D2.1 Use Cases and Requirements

Project Acronym	SONATA
Project Title	Service Programing and Orchestration for Virtualized Software Networks
Project Number	671517 (co-funded by the European Commission through Horizon 2020)
Instrument	Collaborative Innovation Action
Start Date	01/07/2015
Duration	30 months
Thematic Priority	ICT-14-2014 Advanced 5G Network Infrastructure for the Future Internet

Deliverable	D2.1 Use Cases and Requirements
Workpackage	WP2 Use Cases definitions, Requirements, Architecture and Market watch
Due Date	M3
Submission Date	2015/10/27
Version	1.0
Status	Final
Editor	Shuaib Siddiqui/Jose Aznar (i2CAT)
Contributors	All partners
Reviewer(s)	Johannes Lessmann (NEC)/Panos Trakadas (Synelixis)

Keywords:

use cases, requirements, service platform, software development kit

Deliverable Type		
R	Document	**X**
DEM	Demonstrator, pilot, prototype	
DEC	Websites, patent filings, videos, etc.	
OTHER		
Dissemination Level		
PU	Public	**X**
CO	Confidential, only for members of the consortium (including the Commission Services)	

Disclaimer:

This document has been produced in the context of the SONATA Project. The research leading to these results has received funding from the European Community's 5G-PPP under grant agreement n° 671517.

Executive Summary:

SONATA aims to address the development and deployment technological challenges for complex user-facing applications and services perceived for 5G networks, by proposing a customised SDK developed to boost the efficiency of developers of network functions and composed services, and a novel service platform to manage service deployment and execution. This document describes the use cases envisioned for the SONATA framework and the requirements extracted from them. The use cases enable to highlight the full reach of SONATA framework on the ICT horizon from the perspective of 5G networks. The use cases facilitate in identifying the business, functional and non-functional requirements of the SONATA framework. The functional requirements are further classified in to Service Programming, SDK, Service Platform and Service Monitoring requirements in perspective of SONATA framework. The requirements specified provide an initial input for designing and developing the initial SONATA architecture. The document also discusses the state of the art service in terms of platforms designs and operations including service programming models, service programming tools, service platforms and service monitoring, while highlighting their limitations. It also provides an insight in to how SONATA aims to tackle these limitations. Finally, the document concludes with a set of recommendations for the SONATA architecture.

Contents

List of Figures

List of Tables

1 Introduction

SONATA is based on the rise of two profound and fundamental new technologies - cloud computing and software defined networking (SDN) which have been taken to the Telecommunications world with network functions virtualisation (NFV). These technologies provide mechanisms to automate resource intensive processes of network operators. However, on their own, they are not enough to drive transformation.

To use an analogy, the building blocks provided by Cloud, SDN and NFV are like a symphony orchestra which can play a great variety of music. However, the orchestra can only play if it has music and a conductor. SONATA is about providing the conductor for the orchestra, the SONATA Service Platform, and a toolkit, the SONATA Software Development Kit (SDK), to help composers with the composition of orchestral music.

The use cases and requirements set out in this deliverable aim to ensure that the SONATA Service Platform does achieve the level of automation necessary to realise both the cost saving benefits and the new service benefits achievable across the full range of processes. They also aim to test the efficacy of the SONATA SDK for simplifying the definition of new services.

1.1 Foundations in Virtualisation Technologies

While arising in apparently different industries, Cloud Computing and SDN share a vital common objective, the creation of virtual functionality. Cloud computing enables the creation of a virtual machine (VM) which appears to be a fully functional server, but decoupled from the actual real hardware server(s) on which it is hosted. Virtual machines can be created, modified, moved, and destroyed with great speed and flexibility and certainly a great deal more quickly and more cheaply than installing, modifying, moving, or destroying services and applications on real hardware servers. SDN similarly allows telecoms networks to create virtual networks, especially at the Ethernet layer, so that E-LAN, E-Tree, or E-Line services can be created, modified, moved, and destroyed with similar speed and flexibility and without making any modifications to network physical equipment. In the current world of cloud computing, SDN is used by cloud technology to create flexible and secure network interconnection between its virtual machines.

More recently, the NFV initiative seeks to combine these two technologies. NFV takes the cloud computing technology of virtual machines and implements some of the more complex networking functions as virtual machines running on commercial off-the-shelf (COTS) servers. Given that telecommunications networks are inherently distributed, the flexibility offered by virtualization is especially valuable. NFV makes it possible to deploy networking functions on general hardware at distributed sites. However, in contrast to legacy equipment, the exact functionality they perform as well as the exact capacity they require can be determined and changed independently to the deployment of the equipment which provides the processing, storage, or transport capacity.

