possibility to automatically perform a static validation on given packages. All schemas are maintained in a public GitHub repository [10] so that they are always accessible to project partners and any external developer. The exact package format will be refined and described in more detail with the first prototype release.
3 SDK

The 5GTANGO SDK aims at providing the developer with a powerful set of tools to develop, test and evaluate NFV-based Network Services. The resulting toolset builds further on the ecosystem of SONATA SDK, and further evolves it, given the changing state-of-the-art, modified programming model and Service Platform, providing support for the 5GTANGO Validation and Verification Store. Fig. 3.1 provides an overview of the high-level components of the 5GTANGO SDK, and highlights the changes and innovations (coloured in green) compared to its departing point, the SONATA SDK.

The central SDK toolset consists of a set of well-integrated, command-line interface (CLI) applications son-cli developed in SONATA. This includes tools for workspace/project creation, validation of involved descriptors, packaging functionality and an upload tool son-access to onboard packages on either the SONATA Service Platform or the SONATA emulator. 5GTANGO will further adapt and extend these CLI tools in order to support improved descriptor formats, functionality to ease the process of NF image creation, as well as related packaging towards a range of platforms (for example, including OSM support). The Upload functionality will be further extended to enable interaction with the 5GTANGO V&V Store, as well as enable interaction with NS/NF catalogues.

The robust and efficient creation of descriptors is becoming an increasingly difficult task in a landscape of services consisting of plenty of components, requiring to run on a range of infrastructures, and supported by multiple MANO platforms. The 5GTANGO SDK will dedicate a Descriptor Generator component to increase the speed and robustness of creating descriptors.

A key innovation of the SONATA project consists of the concept of function and/or service-specific managers (FSM/SSMs). These components enable services to dynamically influence the virtualized lifecycle in order to support scaling or failover. The events often require adequate state handling. 5GTANGO will therefore develop a Specific Manager Development component assisting the developer in the creation of management functionality.

Validation and verification are central points of innovation in the 5GTANGO project. The execution of tests and associated collection of performance results is one of the functionalities of the 5GTANGO V&V Store. The SDK will support this new environment by providing an extensive

Figure 3.1: Service Development Kit’s architecture
toolkit to develop a range of tests which can be executed either on the V&V store or on the local developer emulator. Test results can be collected and attached to the resulting service package. To enable the developer to process the gathered results for the purpose of debugging or improving service performance, the SDK will be extended with a new Execution Analyzer component in order to assist the developer in maximally using test results to improve is/her service.

The SONATA emulator (son-emu) has been one of the cornerstone achievements of the SONATA SDK, enabling rapid local NS evaluation on multiple PoP setups. This will be further extended in order to support local test execution by the developer, potentially relying fully or partly on a local deployment of the V&V Store.

3.1 SDK Workflow

The 5GTANGO SDK will build further on the successful SONATA SDK workflow approach, and extend it with improved support for rapid and robust development, different platforms, dynamic service management, testing and analysis. The SDK follows an iterative process as documented in Fig. 3.2. The resulting workflow is flexible and allows for a range of different sequences of steps. In addition, some steps are optional. Below, we document the canonical workflow, and indicate potential, alternative workflows.

3.1.1 Workspace and Project Creation

The first phase in development consists of the setup of a workspace and project environment. This provides a basic file/folder structure under which images and descriptors required for the service can be accommodated. In addition, the workspace environment as provided by the SONATA son-workspace tool, provides support for storing authentication profiles for multiple service and or V&V platforms. Subsequent phases involve a range of novelties, and require a more in depth discussion. Therefore each of the next phases will be assigned a dedicated sub-section.

3.1.2 Image Creation

Given the local work- and project space, core NS functionality consists of a range of NFs which will compose the service. The SONATA platform supports both container, and VM-based VNFs, however the process to create those is rather manual, error-prone, and does not stimulate re-use or integration of packet-processing libraries or toolkits such as Click Modular Router, or support for unikernel VMs. Image creation in 5GTANGO aims at improving this issue by introducing an additional tool bundling and automating existing approaches for VM, container, or unikernel image creation. This will allow rapid reuse of existing state-of-the-art functionality, while ensuring compatibility with the 5GTANGO platform. In case the image is already available from a catalog, this step can be bypassed.

3.1.3 Fetch from Catalog

Developers can rapidly start their project by fetching images, descriptors, tests or Specific Managers from an external catalog. This enables easy re-use of existing components, which can be blended into the overall service under creation. Obviously, this step is optional.

3.1.4 Management Component Development

Next to the individual NF images, functionality in order to manage virtualized instances of these images enabling dynamic scaling, or seamless failover is still a matter of highly service-specific efforts.
Figure 3.2: 5GTANGO service development kit’s workflow
by NF developers in any existing NFV platform. 5GTANGO aims to capitalize the SSM/FSM concepts of SONATA, by providing reusable mechanisms across services and NFs enabling scaling, migration and/or failover of components. This involves mechanisms to handle session state during such events. Alternatively, the SM functionality can be downloaded from a catalog.

3.1.5 Descriptor Editing

Once the individual images and management components are available, composition of the service consists of describing these components as well as their interactions (w.r.t. topological or forwarding behaviour) into descriptors. Current descriptor formats are highly varying among existing platforms, although many of those, including SONATA (via the son-editor GUI), have tools to generate them in a user-friendly manner. However, many of these tools only support a limited set of descriptor functionality, resulting into the need for manual editing of YAML or XML files. This is often painful, slow and prone to failures. In 5GTANGO, this phase will be assisted through a Descriptor Generator component easing this process, by providing templates and generators to quickly derive descriptors, as documented in later subsections.

3.1.6 Test Creation

The introduction of the V&V in 5GTANGO, increases the focus on the robustness of services and its components. For this reason, the SDK is extended to enable developing compliance-, functional-, security-, or performance-focused tests through a set of test development support tools. These tests can be executed partly or entirely on a range of platforms, including the SDK emulator, one or more Service Platforms, or one or more V&V stores.

3.1.7 Descriptor Validation

In order to ensure that constructed services and associated descriptors are sound and ready for deployment, validator functionality inheriting from the SONATA son-validate tool enables to check for both syntactical and semantic errors.

3.1.8 Packaging

The format of exchange between main parts of 5GTANGO is the (service) package. This package contains the descriptors of involved components, and optionally images and tests available for the service or its components. Packages can be onboarded on the emulator, one or more V&V stores, or Service Platforms. Depending on the selected platform, the Uploader tool can be used to onboard and/or instantiate it on the emulator or platform.

3.1.9 Emulation

The 5GTANGO project will further extend the son-emu emulator from the SONATA project. This component provides a light-weight local rapid-prototyping environment for debugging developed services and components. The emulation platform will be extended to support the execution of some of the tests, e.g., functional tests since performance tests might not be compatible with local environments. In addition, the emulator will be extended to support 5GTANGO’s extended package format.
3.1.10 Test Result Package Generation and Analysis

Depending on the platform responsible for executing the developed service, tests can be executed, and associated monitoring results can be gathered into a package referring to the unique service package that was tested. The resulting data can be fed back into the SDK Execution Analyzer component for debugging or performance tweaking analysis.

3.2 SDK Components

This section describes all SDK components as documented in the overview and workflow section. We focus on the architectural properties (main subcomponents and interfaces), as well as on the novelties of the SDK components which differ sufficiently from the SONATA SDK.

3.2.1 SDK Descriptor Support

As indicated in sec. 2.4, descriptors of VNFs or network services can contain a large amount of different fields. Currently, new VNFDs or network services have to be written from scratch or copied and adjusted from other VNFs or network services. This makes the creation or configuration of network services cumbersome, time-consuming, and error-prone – especially for large or complex network services.

Therefore, 5GTANGO will introduce tools to support and alleviate the process of creating or adjusting descriptors. The goal is to both minimize the number of manual steps and to simplify the remaining manual steps when creating or adjusting network service descriptors.

3.2.1.1 Minimization of Manual Steps Through Use of Default Values

A considerable part of VNFDs and NSDs is similar across all VNFS and network services. For example, all common VNFs and network services (and even VDUs) have three connection points to the outside world: Input, output, and management. Such repetitive information could be generated automatically, using common default values. This would relieve developers from performing repetitive, error-prone tasks.

For further automation, constituting VNFs should be categorized, e.g., into firewalls, IDS, etc. VNFs of the same category, e.g., different implementations of a firewall, tend to have similar descriptions. Hence, more fields can be automatically generated and filled with suitable default values for these categorized VNFs. For example, resource requirements or monitoring metrics might be similar within each category.

Of course, developers can still modify automatically created descriptions. Nevertheless, the automatic generation of standard fields and descriptions helps avoid errors and reduce development time. It also ensures that developers only deviate from the default if they actively choose to do so, not by mistake.

3.2.1.2 Simplification of Remaining Manual Steps

Clearly, not all information can be generated automatically, even with categorization of VNFs. Developers still need to decide and specify which VNFs or network services are involved in a network service and how they are connected.

However, 5GTANGO aims at simplifying these remaining development steps as far as possible. Developers will be able to specify all relevant information in a high-level way. For example, providing a list of references to all involved VNFs and network service could be sufficient to automatically
create the corresponding VNFDs and descriptions in the NSD. Similarly, it is conceivable that describing the order of VNFs/NSs in a high-level way might suffice to automatically generate the corresponding virtual links and forwarding graph.

### 3.2.2 Packaging Tool

Network service packages are the main artifacts exchanged between 5GTANGO’s SDK, V&V, and service platform (cf. Sec. 2.5) and can contain network services, network functions, and all kinds of additional meta data to describe the service or to support its instantiation and operation. To create these packages, the 5GTANGO SDK contains a packaging tool that allows network service and VNF developers easily generate and handle 5GTANGO compatible packages.

#### 3.2.2.1 Existing Implementation

The packaging tool will be based on the SONATA implementation that is part of the SONATA SDK. This tool supports the generation of SONATA compatible service packages from a SONATA SDK project. During this process, the tool utilizes the validation functionalities to automatically validate the packaged network service, e.g., by validating the involved descriptors against the schemas. Additionally, the tool automatically resolves dependencies and can pull-in external resources that are referenced by the network service project. Finally, the tool can sign the entire package using the private key of the developer to ensure integrity when the package is uploaded to a catalog or service platform. Fig. 3.3 [48] shows the SONATA packaging tool and its interactions with other tools of the SONATA SDK, namely son-access which is used to resolve external dependencies and son-validate that allows descriptor validations.

The tool is implemented in Python and is available under Apache 2.0 license. It offers a user-friendly CLI interface. More information about the SONATA packaging tool and package format can be found in SONATA’s deliverables D3.1, D3.2, and D3.3 [47, 48, 49].
3.2.2.2 Required Extensions

5GTANGO’s packaging tool needs several novel features and extensions compared to the existing SONATA packaging tool. The required extensions are described in the following:

- **Support for new package format**: The 5GTANGO package format described in Sec. 2.5 is a much more advanced packaging format than the simplistic one used by SONATA. Especially the support for different package layers and package sections signed by different parties, require an substantial extension of the existing packaging toolchain.

- **Packaging of different descriptors**: 5GTANGO targets more than one service platform to execute tests or deploy services. Thus the packaging format shall support different kind of service and function descriptors. This generic packaging approach will require the packaging tool to understand and deal with different schema definitions, e.g., OSM service descriptors.

- **More and cleaner APIs**: The existing packaging tool focuses on usage on the developer’s laptop through its built-in CLI interface. In 5GTANGO, not only the SDK will use the packaging tool but also other components, like the V&V, can benefit from using this tool to extract and handle package contents. This will simplify package format changes since only one code base needs to change along with the package format. However, in order to let other components use the packaging tool, it should provide well-designed and easy-to-use APIs. On the one hand, the 5GTANGO packaging tool should expose its internal Python APIs so that it can simply be imported as additional module to other Python projects. On the other hand, an REST API for the packaging tool might be considered so that it can be packaged as a micro service and easily be integrated into the Gatekeeper or the V&V.

- **More features are expected to be developed throughout the project when the 5GTANGO package format evolves.**

3.2.3 Catalogue and Platform Access

In SONATA, the catalogue is part of the service platform, with each developer accessing it through a tool named `son-access` [52]. In 5GTANGO this implementation will have to evolve into a tool that supports the following:

1. Supports other use cases than the developer or the service platform accessing the catalogue.

2. Supports additional interactions between SDK and V&V, e.g., upload new test specifications to the V&V or download signed test results for further processing.

3. Enable the interactions between the developer and the stored information in different ways, like when a given VNF is suggested to the developer based on additional metadata annotations.

The interactions between the SDK and the catalogue can be found in Sec. 6.1.2.

3.2.4 Emulation Platform

This emulation platform was created under the name `son-emu` as part of the SONATA SKD [47, 48, 49] to support network service developers to locally prototype and test their network services in realistic end-to-end multi-PoP scenarios. It allows the execution of real network functions, packaged as Docker containers, in emulated network topologies running locally on the developer’s
The emulation platform also offers OpenStack-like APIs for each emulated PoP so that it can integrate with MANO solutions, like OSM [17]. The core of the emulation platform is based on Containernet [37] which is an actively developed open source project maintained by Paderborn University. The main concepts of the emulation platform have been described in several scientific publications [40, 39, 60].

During the SONATA project, the emulation platform raised a lot of community interest and was finally adopted by OSM and integrated into the third OSM release under the name vim-emu [35]. There it is developed as part of OSM’s DevOps MDG.

### 3.2.4.1 Scope of the Emulation Platform

Fig. 3.4 shows the scope of the emulator and its mapping to a simplified ETSI NFV reference architecture [45] in which it replaces the network function virtualisation infrastructure (NFVI) and the virtualised infrastructure manager (VIM). The design of the emulator is based on a tool called Containernet [37] which extends the well-known Mininet emulation framework [31] and allows us to use standard Docker containers as VNFs within the emulated network. It also allows adding and removing containers from the emulated network at runtime which is not possible in Mininet. This concept allows us to use the emulator like a cloud infrastructure in which we can start and stop compute resources (in the form of Docker containers) at any point in time.

### 3.2.4.2 Architecture

The emulator design follows a highly customizable approach that offers plugin interfaces for most of its components, like cloud API endpoints, container resource limitation models, or topology generators.

In contrast to classical Mininet topologies, emulator topologies do not describe single network hosts connected to the emulated network. Instead, they define available PoPs which are logical cloud data centers in which compute resources can be started at emulation time. In the most simplified version, the internal network of each PoP is represented by a single SDN switch to which compute resources can be connected. This can be done as the focus is on emulating multi-PoP environments in which a MANO system has full control over the placement of VNFs on different PoPs but limited insights about PoP internals. We extended Mininet’s Python-based topology API with methods to describe and add PoPs. The use of a Python-based API has the benefit that developers can use scripts to define or algorithmically generate topologies.

Besides an API to define emulation topologies, an API to start and stop compute resources within the emulated PoPs is available. Von-emu uses the concept of flexible cloud API endpoints. A cloud API endpoint is an interface to one or multiple PoPs that provides typical infrastructure-as-a-service (IaaS) semantics to manage compute resources. Such an endpoint can be an OpenStack Nova or HEAT like interface, or a simplified REST interface for the emulator CLI. These endpoints can be

---

Figure 3.4: Scope of the emulation platform in the simplified ETSI NFV reference architecture
Figure 3.5: Deployment example: Emulation platform that emulates four PoPs and controlled by the SONATA service platform and OSM

easily implemented by writing small, Python-based modules that translate incoming requests (e.g., an OpenStack Nova start compute) to emulator specific requests (e.g., start Docker container in PoP1).

As illustrated in Fig. 3.5, our platform automatically starts OpenStack-like control interfaces for each of the emulated PoPs which allow MANO systems to start, stop and manage VNFs. Specifically, our system provides the core functionalities of OpenStack’s Nova, Heat, Keystone, Glance, and Neutron APIs. Even though not all of these APIs are directly required to manage VNFs, all of them are needed to let the MANO systems believe that each emulated PoP in our platform is a real OpenStack deployment. From the perspective of the MANO systems, this setup looks like a real-world multi-VIM deployment, i.e., the MANO system’s southbound interfaces can connect to the OpenStack-like VIM interfaces of each emulated PoP. A demonstration of this setup was presented at NetSoft 2017 [38]. Are more detailed overview and technical details about the emulator can be found in SONATA’s deliverables D3.1, D3.2, and D3.3 [47, 48, 49].

3.2.4.3 Usage and Vision for the Emulator in 5GTANGO

The emulation platform, with its unique capabilities as described above, offers a couple of possibilities to be integrated into the 5GTANGO environment and tool chain. Besides this, it allows the project to continue the close collaboration with OSM that was started in SONATA. Based on this, 5GTANGO can directly contribute to OSM by extending and improving the emulation platform. Technically a couple of novel usage scenarios for the emulation platform will be exploited throughout the project.

- First of all, the emulation platform will act as a local test execution platform that can be installed locally on a developer’s laptop. The 5GTANGO SDK as well as the V&V can then interact with this emulated environment to execute test cases for single VNFs and entire network services. The benefit of such a local platform are the quick turnaround times for the
developer who tries to fix bugs inside a network service. At the same time, test developers benefit from the availability of an easily accessible test environment. However, due to the nature of such an emulation environment, will the local deployment mostly focus on functional tests rather than on performance tests. Fig. 3.6 shows the envisioned setup.

- Next, the emulation platform can utilize its ability to emulate VIM interfaces (e.g., OpenStack) to be used to test different MANO and service platform solutions. In fact, this use case is also planned for the OSM project.

- Another, less complex, improvement of the emulator will be the extension of its simple, build-in orchestrator to be able to understand the extended 5GTANGO descriptors and service packages.

- Finally, the emulation platform with its Mininet core seems to be a good candidate to perform experiments with the developed slicing solutions.

**3.2.5 Test Definition Support**

Validation and verification of NFV services and components is one of the core innovations provided by the 5GTANGO project. This is embodied by the introduction of the V&V Store, as well as by the additional test support provided by the 5GTANGO service platform. However, the development of procedures enabling verification, validation and performance profiling currently consisting of very service-specific, manual effort.

The 5GTANGO SDK intends to address this gap, by introducing a test developer component, consisting of the following parts:

- A test setup specifier
- A traffic specifier
- A profile & condition specifier
- A test validator
The role and interaction of each of these components is further described in Fig. 3.7. The outcomes of test development result into a test package which is embedded into the service package. The latter can be onboarded and executed on: the SDK emulator, one or more service platforms or V&Vs. Executed tests on these platforms collect monitoring data and the outcomes of a number of validations which can be stored into a test result package.

The role of the different test development components will be exemplified through a virtualized DHCP service. This will make the role of the different parts more tangible, and provides a first, simple case to validate the functionality of the components, once implemented.

