

H2020 5G-TRANSFORMER Project

Grant No. 761536

SEBASTIAN user and development guide

Abstract

This document provides the user and software development guide for the 5GT-VS reference implementation, i.e. for SEBASTIAN (SErvice BAsed Slice Translation, Integration and AutomatioN) prototype.

Document properties

|  |  |
| --- | --- |
|  |  |
| Document title | SEBASTIAN user and development guide |
| Document responsible | Giada Landi (NXW) |
| Document editor | Giada Landi (NXW) |
| Editorial team | Giada Landi (NXW) |
| Target dissemination level | Public |
| Status of the document | In progress |
| Version | 0.2 |

Document history

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Issued by | Description |
| 0.1  0.2 | 1 May 2019  7 May 2019 | Giada Landi (NXW)  Giada Landi (NXW) | Initial version  Minor additions |

Disclaimer

This document has been produced in the context of the 5G-Transformer Project. The research leading to these results has received funding from the European Community's H2020 Programme under grant agreement Nº H2020-761536.

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.

Table of Contents

[List of Figures 4](#_Toc7626357)

[List of Tables 4](#_Toc7626358)

[List of Acronyms 5](#_Toc7626359)

[1 Vertical Slicer Reference Implementation 6](#_Toc7626360)

[1.1 SEBASTIAN functionalities 6](#_Toc7626361)

[1.2 SEBASTIAN user guide 7](#_Toc7626362)

[1.3 SEBASTIAN software architecture and development guide 14](#_Toc7626363)

[1.3.1 Software components 16](#_Toc7626364)

[1.3.2 How to extend the Vertical Slicer prototype 18](#_Toc7626365)

[2 References 22](#_Toc7626366)

# List of Figures

[Figure 1: SEBASTIAN admin GUI – Main page 8](#_Toc7626367)

[Figure 2: SEBASTIAN admin GUI – Groups visualization 8](#_Toc7626368)

[Figure 3: SEBASTIAN admin GUI – Group creation 9](#_Toc7626369)

[Figure 4: SEBASTIAN admin GUI – Tenants visualization 9](#_Toc7626370)

[Figure 5: SEBASTIAN admin GUI – Tenant creation 9](#_Toc7626371)

[Figure 6: SEBASTIAN admin GUI – Wizard to onboard a VSB 10](#_Toc7626372)

[Figure 7: Example of VSB in JSON format 10](#_Toc7626373)

[Figure 8: SEBASTIAN admin GUI – Wizard to onboard a VNF package 10](#_Toc7626374)

[Figure 9: SEBASTIAN admin GUI – Wizard to specify translation rules 11](#_Toc7626375)

[Figure 10: SEBASTIAN user GUI – Main page 12](#_Toc7626376)

[Figure 11: SEBASTIAN user GUI – VSB visualization 12](#_Toc7626377)

[Figure 12: SEBASTIAN user GUI – VSD creation 13](#_Toc7626378)

[Figure 13: SEBASTIAN user GUI – VSDs visualization 13](#_Toc7626379)

[Figure 14: SEBASTIAN user GUI – VSI instantiation 14](#_Toc7626380)

[Figure 15: SEBASTIAN user GUI – VSI visualization 14](#_Toc7626381)

[Figure 16: SEBASTIAN Software Architecture 15](#_Toc7626382)

[Figure 17: SEBASTIAN software architecture – Translator and Arbitrator 18](#_Toc7626383)

[Figure 18: SEBASTIAN software architecture – Translator Java Interface 19](#_Toc7626384)

[Figure 19: SEBASTIAN software architecture – Arbitrator Java Interface 19](#_Toc7626385)

[Figure 20: SEBASTIAN software architecture – new NFVO drivers 20](#_Toc7626386)

[Figure 21: SEBASTIAN software architecture – NFVO drivers class diagram 21](#_Toc7626387)

# List of Tables

[Table 1: SEBASTIAN functionalities 6](#_Toc7626388)

# List of Acronyms

|  |  |
| --- | --- |
| Acronym | Description |
| 5GT-SO | Service Orchestrator |
| 5GT-VS | Vertical Slicer |
| API | Application Programming Interface |
| AppD | Application Descriptor |
| CRUD | Create-Read-Delete-Update |
| DB | Database |
| DF | Deployment Flavour |
| eMBB | Enhanced Mobile BroadBand |
| ETSI | European Telecommunication Standardization Institute |
| GUI | Graphical User Interface |
| IFA | Interfaces and Architecture |
| IL | Instantiation Level |
| JPA | Java Persistence API |
| LC | Lifecycle |
| LCM | Lifecycle Management |
| mIoT | Massive Internet of Things |
| MEC | Multi-access Edge Computing |
| NBI | Northbound Interface |
| NF | Network Function |
| NFV | Network Function Virtualization |
| NFVI | Network Functions Virtualisation Infrastructure |
| NFV-NS | NFV Network Service |
| NFVO | NFV Orchestrator |
| NS | Network Slice |
| NSD | Network Service Descriptor |
| NSI | Network Slice Instance |
| NSMF | Network Slice Management Function |
| NSSI | Network Slice Subnet Instance |
| NSSMF | Network Slice Subnet Management Function |
| OSS | Operating Support System |
| REST | Representational State Transfer |
| SBI | Southbound Interface |
| SEBASTIAN | SErvice BAsed Slice Translation, Integration and AutomatioN |
| SLA | Service Level Agreement |
| SST | Slice/Service Type |
| URLLC | Ultra-Reliable Low-Latency Communication |
| VNF | Virtual Network Function |
| VNFD | VNF Descriptor |
| VSB | Vertical Service Blueprint |
| VSD | Vertical Service Descriptor |
| VSI | Vertical Service Instance |

# Vertical Slicer Reference Implementation

The 5G-TRANSFORMER Vertical Slicer (5GT-VS) reference implementation, called SEBASTIAN (SErvice BAsed Slice Translation, Integration and AutomatioN), is an open source software prototype developed in Java and based on the Spring framework,[[1]](#footnote-1) which provides all the major 5GT-VS functionalities required by 5G-TRANSFORMER use cases [3].

## SEBASTIAN functionalities

The list of 5GT-VS functionalities and features implemented in SEBASTIAN is reported in Table 1 (for further details about the 5GT-VS architecture and functionalities see [1]).

Table 1: SEBASTIAN functionalities

|  |  |
| --- | --- |
| **Functionality** | **Description** |
| Administrative REST API | REST-based NBI for administrative functions:   * Management of tenants, groups, Service Level Agreements (SLAs) and policies * On-boarding of Vertical Service Blueprints (VSBs) * Retrieval of Network Slice Instances (NSIs) |
| Operational REST API | REST-based north-bound Interface (NBI) for verticals:   * Retrieval of VSBs * Definition of Vertical Service Descriptors (VSDs) * Management of Vertical Service Instances (VSIs) |
| Authentication and authorization | Mechanisms to authenticate the user and authorize REST API requests based on tenant’s profile. |
| Web-based GUI for 5GT-VS administrators | Web GUI to access the functionalities exposed by the administrative REST API. |
| Web-based GUI for verticals | Web GUI to access the functionalities exposed by the operational REST API. |
| Web-based GUI for monitoring | Web GUI page to visualize monitoring data on a per-service basis. |
| Translation between VSD and Network Service Descriptor (NSD) | Implementation of 5GT-VS Translator. |
| Arbitration for 1:1 mapping between VSIs and NSIs | Basic implementation of 5GT-VS Arbitrator:   * SLA verification based on resources consumed by each tenant * Mapping of VSI on single NSI (and vice versa) |
| Arbitration for N:M mapping between VSIs and NSIs | Full-featured implementation of 5GT-VS Arbitrator:   * SLA verification based on resources consumed by each tenant * Management of NSI composition * Mapping of multiple VSIs on single, composite NSIs * Sharing of Network Slice Subnet Instances (NSSI) among multiple, composite NSIs * Arbitration among multiple vertical service instances belonging to the same tenant and with different priorities (including triggering of scaling actions) * Management of service isolation requirements |
| Basic lifecycle management of “non-composite” services | Management of instantiation and termination procedures for “non-composite” services (i.e. VSIs mapped on NSIs without NSSIs) |
| Basic lifecycle management of “composite” services | Management of instantiation and termination procedures for “composite” services (i.e. VSIs mapped on NSIs including one or more NSSIs) |
| Policy management | Management of policies to provide the 5G-TRANSFORMER Service Orchestrator (5GT-SO) with directives about the instantiation of Network Function Virtualisation (NFV) Network Service instances (NFV-NSIs) |
| Service scaling | Management of scaling procedures in VSI and NSI lifecycle |

## SEBASTIAN user guide

This section provides a brief guideline about how to use SEBASTIAN from its web Graphical User Interface (GUI), for administrative actions and verticals’ actions.

It should be noted that SEBASTIAN GUI interacts with SEBASTIAN core using its REST APIs. It is thus possible to use the same REST APIs in order to interact with the system from external components, e.g. from a vertical-managed application that requests a VSI-related operational action in a programmable manner.

In the following, we assume that SEBASTIAN has been deployed, installed and correctly configured with both its core system and its GUI, with the GUI accessible at a generic X.X.X.X IP address. Installation and usage instructions are described in the SEBASTIAN/README.md file for SEBASTIAN core part, and in the SEBASTIAN\_WEB\_GUI/README.md file for the GUI. Further details about installation and configuration mechanisms are available in D3.4 [2].

SEBASTIAN GUI can be accessed from a web browser at the following link: <http://X.X.X.X/sebastian_web_gui/index.html>. At the beginning, the GUI will initially visualize the authentication page where the user can enter its username and password.

### SEBASTIAN Administration GUI

In order to enter SEBASTIAN administration GUI, the user should insert “admin” and “admin” as username and password in the authentication page (admin credentials can be modified through the SEBASTIAN configuration file). The main page shown in Figure 1 will be visualized.