### 3.2.5.1 Test Setup Specification

The test setup specifier will provide an environment to quickly describe test setups. This will focus on re-usability, templates and human and machine readability. The Test Specifier will provide means to describe at least the following test setup aspects:

- The service, NF(s) or SSM/FSMs to be tested
- The involved network topology and/or connectivity
- Additional components to create the test conditions (e.g., generation of user requests)
- The resources required in order to deploy above requirements involving platform (e.g., version and functionality in the V&V/service platform) and physical requirements (e.g., CPU, memory or storage requirements)

Note that tests might require that the above aspects change over time. A virtualized DHCP service can for example handle increasing traffic by increasing the number of DHCP server instances behind a load balancer (see Fig. 3.8). A test setup, therefore, should be able to describe the time-dependency in this setup, either by listing discrete setups which need to be evaluated in time, or by describing test events coupled to their occurrence timing. Test execution potentially might use the concept of Test Specific Managers (TSMs) in analogy to the SSM/FSM to control the events and associated changes needed in the environment/platform to steer the test. Potential events might involve:

- Instructing a test component (e.g., traffic generator) to generate other types or rates of events
• Instructing a change in the test setup

• Changing the resources available for the component

3.2.5.2 Traffic Specification

Test setups often include components for generating network traffic in order to test the service that is provided. The network traffic that can be handled is often very specific to the service which is tested. Generating different types or rates of traffic is therefore often a challenge in testing environments. The TrafficSpecifier component aims to ease this by bundling a range of existing tools (e.g., hping, pktgen, scapy, ostinato) into a homogenous environment in which it is easy to:

• Provide highly service-specific traffic generation components

• Provide traffic generation components which can be configured to generate different types of requests across network layers (e.g. from application-layer web requests, to L3/L2 DHCP control traffic requests

• Generate traffic at different throughput rates, enabling performance profiling type of tests

• Reproduce traffic generator configurations as well as generated traffic in a sound manner

3.2.5.3 Profile & Condition Specification

Traditional desktop or cloud software have introduced a number of testing and debugging concepts which, for the moment, have no counterpart in NFV context. The profile and condition specifier aims to change this by introducing functionality to easily specify, monitor, log and inspect:

• NF- and service specific state: Persistent state or volatile variables during the runtime of a service (e.g., routing forwarding table, DHCP server cache, or TCP session state)

• CConditions: Assertions or state of conditions during the runtime of a service (e.g., double use of DHCP assigned address)

• Performance metrics: Quantitative measure of service/component performance at any layer of the network stack (e.g., number of web or DHCP requests per minute that can be handled).

State, conditions or metrics should enable composable and re-usable specification, enabling that the similar conditions can be described in a similar way, and stored for later use. A monitoring framework should enable monitoring and logging time-dependent state as well as metrics, enabling to replay test chronology needed for debugging a service. Time-related logs of performance metrics, together with the description of the environment driving the test (as defined by the test setup specifier) provides a good basis for a performance profile of NFV services or components. Fig. 3.8 shows an example test setup for a scalable DHCP service.

3.2.5.4 Test Validation

Tests involve description of the setup, involved traffic as well as metrics and conditions to be provided. All of these will involve additional descriptors, images and configurations which might contain errors or inconsistencies. Similarly as Service Packages and their descriptors, tests also require validation functionality which ensures that no syntactical or semantic errors are in the test descriptions. The SDK therefore involves a Test Validator component in charge of these. The Validator might be integrated with the already existing Validator part of the son-cli tools.
3.2.5.5 Test Execution & Analysis

Once a test is packaged in addition to the service description itself, it might be onboarded and executed on a V&V/Service platform or emulator. According to the instructions available in the test description, as well as the policies followed by the platform, a range of test results might be gathered and stored into an additional test result package, referring to the unique service it has tested. This package might contain performance recordings (profiles), event logs, conditions checked over time, amongst others. Such data provides an excellent basis for post-processing for either debugging purposes (what went wrong) or performance assessment (how good did it perform). Adequate analysis of such data can be tremendous value for the developer. The Execution Analyser component aims to provide a set of tools for the statistical analysis of test result packages. The following functionality is envisioned:

- Replaying the chronology of recorded events and conditions, enabling the developer to inspect state in a chronological manner which can be mapped against provided input parameters (e.g., network traffic).

- Tools for anomaly detection in recorded logs.

- Tools for statistical model construction & evaluation.

Performance modelling of software-based systems is often a challenging task because of the variance and uncertainty involved in network conditions, input and resource conditions. Performance profiles or controlled recordings of well-chosen performance metrics, together with their environmental setup provide invaluable data to build statistical models of the correlation between input (e.g., traffic and/or events) and output parameters (e.g., performance metrics such as throughput). These models can vary between simple linear models to advanced deep neural networks. Building such data-based statistical models can be a lot more tangible than constructing performance management functionality by hand. The use of such performance characterization models might therefore be very adequate input for constructing service or function scaling management components (SSM/FSM), as depicted in Fig. 3.7.
3.2.6 Monitoring and Profiling Data Analysis

5GTANGO does not only focus on the functional testing of VNFs and network services, but also on non-functional testing. This explicitly includes the execution of performance tests, also referred to as VNF/network service profiling or VNF/network service benchmarking [39, 36]. In contrast to functional tests, which usually produce discrete results, like pass, error, or fail, performance tests produce more complex datasets that contain input parameters, like configurations, resource assignments, and traffic properties, as well as output metrics, like throughput, delay, or application specific performance metrics. To understand these results, more advanced analysis mechanisms will be available in order to support developers, service operators, and MANO systems, to extract the required information from the raw results.

3.2.6.1 Network Functions and Services Profiling

Taking into account the results realised from the set of performance tests, profiling of the developed network functions and services is going to be realised, focusing -among others- on their resource usage profile characteristics. The profiling results are going to be included in the metadata of the associated software. Towards this direction, component-based and graph-based profiling is going to be realised.

The component profiling approach regards the processing of the collected data regarding the performance of a VNF and their compilation in terms of a resources usage profile along with the identification of the main performance metrics that affect the behaviour of the specific VNF. The graph profiling approach regards the extraction of profiling information based on the analysis of the relationships of the VNFs included in a network service (graph). The graph profiling approach combines results from VNF-based profiling and network service-based (graph) profiling. Analysis and identification of how graph relationships affect the overall performance is going to be performed.

Graph profiling is going to support the identification of bottlenecks in specific software components that deteriorate the performance of the overall graph and eventually help software developers to improve this part of software, as well as to examine how different QoS parameters may affect the overall graph performance.

Component and graph profiling regard an offline analysis process taking into account data collected from the various tests performed, as well as data coming from the instantiation of a network service by a service provider. In the latter case, profiling may be performed by the service provider having ownership of the data. In terms of mechanisms, graph-based data analytic frameworks are going to be used.

3.2.6.2 Importance Factors Identification

In the case of composite services, challenges arise given that many different entities (developers, service operators, and MANO systems) are setting their requirements for the overall service. Those entities may have specific preferences for resource parameters and potentially additional parameters that can be monitored (e.g. number of sessions). At the same time, the service operators may update their offerings according to their strategy relationships with the service developer. In this case, among with the datasets produced by the performance tests, identifying the impact of a service parameter appears to become crucial.

To address the aforementioned challenges, we need to set a mechanism that analyses monitoring data and performance information in order to identify dependencies between the parameters and how these dependencies affect the overall performance. The latter will be reflected to the so called “importance” factors / weight factors per parameter. This is in fact an on-line learning process.
that updates and dynamically evolves the profiling analysis, as already mentioned in the previous paragraphs.

### 3.2.7 State Handling Support

Many network functions need to store a certain amount of state. They need to keep track of past events which might influence their current and future behaviour. Examples include the currently open TCP connections flowing through a firewall, and the desired quality-of-service of a particular video stream. The type and the extent of state that a particular network function keeps depends on the purpose of a function as well as on its implementation.

The state is of utmost importance to the proper functioning of a service. It therefore needs to be kept somewhere and should ideally not be lost, even in case of a crash. Storing state in volatile memory in a single location is therefore not enough. State keeping can be realized in different ways. Examples include local files, remote files, databases, and replicated storage.

Besides storing and reloading the state of network functions in order to minimize the impact of a crashing service, state externalization is a crucial feature for migrating, upgrading, and scaling services. In order to migrate a network service from its current location to a new one, its state needs to be transferred together with it. This means that state has to be externalized, transferred, and then reloaded by the service at the new location. Ideally, that transfer process is unnoticeable to the function’s user.

Upgrading a service is similar to migration, with the difference that the location of the function (usually) stays the same, while the software code is changed to a newer version. In a new release of a function, the structure of the function’s state might have changed. New information might have been added and/or some information might have been removed. When doing an upgrade, the state must be transformed from the old structure to the new one. Some transformation component is therefore needed. It is an open architectural question whether this component is an integral part of the function or whether to keep it separate from the function’s main code.

Scaling out functions typically requires spinning up new function instances (e.g., in the form of virtual machines or containers). Similarly, scaling in a function means shutting down some instances. Independent of adding or removing instances, the combined state across all instances of a network function is not affected by these actions. For example, the set of open connections passing through a distributed firewall is not affected by the current number of firewall instances. This independence implies that the state might need to be redistributed across the active function instances when the set of instances changes. The redistribution can happen immediately or gradually (e.g., assign new connections to a new function instance and keep existing ones where they are). Looking at individual function instances, their state needs to be split or merged upon scaling out or in.

In addition to the function-internal state looked at so far, network forwarding state needs to be considered. Network forwarding state refers to the rules which steer network traffic to a particular function at a specific location in the network. When migrating or scaling functions, the network forwarding state needs to be adjusted as well, in line with the changes done to the set and location of network function instances and, more importantly, their state.

A MANO system needs to take into account state management. Although the state itself will be managed by each individual VNF, the MANO system will usually trigger the migration, upgrading, and scaling of functions. It therefore needs to trigger state externalization and recovery in line with service management schedules and needs to consider service level agreements at the same time. 5GTANGO will be looking at how such interaction can be performed. In particular, the service development kit is concerned with the development of VNF and network service management aspects, of which state management is part and needs to be addressed properly. As state management goes beyond the development phase, the developments within 5GTANGO will cover
all phases of the service life cycle, including the development phase, the testing phase, and the operational phase.
4 Verification and Validation (V&V)

The V&V has previously been described in some detail in deliverable D3.1 [12]. The following provides an overview and synopses of the V&V. It explains the internal working of the V&V and shows how the V&V will be integrated into the overall 5GTANGO architecture. For more detail on the V&V, the reader is directed to deliverable D3.1 [12].

4.1 V&V Overview

The V&V provides a Verification and Validation service that tests submitted VNFs or network services to ensure they pass a range of tests. These tests are designed to ensure that the systems which have been tested meet a core set of requirements. These requirements are not fixed but change over time.

4.1.1 How the V&V Helps in Avoiding Technical Debt

The V&V exists in a DevOps world, where network services and the network infrastructure itself are continually changing. New features and functionality are rolled out onto the network on a daily or even hourly basis. At the same time, issues and problems with the network, the network services it hosts or even specific VNFs are identified, and corrected. As this happens new tests are introduced to the V&V which cover newly discovered scenarios or issues which have been encountered during the DevOps processes.

When new tests are introduced the definition of an element which has been “Verified and Validated” changes. A component which passed the V&V process last year, may not pass this year. This is the technical debt.

The network services and VNFs will need to be maintained. To support this, the V&V monitors the set of network services and VNFs it has been asked to verify and the set of tests which are necessary for the network service or VNF to pass. If a test is changed then the V&V will try to retest and revalidate those network services or VNFs which the modified test covered. Likewise when a change is made to a known network service or VNF then it will be retested.

This ensures that when a developer, network operator, or other user obtains a VNF or network service from the V&V they can be sure that its V&V status is valid and up to date, and that the network service or VNF has passed the V&V process. Simply put the item is free from such technical debt.

4.1.2 Multiple Test Environments

The V&V runs the network service or VNF in a test environment. In this environment it stresses the network service/VNF (known as the System Under Test) and observes the results. The V&V could test network services or VNFs which have been prepared for any runtime environment. The runtime environment for network services or VNFs is often an orchestrator. There are a number of differing orchestration environments available, both in the open source world and also in the proprietary world. To ensure that the V&V is relevant, it must support a number of environments. To this end the V&V has a set of Test Platform Management layers which abstract away the test
environment and provide a framework which is easily extendible to other platforms and runtime environments.

4.1.3 Metadata and Packages and Continuous Testing

The V&V considers that all network services or VNFs are packaged with a set of metadata which identifies information about that system. This is used to identify the functionality that the associated software provides. Similarly the V&V considers that all tests are accompanied with a set of metadata. This metadata identifies what type of functionality the test provides, and this can then be matched to the metadata provided with each network service or VNFs. Overlaps in the data would indicate that the test should be run against the overlapping network service or VNF.

The metadata describing a network service or VNF is tightly coupled with the network service and 5GTANGO assumes that once packaged together the “metadata and network service/VNF” set becomes one atomic item; the metadata cannot be modified, and likewise the software of the network service/VNF cannot be modified separately. If either of these are modified then the V&V considers that the network service/VNF combination becomes a new network service.

Tests are treated differently. To support the rapidly changing DevOps world the V&V considers the metadata of a test and the test code itself as two closely related but independent items. Changes to a test’s metadata and separate changes to the test themselves are allowed.

To support automated testing and the avoidance of technical debt, any changes to the metadata of a test, the test itself, or the addition of a new network service/VNF will automatically trigger a new set of tests (test passes).

4.2 V&V Workflow

This section provides an overview of the major functionality that the V&V provides and details how each of the internal components works together to provide the necessary functionality.

It is envisioned that people who fulfil the following roles will want to use the V&V:

- Developers creating VNFs
- Developers creating network services composed of VNFs which have previously been certified as V&V compliant
- Network Operators
- Third party V&V test houses
- Administrators / Super users

For the purposes of this section all of these roles are encompassed by the token “User”. The specific set of role types, their access permissions and hence scenarios in which they can act needs to be defined for the V&V and will follow in a subsequent update. This update will introduce a 5GTANGO wide taxonomy of user roles.

4.2.1 Upload of Network Services/VNFs with Metadata for Testing

The user has created a network service and is uploading it to the V&V (Fig. 4.1). Note that there is no way to modify a network service/VNF once it is uploaded. Instead, the user must upload a completely new version of the network service/VNF.
4.2.2 Manual Triggering of Test Pass

The user can request that a specific test should be executed against a given network service or VNF. To do this the user can enumerate the available network services or VNFs, identifying the service they wish to test, then query to the V&V to see the available tests. Alternatively, the user may already know the GUID of the service and invoke it directly. Once a specific test has been identified, the user can request that this network service or VNF is scheduled for testing. Fig. 4.2 illustrates this workflow.

4.2.3 Upload of Tests and Metadata

The test developer has previously created and validated a test and is uploading it with the metadata to the V&V for the first time (cf. Fig. 4.3).

4.2.4 Modification of Tests or Test Metadata

The test developer wants to update a test already available in the Test Repository. He needs to retrieve the unique identifier of the test to be modified in the test repository and make a change request towards the test under this identifier (cf. Fig. 4.4). Alternatively, he needs to update the metadata associated to the test.

4.2.5 Automatic Test Execution via Modified Event

The test invoker has received a notification of a change or other event which should trigger a set of automatic tests. The list of events that can cause these changes are (also illustrated in Fig. 4.4): Adding (uploading) new tests, test modification (test or the test’s metadata), and network service/VNF upload.
Figure 4.2: Manual test triggering

Figure 4.3: Test upload
Figure 4.4: Modification of tests or test metadata
4.2.5.1 Test Modification and Test Addition

The sequence diagram in Fig. 4.5 illustrates what steps the Test Invoker performs when one of the following events are received:

- New test has been added to the V&V
- An existing test has been updated
- The metadata associated with a test has been modified, increasing the scope (set of all network service or VNFs which would be covered by the metadata) of the test

The sequence diagram Fig. 4.5 highlights the need for a search / query API on the network service catalogue that would allow a quick search for matching network service/VNF based on a set of supplied metadata.

4.2.5.2 Network Service Upload

The sequence diagram in Fig. 4.6 illustrates what steps the Test Invoker performs when a network service upload event is received.

4.2.6 Obtaining Test Results

The user wants to obtain the results of a test pass. A user can either be notified when a test run has been completed, or can search and enumerate the already stored results and select one to obtain (see Fig. 4.7).

4.2.7 Tested Package Push/On-boarding

The user uploads an network service or VNF and it is automatically scheduled for testing, tested, and results are stored inside the V&V. The V&V integration component checks to ensure that the all tests are passed before uploading the network service/VNF to an external (to the V&V) catalogue. The external catalogue could be:

- A 3rd Party catalogue
Figure 4.6: Network service upload

Figure 4.7: Obtaining test results
Figure 4.8: Pushing/On-boarding a tested package

- A catalogue with the 5GTANGO service platform
- A catalogue within as yet known service platform

At a high level the process is rather simple as illustrated in Fig. 4.8.

### 4.2.8 Automated Package Retest/Push

The V&V can be configured to publish the NS/VNF which pass the V&V tests directly to an external (non-V&V) catalogue for future deployment or developer reference. However the V&V is continually testing all the NS/VNF, this in turn suggests that a number of previously tested and V&V passed NS/VNF which have been published to the external catalogue will now no longer be valid. Therefore the V&V needs to adopt an update policy to keep the external catalogue in sync with the current test state of the NS/VNF. The following rules outline how this could be achieved:
1. IF the NS/VNF fails the V&V test AND it does NOT exist in the external catalogue THEN do NOT publish it to the external catalogue.

2. IF the NS/VNF passed the V&V test AND it does NOT exist in the external catalogue THEN it is published to the external catalogue.

3. IF the NS/VNF passes the V&V test AND it DOES exist in the external catalogue THEN publish an updated NS/VNF with updated test data included, showing the “PASSED” state.

4. IF the NS/VNF fails the V&V test AND it DOES exist in the external catalogue THEN publish an updated NS/VNF with updated test data including showing the “FAILED” state.

4.3 V&V Components

The major components which comprise the V&V are illustrated in Fig. 4.9 and described in further depth in the following.

4.3.1 Component Description

The components depicted in Fig. 4.9 are group in to related areas and described below.

4.3.1.1 Public API End Point and V&V Integration

- **“AAA”** (Authentication, Authorisation and Accounting): The V&V does not implement its own user management solution, instead it provides as a plug-in architecture allowing an existing user management solution to be supplied. The AAA is the end up validation that communicates to the shared authentication system.

- **V&V API Gateway**: The API gateway provides a common entry point for all V&V API calls.
• **V&V Integration**: This provides a set of integration functions that allows the V&V to provide an automated upload, test (V&V passes) and the upload the results and verified NS up to either a service platform’s catalogue or to a 3rd party catalogue.

• **V&V GUI**: This component provides a user friendly way to exercise the V&V APIs, and Integration elements.

### 4.3.1.2 V&V Storage

• **Test repository**: The test repository will allow tests and their associated metadata to be managed.

• **Network service catalogue**: For the V&V to be able to schedule and execute tests, it must know from where to obtain the network service for testing, and the state of the network service (modified, etc.). To fulfill this, the V&V supplies a network service library which stores the network service ready for deployment in the network service environment.