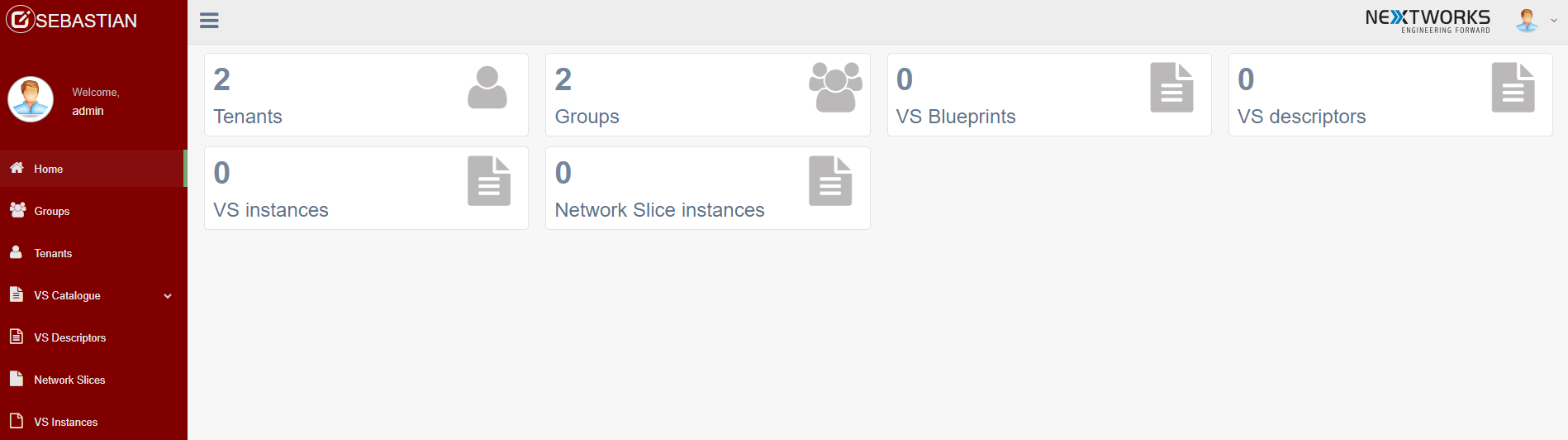


Figure 1: SEBASTIAN admin GUI – Main page

From SEBASTIAN admin GUI the following actions can be performed:

* Creation, visualization and deletion of groups and tenants (i.e. verticals);
* Specification of SLAs for tenants, defining the maximum amount of [Global, Cloud, Multi-Access Edge Computing (MEC)] resources available for each tenant;
* Definition of VS Blueprints (VSB):
  + Configurable parameters of VSB;
  + Associated NSD, Virtual Network Function (VNF) Packages (optional) and MEC App Packages (optional);
  + Translation rules, specifying how to map different ranges of service parameters into NSD deployment flavours (DF) and instantiation levels (IL).
* Visualization of VSD, vertical service instances and network slice instances.

Clicking on the Groups button on the left side of the main page, all the configured groups of tenants will be visualized (Figure 2). Clicking on the Add Group button it is possible to add a new group specifying its name (Figure 3).

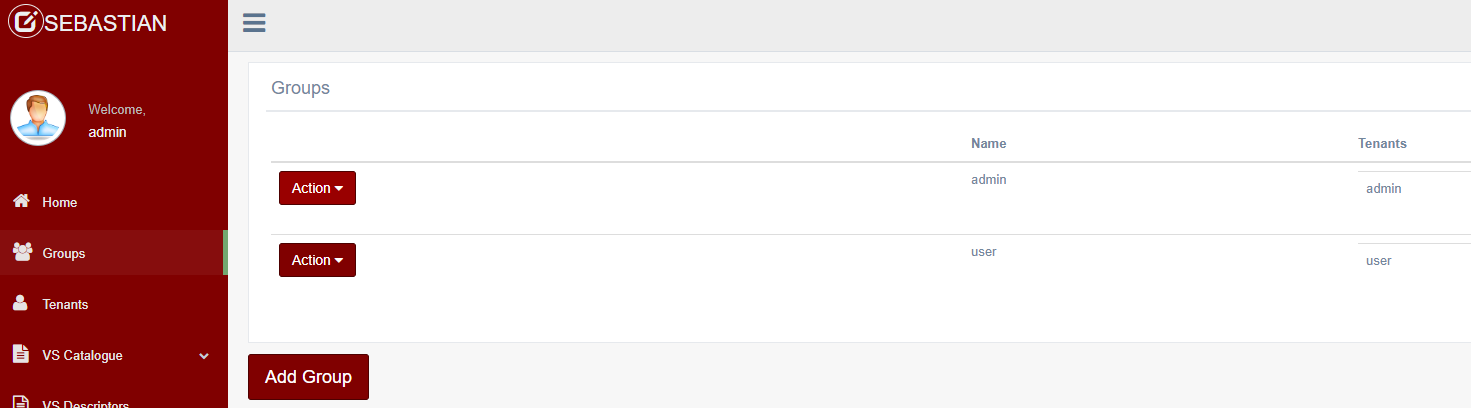


Figure 2: SEBASTIAN admin GUI – Groups visualization

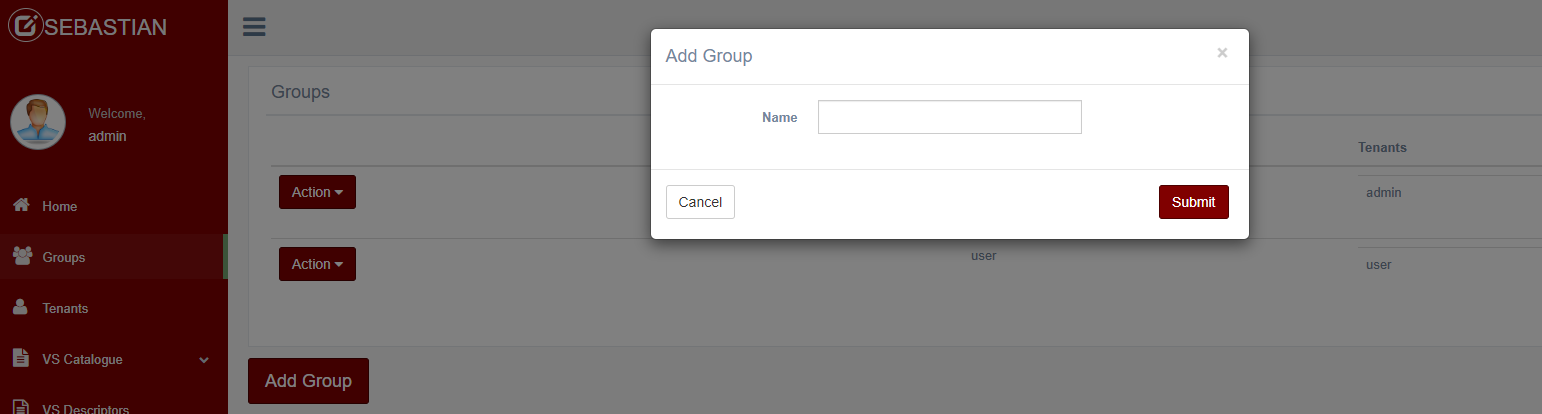


Figure 3: SEBASTIAN admin GUI – Group creation

Clicking on the Tenants button on the left side of the main page, all the tenants belonging to a given group will be visualized (Figure 4). The group can be selected from the “Selected Group” menu on the top of the page. For each tenant, the page visualizes the currently used resources. Clicking on the Add Tenant button it is possible to add a new tenant, selecting its group and specifying its username and password (Figure 5).

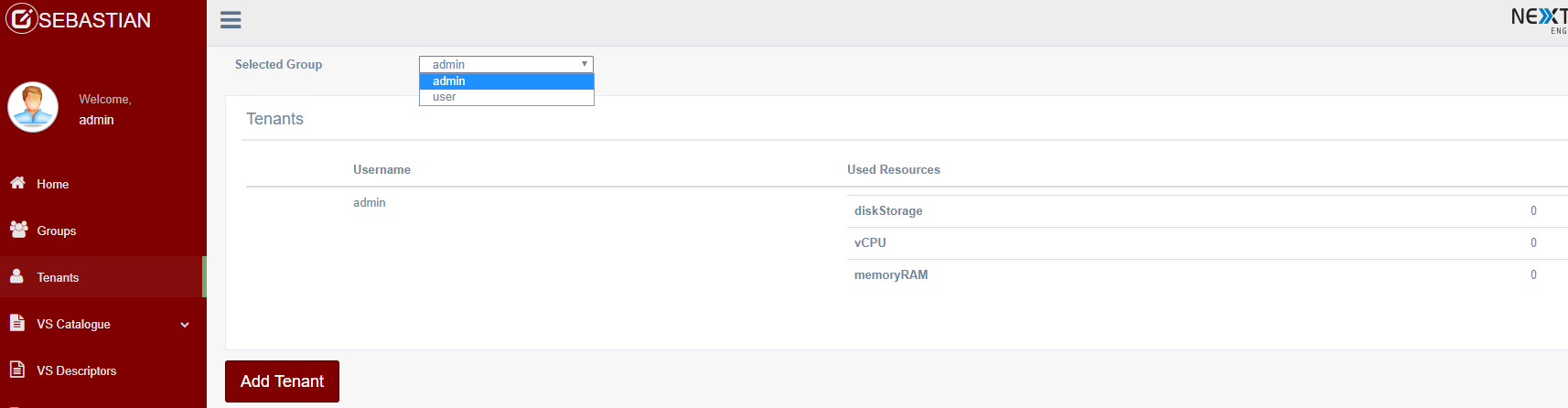


Figure 4: SEBASTIAN admin GUI – Tenants visualization

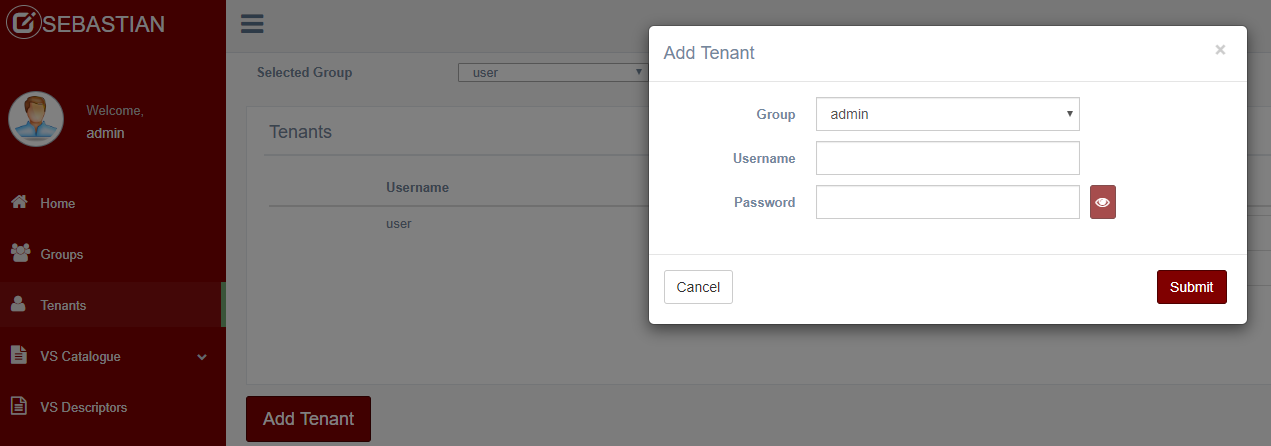


Figure 5: SEBASTIAN admin GUI – Tenant creation

Clicking on the Vertical Service (VS) Catalogue button on the left side of the main page, the VSB menu opens up. Clicking on the “VS Blueprint on-boarding”, the wizard to onboard a new VSB is opened (Figure 6).

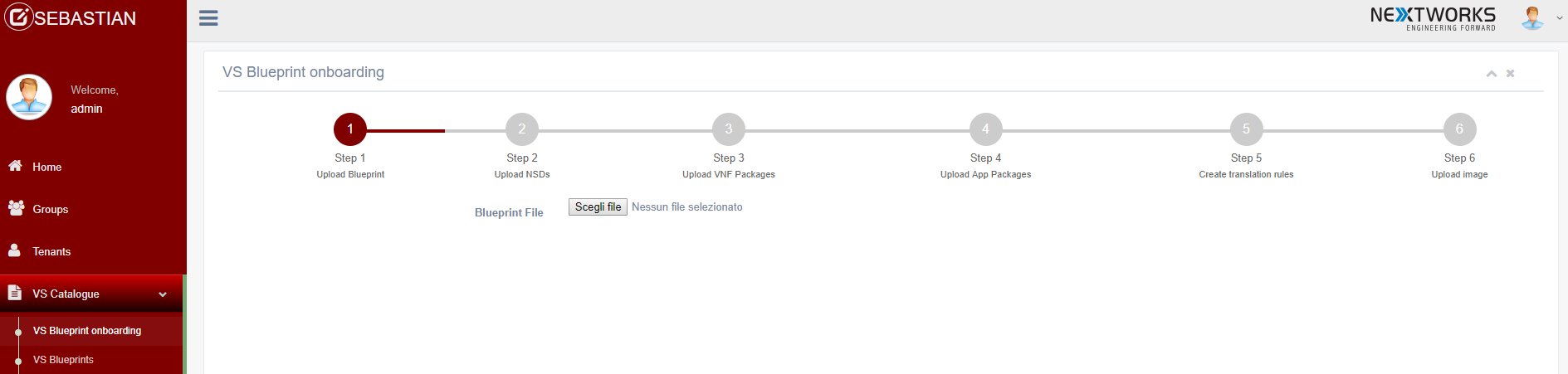


Figure 6: SEBASTIAN admin GUI – Wizard to onboard a VSB

The first step to onboard a VSB is to select a VSB file in json format from the local PC. A simple example of VSB is provided in Figure 7.

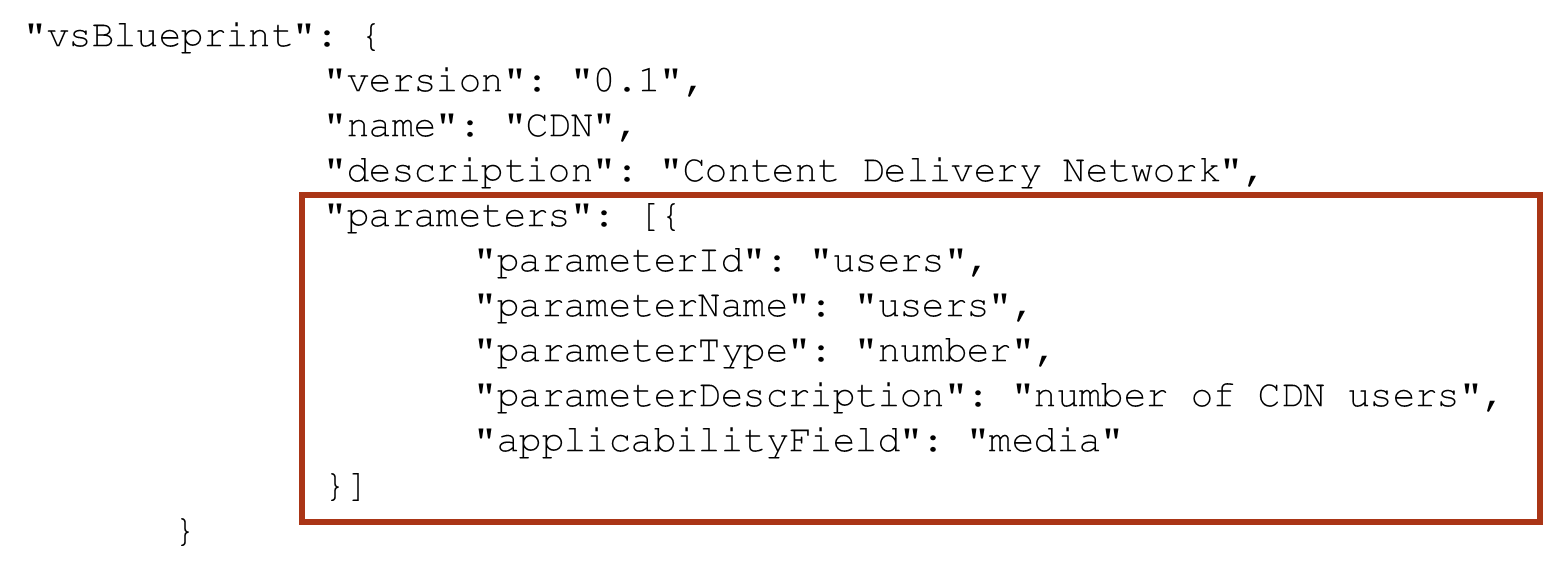


Figure 7: Example of VSB in JSON format

The second step requires to onboard an NSD, still selecting the json file from the local PC. In the third and fourth step, the VNF packages (Figure 8) and the MEC app packages (optional) are requested. In this case, the following information must be provided: name, version and provider of the VNF; checksum (this field is not currently processed); VNFD path, with the URL where the VNF package can be downloaded.

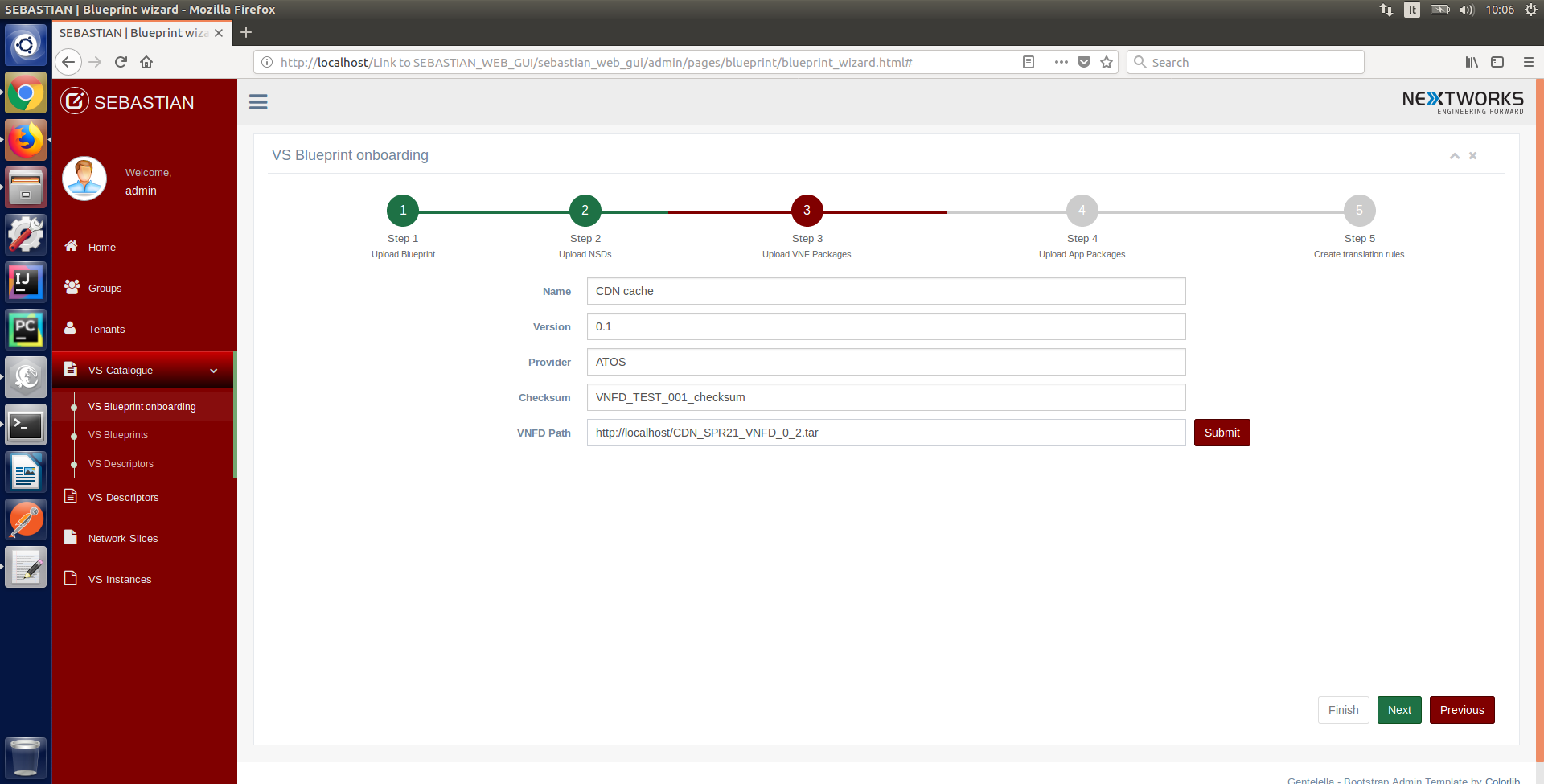


Figure 8: SEBASTIAN admin GUI – Wizard to onboard a VNF package

The final step allows the definition of the translation rules (Figure 9) that will be used by SEBASTIAN Translator to map a VSD (i.e. the VSB with its configurable parameters filled with actual values) into the triple <NSD, DF, IL>, which define how to instantiate a network slice able to meet the service requirements.