• **Test results repository**: The V&V needs to record the results of each test pass, and of aborted tests. This component stores such details and ultimately makes them available to users and other software entities.

### 4.3.1.3 Test Execution

• **Test invoker**: This component watches for events which can trigger automated test execution. These are triggered by changes to test environment configuration, test metadata, test, network service, or network service metadata. The test invoker receives notifications of changes and then generates the list of tests which need to be executed and passes that on to the test engine.

• **Test engine**: The test engine a network service testing specific workflow engine. The test engine is responsible for executing tests and controlling the queue of tests due to be executed. The test engine generates its own events when tests are being executed or queued (queued, executing, and complete).

### 4.3.1.4 Interface with Multiple Test Platforms

• **Test platform manager**: This provides the test engine with an interface for network service deployment. This component contains a lot of the logic required to actually deploy, run, and tear down network services. It also provides a high level interface to allow the collection of advanced metrics. The test platform manager supports multiple orchestration platforms. The details of this are provided in D3.1.

### 4.3.2 Test Repository and Test Result Repository

This section introduces the test repository and test result repository.

#### 4.3.2.1 Test Repository

This component’s role is to provide a centralized storage for all the abstract tests that are available in one V&V platform. It provides the following capabilities:
• **Storage of test specifications and metadata.** The test specifications are written in a platform independent language (TTCN-3) along with the associated metadata, such as test categorization and scope. This extra information is useful for discovery, selection and definition of the execution priority of tests. A unique identifier (UID) is attributed to each test specification and its associated metadata stored in order to facilitate all the management upon the tests.

• **Test management.** The test repository provides APIs for V&V internal components to manage test specifications and the metadata. The most typical test management operations are CRUD (Create, Read, Update and Delete).

• **Test discovery.** The repository provides the discovery of tests based on searching criteria, such as a filter that selects only the tests with “functional test category” in its metadata.

• **Test modification notification.** The test repository generates events (notifications) when a test specification or its associated metadata is changed. These change events can be used to trigger the re-running of tests on already validated NSs that are stored in the NS catalogue.

4.3.2.2 Test result repository

The information that is needed in the test results includes (more details of the test results content are provided in D3.1 [12]):

• Information about the test plan and test profile(s) applied on the target system under test (SUT)

• Test trace-ability that provides the matrix between the requirements and the tests that check the requirements

• The test environment configuration for the test result comparability purpose

• The test results or conclusion

Tbl. 4.1 shows the structure of a test result stored in the repository.

<table>
<thead>
<tr>
<th>Test result id</th>
<th>Target SUT</th>
<th>Executed test</th>
<th>Test profile or test plan</th>
<th>Covered requirements</th>
<th>Test environment configuration</th>
<th>Test result or conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

There is no assumption on the number of test result repositories that can be attached to one single V&V platform. One repository is sufficient in most of the cases because the test results are retrievable and search-able regarding the SUT, the type of test (i.e. functional or performance), the test execution time, etc. However, a V&V provider is free to make their own decision on the number of test result repositories. It provides the following capability:

• **Gather the test results related to one single NS and wrap them as a retrieveable package.** The package is made based on the same SUT and the same test campaign (a test campaign refers to a “pass” of a set of tests that covers one or more test profiles).

• **Test results management**, such as CRUD operations on a single test result or a test result package.
4.3.3 Test Engine

As mentioned in D3.1, the test engine interacts with the test platform via the test platform manager to provision, test, and shutdown a VNF or network service.

A test pass is an execution of a test against a network service or VNF. The entire V&V process will include multiple Test Passes.

The test engine is essentially a workflow engine that maintains a queue of pending tests, an interface to multiple test environments via the test platform manager, and supports a number of differing test technologies via the test plugin manager. The general architecture of the test engine is shown in Fig. 4.10.

4.3.3.1 Test Management

The test engine also provides the ability to manage the test passes. Test passes can take a long time to execute, it is therefore important to be able to provide information on the status of a test pass. It also gives users or other entities the ability to cancel an executing test.

Manual Triggering of a Test Pass

Please see Sec. 4.2.2, where this action has already been detailed.

Test Execution and State Notification

Once a test has been scheduled for execution, the state of this test can optionally be reported back to the user which requested the test pass. The states in Tbl. 4.2 exist and are indicated to the caller (see Fig. 4.11). Depending on the state the test pass is in, the user may be able to cancel the test.

Note: The results of the test pass, “the VNF/network service passed the test”, “failed the test” are separate to the state of the test pass as it is being executed.

The following sequence diagrams illustrate how the Test Engine responds to requests for test execution, test state queries and test cancellation. The entity invoking these actions can either be a user, via the V&V API or it could be another system component, like the Test Invoker, or V&V Integration components. For simplicity “User” is simply used as the actor.
Table 4.2: Test state overview

<table>
<thead>
<tr>
<th>Test state</th>
<th>State description</th>
<th>Actions permitted</th>
<th>Results available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test queued</td>
<td>The test has initially been queued.</td>
<td>Cancel test</td>
<td>No</td>
</tr>
<tr>
<td>Test preparation</td>
<td>The test engine has selected the test and the engine is preparing the test environment to commence testing.</td>
<td>Cancel test</td>
<td>No</td>
</tr>
<tr>
<td>Test executing</td>
<td>The test is currently being executed.</td>
<td>Cancel test</td>
<td>No</td>
</tr>
<tr>
<td>Test shutdown</td>
<td>The test pass has completed, and the results are being stored.</td>
<td>Nothing</td>
<td>No</td>
</tr>
<tr>
<td>Test complete</td>
<td>The test pass has been completed and the results have been sent to the test results repository for later retrieval</td>
<td>Nothing</td>
<td>Yes</td>
</tr>
<tr>
<td>Test Cancelling</td>
<td>The test pass has been instructed to stop, or to cancel and the Test Engine is in the process of cancelling the test</td>
<td>Nothing</td>
<td>No</td>
</tr>
<tr>
<td>Test abort</td>
<td>The test pass has been skipped (cancelled by user, or error occurred)</td>
<td>Nothing</td>
<td>No</td>
</tr>
</tbody>
</table>

**Test State Query**

At any time during or after a test pass, the user may query the status of the test pass (Fig. 4.12). If the test pass is not found, an error response is returned.

**Test Cancellation**

As illustrated in Fig. 4.13, an interface exists which allows the test engine to cancel tests. If the test cannot be found or if the test pass is now in a state in which it cannot be cancelled, then an error message is returned.

4.3.3.2 Test Execution

The test engine is also responsible for executing test passes. As this execution requires heavy collaboration with the service platform, it is detailed in the corresponding interaction section (Sec. 6.4).

4.3.4 Test Platform Manager (TPM)

This component’s role is to provide a high-level set of logical workflow steps to the test engine. It then translates these into API calls to one or more service platforms. The TPM supports a wider set of interfaces than just service platform integration, it also supports advanced metric collection and processing. Collecting metrics from a system being verified and validated is essential and can require deeper integration with the service platform than simply providing a “driver” like interface.

4.3.4.1 Test Platform Manager Design Goal

The goal of the TPM is to provide a framework which can quickly and easily support multiple test platforms, from the SONATA emulator environment through to OMF, ONAP, or other yet to be defined service platforms. To enable this quick support, the TPM breaks down the work necessary to support these service platforms into three key layers (also see fig. 4.14.

This layering and design pattern is a replication of the Logical Device Driver (LDD) / Physical Device Driver (PDD) pattern often seen in operating systems [1], it allows a separation of concern and allows the TANGO project to focus on the logic required by the workflow engine. The LDD focuses on the logical interaction with the PDD (device driver), the PDD is the entity which communicates with the actual service platform. So if a non-MANO compliant service platform is
Figure 4.11: Test States
selected in the future it is possible to swap out layer 2 (logical device driver) and layer 3 (physical device driver). But if multiple MANO compatible service platforms are to be supported then this can be achieved by simply supplying a new layer 3 (physical device driver).

Additionally this break down allows the V&V to support multiple drivers for a single service platform / test environment. Although not illustrated below, it would be possible to split the drivers, allowing for separate drivers and technology to be use for metric collection and for service platform interaction. For simplicity the following simply refers to a single driver for the service platform.

For the Test Platform Manager, this breaks down as follows:

- **Layer 1 (TPM):** The north bound interface connects to the test engine. This provides an interface specifically designed to support the test engine's workflow, e.g. an interface which will “start test”, but under the hood it invokes Layer 2 interfaces to carry out this work.

- **Layer 2 (Orchestration Driver Framework API):** The middle - logical layer. This provides high level functions, for instance with the example above, these interfaces may include one to provision an initial service platform instance, and a second interface which will provision a NS, this will require multiple calls to the 3rd layer, to push the NS to the service platforms catalogue and then ask the service platform to deploy it. This mirrors the Logical Device Driver concept.

- **Layer 3 (Orchestration Driver Framework):** The traditional driver interface. This provides basic service platform agnostic wrapping to common functions, e.g. the interface to upload a NS to a service platform's catalogue is going to exist for many MANO compliant,
Figure 4.13: Test cancellation

Figure 4.14: Layers of the test platform manager
and even many non-compliant orchestration platforms, however the exact interface can and often does vary. This mirrors the Physical Device Driver concept.

4.3.4.2 Test Execution

The best way to describe the TPM is to examine how it supports the execution of a test. The next sections of this document will describe the steps the test engine goes through as it executes a test. This sequence assumes that the test engine receives a test to be scheduled for execution. This test could be a result of either a manual test request from a user, or via an automate test schedule, triggered by the Test Invoker. The test request is always placed on the internal Test Engine queue.

As discussed in the Test Engine section the Test Engine publish test state information. The following sequence diagrams map to the following 3 states and illustrate the work the Test Engine, and primary the Test Platform and the management frame undertakes. The remaining states are not shown as this don’t involve any significant interaction with the Test Platform Manager. The three states are:

1. Test Preparation
2. Test Execution
3. Test Shutdown

Test State: (1) Test Preparation
In test preparation the test environment must be setup, this involves starting an instance of the service platform (not shown), if needed. It also involves populating the service platform’s catalogue with the SUT and with any additional test framework packages. Once this is complete the services can be instantiated, and only then is the Test Engine ready to actually start executing the tests.

Test State: (2) Test Executing
Once the environment is configured and setup the tests can be executed (Fig. 4.20).
Figure 4.15: Test preparation - shutting down services

Figure 4.16: Test preparation - test environment catalogue management
Figure 4.17: Test preparation - pushing new services

Figure 4.18: Test preparation - starting services
Figure 4.19: Test preparation - configuring data collection

Figure 4.20: Test executing
Test State: (3) Test Shutdown

Once the tests are complete the Test Engine and the Test Platform Manager need to shutdown the SUT and test service and then return the test environment’s service platform to a state where it can be used for another test. This follows the inverse of the Test Preparation.

Shutdown Metric Collection First step is shutting down any supporting infrastructure which is collecting metrics (Fig. 4.21).

Shutdown Running Services The SUT and supporting test services need to be shutdown (Fig. 4.22).

Test Environment Catalogue Cleanup The final step is to clean up the test environment’s service platform (Fig. 4.23).

4.3.5 5GTANGO Catalogue in the V&V

The 5GTANGO catalogue deployment in the V&V is shown in Fig. 4.24. The V&V platform will use an instance of the 5GTANGO catalogue for its functions. As illustrated, the catalogue in the V&V platform will store the untested VNFs/network services and their metadata (descriptors), which come from the SDK. The test engine within the V&V component will acquire those for testing. The results that are generated, will be stored in a test result repository. Then, the VNFs/network services and metadata (descriptors, including the V&V test results) can be pulled from the V&V platform through its API. The service platform can only retrieve VNFs/network services (and their metadata) that have passed the V&V tests.
Figure 4.22: Shutdown running services

Figure 4.23: Test environment catalogue cleanup
Figure 4.24: V&V and Catalogue interaction
5 Service Platform

The service platform in the 5GTANGO architecture (see Fig. 2.1) provides the service and function orchestration features, plus all the needed complementary and supporting features, like slice management, policy and SLA management, user access management and infrastructure adapter. These complementary features mentioned were considered in order to enhance the existing solution, as well as better support the vertical’s requirements.

5GTANGO’s service platform will support two distinct forms of deployment:

1. As a test platform for the V&V platform
2. As a production Service platform

These two forms of deployment imply a design that supports two very different sets of user roles, which will be addressed in detail next.

The service platform’s high level architecture is shown in Fig. 5.1. Each one of these modules will be addressed in the upcoming sections.

The following sub-sections describe in more detail each one of these modules, starting by the description of the needed workflows, from which components are derived.

5.1 Service Platform Workflows

This section describes service platform end-to-end workflows, from which its components will be derived. Some of these workflows and components exist already in SONATA ([1]), but are being adapted to 5GTANGO’s requirements.

5.1.1 User Management Workflows

Given a 5G environment, in which multiple service developers will need access to a service platform, submitting his/her services, and the multiple forms a service platform can be deployed, either serving a V&V platform as a test platform or as an independent platform, different kinds of users playing different roles, must be registered and authenticated. Adding to that, and considering a modular platform design based in micro-services, the service platform will also have to support the registration and authentication of those micro-services, thus guaranteeing that only those micro-services that are known can provide services to the Platform.

For the sake of simplicity, we consider here the generic user registration, which will be detailed later in the more specific work packages. User management is often neglected until the very last moment, when we want to demo different roles. Adding user authentication in a later stage costs much more time and effort than having it in mind from the beginning Therefore this section. This component is generic and can be instantiated in other components of the 5GTANGO platform.

5.1.1.1 User registration

Users must register in the platform before being able to use it. Different user roles will be given in a secured way to the different users registering with the platform.
5.1.1.2 User authentication

To use the service platform, any user has to obtain a valid (i.e., authenticated) token. Fig. 5.2 shows the corresponding sequence diagram. The returned token will have a limited life-span (configurable).

5.1.1.3 Features Registration

Service platform features must be registered, before being allowed to be used by the service platform’s users (authorization, see below). Features are authorized by user role.

In order for the service platform manager user role to have full flexibility in (re-)defining which features are available, the service platform’s GUI may provide a feature for managing those features. Otherwise, a simple file-based configuration of the available features can be provided instead.

5.1.1.4 User authorization

User actions on the service platform features are validated (authorized) before being executed. Fig. 5.3 shows the corresponding sequence diagram. Step 8 is the call(s) that is(are) made to the micro-service(s) that implement the authorized feature.

5.1.2 Network Services and Virtual Network Functions

This section covers the service platform workflows that deal with network services and the VNFs that are part of those network services.

We start by describing how network services/VNFs reach the service platform’s catalogue and then we describe the network service lifecycle, after which we cover the different operations of the...
Figure 5.2: User authentication on the service platform

Figure 5.3: Service platform’s user authorization
network service instance life cycle. Workflows covering how an network service instance can be monitored closes the section.

### 5.1.2.1 Package On-boarding

Network services and VNFs are on-boarded on the service platform as being part of a package. Packages have other assets than services and functions (e.g., test specifications), which are treated in Sec. 4.

After a package has been successfully on-boarded, the service it describes becomes ready for instantiation. In the following sequence diagrams it is assumed that users are already *authenticated* and *authorized*.

#### Package is Pushed into Service Platform

The package on-boarding scenario implemented in SONATA ([1]) is the SDK pushing the package to the service platform In 5GTANGO this might also be the case, with the V&V also being able to push the package to the service platform. This is described in Fig. 5.4. This scenario illustrates the case when the V&V platform and the service platform belong to the same entity (the service provider), so there is a strong degree of trust between them.

#### Package is Pulled from the V&V Platform

An alternative scenario for the package on-boarding is a service platform manager to select a package from the V&V Platform, which is then pulled by the service platform. This is described in Fig. 5.5. This scenario illustrates the case when the V&V platform is independent of the service platform (i.e, belong to different entities), so there is a weak degree of trust between them.
Figure 5.5: Package on-boarding (pulled from the V&V Platform)
5.1.2.2 Network Service Lifecycle

Successfully on-boarded network services can then be instantiated. We are assuming the state diagram for a service instance as shown in Fig. 5.6. An instantiation request takes some time to be executed, during which no other event can be executed. This is modelled by the Instantiating state. When the instantiation is finished, the service instance goes into the Instantiated state, from which it can be updated (Updating state) or terminated (Terminating state).

When the service is Instantiated (a super-state), by default it starts as being Stopped. While Stopped, the instance can be configured (by receiving the configure event, and still staying in the Stopped state), terminated or run (goes into a Running sub-state). While in this state, the event stop returns it to the Stopped state. The service instance can also be configured while Running, depending on the specific service.

Sec. 5.1.2.2, Sec. 5.1.2.2 and Sec. 5.1.2.2 below detail the actions taken in these states.

Instantiating a Service

The value of services in a service platform reveals only when they are instantiated.

We assume that customers are the ones requesting the instantiation of a service, but it can be easily generalisable to being an Operation Support System (OSS)/Business Support System (BSS) command, or any of the other 5GTANGO platform components.

Note that we are separating the instantiation of a service from the starting of a service instance (see the next section). This separation gives the 5GTANGO service platform an extra degree of flexibility in the control of the service instance lifecycle, allowing (e.g.) the configuration of the service instance before it starts.

For the sake of readability we have opted to separate the sequence diagram in two, shown in Fig. 5.7 and Fig. 5.8. In Fig. 5.7 the interaction goes from the customer to the MANO framework, the network services orchestrator (see [26]). The customer uses the GUI to choose one of the services to be instantiated. The current implementation takes into account two kinds of services, specified in the Network Service Descriptor (NSD):

- **Public services**: By default, if nothing is written in the NSD, or if its licence_type is defined as being public, any Developer can download the service’s meta-data and any customer can instantiate it.
• **Private services**: Indicated in the NSD as a *private licence type*, together with a URL to be used to validate the licence. For a developer to see the service’s meta-data or a customer to instantiate it, he/she must go into the licence store and buy a licence (for simplification, money transactions are out of scope of the service platform, but can easily be implemented).

So, instantiable services are those that are either *public* or *private* and *licensed* to that customer. Chosen the service, it’s NSD (and VNFDs) are grabbed from the *catalogues* and passed to the MANO framework to execute the instantiation.

When the MANO framework receives the service instantiation request, it calculates the *optimal placement* of the service instance, and requests the Infrastructure Abstraction for the needed resources.

After the VIM/WIM return the resources effectively allocated, the service instance record is saved in the Repositories and the Monitoring Manager is notified with the monitoring parameters that need to be collected.
Figure 5.8: Service instantiation (2)
Run, Configure, and Stop a Service Instance

Actions changing the lifecycle of an instantiated service are received by the MANO framework, decomposed into commands to each VNF instance that are part of the service. These VNFs are made up of Virtual Machines (VMs). The generic instantiated service lifecycle sequence diagram is shown in Fig. 5.9.