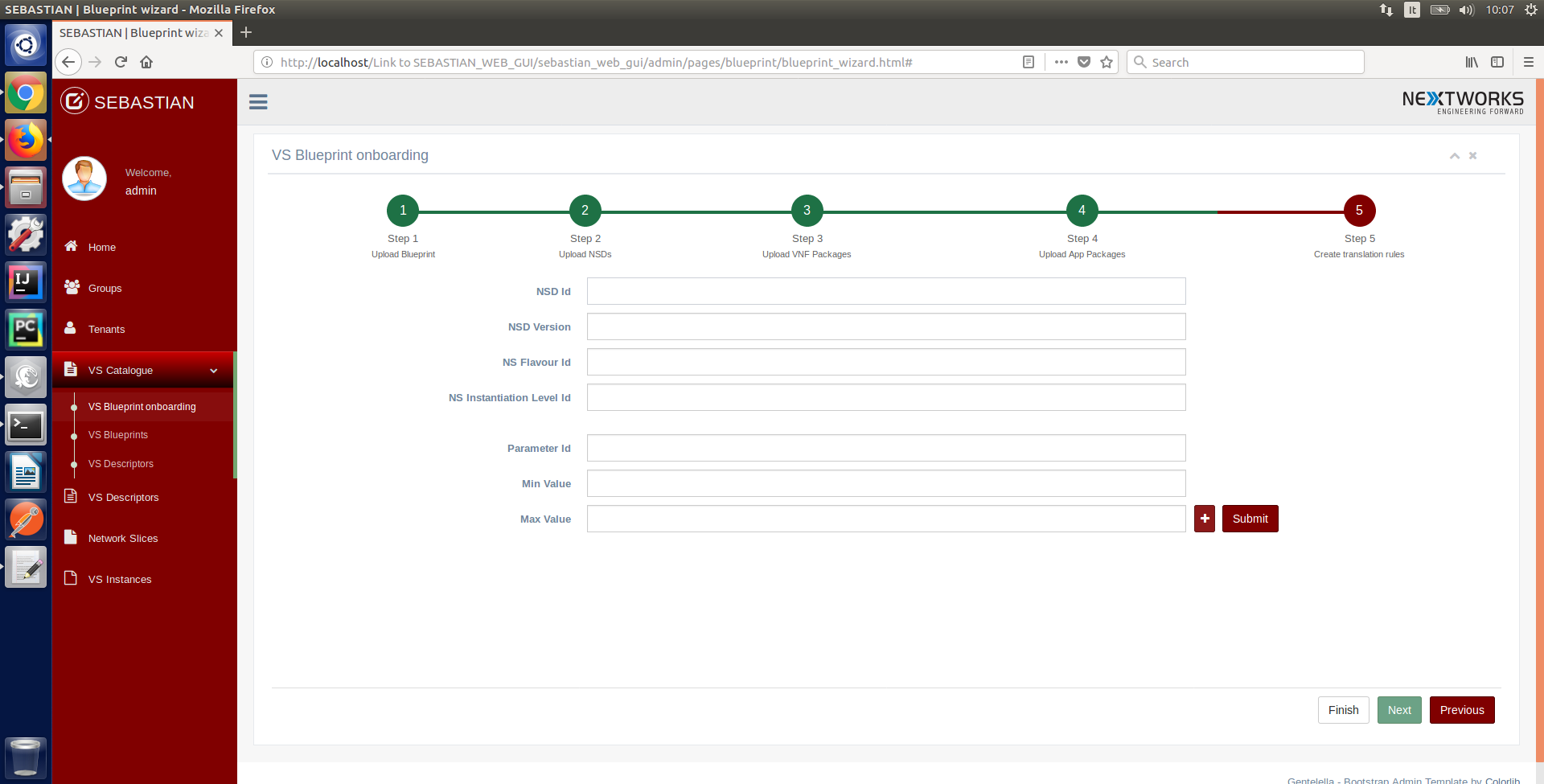


Figure 9: SEBASTIAN admin GUI – Wizard to specify translation rules

A typical translation rule includes the following fields:

* NSD ID, NSD version, NS Flavour ID, NS Instantiation Level ID: these fields identify the NSD, the deployment flavour and the instantiation level that will be used to instantiate the NFV-NSI corresponding to an NSI, which is able to run a service with the value of the configurable parameters compliant with the given ranges.
* A list of <parameter Id, minimum value, maximum value>: defines a list of ranges of values for the service configurable parameters that will be filled in the VSD. Whenever the values provided by the vertical are included in these ranges, SEBASTIAN translator will select the triple <NSD, DF, IL> specified above.

The last three buttons on the left side of the home page allow the visualization of information about the entities available in the system, i.e. the VS Descriptors on-boarded by the verticals, the network slices and vertical services instantiated in the platform.

### SEBASTIAN User GUI

In order to enter SEBASTIAN user GUI, the user (a vertical industry using the 5G-TRANSFORMER system) should insert his/her username and password (as configured in the tenant section of the admin GUI) in the authentication page. The main page shown in Figure 10 will be visualized.

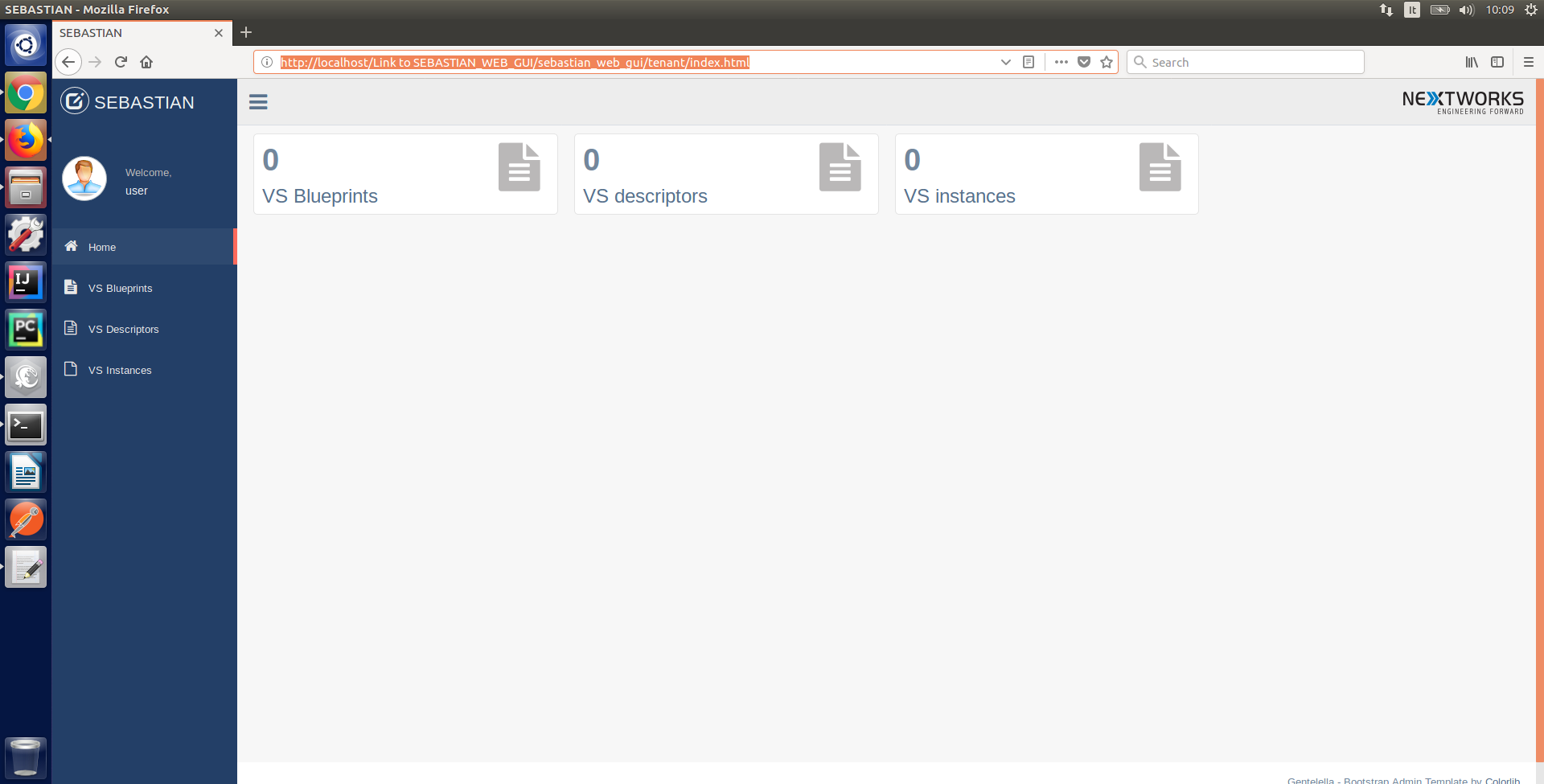


Figure 10: SEBASTIAN user GUI – Main page

From SEBASTIAN user GUI, the following actions can be performed:

* View VS Blueprints;
* Define new VS Descriptors (VSD) from a VS Blueprint;
* Manage the VSDs owned by the user;
* Request operational actions on the VS instances owned by the user:
  + Instantiate a new VSI from VSD;
  + View the information of an existing VSI;
  + Terminate an existing VSI;
  + Modify an existing VSI.
* Visualize the monitoring data for an active VSI;

Clicking on the VS Blueprint button on the left side of the main page, all the VSBs available in the catalogue will be visualized in a summarized list (Figure 11). The “Action” button on the left of each VSB entry allows to visualize further information about the VSB or to create a new VS Descriptor starting from the VSB.

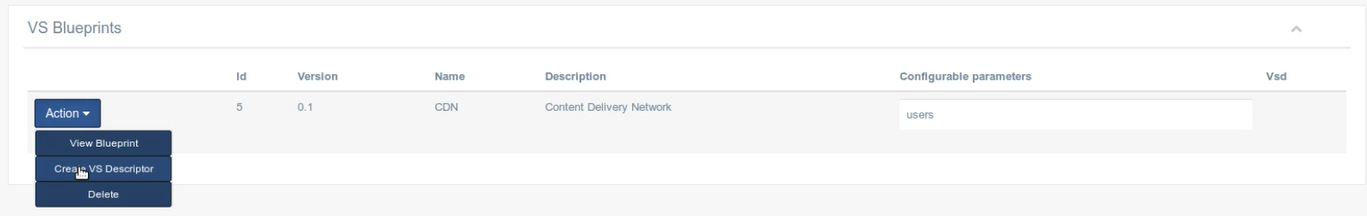


Figure 11: SEBASTIAN user GUI – VSB visualization

Clicking on the “Create VS Descriptor” entry in the menu, a new window is visualized in order to create a new VSD from the given VSB (see Figure 12). The user needs to enter a name and a version for the new VSD and to provide values for the configurable parameters that were defined in the VSB (the number of users in the example). Moreover, the user needs to indicate the Slice Service Type (SST) to be used in the NSI which will be used to run the service (for example, an enhanced Mobile BroadBand – eMBB – slice), the entity responsible for the management of the slice (i.e. whether the slice is provider-managed or tenant-managed[[2]](#footnote-2)) and if the VSD is public. Public VSDs can be used also by other verticals to instantiate their own vertical services.

Once the VSD is created, it becomes available for the instantiation of new vertical services. The list of available VSDs can be visualized clicking on the VS Descriptors button, on the left side of the main page (Figure 13).

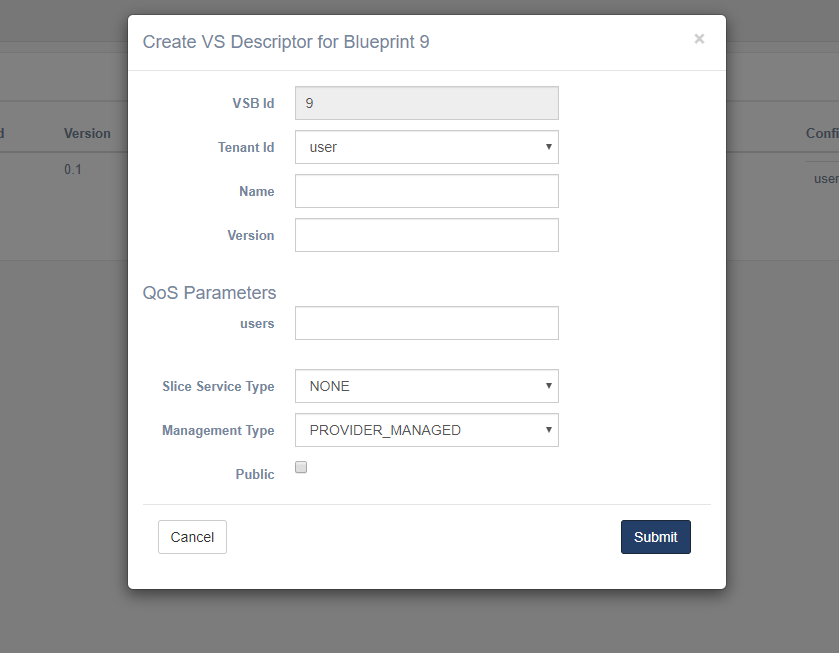


Figure 12: SEBASTIAN user GUI – VSD creation

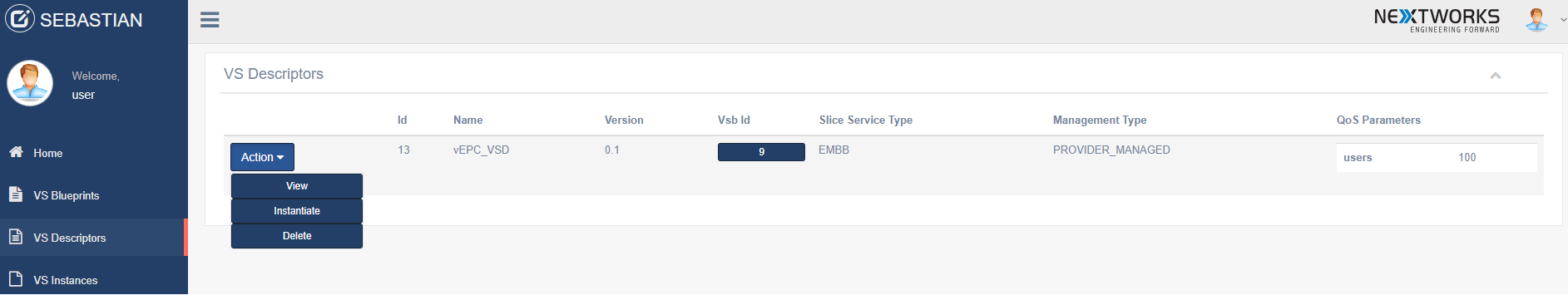


Figure 13: SEBASTIAN user GUI – VSDs visualization

Clicking on the Action button on the left of each VSD entry, it is possible to view the details of the VSD, instantiate a new VSI from the VSD, or delete the VSD. In order to instantiate a VSI, the user should specify a service name, the name of the tenant and a service description (see Figure 14).

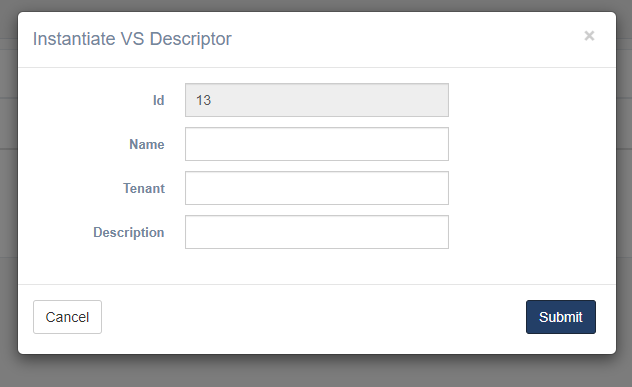


Figure 14: SEBASTIAN user GUI – VSI instantiation

Once the service instantiation is requested, the user can visualize the status and the information related to the service clicking on the VS Instances button on the left side of the main page, which opens the list of available VSIs (Figure 15).

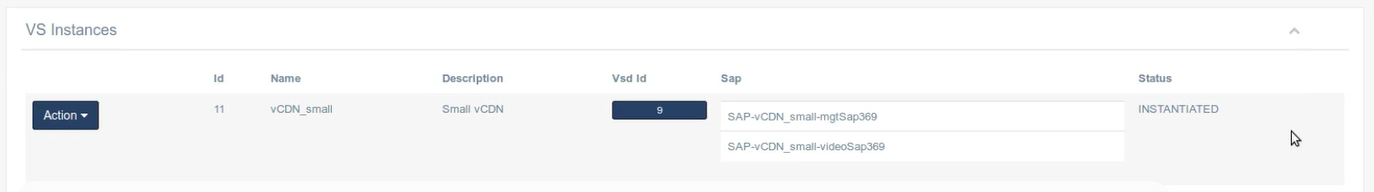


Figure 15: SEBASTIAN user GUI – VSI visualization

## SEBASTIAN software architecture and development guide

As mentioned, SEBASTIAN prototype is developed in Java and it is based on the Spring framework[[3]](#footnote-3), adopting Apache Maven[[4]](#footnote-4) as build automation tool. The software architecture, shown in Figure 16, follows a modular approach designed to simplify the introduction of future extensions in the system. The internal modules of SEBASTIAN implement Java interfaces to provide their functions towards other entities. A RabbitMQ[[5]](#footnote-5)-based message bus (with messages encoded in JSON format) enables the asynchronous interactions among the different software modules. This choice allows an easy extension of such communications to external entities that may use the same message bus to interact with specific system components, following a subscription/notification approach. A PostgreSQL[[6]](#footnote-6) database is used to implement SEBASTIAN catalogues and records, as well as to maintain its internal information persistency. The management of the databases within the software is wrapped through the usage of the Java Persistence API (JPA). On top of the system core, SEBASTIAN implements also the web-based GUI described in Section 1.2.

SEBASTIAN north-bound interface (NBI) is implemented through a set of REST controllers that implement the 5GT-VS REST APIs and invoke the corresponding services in the system core. The CRUD (Create-Read-Delete-Update) requests received at the management REST API (e.g. related to user groups, tenants, SLAs, or policies configuration and VSB creation) are elaborated through services that interact in read/write with the associated databases, modelled as JPA repositories. The REST requests for operational actions on VSIs are handled through the *VS operation service*, which in turn invokes the Engine dispatcher. From here, the processing of the requests is handled in an asynchronous manner, coordinating the exchange of ordered messages between the software modules cooperating for the execution of the given command. For example, a request to create a new VSI is initially processed at the corresponding *VSI Lifecycle (LC)* *Manager,* which invokes (synchronously) the *VSD-NSD Translator* and then the *Arbitrator*. The arbitrator implements algorithms embedding the logic of network slice sharing and multi-service arbitration. It provides in output the specification of the NSI to be deployed and, if needed, information about the NSSI to be re-used and if/how they must be scaled. Moreover, in case of arbitration decisions involving service arbitration, the arbitrator output specifies also the actions (i.e. scaling or termination) that must be performed on existing VSIs/NSIs before instantiating the new one.

The requests for new NSIs are handled at the *NSI LC Manager*, which invokes the NFV Orchestrator *(NFVO) Service* to request the creation of the corresponding NFV-NSI to the underlying 5GT-SO. If an NSI includes existing NSSIs, the NFV-NSI request will include the references to the IDs of the nested NFV-NSIs. In all the cases where actions on existing VSIs must be performed before instantiating the current one (and thus its NSI), a new instance of *VSI Group Coordinator* is created, with the responsibility of handling the ordered list of needed actions.