The customer selects in the GUI one of his/her instantiated services and applies a command (e.g., run, stop, config, etc.). Note that available commands depend on the concrete service configuration and current state (e.g., update, an example of a config command, can only happen when an actual updated version of the service is available). The selected commands reach the MANO framework through the Gatekeeper and the MANO framework executes them for each one of the Virtual Machines (VMs) that implement the service to be ran, configured or stopped.

Note that this instantiated service lifecycle management might interact with the Element Manager (EM) (see [25]) of each one of the VNFs that are part of the service, if they are present.

Note also that configuring a service instance might include more complex tasks such as scaling, migrating, etc. Service (re-)configuration commands can have two sources: those that can be triggered by the user from the GUI (e.g., updating a service instance) and those that are triggered from within the service platform (e.g., the policy manager notices the violation of a policy, which triggers a scaling event for the service instance).

If the Policy Manager informs the MANO Framework that a network service is running out of resources (i.e., one of the VNFs of that service is exhausting its assigned resources), the MANO Framework needs to dynamically reconfigure the network service to guarantee service continuity to its users. Therefor, 5GTANGO will investigate the implementation of scaling and migration workflows in the MANO Framework. A scaling workflow provides more resources (e.g., an additional CPU) to a single VNF instance (scaling up), or creates additional instances of that VNF to overcome shortage of resources (scaling out). In the second scenario, the connectivity between the VNFs in the service needs alteration, so that the load is balanced among all the available VNF instances.

Once the MANO Framework notices that a scaled service has too many resources assigned compared to its load (e.g., a VNF is only using 5% of its assigned CPU power), a scaling down (lowering the resources of a VNF) or a scaling in (reducing the amount of VNF instances) workflow can be executed. A different solution for a shortage of resources might be obtained through the migration of a VNF. By relocating a VNF that is exhausting its resources to a different server, more resources might be available to it. In such a scenario, the MANO Framework needs to instantiate a new instance of the VNF on the new location and provide a state sharing protocol, so that the old VNF can inject its gathered state into the new VNF. More details on scaling and migration workflows will be included in D5.1.

Terminating a Service

Only customers who have requested the instantiation of a service can terminate that service instance. This terminate command can also be issued by a Operation Support System (OSS)/Business Support System (BSS), or any of the other 5GTANGO platform components.

For the sake of readability we have opted to separate the sequence diagram in two, shown in Fig. 5.10 and Fig. 5.11. In Fig. 5.10 the interaction goes from the customer to the MANO framework, the network services Orchestrator (see [26]).

The customer uses the GUI to choose one of the service instances to be terminated. Chosen the service, it’s NS records (and VNF records) are grabbed from the Repositories and passed to the MANO framework to execute the termination.

When the MANO framework receives the service instance termination request, it:
Figure 5.9: Instantiated service lifecycle commands
Figure 5.10: Service termination (1)
Figure 5.11: Service termination (2)
1. Requests the service to terminate, which means requesting each VM supporting each one of the functions of the service to terminate.

2. Requests the Infrastructure Abstraction the release of the allocated resources, which is passed to the VIM/WIM.

After the VIM/WIM returns the released resources the service instance record is updated in the Repositories and the Monitoring Manager is notified about the parameters that do not need to be monitored anymore.

5.1.2.3 Network Service Instance Monitoring Data

In the 5GTANGO ecosystem, a plethora of entities (either persons or automated processes, such as service platform administrator, service provider or service developer, specific FSMs and/or SSMs, SLA Manager, etc.) might be entitled to request monitoring data from instantiated network services for totally different purposes. Thus, different ways of accessing and consuming monitoring data is mandatory in 5GTANGO.

This section provides the sequence diagrams, along with explanatory text, for each one of the three foreseen ways to provide monitoring data collected from VNFs and network services, namely: 1) via alerts from 5GTANGO message bus, 2) via Monitoring Manager RESTful API and 3) via streaming monitoring data through websocket server.

Receive Alerts from 5GTANGO Message Bus

One of the conceptually central components of 5GTANGO service platform is the message bus that offers the means for communication between service platform components.

This message bus is utilized also by the monitoring framework to inform subscribed entities about alerts. As shown in Fig. 5.12, the Monitoring Server collects data periodically from the probes installed within the VNFs. The data is processed and calculated against the set of rules and thresholds, creating alerts upon violations. These alerts are sent to the message bus and the entities are able to receive them in case they are already subscribed to the particular topics.

It is highlighted that the term entity has been selected in this case, since the subscribed users could be specific FSM or/and SSM plugins that receive the alerts and act accordingly or even the Policy Manager that, based on the alerts that it receives, enforces different policies and instructs the responsible components to act on the network service.

Collect Monitoring Data from Monitoring Manager API

The monitoring manager that has been implemented in SONATA and will be utilized and extended in 5GTANGO, offers a rich RESTful API for the users to communicate with in order to receive monitoring data from their network services or their respective VNFs. The interested reader can refer to [2].

Since the components comprising the monitoring framework are internal to the service platform, all information exchange between the users and these components passes through the gatekeeper, as shown in Fig. 5.13.

Upon the need of the service provider or the service developer to collect monitoring data for a specific set of metrics with respect to a specific network service or VNF, the user sends a request to the gatekeeper that is forwarded to the Monitoring Manager, asking for the list of metrics available for this specific VNF or network service. When the user receives this list, then it selects the metrics she is interested in, she sets the time frame that requests data for and sends the request to retrieve the data from the time-series database of the monitoring framework.
Figure 5.12: Receive alerts from 5GTANGO message bus

Figure 5.13: Collect monitoring data from Monitoring Manager API
Request Streaming Monitoring Data through Websocket

As shown in Fig. 5.14, the monitoring framework supports the collection of streaming data via a websocket. This is highly convenient for the service developers in order to timely debug the service, while it provides a way to measure its performance.

The user sends a request of the metrics (for a particular VNF or network service, as in the previous case) she is interested in and the time frame and she receives a URL that she can connect to in order to collect the requested data.

5.1.3 Network Slice Lifecycle

This section describes the Network Slice Lifecycle Management (LCM) operations. Slices are initially on-boarded in the service platform using Network Slice Templates (NST), making them ready for creation. Using the NST, multiple Network Slice Instances (NSI) can be created. During runtime, NSIs can be scaled, adjusting the slice (scale-out/in) to the desired performance level. Finally, NSI can be terminated, terminating all functions and services constructions and releasing the associated resources. This life cycle is depicted in Fig. 5.15.

5.1.3.1 On-boarding a Network Slice Template

Network Slices are on-boarded on the service platform using Network Slice Templates (NST). Templates describe how a slice instance can be built by using ETSI NFV pieces as building.

According to NGMN [5], 5G Americas [6] or ETSI NFV [20], a Network slice Instance (NSI) can be defined by combining a list of one or more network services. Eventually, other constructions, such as VNFs, connectivity, or others may be used for the purpose of this integration.

Once a NST is on-boarded, it becomes ready for creation. Based on this template, multiple NSIs can be created with similar characteristics.
Figure 5.15: Slice Lifecycle Management

In the following sequence diagrams it is assumed that users are already authenticated and authorized.

The Slice Manager (SM) is the service platform new component, responsible to manage the lifecycle of Network Slices. The on-boarded NSTs are stored in the Catalogue after previous validation. The Slice template on-boarding workflow is depicted in Fig. 5.16. Depending on the validation level to be performed, the MANO Framework could be also involved, to verify whether the involved NFV building blocks (network services) are available.

5.1.3.2 Creating a Network Slice Instance

Once a Network Slice Template (NST) is on-boarded, multiple Network Slice Instances (NSI) can be created using this template. The Network Slice instantiation triggers the instantiation of the list of Network Services (NSs) defined in the NST, which are used as building blocks to compose the NSI. The multiple NSs may need to be combined to form a meaningful network/service.

NSIs are usually large-sized constructions, capable to provide end-to-end (e2e) services to end users. They are delivered to NSI customers in a SliceaaS model, hiding the complexity of underlying resources. NSI clients can only focus on the network/service level management and commercial exploitation.

As an example, an NSI can implement a full Mobile Network, which can be composed by two Network Services (NSs): C-RAN (radio part) and Core Network (centralized part). The NSI customer can be an MVNO (Mobile Virtual Network Operator) with basic network management and commercial capabilities. The NSI can include an NMS (Network Management System) tool to manage the network components, hiding the underlying resources.

OSSs/BSSs is the entity that usually triggers the Network Slice instantiation, although it may also be triggered automatically (autonomous behaviour). The NSI creation workflow is depicted in Fig. 5.17.

Note: For the sake of simplicity, the workflow does not show the situation where any NS cannot be deployed (e.g. lack of resources). In this case, the NSI creation is aborted and all NSs already deployed are terminated.
5.1.3.3 Scaling a Network Slice Instance

After a life time period, a NSI can be adjusted, in order to accommodate the performance required. NSIs can be increased in their performance (scale-out) to cope with additional service/network load, or decreased in their performance (scale-in) when the installed service/network load is higher than required.

The NSI scaling is performed by scaling the underlying network services. In turn, this will lead to the change on the VNFs composing the network service, scaling them too, by increasing or reducing the infrastructure resources allocated to them (e.g. from "3 CPUs" to "5 CPU"). Any resources (e.g. connectivity) used to combine the network services may also require to be scaled (e.g. link bandwidth increased from 10Mbit/s to 20Mbit/s).

OSSs/BSSs is the entity that usually triggers the NSI scaling, although it may also be triggered automatically (autonomous behavior). The NSI scaling workflow is depicted in Fig. 5.16.

Note: For the sake of simplicity, the workflow does not show the situation where any network service cannot be scaled (e.g. lack of resources). In this case, the NSI scaling is aborted and either, all network services scaled to the original size, or (if feasible) the NSI remains partially scaled.

5.1.3.4 Terminating a Network Slice Instance

After a life time period, a NSI can be terminated, terminating the underlying network services and releasing any additional resources used to combine network services. After NSI termination, the information related to the NSI and respective list of network services could be destroyed. Underlying resources associated to the network services are released. OSSs/BSSs are the entity that triggers the NSI termination, although it could also be triggered automatically (autonomous behaviour). The NSI termination workflow is depicted in Fig. 5.19.
Figure 5.17: Slice instantiation
Figure 5.18: Slice instance scaling

Figure 5.19: Slice instance termination
5.1.3.5 Providing Network Slice Instance Monitoring Data

In accordance to the requirement to monitor each particular network service and the VNFs that it comprises of, the performance of each Slice Instance must be monitored in order to support the slice instance life cycle management, as described in the previous sections.

From the development and implementation effort on SONATA, monitoring framework is already capable of collecting data from OpenFlow controllers, taking advantage of the OpenDayLight API, and also collect data from OpenStack environments, utilizing OpenStack API.

However, during 5GTANGO, other types of resources or technologies must be integrated with the Monitoring Framework in order to collect monitoring data, in the form of new monitoring probes (or adapters with available APIs providing technology-specific monitoring data).

Beyond that, slice monitoring should be also able to monitor the service level data from the VNFs and network services deployed as part of this slice. This requirements brings the distributed approach on slice monitoring in place, since the slice may consist of parts of different PoPs.

Finally, a new feature that 5GTANGO must fulfil is to enhance monitoring functionality in such a way that it would be able to provide isolation between monitoring data from different slices (a slice owner cannot see other’s slices data). This applies both for the infrastructure and service level monitoring.

5.1.4 Policy Workflow

The policy manager requires having a view of the registered, allocated and available resources (compute, storage, network) at any point in time in order to be able to suggest actions related to resources management as well as to use such information during the policies formulation process. Such actions may regard reservation or release of the resources required for the provision of a service or dynamic management of the resources allocated to a network slice. Towards this direction, interaction among the policy manager, the Slice Manager, the MANO framework and the Infrastructure Abstraction is realised, as denoted at Fig. 5.20. The information fetched per request regards part or all the registered resources/slices.

5.1.4.1 Policy Formulation for a Network Service - Part I

The previous sequence diagram can be optionally used by the service platform manager in order to get an updated view of the available resources and slices in order to proceed to policies formulation.

Following, two diagrams are created, depicting the process regarding the formulation of a policy and its association with a network service. In part I of the diagram, depicted at Fig. 5.21, the service platform Manager selects the network service and examines set of available information that can be proven helpful for a policy formulation or update. The selection of the network service is realised through a policies editor that is aware of the set of supported NSs. Specifically, the service platform Manager may request information regarding the available descriptors and the relevant metadata, results provided through the profiling mechanisms and made available by the catalogue, as well as go through already defined policies for this NS.

5.1.4.2 Policy Formulation for a Network Service - Part II

In part II of the diagram, depicted at fig. 5.22, the service platform manager creates a new policy or updates an existing one. The creation of a new policy means that the service platform Manager may edit the existing rule expressions -as they are provided by the Developer in the form of a descriptor included within the software package- (e.g. change a threshold based on the knowledge of the infrastructure where the network service is going to be deployed), combine or split existing rule
Figure 5.20: Resources and slices overview
Figure 5.21: Policy formulation for a network service (1)
expressions, as well as add extra rules. Updating an existing policy is following a similar process, except that before editing the policy, end user retrieves it from the service platform catalogue.

It should be noted that any changes applied to expressions already provided by the developer for a network service may be realised by a Service Provider, however impacting only the instantiations/deployments of the network service within its service platform. In this way, Service Provider specific customizations may be realised in this way. For instance, a Service Provider may specify set of thresholds taking advantage of the knowledge of its infrastructure, while another Service Provider may define different thresholds, though the rules of the network service are the same in both cases.

5.1.4.3 Policy Enforcement of a Network Service

The enforcement of a policy includes a set of mechanisms regarding the initiation of the appropriate monitoring feeds upon a network service deployment, the publishing of the produced alerts in the message broker, the consumption of such messages by the policy manager and the support of inference mechanisms -as detailed in the policy manager section of this document-, the publishing of the inference results (suggested actions) to the message broker and the consumption of such results
Figure 5.23: Policy enforcement for a network service

by the various orchestration mechanisms. Such mechanisms are activated during the network service deployment and remain active during the overall lifetime of the NS. This workflow is depicted at Fig. 5.23.

5.1.4.4 Policy Expressions Formulation During Development

The network service developer is able to define set of rules that can be used towards runtime policies enforcement, aiming at optimal execution of the provided software. Such rules should be declared in an infrastructural-agnostic way, given that the developer is not aware of the exact infrastructure where the network service is going to be deployed. Upon the collection of data from set of tests as well as the consideration of any data made available by service providers deploying the NS, the rules can be updated in order to fit better realistic deployment scenarios. This workflow is depicted at Fig. 5.24.

5.1.5 SLA Workflow

In order for a network service to be deployed in the service provider’s infrastructure, several steps should be implemented with regards to the performance estimation and the required quality, that should be agreed and signed in the SLA.

While there are several SLAs that can be potentially managed between different actors, the SLA management of 5GTANGO primarily focuses on the interaction between the service provider and the end user/customer.

5.1.5.1 SLA Template Generation

During the SLA template timeline an initial SLA template will be created for the service provider. The SLA generator will analyse the metadata for a given network service and generate in a dynamic way a structured and at the same time generic SLA template as shown in Fig. 5.25.
Figure 5.24: Policy expressions formulation during development

Figure 5.25: SLA template for service provider/infrastructure Provider Workflow
Figure 5.26: Signed SLA workflow part I - If end-user NOT accept the SLA Template

5.1.5.2 Signed SLA - Part I

In Fig. 5.26, after the SLA template becomes available to the end-user, considering his/her business aspects, he should accept and sign it as is, or should define some new high-level requirements (e.g. availability, response time, etc.)

5.1.5.3 Signed SLA - Part II (a)

Once the SLA Manager gather all the appropriate datasets, checks if there is already a combination between the latter and already existing mapping results, in order to decide whether the process of the mapping mechanism should be done or not. In case the mapping process need to be done, combination (mapping) of the end-user-defined service requirements (i.e. workload parameters), and policies to the resource level attributes takes place, in order to foresee the performance estimation and the required quality, that should be agreed and signed in the SLA, as described at Fig. 5.27.

5.1.5.4 Signed SLA - Part II (b)

In case there is already a combination between the input dataset (requirements obtained from the service provider and the End-User) and the stored mapping results, at Fig. 5.28, the SLA Manager bypasses the mapping process and create dynamically the SLA.

5.1.5.5 SLA Template Generation Optimization

The SLA management is performed according to the sequences above. Given that the template generation is based on different datasets, the goal is to optimize the generation process by analysing historical data and data obtained from (runtime) executions. To this end, as shown in fig. 5.29,
Figure 5.27: Signed SLA Workflow Part II(a) - If there is NOT mapping yet

Figure 5.28: Signed SLA Workflow Part II(b) - If there is mapping
the SLA management framework collects run-time monitoring data and considers them during the template generation process (e.g., identification of values resulting to violations).

### 5.2 Service Platform Components

Given workflows defined in the previous section, we describe in this section the service platform’s components that allows to implement those workflows.

#### 5.2.1 Gatekeeper

As seen in Fig. 5.1, the gatekeeper is the service platform component that is in charge of controlling who can interact with the service platform, make some preliminary validations, collect usage KPIs, etc.

The existing ([1]) gatekeeper’s internal architecture ([52]) is shown in Fig. 5.30.

These modules have the following responsibilities:

- **GK API**: It is the API Gateway, validating and routing requests to the adequate module
- **Package Manager**: Manages packages on-boarding and downloading of both the package file and its meta-data

- **Service Manager**: Manages services meta-data downloading

- **Function Manager**: Manages functions meta-data downloading

- **Record Manager**: Manages services’ and functions’ records downloading

- **VIM Manager**: Manages the information the Infrastructure Abstraction provides, both about VIMs and WIMs

- **User Manager**: Manages all the needed authentication and authorization (humans and modules)

- **KPIs Manager**: Manages the usage KPIs collected

- **Licence Manager**: Manages the simple licensing model implemented in SONATA: packages, services and functions are by default public, but a developer can declare them private and provide a URL for the service platform to validate the existence/validity of a licence for the given package, service or function (i.e., all the licensing aspects remain out of scope of the service platform)

- **ConsumedServices**: Abstracts all the gatekeeper’s consumed services, like the MANO framework, the catalogues, etc.

Each one of these modules is deployed using a docker container ([24]). Other helper containers, not represented in the above figure, such as databases and message brokers, are also deployed using the same technology and shared among several modules.

This architecture will evolve to accommodate the new requirements from 5GTANGO ([9]). For example, there will be at least three new modules that will be added to the service platform architecture, namely the Slice, the Policy and the SLA Managers. Like the MANO or the Monitoring frameworks, each of these new modules will have their features exposed to the outside world of the service platform through the gatekeeper (see Sec. 6).

### 5.2.2 Catalogues

In order to support the lifecycle management and operation of services the 5GTANGO catalogue in the service platform provides management and storage of network service descriptors (NSD), Virtual Network Function Descriptors (VNFD), Package Descriptors (PD) and other asset descriptors like Monitoring Descriptors. The stored descriptors are required for the selection, development and instantiation of network services as well as for data analysis in order for the added value services that the 5GTANGO catalogue in the service platform provides to work (Decision Support Mechanism, Continuous Optimization Mechanism explained in deliverable D3.1 - “V&V Strategy and metadata management” [12]). The catalogue is accessed only via the gatekeeper Component. Thus the following are defined:

- **Network Service Stored Objects**: In the 5GTANGO catalogue in the service platform, it will be feasible to store information about all the on boarded network services, aiding the creation and management of the network service deployment templates. The aforementioned stored information refers to the NSD which contains references to the constituent VNFs,
Virtual Links and the Forwarding graph, associated SLAs and Policies. The VNF forwarding Graph Descriptor contains references to virtual link elements thus providing information about the chaining of the VNFs that compose the network service. The Virtual Link Descriptor (VLD) contains information about the connection points participating in each virtual link. Further Metadata like test results (e.g. test score, test platform configuration) and Monitoring Descriptors may also be included.

- **Virtual Network Function Stored Objects:** It will also be feasible to store information about VNFs in the catalogue. This information refers to the VNFD which contains the VNF description template, reference to the software images (or even the software image itself) and manifest files. Further metadata like test results (e.g. test score, test platform configuration) and Monitoring Descriptors may also be included.

More information about the included metadata is available in deliverable D3.1 - “V&V Strategy and metadata management” [12].

### 5.2.3 MANO Framework

The MANO framework is the service platform component responsible for managing the life cycle of the instantiated network services and the orchestration of the infrastructure that is used to host these network services on.

In 5GTANGO, the MANO framework component is further divided into two main subcomponents: The *network service Orchestrator* (NSO) is responsible for all decisions on the level of a single network service, whereas the *Resource Orchestrator* (RO) handles decisions on a larger scale, with the entire infrastructure and all instantiated network services in scope.

The design for the MANO framework, which is in accordance with the ETSI model, is shown on Fig. 5.31.
5.2.3.1 Network Service Orchestrator

The network service Orchestrator (NSO) is responsible for all MANO framework tasks on the service level and thus needs to be able to execute a set of workflows. Among others, it needs to provide instantiating, terminating and updating workflows for the gatekeeper to trigger, and both the resource orchestrator (RO) and the NSO itself are allowed to request scaling and migration workflows for a service.

These workflows can be specified by constructing them out of chains of basic tasks (or low-level workflows), such as

1. Calculating the placement for a service
2. Generating, storing and updating records for a service
3. Deploying or terminating a VNF
4. Instantiating or removing a service function chain (SFC)
5. Transferring the state from one VNF to another
6. Obtain characteristics of the slice that hosts the service

This mapping between the supported workflows and a chain of low-level tasks is organised by the Service Lifecycle Manager (SLM), the core component of the NSO. It is the recipient of all workflow requests, makes sure that all tasks associated with this workflow are executed in the correct order and reports on the result of the workflow to the initiator of the request. The SLM invokes the modules that are responsible for each task in the correct order. These modules, and their responsibilities, are:

1. **Slice Manager**: Provide resource information for the slice that is being used for the service
2. **Function Lifecycle Manager** (FLM): Handles workflows on the level of the VNF (e.g. instantiate a VNF, terminate a VNF), also by chaining together a list of tasks (e.g. prepare VIM, instantiate VNF, store record)
3. **Placement Manager**: As part of the RO, the Placement Manager calculates the optimal placement of a VNF
4. **Monitoring Manager**: To start or reconfigure the monitoring of a service
5. **Repositories**: To store service and VNF records

Both the SLM and the FLM translate workflows into chains of tasks. To increase the flexibility of the MANO framework, the 5GTANGO service platform allows for the inclusion of Service Specific Managers (SSM) and Function Specific Managers (FSM) within service and VNF descriptors. SSMs and FSMs are executable code segments that customize the behavior of either the SLM or FLM. They are created by service and VNF developers and each one is wrapped in a separate container. The MANO framework will execute such SSM and FSM containers when it receives a request to instantiate a service or VNF which specify SSMs or FSMs in their descriptors. At this point, the deployed SSM or FSM will customize the behavior of the MANO framework. They can overwrite the chain of tasks that is associated with a workflow in the SLM or FLM with a different chain. They can provide new tasks or overwrite the functionality of existing tasks (e.g. an SSM can instruct the SLM to use an embedding algorithm implemented by the SSM, instead of using the Placement Manager from the RO).
5.2.3.2 Resource Orchestrator

The Resource Orchestrator (RO) is responsible for decisions that affect the entire infrastructure that is used by the service platform. It calculates the optimal placement of VNFs of new network services. It can provide, abstracted if needed, topologies of the available infrastructure to other parties (e.g. parent service platforms). It monitors the infrastructure usage to evaluate how migrating some of the deployed images further optimizes the overall QoS of all deployed network services. To this end, the RO evaluates whether a VNF could be moved to improve its performance, to anticipate on future scaling requirements, to optimize the (power) usage of the infrastructure, etc. To find the optimal resource embedding, all services should be considered at once. This re-optimization evaluation should be frequently executed, as the outcome might have changed due to varying loads in VNFs or terminated network services. Once the RO decides on a list of network service life cycle events, i.e. network service migrations, it instructs the NSO to execute them.

The RO requires input from the policy manager and the SLA Manager. These modules indicate to the RO whether services or VNFs are (close to) violating policies or SLAs. The RO will then search for a new orchestration for the entire infrastructure, taking slices into account, that resolves these violations while minimizing the impact on the deployed services (i.e. minimizing the number of VNFs that require migration). This task cannot be executed by the NSO, which operates per service. A scenario with network services that require migration but can’t be migrated due to other network services occupying the required resources can only be resolved by an RO that can decide to migrate a healthy service to make resources available for services that perform sub-optimal.

5.2.3.3 MANO Framework and Slicing

When a new service needs to be instantiated, the gatekeeper should inform the MANO framework by which slice it should be hosted. This data will be used by the MANO framework to request the available resources associated with this slice from the Slice Manager. These available resources will then be used as input for the embedding calculation that takes place in the Placement Manager. Changes in the life cycle of a service instance will be communicated to the Slice Manager, allowing it to keep up to date with the latest configurations.

The MANO framework should be able to function independent of the presence of the Slice Manager. This ensures that the 5GTANGO service platform can be used in a context without network slicing. In such a scenario, the available resources are not requested from the Slice Manager, but from the Infrastructure Adaptor. Instead of calculating the embedding within the resources of a slice, all the resources available to the service platform are considered.

5.2.3.4 Flexibility

In terms of flexibility, the MANO framework follows the line of the entire 5GTANGO service platform by using micro services for the different modules in the architecture. Since micro services are stand-alone processes that use a network to communicate, there is no tight coupling between the MANO framework modules. This allows the service platform owner to easily alter the behaviour of the MANO framework by replacing one of the modules by a module that incorporates a different functionality, all during runtime. The communication between the micro services is chosen to be asynchronous and thus stateless, which further increases flexibility.

A second display of flexibility comes from the SSMs and FSMs, which allow the developers of network services and VNFs to customize the MANO framework behaviour to better support the different life cycle events of their service or VNF.
5.2.4 Network Slice Manager

A wide range of use cases from the vertical industries with different requirements (e.g., security, latency, resiliency, and bandwidth) shall be accommodated within the 5G architecture. In order to cope with this challenge, the NGMN has proposed the concept of network slicing.

5.2.4.1 Network Slice Definition

Network Slice Instance (NSI) is defined in [23] as a set of network functions and the resources for these network functions which are arranged and configured, forming a complete logical network to meet certain network characteristics.

3GPP has proposed a data model [8], which consists of a list of Network Slice Subnetworks instances (NSSI), which contain a set of network functions and the resources for these network functions which are arranged and configured to form a logical network.

The NSI contains NSSI, which in turn contain NFs (e.g., belonging to Access Network (AN) and Core Network (CN)) as well as all information relevant to the interconnections between these NFs like topology of connections and individual link requirements (e.g. QoS attributes). The NSI is created by using a Network Slice Template (NST).

ETSI NFV ISG is actually highlighting the relationship between network services and Slices / SubnetSlices [20]. This is important since the SP is familiar and supports the network service (and VNF) constructions. Fig. 5.32 presents the relationship between 3GPP and ETSI NFV data models. It can be observed that the virtualised resources for the slice subnet and their connectivity to physical resources can be represented by the nested network service concept, or one or more VNFs and PNFs directly attached to the network service used by the network slice. ETSI states that “an NFV Network Service (network service) can thus be regarded as a resource-centric view of a network slice, for the cases where a Network Slice Instance (NSI) would contain at least one virtualised network function” [20].

In order to align with both standards bodies, 5GTANGO will extend, implement and validate the architecture proposed by ORANGE in [15], where ETSI and 3GPP SDOs combine their existing
solutions in order to offer Network Slicing. The workflow between Network Slice Manager and the rest of Service Platform components is described in previous sections.

### 5.2.4.2 Network Slice Manager Internal Architecture

The proposed Network Slice Manager internal architecture is shown in Fig. 5.33. The Network Slice Manager is a Service Platform (SP) functional block that interacts with OSS and is responsible for interacting with the SP MANO framework to control slicing. The Network Slice Manager is a consumer of the REST APIs exposed by the MANO framework (reference point Os-Ma-nfvo [19]) and the IA. Within a Network Slice Manager, one can identify two main functions:

- The first function is responsible for assigning services and applications to network slices and for managing the lifecycle of these slices. We will call it the Slice Lifecycle Manager.
- The second function is responsible for mapping network slices to NFV Network Services (NS). We will call it the Slice2NS Mapper.

#### Slice Lifecycle Manager

The Slice Lifecycle Manager is responsible for the entire lifecycle management of the created network slice instance, until it is terminated. Subsequent lifecycle events are likely to have an impact on the lifecycle of the underlying Network Services (NS), but not systematically.

The Slice Lifecycle Manager function decides whether a customer facing service is to be assigned to an existing network slice or sub-network slice or whether a new network slice or subnetwork slice is to be created. If the service is assigned to an existing slice, the Network Slice Template (NST) is updated while in the second case, a new NST is generated.

NSTs are then converted by the Slice2NS mapper into NSDs and flavours. Once the NSD has been on-boarded, a slice instantiation request can be triggered, resulting in an NS instantiation request with the appropriate flavor identifier being sent to the NSO.

#### Slice2NS Mapper

The Slice2NS Mapper functionality has to maintain an association between NST, and NSDs with applicable deployment flavour identifiers, as well as an association between slice identifiers and NS instance identifiers. It also deals with the required combination/integration of NSs.

This behaviour might involve automatic generation of a new NSD, in case that the slice requirements do not map to an already on-boarded NSD (i.e., available in the NSD catalogue). A network
slice intended to support totally new customer facing services is likely to require a new Network Service and thus the generation of a new NSD.

5.2.5 Infrastructure Abstraction

The infrastructure abstraction is a component of the 5GTANGO service platform located in the lower layer that enables the communication between the 5GTANGO service platform to multiples VIMs and WIMs.

The existing ([1]) infrastructure abstraction internal architecture [46] is shown in Fig. 5.34. In this figure, four components can be identified. The North Bound Interface (NBI), exports a common interface to manage computational and storage resource, deploy and manage services, provide necessary instance information for the monitoring and management facilities of the service platform. The VIM and WIM Adaptors are specific wrappers for different VIM/WIM implementations. The South Bound Interface enables the communication from the IA to the VIMs and WIMs. VIMs and WIMs as detailed in Sec. 5.2.9 are the infrastructures attached to the 5GTANGO infrastructure abstraction where the network service will be deployed.

5.2.5.1 Infrastructure Abstraction Interfaces:

In SONATA “service platform First Operational Release and Documentation” ([51]) were defined the Infrastructure abstraction interfaces. The IA is internally organised as wrappers with the purpose of hiding states, configurations, functions and implementations which are specific for each technology (OpenStack, OpenVIM, vCloud, Kubernetes, OpenDaylight, VTN and even other 5GTANGO service platform). Wrappers are divided into four categories:

- **Compute Wrapper**: Wraps a VIM to deploy virtual machines, create virtual networks, virtual routers and virtual ports.

- **Network Wrapper**: Wraps a VIM to deploy rules and rule sets to enforce Service Function Chaining or other network policies.
• **Storage Wrapper:** Wraps a VIM to store and retrieve virtual deployment unit (VDU) images.

• **WIM Wrapper:** Wraps a WIM to configure a WAN or a portion of a WAN to enforce intra-PoP connectivity for services.

Each of these wrappers is designed to be an interface, offering specific functions which are combined to implement the IA API. Compute, Network and Storage wrappers are considered as the three parts of an NFVI-PoP, so to be able to store Virtual Machine (VM) images, deploy these VMs and configure their internal networking. For this reason, for each Compute wrapper registered to the IA, there must be a relevant Storage wrapper to store the images needed for the service deployment, and a relevant Network wrapper to enforce network configuration on the instances of these images.

### 5.2.5.2 Infrastructure abstraction functionalities

• **OpenStack Heat Wrapper:** This offer a translation model that enables the deployment of the network service in multiples PoPs.

• **OVS Networking Wrapper:** This offer a custom SFC agent that provides a way to create service function chaining in an OpenStack node.

• **VTN WIM Wrapper:** This wrapper uses OpenDayLight Virtual Tenant Network (VTN) Plugin to set traffic flow rules, that allow or prohibit communication, as well as redirect packets that meet a particular condition. Other REST APIs might be considered through the duration of the project, such as ONF Transport API [42] and IETF ACTN [32].

### 5.2.6 Baseline Monitoring Framework

This section provides an overview of the Monitoring Framework that has been implemented and integrated within SONATA project and which will be enhanced and extended to cover the requirements of 5GTANGO.

In a nutshell, the SONATA monitoring framework collects and processes data from several sources, providing the developer the ability to activate metrics and thresholds in order to capture generic or service-specific behaviour. Moreover, the developer can define rules based on metrics gathered from one or more VNFs deployed in one or more NFVIs in order to receive notifications in real time. In general, the developer is able to subscribe to the message bus or she can get the alert notifications by email and/or SMS on her smartphone. Most importantly, monitoring data and alerts are also accessible through a RESTful API or directly accessing a websocket URL. The internal architecture of Monitoring Framework is depicted in Fig. 5.35 and the functionality of each module is explained in the next subsections.

#### 5.2.6.1 Collecting Data From Several Sources

It is of paramount importance to collect monitoring data from as many as possible sources. In the implemented framework, there are four different types of sources for collecting data:

1. Container exporter which runs inside the container of the VNFs to collect data from its performance

2. VM exporter that collects data from Virtual Machines (VMs) hosting VNFs
3. OpenFlow exporter which is a Python software that utilizes OpenDayLight API to collect data from the OpenFlow controller

4. OpenStack exporter that has also been developed as a software module (in Python language) that uses OpenStack API to collect data from all OpenStack components

5.2.6.2 Push Gateway

This component is part of the open source Prometheus monitoring solution that has been adopted and extended to cover the needs of 5GTANGO Monitoring Framework. Push Gateway is utilized so that the exporters/sources use HTTP POST method to “push” monitoring data to the Push Gateway, while Prometheus server collects the data in a predefined time interval. The advantage of this approach is that in the case of the deployment of a new service, there is no need for the Prometheus monitoring server to search for data related to the newly deployed VNF, but rather collect them from the PushGateway.

5.2.6.3 Monitoring Server

The implementation of 5GTANGO Monitoring Server will capitalize on Prometheus monitoring tool [41], enhanced to address 5GTANGO requirements. Prometheus is an open-source service monitoring system, based on time series database that implements a highly dimensional data model. A time series entry is identified by a metric name and a set of key-value pairs. Prometheus has a sophisticated local storage subsystem (LevelDB), which is essentially dealing with data on disk and relies on the disk caches of the operating system for optimal performance. Prometheus server is responsible for collecting the data and communicating with the time-series database for retrieving data upon request.
5.2.6.4 Monitoring Manager

Monitoring manager is a Django-based server that offers RESTful APIs to the users with respect to the monitoring data of their instantiated 5G services, including: 1) the relation among services, network functions, NFVIs and users, 2) the ability to modify rules and thresholds during service/function runtime, 3) the reconfiguration of Prometheus server, 4) the ability to define the notification methods in case of alert generation, 5) the definition of a new websocket to get data in real-time, and 6) the capability to provide information on collected monitoring metrics.

5.2.6.5 Alert Manager

As previously discussed, the Alert Manager is responsible (along with the implementation of a message queuing mechanism, such as RabbitMQ) for sending notifications about firing alerts to the subscribed users. After this notification, the user can take advantage of the API to further investigate the fault or activate a websocket to receive real-time monitoring data.

5.2.6.6 WebSocket server

The implementation of websockets (Tornado web server) allows the user to collect streaming data from VNFs that have been deployed in the Service Platform. This is highly beneficial to the developers, as they would be able to monitor the performance of a new service in real environment. Prior to the establishment of a new websocket, the user must be aware of the metrics collected per VNF, the VNFs comprising his deployed Network Services and other related information and this information is already provided by the existing Monitoring Manager API framework. After selecting the VNF and the respective metrics to be sent, the user requests the creation of a new websocket from the Monitoring Manager. After checking the validity of the request, the Monitoring Manager communicates with the Websocket server that creates and sends new URL for the user to connect to and where metric values are pushed into.

5.2.6.7 Scalability and Distributed Architecture

One of the cornerstones of the monitoring framework implementation was to deliver a carrier-grade solution that would fulfill scalability requirements in a multi-PoP environment. As it is noticed in Fig. 5.36, several components of the Monitoring Framework had to be distributed across the 5GTANGO Points of Presence (PoPs). First, each PoP must have its own websocket server to accommodate developers’ demands for streaming data, although the management of websockets is handled by the Monitoring Manager instance in a centralized way. Second, Prometheus Monitoring servers follow a distributed (cascaded) architecture. The local Prometheus servers collect and store metric data from the VNFs deployed in the PoP, while only the alerts are sent to the federated Prometheus server for further processing and forwarding to the subscribed users.

Moreover, the alerting rules and notifications are based on monitoring data collected in different PoPs and thus the decision must be made on a federation level. Another scalability requirement concerns the large flow of data from the monitoring probes to the Monitoring Server and its respective database that might affect the service performance in extreme cases. In this respect, an architectural decision to address this scalability issue was to support a distributed architecture regarding the monitoring server and its database, working in a cascaded fashion along with proper modifications on component level. In particular, the functionality of the monitoring probe will change so that it will not send data to the monitoring server in cases where the value difference is less than a threshold defined by the developer. The same will be the case in the communication between the monitoring server within a NFVI and the monitoring server in the Service Platform.
5.2.7 Extensions to Existing Monitoring Framework

There are three extensions of the current Monitoring Framework implementation that will be addressed in 5GTANGO, as described in the following sections.

5.2.7.1 Support V&V Testing Methodology

The main driver for this extension is to support testing of NSs and VNFs by providing monitoring data from their performance in the V&V environment.