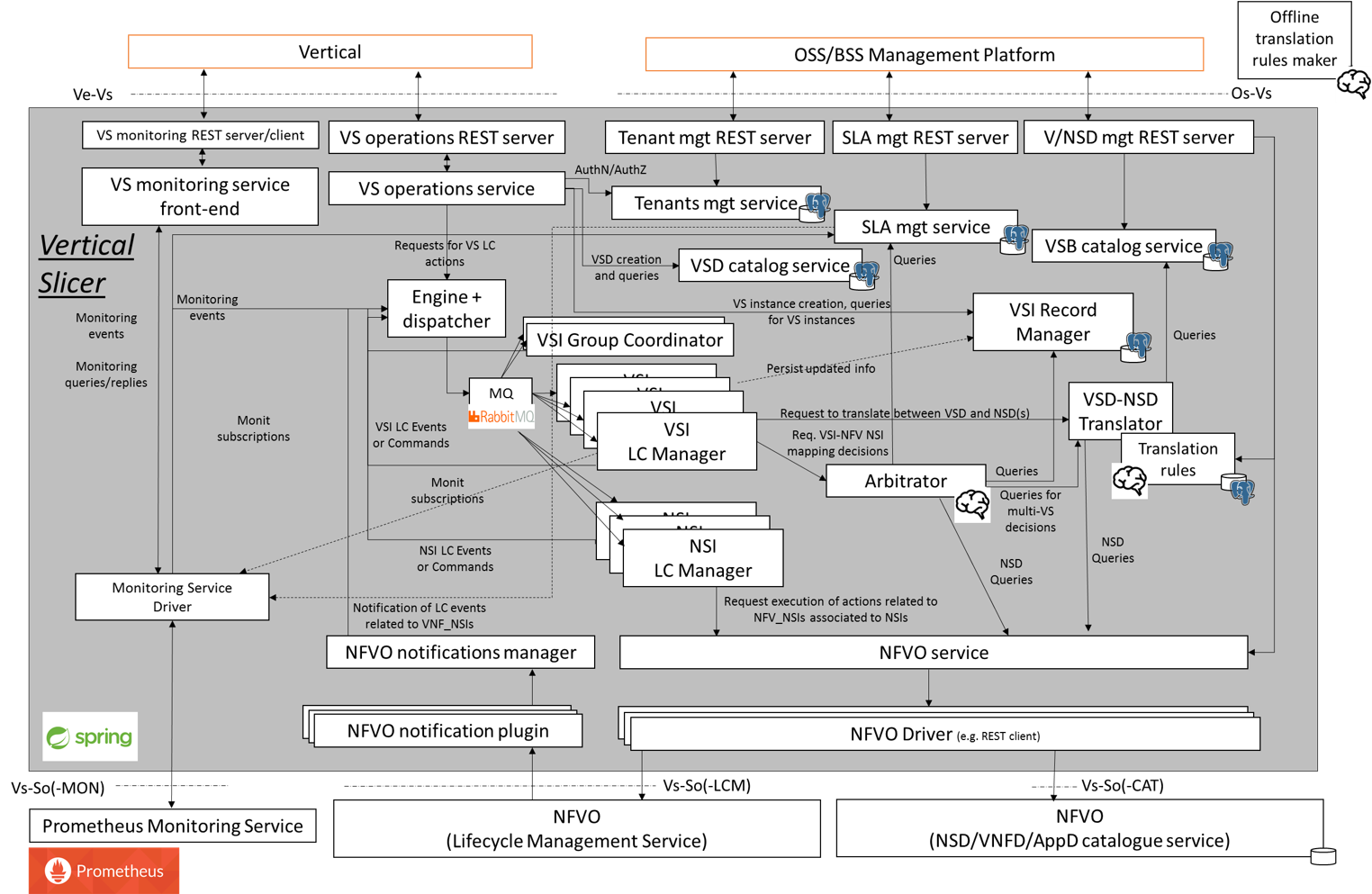


Figure 16: SEBASTIAN Software Architecture

At the southbound interface (SBI), the interaction with the 5GT-SO is handled through a specific driver which embeds a REST-based implementation of the consumer side of the ETSI NFV IFA 013 interface [4], selected as reference NBI of the 5GT-SO. Different *NFVO drivers* can be supported, enabling the interaction with other NFVOs (see Section 1.3.2 about how to extend SEBASTIAN to integrate new NFVO drivers). The communication between the internal modules of the 5GT-VS and different NFVO drivers is mediated through the *NFVO Service*, which provides a single service access point and dispatches the incoming requests towards the correct driver.

### Software components

This section provides a list of the major software components of SEBASTIAN.

Engine dispatcher

The *Engine dispatcher* (package it.nextworks.nfvmano.sebastian.engine) is a centralized entity that dispatches the internal messages among the major components of SEBASTIAN in charge of handling the lifecycle of the controlled entities (i.e. vertical services and network slices). This approach enables an asynchronous processing of the different operations related to multiple entities. For example, the dispatcher distributes messages encoding lifecycle actions/events targeting a given Vertical Service towards the corresponding VSI LC Manager. Message forwarding is based on the RabbitMQ message bus, where VSI LC Managers and NSI LC Managers are the consumers of the messages and routing is performed through the RabbitMQ topics.

VS operations service

The *VS operations service* (package it.nextworks.nfvmano.sebastian.vsnbi) processes the requests related to the management of VSIs. These requests are issued by the REST controller (called *VS operations REST server* in Figure 16) receiving the related REST requests on the NBI. Queries are handled internally at the VS operations service, interacting with the Vertical Slicer database. On the other hand, requests related to lifecycle actions like VSI creation, modification or termination, are forwarded to the *Engine dispatcher*, as they are treated in an asynchronous manner.

VSI LC Manager

*VSI LC Managers* (package it.nextworks.nfvmano.sebastian.engine  
.vsmanagement) are entities created at runtime to control the whole lifecycle of a single VSI, from its creation up to its termination, including the scaling action. Each VSI LC Manager implements the finite state machine regulating the lifecycle of a vertical service and keeps the status and the internal information related to the specific VSI (also stored persistently in the database, see the *VSI Record Manager* component in Figure 16). It triggers all the actions needed at the different stages of the VSI existence, based on external events or commands.

Arbitrator

The *Arbitrator* (package it.nextworks.nfvmano.sebastian.arbitrator) maps the requests for VSIs associated to a given vertical (i.e. tenant) into one or more network slices described through the triple <NSD, DF, IL> returned by the *VSD/NSD Translator*. The Arbitrator support both 1:1 and N:M mapping between VSIs and NSIs. In the former case, for each VSI a new, dedicated NSI is created. In this situation, the Arbitrator only verifies the feasibility of the new NSI creation, based on the amount of resources still available for the given tenant, according to the established SLAs. In the N:M mapping between VSIs and NSIs, the Arbitrator manages the decomposition of network slices in multiple network slice subnets and the sharing of such subnets across multiple slices. This is regulated according to the service isolation requirements defined by the vertical in the VSI requests. Moreover, the Arbitrator also supports the arbitration among concurrent services with different priorities and belonging to the same tenant, selecting the services (and related slices) to be terminated or scaled down in case of scarcity of resources in the established SLAs.

NSI LC Manager

The *NSI LC Manager* (package it.nextworks.nfvmano.sebastian.engine  
.nsmf) manages the lifecycle of NSIs focusing on commissioning (creation), operation (activation and deactivation), scaling and decommissioning (termination) phases. It also permits to query the instantiated network slices or a specific one to view the already deployed network slices and their related information. Network Slice Subnet LCM is internal to Network Slice LCM. Both 1:1 and 1:M mapping between a network slice and network slice subnets are supported, i.e. a network slices may be composed of one or more network slice subnets.

VSD/NSD Translator

The *VSD/NSD Translator* (package it.nextworks.nfvmano.sebastian  
.translator) translates a VSD into a triple <NSD, DF, IL> which describes the characteristics and the size of the network slice to be deployed for the vertical service in the given VSD. The translation is performed through a set of “Translation Rules”, which are defined by the administrator during the on-boarding of the VSB and identify the type of network slice to be instantiated for different QoS parameters specified by the vertical in the VSD. The Translator is also able to select service-specific policies to be applied to the resulting NFV-NS instance, based on per-service SLA requirements defined by the vertical user in the VSD.

VSI Group Coordinator

The VSI Group Coordinator (package XXX) coordinates aggregated groups of lifecycle actions (e.g. ordered set of VSI/NSI scaling or termination actions) that must be performed jointly as a consequence of an Arbitrator’s decision. This is adopted in two major cases: (i) when a group of network slice subnets must be scaled up or down as consequence of a network slice sharing decision and (ii) when a number of VSIs must be terminated or scaled down to make room for a new VSI with higher priority.

NFVO Service

The *NFVO Service* provides a single service access point towards the underlying NFVOs for all the components of the 5GT-VS. This is achieved by exposing a java interface which implements the ETSI NFV IFA 013 [4] abstract messages for management of NSDs/VNFDs and lifecycle management (LCM) of NFV Network Service instances. The NFVO Service builds and handles all the different NFVO drivers, which in turn translates from the IFA 013 abstract primitives to the particular messages and information models adopted by the NBIs of each NFVO.

NFVO Driver

In SEBASTIAN we use a driver-based approach to interact with the system components underneath, i.e., the 5GT-SO or other NFVOs. Each driver implements a common interface, based on ETSI NFV IFA 013 [4]. SEBASTIAN included an implementation of the driver for the 5GT-SO (in the *drivers/SM\_DRIVER* folder in the repository) with an initial set of functions for the instantiation, scaling and termination of a network service, namely createNsIdentifier, instantiateNs, scaleNsm terminateNs and getOperationStatus. Moreover, it supports the on-boarding of NFV-NS Descriptors (NSDs), VNF Descriptors (VNFD) and Multi-Access Edge Computing (MEC) Application Descriptors (AppD).

### How to extend the Vertical Slicer prototype

SEBASTIAN has been designed to facilitate its extension with new algorithms and lifecycle management logic, as well as its interfaces towards other external entities, in particular to other NFVOs. In the following we provide some details about how to extend SEBASTIAN to:

* Introduce new algorithms for 5GT-VS Translator and 5GT-VS Arbitrator;
* Introduce new NFVO drivers to interact with different types of NFVO.

New 5GT-VS Translator and 5GT-VS Arbitrator algorithms

SEBASTIAN can be easily extended with new algorithms for (i) the translation between VSD and the triple <NSD, DF, IL> that identifies the kind of network slice to be instantiated for the given vertical service and (ii) the service arbitration decisions, in particular regarding the sharing of network slices and network slice subnets, as well as potential termination or scaling actions to be performed on low priority services to guarantee the resources for services with higher priority. As shown in Figure 17, SEBASTIAN translation and arbitration algorithms are placed in the packages *it.nextworks.nfvmano.sebastian.translator* or *it.nextworks.nfvmano.sebastian.arbitrator,* respectively.

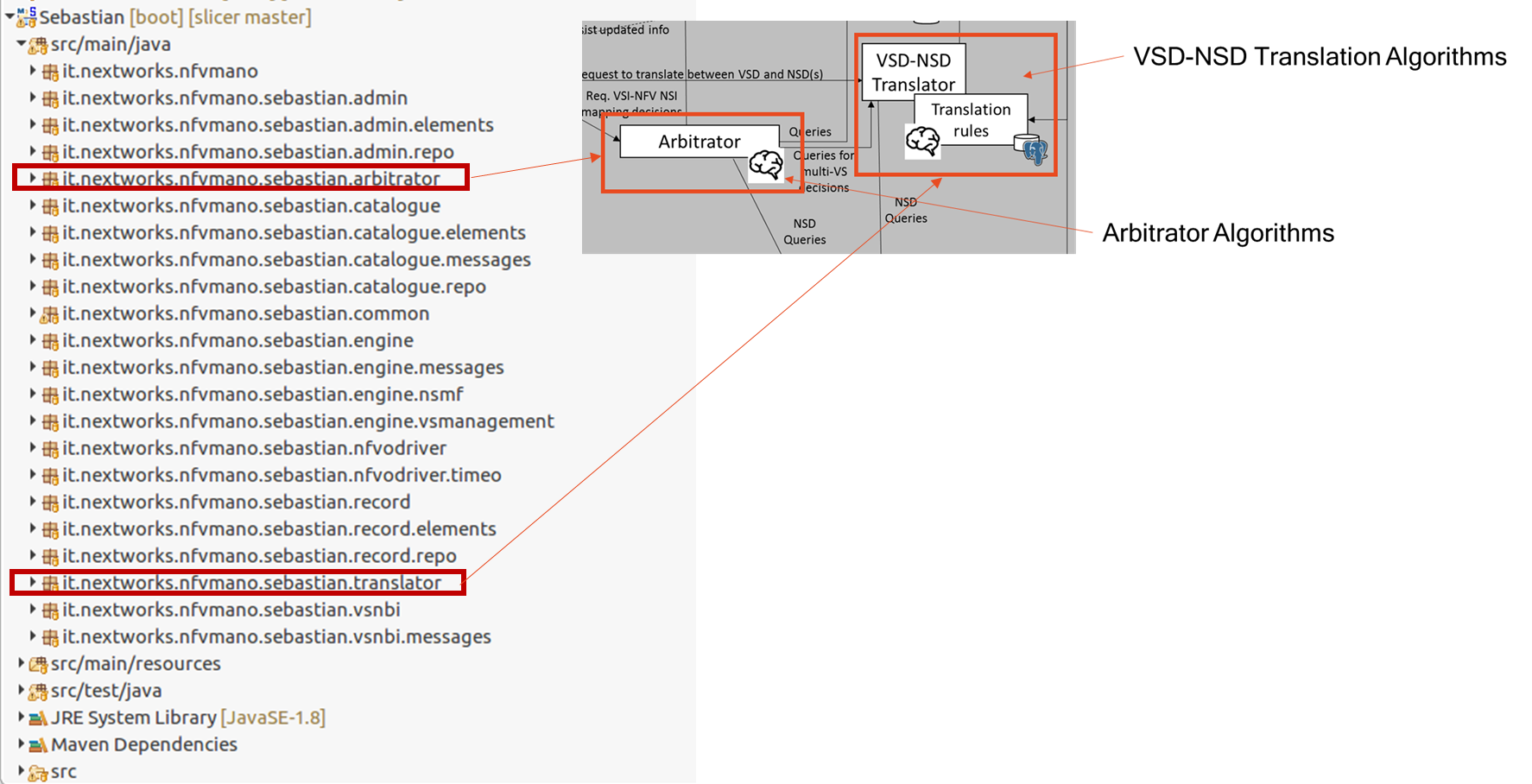


Figure 17: SEBASTIAN software architecture – Translator and Arbitrator

The implementation of a new translator algorithm requires to write a new java class that extends the *AbstractTranslator* abstract class, which in turn implements the *TranslatorInterface* Java interface (see Figure 18). The new class must implement the *translateVsd* method, which gets as input a list of VSD IDs and must return, for each of them, the characteristics of the network slice needed to support them. Such characteristics are expressed in terms of NSD of the required NFV-NS and the pair <DF; IL> that describes how to instantiate the NFV-NS, so that it meets the network slice requirements.

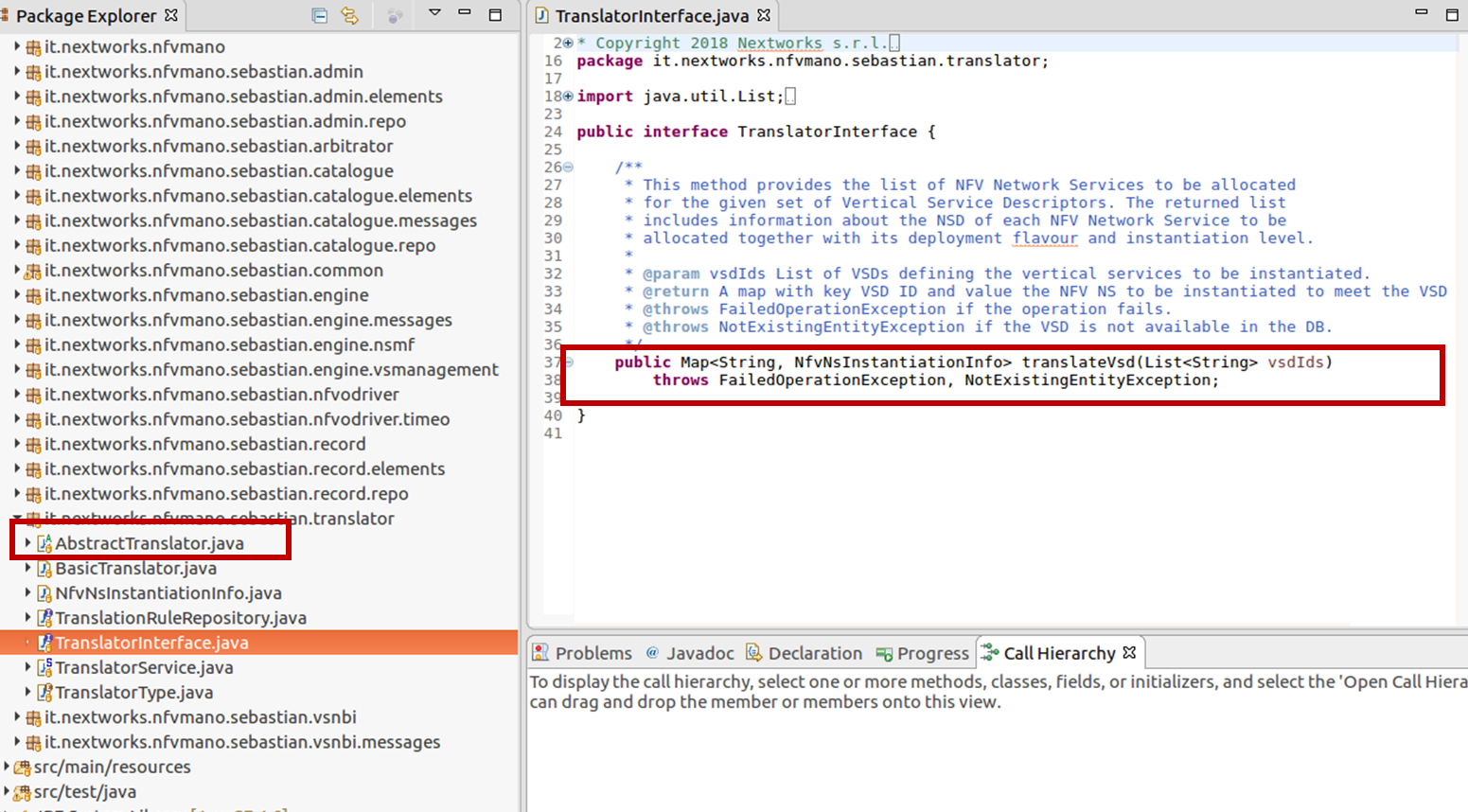


Figure 18: SEBASTIAN software architecture – Translator Java Interface

A similar approach must be followed to introduce a new arbitration algorithm. In this case, the developer needs to implement a new arbitrator class that extends the *AbstractArbitrator* abstract class and implements the *computeArbitratorSolution* method of the *ArbitratorInterface* Java interface (see Figure 19).