The main objectives are:

1. Provide automated deployment of tests and auxiliary services as an integral part of the deployment process defined in the generic framework of the 5GTANGO project.

2. Provide a set of supported metrics and measurements, mainly covering active monitoring, since passive monitoring is preferred for off-line processing.

The Fig. 5.37 presents a basic diagram of the monitoring components included in NS testing process within the V&V environment.

In the general case, the Network Service might consist of two VNFs deployed in two remote PoPs, as depicted in Fig. 5.38. VNFs can also be deployed by utilizing several recent technologies, including Virtual Machines and/or Linux or Docker Containers. Each software block has two network interface cards (NIC); one connected to a public network being used as a management network for remotely configuring virtual switches via the vSwitch Controller, based on testing conditions, defined in the Test Monitoring Controller. The second NIC is used to connect the two VNFs comprising the network service or one VNF with an auxiliary function/service (software module) deployed solely for the purposes of testing (e.g. when a web server reachability is tested via ping testing). The NICs are connected through proper LAN/WAN network, as shown in Fig. 5.38. Moreover, as it is obvious from Fig. 5.38, network monitoring protocols already existing in the routers can be supported. For example, OVS already supports NetFlow and SFlow.
5.2.7.2 Support Monitoring During Network Service Runtime

The objective of this process is to enlarge the set of monitoring metrics with passive and active probes, since the framework to be used in this case is almost identical with the one developed in SONATA.

In a typical Network Service deployment, the VNFs are typically installed over several physical servers (allocating required resources as demanded by the VNFD/NSD) and the network is configured to correctly forward packets along the VNF Forwarding Graph (also described in the VNFD/NSD). Then, the service provider (manually or automatically) must orchestrate and manage the VNF components throughout their lifecycle to ensure SLA compliance. In this respect, monitoring framework directly supports SLA so that infrastructure resources are properly allocated (and not wasted) and timely identifying the need for spawning new VMs or scale VNFs to meet increased loads as shown in Fig. 5.38. The architecture is based on the existing Monitoring Framework developed in SONATA, but extended with (mainly active) network monitoring probes to cover all possible requirements of the platform.

It is mentioned that current implementation includes collecting data from OpenFlow Controllers, but it needs to be extended by configuring and activating well-known network monitoring protocols. Moreover, if deemed necessary, port mirroring techniques could be implemented in the virtual switches for passive monitoring processing.

5.2.7.3 Enhanced Monitoring Framework Functionalities

Probes collect monitoring data from running services and (currently) according to pre-defined thresholds the information is sent to the server, which analyses it and takes actions based on the QoS criteria that have been set.

The proposed architecture aims at enabling decision-taking at the probe level. The goal is to
facilitate decision making with respect to:

1. When to transmit the monitoring data
2. Which data to transmit
3. How to optimize the process

To this end, we propose to implement a mechanism that will allow the aforementioned decision considering:

1. The time intervals for transmitting the monitoring data by obtaining from the server the QoS parameters and evaluating how the VNF monitoring data “evolve” towards the QoS thresholds (e.g. linear or exponential change of specific values).

2. The data to be transmitted. For example, if CPU remains unchanged but memory data demonstrate continuous or significant changes, partial CPU data will be sent to the push gateway. This can be done using the importance of different monitoring data. An approach will be incorporated to compile importance weight factors of the monitoring data by correlating and analyzing the SLAs and the associated policies.

3. The proposed optimization approaches in the overall process through potential collective knowledge emerging from different sources: If there are several sources in a given Network Service, information regarding monitoring parameters, time intervals, importance of parameters, etc. could be exploited by sharing this information directly between the VNFs. Also, the (time interval) pattern to submit the data from the probes towards the push gateway could be such that a probe collects information from other probes, aggregates and submits the information. The latter implies the development of a mechanism that decides during
Figure 5.39: Frequency Analyzer Mechanism

runtime according to several conditions (e.g. VNF status, network status for all VNFs, etc.) which is the optimum pattern. It is also heavily dependent on the use case.

The Fig. 5.39 is showing the mechanism mentioned above which is the frequency analyzer. It will be among with others mainly responsible for the decision making in respect of “when” to transmit the monitoring data.

5.2.8 Network Service and Function Repositories

The service platform contains repositories which implement databases to hold runtime information about network services, network functions, and further runtime information. In contrast to the network service and function descriptors, which contain a static description of the network service and functions to be instantiated, the repositories hold dynamic information for every instantiated network service or function. These additional information are, for example, provided by the infrastructure abstraction upon service instantiation and added to the record that belongs to the service or function instance. A typical example for this are management IP address which are assigned to each VNF by the NFVI.

An important benefit of the repository approach is that individual components like the MANO framework are not required to hold the state of the service deployments by their own and can operate more or less state-less. This simplifies update and restart procedures and contributes to the platform’s resiliency and scalability. Further details about the repository implementations used in 5GTANGO can be found in [51, 52].
5.2.9 Network and Virtualised Infrastructure Management

ETSI ISG NFV initial document are anticipating infrastructure specific managers in order to manage and control the resource allocation within the physical infrastructure substrate where the VNFs are deployed and chained together. These managers are (i) the Virtualised Infrastructure Manager (VIM) and (ii) the WAN infrastructure management. Next couple of paragraphs provide a summary of their functionalities and their role in the NFV framework.

5.2.9.1 VIM

The generally accepted view of NFV deployment scenarios, regardless if multi-domain or single-domain is included, the instantiation of VNFs is considered to happen in various locations within the network topology (i.e core and edge). These locations are anticipated to host datacenter infrastructures (at various scales) that are interconnected via Wide Area Network (WAN) infrastructures. These designated locations are called Network Function Virtualization Infrastructure Points of Presence (NFVI-PoP) according to ETSI NFV. Within each NFVI-PoP the resources (i.e computing, storage and network) are managed via the entity called Virtual Infrastructure Manager (VIM). The actual implementation of the VIM is directly influenced by and dependent on the technology that is used in order to provide virtualization support within the NFVI (i.e. hypervisor) and the type of resources actually being virtualized. A VIM may be specialized in handling a certain type of NFVI resource (e.g. compute-only, storage-only, networking-only), or may be capable of managing multiple types of NFVI resources (e.g. in NFVI-Nodes). Hence, a VIM can be found in a variety of implementations, each exposing their own set of Northbound and Southbound interfaces and supported functionalities. Below you may find a summary of possible VIM solutions that are being considered:

- **Openstack HEAT**: OpenStack is an open source software for creating private and public clouds. It is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. Heat is the main project in the OpenStack Orchestration program. It implements an orchestration engine to launch multiple composite cloud applications based on templates in the form of text files that can be treated like code.

- **OpenStack Tacker**: Tacker is an official OpenStack project building a Generic VNF Manager (VNFM) and a NFV Orchestrator (NFVO) to deploy and operate Network Services and Virtual Network Functions (VNFs) on an NFV infrastructure platform like OpenStack. It is based on ETSI MANO Architectural Framework and provides a functional stack to Orchestrate Network Services end-to-end using VNFs.

- **OpenVIM**: OpenVIM is a light implementation of an NFV VIM supporting enhanced performance features and control of an underlay switching infrastructure through an Openflow Controller (OFC). OpenVIM interfaces with the compute nodes in the NFV Infrastructure and an Openflow controller in order to provide computing and networking capabilities and to deploy virtual machines. It offers a northbound interface, based on REST (openvim API), where enhanced cloud services are offered including the creation, deletion and management of images, flavors, instances and networks.

- **Kubernetes**: Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery.
5.2.9.2 WIM

WIMs refer to the WAN Infrastructure Managers as shown in Fig. 5.40. Each WIM function is to control the WAN network and be able to control the designated data traffic flow. The WIM, upon request, will be able to redirect and guide the data traffic from the source site towards different PoPs located to different WANs, and finally to its ultimate destination. In the case of WIM, this functional block may be added as a specialized VIM, in particular for new virtualized deployments. In this scope, the functionalities and scope greatly depend on the underlying networking technologies employed as well as on their virtualization capabilities. For 5GTANGO, the provisioning part of the WAN networking links that interconnect the NFVI-PoPs is considered out-of-scope. Simplifying assumptions will be used in place to overcome this in order to support multi-pop use case scenarios, e.g. the tunnels will be pre-provisioned and network matrix (providing resource information) will be available at the infrastructure repository.

5.2.10 Graphical User Interface

The Graphical User Interface (GUI) of the 5GTANGO’s platform will use the same framework as SONATA’s platform with the required extensions to support the new functionality. The GUI will consume a REST API exposed by the involved elements of 5GTANGO platform so it can be easily maintained and extended if more functionalities are needed.

The GUI will be divided into two parts:

- The admin GUI which provides the platform and service administrators the ability to provision, monitor and monetize platform resources.

- The BSS GUI which enables 5GTANGO developers to manage the creation of new services throughout their whole lifecycle.
There will be Admin users with permission to access both GUIs and create users with customer and developer roles. Customer users will be able to access the BSS GUI and developer users will be able to access the Service Platform through the admin GUI, the SDK and also the V&V interface. Admin users will have the ability to enable/disable the use of specific features on a per user basis. Admin users may be provisioned by using a simple CLI tool.

5.2.10.1 Admin GUI

The admin GUI will be divided in these main sections:

1. **Dashboard:** It will allow to display the most important health and performance indicators of the different elements involved in the platform.

2. **Settings:** It will allow to check and set different parameters of the platform.

3. **Catalogue:** It will display the available packages, services and platforms to setup new services.

4. **VIM settings:** This section enables the addition/removal of new Virtual Network Infrastructure and WIM (WAN Infrastructure Manager) to be used for the deployment of the different services.

5. Verification and Validation. It will allow to check and perform automatic tests related to each VNF.

6. **Monitoring:** From this section the admin will be able to monitor and check the health of the deployed services in real-time.

7. **Alerts:** This section will show triggered alarms and configure thresholds and events to trigger the alarms.

8. **SLA metrics:** It will provide information in order to check the SLA and violations (if any).

9. **Policies:** This section will support the formulation of policies based on an editor allowing for the definition of expressions consisting of sets of rules and actions.

10. **User management:** This section will only be visible by Admin users. From this screen it will be possible to create new developer users and assign permissions to them.

5.2.10.2 BSS GUI

This GUI will enable the management of Network Services.

1. Available network services. The visual interface will list all the available services which can be deployed. The Network Services listed here will be available in the Catalogue section.

2. It will be possible to request the instantiation of any of the available Network Services along with the enforcement of the associated policy (if any).

3. Following the request of service instances once their instantiation has been requested so that it is possible to know the status of a request at any time.

4. License store. Network Service use can require a license. From the License store screen will be possible to manage and acquire licenses to be added to the available services in the platform. This interface will also allow acquiring licenses related to specific users.
5. User management. This section will only be visible by Admin users. From this screen it will be possible to create new customer users and it may be possible to assign different permissions.

5.2.11 Policy Manager

The Policy Manager is the entity of the service platform responsible for runtime policies enforcement over the deployed network services. It consists of a set of components supporting the formulation of the policies and their enforcement, including the interfaces for interaction with the 5GTANGO monitoring framework and the message broker for publication and consumption of monitoring data, alerts and suggested actions. The internal architecture of the policy manager is depicted at Fig. 5.41.

5.2.11.1 Policies Definition

Policies definition is based on information that can be provided by the Software Developer of a network service, the service platform manager or both. Definition of rules per policy is supported through the policy editor in a per network service basis. A network service may be associated with a set of policies, however only one can be active during its deployment and execution time. Each policy consists of a set of rules. Each rule consists of the expressions part, denoting a set of conditions to be met and the actions part denoting actions upon the fulfillment of the conditions.

The software developer, having detailed knowledge of the business logic of the developed software, is able to define set of rules that can be applied during the execution of the network service. Such rules are included in the software package of the network service in the form of a descriptor. The service platform manager, having detailed knowledge of the registered infrastructure, is able to add extra rules and/or modify set of thresholds in existing rules provided by the Software Developer. The latter may be supported or not, given the preferences of the software developer. In case it is supported, any change in specified thresholds is going to regard only the service platform instance
In order to facilitate policies formulation as well as for interoperability purposes, policies description is going to be based on the definition of a formal language, taking into account the set of concepts declared in the set of descriptors accompanying a network service (e.g. VNF descriptors, network service descriptor).

5.2.11.2 Policies Enforcement

Policies enforcement is realized through a rule-based management system that attempts to derive execution instructions based on the current set of data and the active rules; rules associated with the deployed network services at each point of time. Towards this direction, the Policy Manager is providing policies enforcement over the deployed network services following a continuous match-resolve-act approach. Specifically, the match phase regards the mapping of the set of applied rules which are satisfied based on the data streams coming from the monitoring infrastructure, the resolve phase regards the process of conflict resolution -if any- among the satisfied rules taking into account the pre-defined salience of each rule, while the act phase regards the provision of a set of suggested actions by the policy manager to the orchestration components of the 5GTANGO service platform.

In more detail, the policy manager -in addition to the policies editor- consists of (i) the working memory; facts based on the provided data, (ii) the production memory; set of defined rules, and (iii) an inference engine that supports reasoning and conflict resolution over the provided set of facts and rules as well as triggering of the appropriate actions. Data is fed to the working memory through the monitoring mechanisms that is responsible to collect data based on a set of active monitoring probes, as well as to support a set of data management and processing operations. The production memory is also fed by policies associated with the deployed network services, as provided through the policies editor - the editor made available to service providers for policies definition, refinement or mapping with a specific network service.

5.2.11.3 Policy-based Slice Management

The policy manager is going also to support slice management functionalities, mainly regarding the management of slices that can be dynamically provisioned and adjusted. Management of a slice may regard the allocation/deprovision of compute, storage and network resources, as well as the activation/deactication of network mechanisms-functionalities supported by the network slice (e.g. dynamic activation of a VNF related to radio access network functionalities). Rules for dynamic configuration of a slice may be related with QoS and/or QoE requirements (e.g. allocation of extra resources in case of high utilization of the available ones or upon reaching a threshold related to a QoS/QoE metric) taking into account the set of services to be provided through the slice and their associated policies and SLAs. Slice resources isolation requirements may be also dynamically applied taking into account the requirements imposed prior to the instantiation of new services.

5.2.12 SLA Manager

In the 5G networks environment, appropriate guarantee for the Network Service (NS) quality can maximize the ability of services and ensure to be carried out effectively and controllably, and that’s what Service Level Agreements (SLA) can offer. To create an operational NS, it is highly envisaged to associate it with their network-related parameters that reflect the corresponding quality levels. These are included in policies but while SLAs target usually business users, there is a challenge for mechanisms that bridge this abstraction gap. To this direction, performance prediction and
modeling are considered to be of major importance, since the service providers need to determine the resources to fulfill the QoS requirements of the service, while at the same time, resource utilization must be maximized.

Considering that the 5GTANGO Service Platform architecture aims at supporting the interaction between a distributed set of users and resources, it requires i) QoS Parameters Prediction - QoS oriented service models for predicting QoS requirements, ii) automated SLA Template Generation - considering provider’s requirements and dynamic discovery of policies expressions and iii) dynamic requirements mapping - map the high-level end-user’s requirements to the low-level Service Provider’s requirements.

5.2.12.1 SLA Definition

An SLA is a contract between a service provider and its internal or external end-users that documents what services the provider will accord, and defines the performance standards that are obligated to be met by the provider. Since each component potentially affects the overall behavior of the platform, any high-level goal (e.g. performance, availability, security) specified for the service potentially relates to all low-level requirements. The metrics that can be included into SLAs may specify i) availability and uptime, ii) Network Service response time, iii) the schedule for notification in advance of network changes that may affect end-users, iv) usage statistics that will be provided and others.

5.2.12.2 SLA Management Framework Internal Architecture

Fig. 5.42 describes that the SLA Management Framework in 5GTANGO should consider two phases: pre-deployment phase (SLA Template Management) and during deployment phase (SLA Information Management).

The SLA Template Management component is responsible for receiving all the appropriate information from the different stakeholders – considering none of the NS run-time monitoring metrics - in order to create in an efficient way a Service Level Agreement. The SLA Register Management takes place after the successful selection of a Network Service from the 5GTANGO Catalogue, and before deploy it at the Production Service Platform. The SLA Runtime Management takes place during the NS runtime and is responsible for obtaining monitoring metrics, analyze them and make any adaption if necessary to the SLA mapping results.

SLA Template Management

SLA Template Management is responsible for receiving the appropriate dataset from different stakeholders, analyze them, map them, and compile a) an SLA Template and b) a signed (final) SLA, in an automated and efficient way.

- **SLA Template Generator**: firstly, creates generic SLA templates for the Provider, and secondly creates the final SLA itself. The SLA Template Generator should have access to associated set of policies, in order to be able to analyze them and generate a structured template with initial objectives about the specific NS.

- **SLA Mapping Mechanism**: mapping between the high-level requirements described by the end-user and the low-level requirements described by the provider, while at the same time decompose service level objectives to associated policies for better Policy Enforcement. The outputs of the mapping mechanism are explicit SLA parameters and metrics.
• **SLA Parameter Analyzer**: It is responsible to decide whether the process of the mapping mechanism should be done or not. It should be noted that the mapping results are due to be stored into a repository. The decision is made after the component searches into the mapping repository and check whether there is a combination between the input parameters and the stored results. In case there is already a combination, the SLA Manager bypasses the mapping process and create dynamically the SLA.

**SLA Information Management**

SLA Information Management is responsible for optimizing the generation process by analyzing historical data and data obtained from (runtime) executions.

*SLA Monitor Analyzer*: Compare the QoS parameters from the Mapping Repository, with the computed monitoring measurements and check if there is any violation. If there is, the Mapping Mechanism will obtain this information as an additional dataset and re-do the mapping process.

**SLA Mapping Repository**

The SLA Mapping Repository is responsible for keeping all the produced mapping results from the Mapping Mechanism component. More specific, it will store the correlations between the high-level and the low-level requirements.
6 Interactions

This section lists the interactions between the main blocks of the 5GTANGO architecture.

6.1 Catalogue Interactions

The following sections focus on all the interactions between the Catalogue and the different 5GTANGO components. In more detail these interactions are:

- Catalogue - Service platform described in Sec. 6.1.1
- Catalogue - V&V described in Sec. 6.1.3
- Catalogue - SDK described in Sec. 6.1.2

6.1.1 Catalogue SP Interaction

The planned interactions of the Catalogue and the Service Platform are as follows:

1. **Package upload/on-boarding:** Whenever the developer thinks he/she has the package ready, it can be pushed into a Catalogue.

2. **Package file retrieval:** Whole package files can be downloaded from the Catalogue, for clarification, reuse, etc. Restrictions (e.g., licences needed) might be applicable.

3. **Package, service, function meta-data (descriptors) retrieval:** Meta-data can be downloaded from the Catalogue, for clarification, reuse, etc. Restrictions (e.g., licences needed) might be applicable.

4. **Search package, service, function meta-data (descriptors):** In order to discover what items are stored in the Catalogue, a list of the available packages, services and function meta-data can be obtained using search parameters to limit that list.