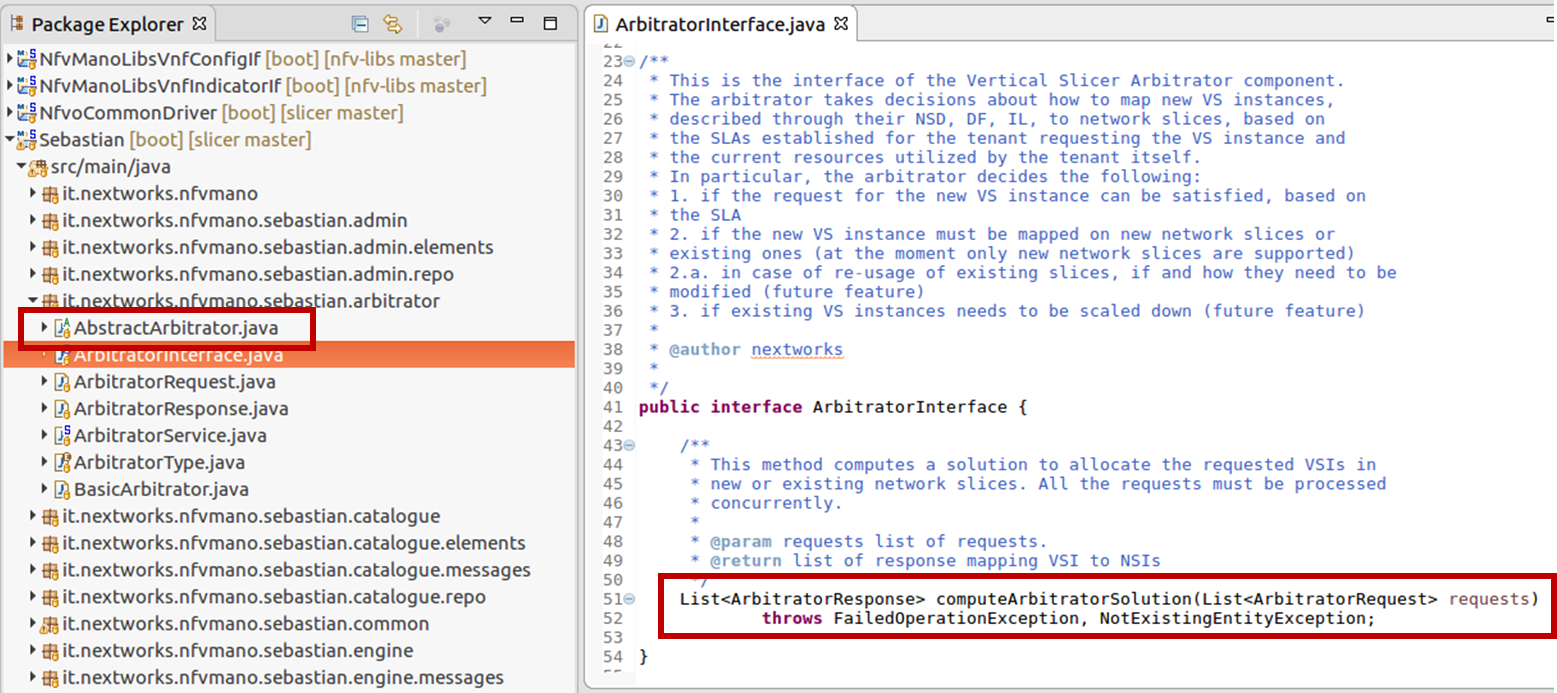


Figure 19: SEBASTIAN software architecture – Arbitrator Java Interface

New NFVO drivers

In SEBASTIAN, an NFVO driver implements the 5GT-VS SBI towards a specific NFVO. In 5G-TRANSFORMER, the interface between 5GT-VS and 5GT-SO is based on ETSI NFV IFA 013. However, different orchestrators may expose different interfaces and information models at their NBI. In order to enable SEBASTIAN interoperability with different NFVOs, the software architecture allows the definition of additional NFVO drivers, each of them implementing the client side of the NFVO-specific APIs, corresponding to the subset of IFA 013 primitives needed for VS-SO interaction (see Figure 20). In particular, the following mechanisms should be at least supported:

* on-boarding and querying of NSDs, VNF Packages, and (optionally) MEC Application Packages;
* lifecycle management actions for creation of NFV-NSI ID, instantiation, termination, scaling, and query of NFV-NSIs.

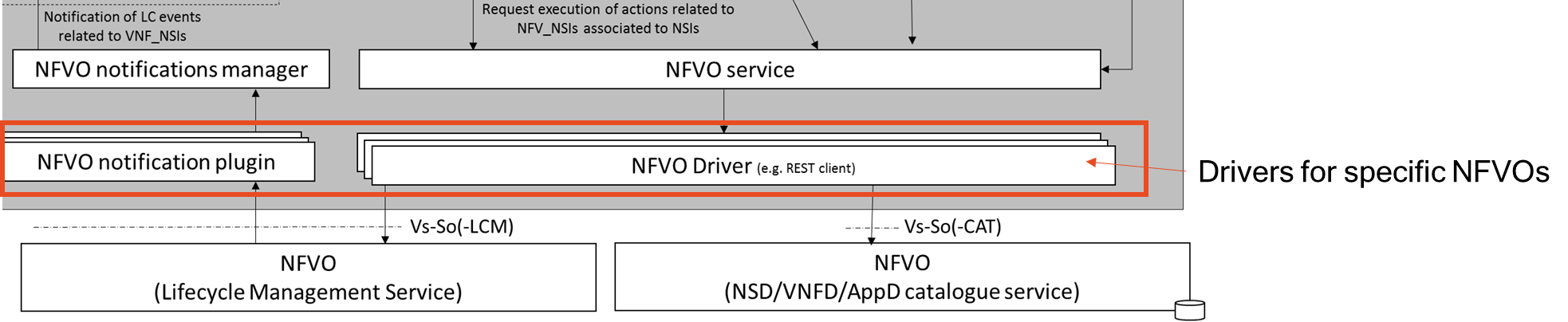


Figure 20: SEBASTIAN software architecture – new NFVO drivers

In particular, an NFVO driver should perform the following actions:

* translate between the NFV information models adopted in SEBASTIAN and the ones adopted in the specific NFVO for descriptors, requests, and responses format;
* implement the mechanisms to interact with the specific NFVO. For example, if the NFVO exposes a REST API, the NFVO driver should implement a REST client;
* implement mechanisms to generate NFV-NSI LCM notifications from the NFVO to SEBASTIAN core, invoking SEBASTIAN’s NFVO notifications manager. In several cases, NFVOs do not support natively asynchronous notifications; in these situations, the driver should implement a polling mechanism to query the NFVO, detect any change in the NFV-NSI status and generate the related notification.

As shown in Figure 21, a new NFVO driver (*NfvoXDriver* in the picture) should be developed as a new class that extends the *NfvoAbstractDriver* abstract class and implements the methods of the Java interfaces implemented by the *NfvoAbstractDriver* class itself. The implementation of the methods should send the corresponding commands and requests to the NFVO, as well as retrieve and return the related response(s). In case of NFV-NSI LCM requests, after sending the message to the NFVO, the driver should also check when the request processing is completed, verify its result, and send the related asynchronous notification invoking the methods of the *notificationHandler* object.

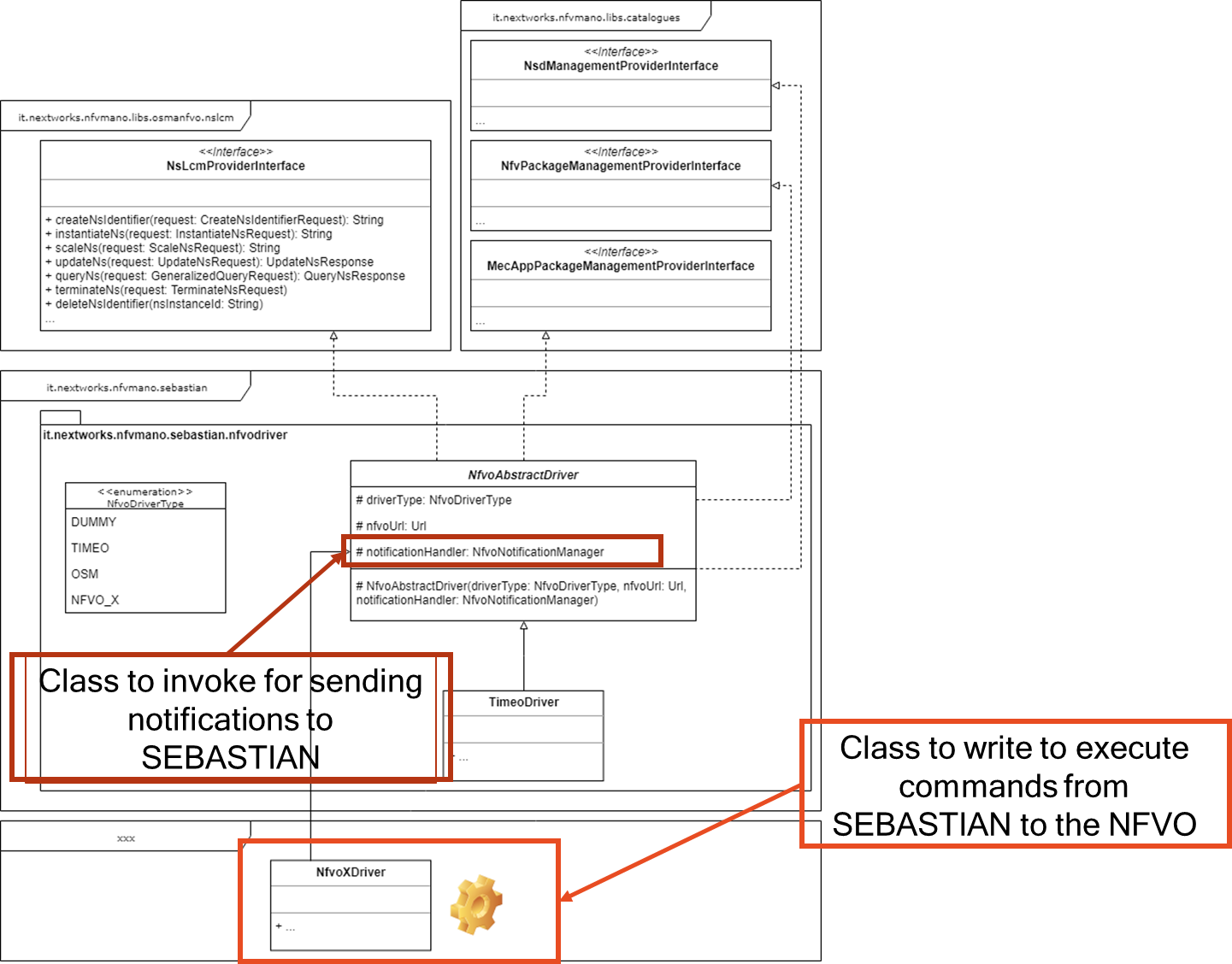


Figure 21: SEBASTIAN software architecture – NFVO drivers class diagram

# References

1. 5G-TRANSFORMER, D3.3, Final design and implementation report on the Vertical Slicer, May 2019.
2. 5G-TRANSFORMER, D3.4, Final design and implementation report on the Vertical Slicer (reference implementation), May 2019
3. 5G-TRANSFORMER, D1.1, Report on vertical requirements and use cases, 2017.
4. ETSI GS NFV-IFA-013, V3.1.1, Management and Orchestration; Os-Ma-Nfvo reference point - Interface and Information Model Specification, 2018.

1. <https://spring.io/> [↑](#footnote-ref-1)
2. Provider-managed services are entirely managed (e.g., scaled up or down, re-configured) by the 5G-TRANSFORMER Service Provider, while in case of tenant-managed services the vertical user can request a modification or a re-configuration of the service. [↑](#footnote-ref-2)
3. <https://spring.io/> [↑](#footnote-ref-3)
4. <https://maven.apache.org/> [↑](#footnote-ref-4)
5. <https://www.rabbitmq.com/> [↑](#footnote-ref-5)
6. <https://www.postgresql.org> [↑](#footnote-ref-6)