5. **Delete package:** Whole package files can be deleted from the Catalogue, on condition that there are no other items depending on them or there are no currently instantiated and running.

6. **Update package:** In order to update the package, a new version of the Package is uploaded/on-boarded on the Catalogue and the referenced annotations to that are updated.

6.1.2 Catalogue SDK Interaction

The planned interactions of the Catalogue and the SDK include the interactions of the Sec. 6.1.1 and the additional following:

1. **VNF recommendation:** A list of VNFs might be suggested to the developer by the Catalogue’s Decision Support mechanism, adequate to his/her profile.
2. Continuous Optimisation:
   - **Feedback to developer:** A snapshot of processed data may be sent back to the developer from the Catalogue’s Continuous Optimisation Mechanism in order to aid in optimisation;

6.1.3 Catalogue V&V Interaction

The planned interactions of the Catalogue and the V&V include the interactions of the Sec. 6.1.1 and the additional following:

1. **Tested Package push/On-boarding:** Whenever a package passes successfully the v&v tests it can be pushed into the Catalogue.

2. **Automated Package Retest/Push:** When an automated test run tests the set of V&V known NS/VNFs and their test state changes, this must be reflected in the Catalogue.

3. **Continuous Optimisation:**
   - **Feedback to V&V:** The V&V component based on actionable information derived from the delta between the monitoring data coming from the production infrastructure and the monitoring data coming from the testing infrastructure the v&v cycle is optimised by triggering new tests considering the real-world behaviour of services.

6.2 SDK and V&V Interactions

NS/VNF developers uses 5GTANGO’s SDK tools to locally create or modify a VNF or a network service. The NS/VNF(s) are then packed into a 5GTANGO package (cf. Sec. 2.5) which can then be uploaded (on-boarded) to the V&V for verification and validation, to the catalogues for sharing or later use, or to the service platform to be instantiated and put to production. This section focuses on the scenario in which a developer interacts with the V&V to test his/hers VNFs or NSs.

6.2.1 Planned Interactions

The following high-level interactions are planned between SDK and V&V:

1. **Establish authenticated user session:** A developer can authenticate himself/herself to a V&V using his/hers credentials. The SDK and its workspace and access components remember this session until the developer logs out or the session times out. During a session, the developer does not need to re-authenticate whenever he/she interacts with the V&V, e.g., to re-upload a package.

2. **Package upload/on-boarding:** Whenever the developer thinks he/she has the package ready, it can be pushed to the V&V to be tested.

3. **Test control:** The test execution can be automatically triggered once the package upload has finished. An alternative is using the SDK’s access component to interact with the V&V’s APIs to manually select and trigger some tests.

4. **Test status query:** The SDK can ask the V&V about test status updates. This might be either implemented through polling mechanisms or with a callback solution to send out update notifications.
5. **Result package retrieval:** Once the V&V has executed all tests, it can store the results to a catalogue. In addition to this, the V&V can create a signed package (cf. Sec. 2.5) that contains or references the test results and return this package to the SDK, i.e., a developer downloads the test results. This allows developers to get detailed insights into the test results and, for example, further analyze performance data generated by performance tests.

### 6.2.2 Example Interaction Sequence

Fig. 6.1 shows a *typical* sequence of interactions that happen between the SDK and a V&V when a developer creates or modifies a NS, packages it, and uploads it for testing. This diagram should be understood as an *example* of a *typical* sequence and skips optional and alternative steps, for example, authentication failures.

### 6.3 SDK and the Service Platform

As described in Sec. 6.2, NS or VNF developers use 5GTANGO’s SDK tools to locally create or modify a VNF or a network service packed as a 5GTANGO package (see Sec. 2.5). Such packages can then be uploaded (on-boarded) to the V&V, to the catalogues, or to the service platform to be instantiated and put to production. This section focuses on the scenario in which a developer interacts with the service platform to on-board VNF or NS packages to be instantiated.

#### 6.3.1 Planned Interactions

Planned interactions between the SDK and the service platform are as follows (all requests start in the SDK, with the Service Platform responding):

1. **Establish authenticated user session:** A developer can authenticate himself/herself to a service platform using his/her credentials. The SDK and its workspace and access components remember this session until the developer logs out or the session times out. During a session, the developer does not need to re-authenticate whenever he/she interacts with the platform, e.g., to on-board a package.

2. **Package upload/on-boarding:** The Service Platform will act as a *proxy* for this feature, towards the Catalogue (see Sec. 6.1.1).

3. **Package file retrieval:** The Service Platform will act as a *proxy* for this feature, towards the Catalogue (see Sec. 6.1.1).

4. **Package, service, function meta-data retrieval:** The Service Platform will act as a *proxy* for this feature, towards the Catalogue (see Sec. 6.1.1).

5. **Service instantiation/termination:** Depending on the platform’s configuration, a developer may be allowed to instantiate or terminate his/her services.

6. **Record retrieval:** Data resulting from the instantiation of services might be retrieved, both for the developer’s SDK and for the V&V internal processes. Restrictions (e.g., licences needed) might be applicable.

7. **Monitoring data retrieval:** Data resulting from the execution of services might be retrieved, both for the developer’s SDK and for the V&V internal processes. Restrictions (e.g., licences needed) might be applicable; This monitoring data might be data produced by a
Figure 6.1: Typical SDK and V&V interactions
currently running service and streamed to the SDK. Filter rules might be applied to reduce the amount of transferred data.

6.3.2 Example Interaction Sequence

Fig. 6.2 shows a typical sequence of interactions that happen between the SDK and a service platform when a developer creates or modifies a NS, packages it, and uploads it. After instantiating this service, the developer can use the SDK to obtain further information about the running service (e.g., records containing floating IP assignments) from the service platform. This diagram should be understood as an example of a typical sequence and skips optional and alternative work flows, for example, authentication failures.

6.4 V&V and the Service Platform

The V&V directly interacts with the service platform on two specific areas:

1. **Using the service platform as a test environment:** When performing a test pass the V&V will drive the service platform via the Sec. 4.3.4.

2. **Automatically pushing updates to the service platform (catalogue):** When automatically updating the catalogue with the results of a V&V test pass. This is detailed in Sec. 6.1.3, and is not covered here.

The V&V also interacts with a stand alone catalogue, allowing NS and VNFs which have passed the V&V process to be stored for reference, referral and searching by other users / developers. However this scenario is predominantly related to V&V and Catalogue interactions. The remainder of this section deals with how the V&V uses the service platform as a test environment. This is closely related to the

6.4.1 Using the Service Platform as a Test Environment

The following details specifically with the first item, test platform integration. During a test pass it is envisaged that the V&V will drive the service platform and will require the following identified functionality.

6.4.1.1 Package Management for Executing a Test Pass

1. Clear the service platform all running NS / VNFs.

2. Clear / Empty the service platform’s Catalogue.

3. Push new NS / VNFs to the service platform’s Catalogue, see Sec. 6.4.1.3.

The V&V maintains its own catalogue of NS/VNF. As the service platform in this instance is only used for testing it does not need to maintain its own catalogue of NS/VNFs, so to avoid data bloat the V&V will wipe out / clear the service platform’s catalogue.

As a new test pass starts it will be necessary to provision a new NS/VNF onto the service platform. The V&V will clear the service platforms catalogue, then upload / provision the necessary packages (services) necessary for the test pass.
Figure 6.2: Typical SDK and SP interactions
6.4.1.2 Service Lifecycle Management During a Test Pass

Test Environment Instantiation
A test pass will need to standup or instantiate the NS or VNF which is under test. Once the NS/VNF is ready the V&V will provision any tests, test harnesses or other test supporting software structures. Once this is complete, and the necessary metric collection and log collection conduits are in place then the tests may start. This leads to the following steps:

1. Instruct the service platform to instantiate NS or VNF.
2. Instruct the service platform to instantiate test harness elements.
3. Provision metric and record collection connections (see below).

Test Environment Termination
After the tests have executed the V&V will need to terminate the currently running test framework, and then terminate the currently NS/VNF. This leads to the following steps:

1. Shutdown metric and record collection connections (see below).
2. Instruct the service platform to terminate currently running test harness elements.
3. Instruct the service platform to terminate currently running NS or VNFs.
4. Remove the NS/VNF that was just tested from the service platform’s catalogue.

6.4.1.3 Record Collection and Monitoring Data
During a test pass the V&V will need to collect information about the performance of the NS/VNF which is under test. This will include information from the VNF/NS itself, and information from the service platform and VIM. This will give an overall view of the system under test and allow the V&V to understand how it behaves under various workloads. There are two main forms of run time information that the V&V would be interested in getting these are:

1. Logs and recorded operational data for a test pass.
2. Monitoring data during the execution of a test pass from:
   3. The system under test
   4. The service platform
   5. The virtual and physical infrastructure

6.5 Service Platform interactions with VIMs and WIMs
The interface between the service platform on one side, and VIMs and WIMs on the other side should support the following interactions:
1. **Resource reservation, usage and freeing:** The service platform needs an interface with each VIM to reserve or free compute, storage and/or memory resources, and to instantiate or terminate an image on those reserved resources.

2. **Service chaining:** The service platform needs an interface with each VIM to create or terminate connectivity between different interfaces of different images hosted on that VIM.

3. **WIM configuration:** The service platform needs an interface with each WIM to instruct it to create or terminate connectivity between two VIMs for a network service that exists of VNFs deployed on both VIMs.

4. **Collection of infrastructure availability:** The service platform needs an interface with every VIM and every WIM to receive data on availability of all the resources under the control of the service platform, and its topology (e.g. which VIMs are directly connected, what is the capacity of the link that combines them, etc.).

5. **Image uploading:** The service platform needs an interface with each VIM to upload an image onto its database.

### 6.6 Recursive Service Platform to Service Platform Interactions

Often delivery of a service to a customer needs the involvement of more than one service platform. For example, the customer requires the service in multiple geographical locations. Since no operator is present in all of them, the service platform that received the request needs to rely on service platforms of other operators to satisfy the request. Another example is where some operators specialize in end-customer-facing operations, whilst others specialize in the “wholesale” provision of infrastructure, or in providing specific types of VNF.

Cooperating service platforms can be organized in a hierarchical architecture (“north-south”) or a more peer-to-peer approach (“east-west”). One “main” service platform provides the end-to-end service to the customer (the one the customer is connected to), and it chooses to involve other service platforms to deliver part of the required capability. From the customer’s perspective, they only interact with, and know about, the main service platform. Further, we believe that the architecture should be recursive, meaning that the each additional service platform that is invoked by the main service platform can in turn arrange for some of the service it provides to be delivered by yet another service platform (and so on).

Based on these requirements, we identify three categories of interactions:

1. **APIs to expose/consult service platform capabilities:** For service platforms to collaborate in delivering a service, they need to expose their capabilities to each other. Among these are their catalogues, to indicate which (segments of) services they can deploy, and infrastructure, to indicate which resources are available to be orchestrated.

2. **APIs to manage and orchestrate VNFs/NSs:** Service platforms need APIs through which they can make/receive requests to/from other service platforms to take part in an orchestration event. An example of such an API is one where the deployment of a single VNF is requested, together with instructions for its connectivity.

3. **APIs that provide feedback (FCAPS Management):** When the orchestration is finished, the main service platform requires monitoring data on the performances of the VNFs deployed in other service platforms, accounting information, etc.
The 5GTANGO service platform needs to both expose and consume these APIs, so that multiple 5GTANGO service platforms can collaborate in delivering services. Next, the 5GTANGO service platform should be able to consume these APIs from other NFV service platforms (e.g. OSM, ONAP, etc.) if they expose multi-SP capabilities.

Further refinement of these APIs and streamlining/collaboration with other organizations such as MEF [3] and 5GEx [4] will be considered in D5.1.
7 Conclusion

This deliverable D2.2 provided a first approach of the overall 5GTANGO architecture. We clarified the relation of 5GTANGO with phase one project SONATA, which we will extend and enhance in 5GTANGO in order to enable a full DevOps workflow including automatic validation of network services. We also described the main components of 5GTANGO, namely the SDK, the V&V platform, and the service platform as well as the interactions between these components.

The SDK supports developers in rapidly creating high-quality network services, covering the entire process from image creation or reuse, generation of corresponding descriptors, and packaging to emulating and testing the developed network service locally. It also provides supporting functionalities to access 5GTANGO’s catalogues and interact with the V&V and service platform.

After developing and testing network services locally, the V&V platform allows developers to upload their services and provides in-depth testing of both functionality and performance (depending on the defined tests). The results of these tests are digitally signed to proof the quality of the network service to others. This will also help operators and third-parties to evaluate existing network services.

Once network services are developed and tested, they can be uploaded to the service platform. The service platform handles the management and orchestration of network services, placing, instantiating, and interconnecting involved VNFs. During execution, the network services are continuously monitored to react to possible changes and ensure quality of service.

The presented architecture covers the main aspects of 5GTANGO and contains contributions from all 5GTANGO partners from academia and industry (telecom, verticals, vendors, integrators, SME). While this deliverable provided an early description of the intended architecture, the implementation will be described in future deliverables. As we gain more experience during the progress of the project and during development of the described components, we will adapt and optimize the architecture in an agile way. The updated architecture and more detailed descriptions of the involved components will be presented in subsequent deliverables.
A State of the Art and Related Work

A.1 EU-founded Collaborative Projects

A.1.1 SONATA

SONATA [1] is a H2020 5GPPP project with the following objectives:

1. Reduce time to market for networked services by shortening service development
2. Optimizing resource utilization and reduce cost of service deployment and operation
3. Accelerate the adoption of software networks in industry
4. Promotion and standardization of project results

These objectives will be achieved by developing the following components:

- **An SDK** that helps and guides the network service/VNF developer in building network service and VNF descriptors and on-bording them in the Service Platform
- **A service platform** that validate who accesses its resources, store network service/VNF descriptors ready to be instantiated, allocates resources for each instantiations request and monitors each instance

A.1.1.1 SDK

The SDK in SONATA is a set of (closely related) tools, written in Python, that help writing network service/VNF descriptors, by automatically validating them. These descriptors are then packaged and submitted to the Catalogue. Packages can be slim (i.e., with just a URL pointing to the VM image(s) that should be used in the service functions instantiations) or fat (i.e., including those VM image(s) – not yet supported in v3).

A.1.1.2 Catalogues

The Catalogue stores package files, as well as packages’, services’ and functions’ descriptors. For slim packages the storage of VM images are not yet supported (v3), they have to be available for download in a storage outside of the platform whenever a service instantiation request is done.

A.1.1.3 Orchestration

The SONATA Orchestrator extends the ETSI MANO, by considering a much more modular and flexible MANO Framework, to which Service or Function Specific Managers can be added (Lifecycle management, placement, scaling, etc.), thus modifying the provided default managers to a specific service or function needs. The overall access to the Orchestrator is controlled by a Gatekeeper. Production monitoring data can be made available to developers during limited time slots. Using OpenStack as VIM (v3).
A.1.1.4 Architecture

Fig. A.1 shows SONATA’s high-level architecture, specifically how it maps into the ETSI model, namely through the defined reference points.

In this figure the service specific managers (SSMs) and function specific managers (FSMs) are particularly prominent, emphasizing one of the innovations of SONATA. The Orchestrator provides a default manager for every network service (at the NFVO level) and VNF (at the VNFM level), but allows this generic behaviour to be adapted for each network service/VNF by their developers, taking into account the needed security levels that have to be met.

A.1.2 T-NOVA

T-NOVA project, Network Functions as-a-Service over Virtualised Infrastructures, aims at promoting the Network Functions Virtualisation (NFV), enabling the migration of certain network functionalities, traditionally performed by hardware elements, to virtualized IT infrastructures, where they are deployed as software components. This is done by a novel framework, allowing operators not only to deploy virtualized Network Functions (NFs) for their own needs, but also to offer them to their customers, as value-added services. Virtual network appliances (gateways, proxies, firewalls, transcoders, analyzers etc.) can be provided on-demand as-a-Service, eliminating the need to acquire, install and maintain specialized hardware at customers’ premises. For these purposes, T-NOVA designed and implemented a management/orchestration platform for the automated provision, configuration, monitoring and optimization of Network Functions-as-a-Service (NFaaS) over virtualised network/IT infrastructures.
This platform leverages and enhances cloud management architectures for the elastic provision and (re-) allocation of IT resources assigned to the hosting of Network Functions. It also exploits and extends Software Defined Networking platforms for efficient management of the network infrastructure. Furthermore, in order to facilitate the involvement of diverse actors in the NFV scene and attract new market entrants, T-NOVA establishes a “NFV Marketplace”, in which network services and Functions by several developers can be published and brokered/traded. Via the Marketplace, customers can browse and select the services and virtual appliances which best match their needs, as well as negotiate the associated SLAs and be charged under various billing models. A novel business case for NFV is thus introduced and promoted.

T-NOVA Orchestrator has been named as TeNOR, which is ETSI-NFV compliant VNF deployment and operation, and whose high level architecture can be seen in Fig. A.2.

Highlights of TeNOR architecture are: 1. Features have been split into three main blocks: 1. Service management and life-cycle, which concentrates all the features at the network service level 1. VNF management and life-cycle, which concentrates all the features at the Virtual network function level 1. WIM, from wide-area network interconnection management, which abstracts away all the interactions with the WAN (e.g., communication between VNFs that may leave in different datacenters, connection to a specific customer’s network, etc.) 1. oth external and internal interfaces are REST/JSON, in a micro-service oriented architecture 1. his kind of architecture allows TeNOR to provide a generic VNF manager, while allowing specific VNF managers to be
provided together with a (set of) VNFs (with an authentication/authorization scheme to control which functions should or shouldn’t be delegated to those ‘specific’ VNF managers. VNFs enter TeNOR coming from the NFStore, while network services do it from the marketplace.

A.2 Open Source Initiatives

A.2.1 Open Source MANO (OSM)

ETSI OSM is an operator-led ETSI community that is delivering a production-quality open source Management and Orchestration (MANO) stack aligned with ETSI NFV information models and that meets the requirements of production NFV networks. The OSM community has set itself the goal of being a world-class production ready solution. OSM Release THREE represents another significant step along this path. It has been engineered, tested and documented to be functionally complete to support Operator RFx processes, and to be a key component for internal/lab and external/field trials as well as interoperability and scalability tests for virtual network functions and services. The OSM community has defined an expansive scope for the project covering both design-time and run-time aspects. Fig. A.3 shows the approximate mapping of scope between the OSM components and the ETSI NFV MANO framework.

A.2.1.1 Run-time Scope of OSM

- An automated Service Orchestration environment that enables and simplifies the operational considerations of the various lifecycle phases involved in running a complex service based on NFV.

- A superset of ETSI NFV MANO where the salient additional area of scope includes Service Orchestration but also explicitly includes provision for SDN control.
- A plugin model for integrating multiple SDN controllers.
- A plugin model for integrating multiple VIMs, including public cloud based VIMs.
- A plugin model for integrating multiple monitoring tools into the environment.
- A reference VIM that has been optimized for Enhanced Platform Awareness (EPA) to enable high performance VNF deployments.
- Support to integrate Physical Network Functions into an automated Network Service deployment.

A.2.1.2 Design-time Scope of OSM

- Support for a model-driven environment with Data Models aligned with ETSI NFV MANO.
- The capability for Create/Read/Update/Delete (CRUD) operations on the Network Service Definition.
- A VNF package generation toolset
- A Graphical User Interface (GUI) to accelerate the network service design time phase, VNF on-boarding and deployment.

A.2.1.3 SDK

OSM Release THREE provides a consistent environment for enhancing interoperability among NFV components (VNFs, VIMs, SDN controllers, monitoring tools) and provides a plug-in framework to make platform maintenance and extensions significantly easier to provide and support.

A.2.1.4 Orchestration

The OSM orchestration stack covers all the aspects of network service lifecycle management: * The launchpad is the interactive GUI into the Run-Time system, and it can be used to manage Lifecycle operations on VNFs and Network Services.

- The Service Orchestration Engine is responsible for all aspects of service orchestration including lifecycle management and service primitive execution. It is effectively the “master” orchestration component in the system that governs the workflow throughout OSM.
- The Configuration Data Store is responsible for persistently storing the SO state, particularly in the context of VNF and NSD deployment records.
- The Network Service Composition Engine is responsible for supporting network service and VNF descriptor composition.
- The Resource Orchestrator Plugin is responsible for providing an interface to integrate the Resource Orchestrator.

A.2.1.5 Catalogues

OSM provides a Catalog Manager, responsible for supporting the Create/Read/Update/Delete lifecycle operations on the defined VNF and network service descriptors and packages, to support the elements in the orchestration stack.
A.2.1.6 V&V

The strict model-oriented approach in OSM allows for a consistent application of descriptors towards all the stages of network service and VNF development and operation, applying validation mechanisms at all stages (development, lab testing, onboarding, deployment). Release THREE incorporates initial support for day N monitoring.

A.2.1.7 Slicing

While slicing is not supported as such in the OSM releases, several experiments has been reported as OSM PoCs, demonstrating the application of OSM to support network slicing.

A.2.2 ONAP

Open Network Automation Platform (ONAP) focuses on real-time, policy-driven orchestration. Fig. A.4 shows ONAP’s overall architecture planned for its first release. ONAP supports the features detailed in the following sections.

A.2.2.1 SDK

The SDK consists of a set of tools that allow the design (onboarding of VNFs/Service Design and Governance), Deployment (service validation, instantiation) & operation of services.

A.2.2.2 Orchestration

The orchestration stack on ONAP provides for service delivery/change/scaling controller instantiation and capacity management across both the application and network controllers. These primary functions are split over a dedicated MSO (master service orchestrator) and corresponding Orchestration functions embedded in the Network and application controllers.
A.2.2.3 Catalogues

The service design and creation (SDC) includes a catalogue providing a VNF/Service & product store with products being added via the design store.

A.2.2.4 V&V

The service onboarding process will allow for a plugin type validation of resource, currently limited to package & descriptors only.

A.2.2.5 Slicing

Planned support in the future for the design of 5G specific services that utilize slicing.

A.2.3 OpenBaton

OpenBaton [21] is an open source project by Fraunhofer FOKUS and TU Berlin. Fig. A.5 shows the architecture (currently in Release 4).

OpenBaton provides multiple interaction channels: Dashboard, a GUI to view and manage the overall framework; CLI, Command Line Interfaces also to manage the framework; REST APIs, to expose NFVO features in a programmatic way; and, SDKs, to extend the OpenBaton features and talk with the NFVO.

Figure A.5: OpenBaton architecture (from [21])
A.2.3.1 SDK

To manage the lifecycle of VNFs, OpenBaton uses VNF managers (VNFMs) that can be described in Java, Python, or Go using the OpenBaton SDK. VNFMs are also partially compatible with Juju charms [59].

A.2.3.2 Orchestration

The core of OpenBaton is its orchestrator, which is an implementation of the ETSI MANO specification [26]. Inside the MANO, the different components like the fault management system, the auto-scaling engine, or the network slice engine, communicate using RabbitMQ [44]. OpenBaton supports automatic scaling, using OpenStack as VIM. However, it is designed to cope with multiple VIMs, by using an infrastructure abstraction layer. Container support (dockers) are foreseen in the next releases.

A.2.3.3 Catalogue

Existing VNFs and VNFMs are stored in a public marketplace, where these packages can be downloaded for free.

A.2.3.4 V&V

OpenBaton does not support full and automatic verification and validation (V&V). However, OpenBaton aims at dealing with faults using a fault management system that automatically triggers alarms if certain thresholds are exceeded. These alarms can be used to react accordingly.

OpenBaton uses by default the Zabbix monitoring system to feed the scaling and fault management features. It can also integrate with other systems.

A.2.3.5 Slicing

Using its Slicing Engine, OpenBaton supports isolated multi-tenancy by managing interconnections with QoS guarantees between different PoPs.

A.3 Commercial Solutions

A.3.1 NetCracker AVP

NEC/Netcracker’s Agile Virtualization Platform and Practice (AVP) contains a Service Platform geared towards telecommunications providers [33]. As shown in Fig. A.6, AVP consists of a number of components geared towards management and operations (MANO) in a telecommunications environment. Most relevant to 5GTANGO are the following components: Hybrid Operations Management (HOM), and Virtualization Development and Operations Center (VDOC).

The Virtualization Development and Operations Center (VDOC) contains tools for modeling and describing network services. It therefore fulfills similar purposes as 5GTANGO’s Service Development Kit (SDK). The Hybrid Operations Management (HOM) component revolves around service deployment, monitoring, control, and assurance. As such, it serves the same architectural purpose as 5GTANGO’s Service Platform (SP). One of AVP’s main goals is service assurance which spans all components inside the AVP. To that end, AVP contains extensive capabilities for monitoring. Such monitoring capabilities are the basis for service testing, both in the functional as well as in
Figure A.6: NEC/Netcracker’s agile virtualization platform and practice

the performance domains. A test management component is responsible for executing tests, collecting monitoring and results information, and for presenting them to users. Catalogues are used extensively within AVP, in order to store and share information across multiple components and external users (cf. Tbl. A.1).

Table A.1: NEC Netcracker feature overview

<table>
<thead>
<tr>
<th>SOLUTION PORTFOLIO</th>
<th>NFVO: RT MANO Network Orchestration SO: End-to-End Service Orchestration VNFM: RT MANO VNF Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTIONAL SUPPORT</td>
<td>Policy &amp; Analytics: Closed Loop Automation &amp; Assurance Security &amp; Licensing: Very comprehensive OSS, BSS and Service Orchestators: Business Enablement Applications</td>
</tr>
<tr>
<td>INTEGRATION &amp; INTERWORKING STATUS</td>
<td>OSS, BSS and Service Orchestators: Business Enablement Applications</td>
</tr>
<tr>
<td>VNS &amp; SERVICE SUPPORT</td>
<td>VNFs Onboarded (VM): &gt; 10 own, 125 3P VNFs Validated (VNFM): As above</td>
</tr>
<tr>
<td>DEPLOYABILITY</td>
<td>Operator &amp; VNF Developer Support Strengths: NG Vendor Ecosystem VDOC Live NFVO Deployments &amp; OSS Integrations: 24 live deployments Including NFVO and OSS integration</td>
</tr>
</tbody>
</table>

A.3.2 NOKIA

Nokia CloudBand helps services providers deliver a better class of cloud service by making it simple to host, orchestrate, automate, & manage VNFs & services. Its modular, scalable & flexible products help reduce time to revenue for new services & use automation & optimization to make network operations lean. CloudBand benefits from the rapid pace of open-source innovation and provides a carrier-grade product that it is ready for deployment (cf. Tbl. A.2).
Table A.2: Nokia CloudBand feature overview

<table>
<thead>
<tr>
<th>SOLUTION PORTFOLIO</th>
<th>NFVO: CloudBand Network Director SO: CloudBand Network Director VNFM: CloudBand Application Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTIONAL SUPPORT</td>
<td>EPA with 3P NFVI Blueprints</td>
</tr>
<tr>
<td>INTEGRATION &amp; INTERWORKING STATUS</td>
<td>20 own, 30 3P</td>
</tr>
<tr>
<td>VNS &amp; SERVICE SUPPORT</td>
<td>Operator &amp; VNF Developer Support Strengths: CloudBand Ecosystem &amp; Cloud Verification Testing Live VNFM Deployments (Own and 3rd party VNFs): 22 own, 12 3P (all VNFM-G)</td>
</tr>
<tr>
<td>DEPLOYABILITY</td>
<td></td>
</tr>
</tbody>
</table>

A.3.3 AMDOCS

Amdocs Network Cloud Service Orchestrator is an open, catalog-driven solution for service providers transitioning from physical networks to dynamic network-clouds. Amdocs Network Cloud Service Orchestrator continuously designs, fulfills and assures network services, from any virtual network functions (VNF) vendor, over all mainstream CMS and SDN-c (cf. Tbl. A.3).

Table A.3: Amdocs feature overview

<table>
<thead>
<tr>
<th>SOLUTION PORTFOLIO</th>
<th>NFVO: Network Cloud Service Orchestrator SO: Network Cloud Service Orchestrator VNFM: Network Cloud Service Orchestrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTIONAL SUPPORT</td>
<td>Data Models, Open Source and / or Other Differentiating Functionality: AT&amp;T ECOMP</td>
</tr>
<tr>
<td>INTEGRATION &amp; INTERWORKING STATUS</td>
<td>VNFs Onboarded (VM): 5 own, ~40 3p VNFs Validated (VNFM): Undisclosed</td>
</tr>
<tr>
<td>VNS &amp; SERVICE SUPPORT</td>
<td>Live VNFM Deployments (Own and 3rd party VNFs): Undisclosed Live NFVO Deployments &amp; OSS Integrations: Undisclosed</td>
</tr>
<tr>
<td>DEPLOYABILITY</td>
<td></td>
</tr>
</tbody>
</table>

A.3.4 ADVA

Ensemble Orchestrator is an ETSI MANO-compliant NFV management and orchestration solution that provides end-to-end VNF and network service lifecycle management. Ensemble Orchestrator encompasses NFV Orchestration and vendor-agnostic VNF Management, along with Analytics for tight-loop feedback. Ensemble Orchestrator enables management and orchestration of virtual services and network functions lifecycles, supporting virtual infrastructure in distributed, centralized, or hybrid models. Orchestrator scales reliably across many parallel clouds and supports resource management and monitoring with correlation between infrastructure, VNFs, and service status (cf. Tbl. A.4).

Table A.4: ADVA Ensemble Orchestrator feature overview

<table>
<thead>
<tr>
<th>SOLUTION PORTFOLIO</th>
<th>NFVO: Ensemble Orchestrator SO: Ensemble Orchestrator VNFM: Ensemble Orchestrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTIONAL SUPPORT</td>
<td>Security &amp; Licensing: Very comprehensive</td>
</tr>
<tr>
<td>INTEGRATION &amp; INTERWORKING STATUS</td>
<td>VNFs Onboarded (VM): 37 3P</td>
</tr>
<tr>
<td>VNS &amp; SERVICE SUPPORT</td>
<td>Live NFVO Deployments &amp; OSS Integrations: 1 live development not integrated with OSS.</td>
</tr>
<tr>
<td>DEPLOYABILITY</td>
<td></td>
</tr>
</tbody>
</table>
## A.3.5 CIENA

Blue Planet helps service providers transform and grow their business by accelerating the delivery of on-demand services, while reducing costs through automation. Deployed in production networks by service providers worldwide, Blue Planet provides integrated network intelligence within an open architecture, to simplify network operations, drive smarter business decisions and elevate the customer experience (cf. Tbl. A.5).

### Table A.5: Ciena Blue Planet feature overview

<table>
<thead>
<tr>
<th>SOLUTION PORTFOLIO</th>
<th>NFVO: Blue Planet SO: Blue Planet MDSO VNFM: Blue Planet</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTIONAL SUPPORT</td>
<td>-</td>
</tr>
<tr>
<td>INTEGRATION &amp; INTERWORKING STATUS</td>
<td>-</td>
</tr>
<tr>
<td>VNS &amp; SERVICE SUPPORT</td>
<td>-</td>
</tr>
<tr>
<td>DEPLOYABILITY</td>
<td>Live NFVO Deployments &amp; OSS Integrations: 13 live deployments including</td>
</tr>
<tr>
<td>NFVO and OSS integration</td>
<td></td>
</tr>
</tbody>
</table>

### A.4 Catalogue State of the Art

For the development of the 5GTANGO Catalogue, we believe the best approach is to use a NoSQL database due to the nature of the metadata that annotate the VNFs/NS. In the following section, we have analysed some of the currently available database solutions to determine which one is the most fitting (cf. Tbl. A.6).

### Table A.6: State of the art catalogue feature overview

<table>
<thead>
<tr>
<th></th>
<th>Cassandra</th>
<th>MongoDB</th>
<th>HBase</th>
<th>Neo4j</th>
</tr>
</thead>
<tbody>
<tr>
<td>Written in Comm Protocol</td>
<td>Java, CQL3/Thrift</td>
<td>C++ BSON (binary JSON)</td>
<td>Java HTTP/REST</td>
<td>Java HTTP/REST</td>
</tr>
<tr>
<td>Scalability</td>
<td>Horizontal, Automatic</td>
<td>Vertical, Horizontal requires sharding</td>
<td>Horizontal, master/slave sharding</td>
<td>Vertical read scaling, master needs to handle writes</td>
</tr>
<tr>
<td>AAA Fault tolerance</td>
<td>Yes Similar nodes, replication</td>
<td>Yes master/slave replication of clusters</td>
<td>Yes master/slave replication of clusters</td>
<td>Yes master/slave replication of clusters variable</td>
</tr>
<tr>
<td>Performance</td>
<td>Based on number of nodes high</td>
<td>high</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>Complexity Cost</td>
<td>None/low FOSS low FOSS, Enterprise edition available</td>
<td>moderate FOSS</td>
<td>high</td>
<td>Free, but features like clustering, replication, caching, online backup, advanced monitoring and High Availability are commercially licensed</td>
</tr>
</tbody>
</table>

- **Apache Cassandra** is an open source, distributed, highly scalable database, capable of providing 100% fault tolerant data without at all compromising on the performance. It is efficient and quick, and hence, especially forms the right choice if you are dealing with a large amount of data spread. It uses CQL3 for database communication which is very similar to SQL. Map/Reduce operations are also possible with Hadoop. All the nodes in a Cassandra setup are similar contrary to the master/slave paradigm, like the one HBase uses. Write operations can be much faster than reads which makes this database perfect for storing analytics.
- **MongoDB** is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas. The main focus of the design of this database is performance over features. Regarding the fault tolerance mechanisms, it offers master/slave replication with built-in sharding. For file storage there is GridFS, a mechanism that allows a file to be split into smaller chunks and saved into the database. If the only reason a relational database can’t be used is the need for pre-defined columns, MongoDB is the one to use.

- **HBase** is an open-source, non-relational, distributed database modeled after Google’s Bigtable and is written in Java. It is developed as part of Apache Software Foundation’s Apache Hadoop project and runs on top of HDFS (Hadoop Distributed File System), providing Bigtable-like capabilities for Hadoop. That is, it provides a fault-tolerant way of storing large quantities of sparse data. Hadoop is the best way to run Map/Reduce operations for huge datasets of information. Because of the underlying HDFS, it is possible to store big files. This database also offers very good random access performance.

- **Neo4j** is a graph database management system developed by Neo4j, Inc. Described by its developers as an ACID-compliant transactional database with native graph storage and processing, Neo4j is the most popular graph database according to DB-Engines ranking. The graph-style of this database is good for analysing data that are complex and interconnected, like social relations, public transport links, road maps or network topologies.

According to the detailed information regarding the metadata to be stored and analyzed one of the aforementioned technologies will be chosen for the 5GTANGO catalogue.
B Acronyms

- 3GPP 3rd Generation Partnership Project:
- 5G Fifth generation of mobile networks
- 5GPPP 5G Infrastructure Public Private Partnership
- API Application Programming Interface
- AR Augmented Reality
- BSS Business Support System
- BW Bandwidth
- CDN Content Delivery Network
- CPU Central Processing Unit
- DASH Dynamic Adaptive Streaming over HTTP
- DE Decision Elements
- DevOps Development and Operations
- DME Decision-Making-Elements
- ECOMP Enhanced Control, Orchestration, Management and Policy
- EPC Evolved Packet Core
- ETSI European Telecommunications Standards Institute
- FSM Function-Specific Manager
- GANA Generic Autonomic Network Architecture
- GPU Graphics Processor Unit
- HDR High Dynamic Range
- HMD Head Mounted Display
- HMI Human-machine interface
- HOT Heat Orchestration Template
- HTML HyperText Markup Language
- IO Input Output
• IoT Internet of Things
• ISG Industry Specification Group
• ISP Internet Service Provider
• JSON JavaScript Object Notation
• MANO NFV Management and Orchestration
• MBT Model-based Testing
• MEC Mobile Edge Computing
• MPC Machine Computer
• NAT Network Address Translation
• NBI northbound interface
• NETCONF Network Configuration Protocol
• NF Network Functions
• NFV Network Function Virtualization
• NFVI NFV Infrastructure
• NFVO NFV orchestrator
• NS Network Service
• NSD NS Descriptor
• NSM Network Service Manager
• OASIS Advancement of Structured Information Standards
• ONAP Open Network Automation Platform
• OpEx Operating Expenditures
• OSS Operations Support Systems
• OTT Over-The-Top
• PC Personal Computer
• PLC Programmable Logic Controller
• PoP Point of Presence
• PSTN Public Switched Telephone Network
• QoE Quality of Experience
• QoS Quality of Service
• QSS Quobis Signalling Service
- RAMI4.0 Reference Architecture Model Industrie 4.0
- REST Representational State Transfer
- RPC remote procedure call
- SBI southbound interface
- SDK Service Development Kit
- SDN Software Defined Networking
- SFU Select Forwarding Unit
- SLA Service Level Agreement
- SP Service Platform
- SPoF Single Points of Failure
- SSM Service-Specific Manager
- SUT System Under Test
- TOSCA Topology and Orchestration Specification for Cloud Applications
- TTCN-3 Testing and Test Control Notation Version 3
- TURN Traversal Using Relays around NAT
- UHD Ultra-High Definition
- URL Uniform Resource Locator
- V&V Verification and Validation
- VIM Virtual Infrastructure Manager
- VLAN Virtual Local Area Networks
- VNF Virtualized Network Functions
- VNFD VNF Descriptor
- VNFFG VNF Forwarding Graph
- VNFM VNF Manager
- VPN Virtual Private Network
- VR Virtual Reality
- WAC WebRTC Application Controller
- WIM WAN Infrastructure Manager
- XML eXtensible Markup Language
C Bibliography


[58] OASIS TOSCA. Tosca simple profile for network functions virtualization (nfv) version 1.0, 2015.