1.2 Automation of Service Instantiation and Service Operation

In the early days of NFV, there was much discussion about the improvement in cost and utilisation of network hardware. However, nowadays most network operators would identify process automation

as the major commercial advantage of NFV.

When a new service is designed, service providers create a service template. Later, during the lifetime of this service, there are a number of processes service providers must undertake in order to successfully provide it:

- instantiation of a service instance using the service template
- modification of a service instance within constraints set by the service template during lifecycle
- monitoring of a service instance during lifecycle
- fault localization a service instance during lifecycle
- restoration/repair processes during lifecycle
- removal of a service instance

All of the above are open to automation, offering benefits to service providers and their customers: automated processes normally cost substantially less compared to operating the network manually. It is still the case that the great majority of operations involving deploying, moving, or repairing hardware must be manual processes. However, as virtualization removes the tie between services and hardware, service processes which were tied to hardware in the past can now be fully automated. The benefits go well beyond the cost reductions associated with process automation. Cost reduction only considers the business of doing the same things more cheaply. Automation also opens up the possibility for the deployment and adaptation of new services, some of which will only emerge over time and may be completely unexpected.

At least some new services are likely to emerge from the dramatically improved flexibility provided by process automation. Processes which used to take days, months, or even years (for example, where substantial roll out of network functionality to local exchanges or mobile base stations is needed), can now be potentially reduced to minutes with NFV and process automation. A whole variety of services which would never be possible because of roll-out time and cost are now viable.

1.3 Automation of Service Design

The automation of the service instantiation and operational processes described above requires a service template. This service template gives a detailed description of all the actions that the SONATA Service Platform must carry out in order to manage the life-cycle of service instances. In practice, these service templates (which include NFV network service descriptors (NSDs) and VNF descriptors (VNFDs)) are complex and need a considerable amount of manual programming in order to create them. They also tend to be inflexible and need manual coding changes even for relatively simple changes.

There are three basic elements to instantiation of a service which the template must describe.

1. Instantiation of the functional components of the service. For example, the instantiation of a network service requires the instantiation of the VNFs that compose it, or the instantiation of a VNF requires the instantiation of its VNF components (VNFCs).

2. Interconnection of the component functions. This is the way in which the VNFs are interconnected in a service, or the way the VNFCs are interconnected in case of a VNF.

3. Configuration of each component appropriate to the context of the service of which it is a part, i.e., the modification of the configuration of an existing functional component to offer service-specific functionality.

As is clear from the examples of the network service and the VNF, the details of each of these are different depending on layer of the entity being instantiated. The most significant layer boundary is between the NVF infrastructure and the VNFs it supports. The components of the VNF are VNFCs which map one-to-one to virtual machines (VMs) in the infrastructure. VMs require relatively few defining parameters and their instantiation is readily handled by cloud management systems such as OpenStack. Similarly at this layer, the interconnection of VMs is by virtual networks (VNs). VNs also have relatively few defining parameters and their instantiation is readily handled by an SDN controller such as OpenDaylight. The major complexity in the automation of service definitions is the third task - the configuration of the components in context. While full automation may not be easily achievable, the manual process can be made considerably simpler if a good development toolkit is available. This is the aim of the SONATA SDK.
One of the first steps toward realizing SONATA's vision for automation of service design, service instantiation and service operation is to establish use cases which coherently express the concerns related to the complete service lifecycle automation in context of 5G networks.

2 Definition and Analysis of Sonata Use Cases

2.1 Introduction to Use Cases

SONATA targets to enable flexible programmability of 5G networks by developing and bringing a service development kit (SDK), to boost the efficiency of developers of network functions, and an orchestrator as service platform to manage service execution in order to facilitate and speedup the complete lifecycle, from development to deployment, of network services. SONATA partners have identified 6 flagship use cases as main motivation and guideline for technology and innovations to be developed and they will serve as basis for the evaluation of project results in work package WP6. The use cases will facilitate in identifying the business, functional and non-functional requirements of the SONATA framework which will in turn be essential for designing the SONATA architecture. Furthermore, these use cases will enable to highlight the full reach of SONATA framework on the ICT horizon from the perspective of 5G networks. The conceived use cases are:

- ***Internet of Things (IoT)*** demonstrates SONATA's ability to monitor, classify, and optimize IoT network traffic as an enabler of Smart City ecosystem.
- ***Virtual CDN (vCDN)*** manifests SONATA's capabilities to enhance a virtual CDN service with elasticity and programmability.
- ***Guaranteed, resilient and secure service delivery in Industrial Networks (Ind Net)*** represents industrial network services with the scope of defining a programming model, the design of SDK and the design of the service execution environment.
- ***Virtual Evolved Packet Core (vEPC)*** exhibits and asses the SONATA system's competence to enhance a virtual EPC service in a LTE mobile network.
- ***Personal Security Service (PSA)*** targets SONATA's potential to offer on-demand network security services to enable secure network access and online presence.
- ***Separate Client and Hosting Service Providers (SCHProv)*** demonstrates SONATA's support for internetworking between a client service provider and a host service provider to accomplish an end-to-end service.

These use cases encompass a wide range of network services related to diverse ICT domains including Internet of Things (IoT), Content Distribution Networks (CDN), industrial networks, LTE mobile networks, cyber security and Infrastructure as a Service (IaaS). Moreover, they are also aligned with the ETSI NFV ISG use cases.

The rest of this section provides both a bird eye view of each use case while mapping them to SONATA architecture blocks, as well as explains their relationship to various stages of service lifecycle and workflows including the deployment topology. Furthermore, the benefits and impact of each use case from a user's perspective is also highlighted.

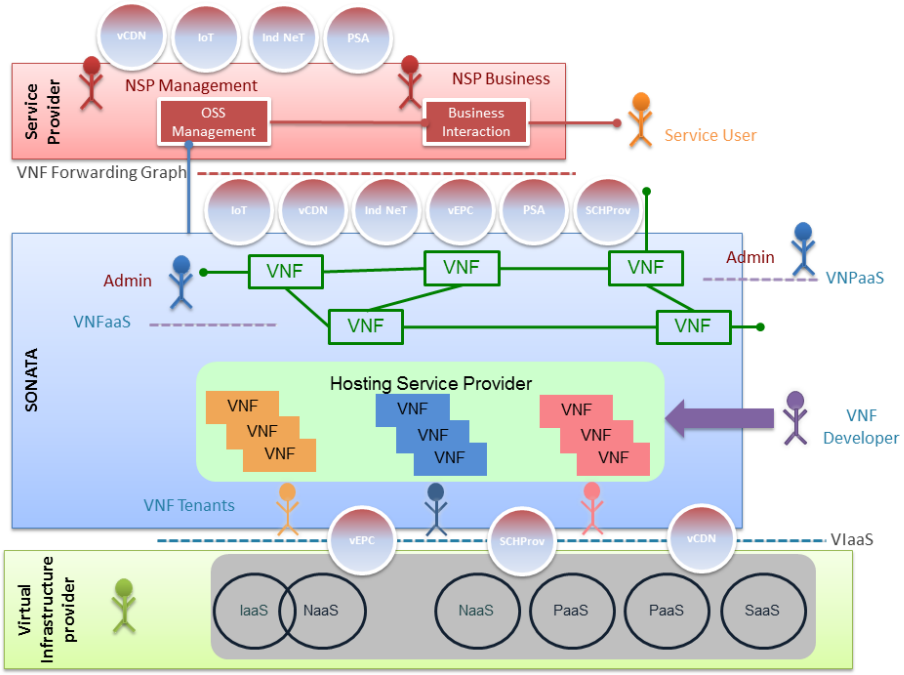

Figure 2.1: Use cases mapped on SONATA Environment

2.2 Use case: Internet of Things

2.2.1 Rationale

Smart city technology deployments will only be possible to exist if a high-functional and highly reliable network infrastructure co-exists with them. According to Garner, Smart homes and smart commercial buildings will represent 45 percent of total connected things in use in 2015, due to investment and service opportunity. Gartner also estimates that this will rise to 81 percent by 2020. By 2017, it is estimated that more than 2674 million devices to be deployed in smart cities. As expressed in 5G challenges, 5G is expected to provide the tools to deal with this increase of devices while also providing higher data rate and lower latency. To achieve these challenges while also reducing capital and operational costs, the network will have to be intelligent enough to optimise its resource usage accordingly with the demand. On a smart cities scenario, a set of sensors may send a lot of traffic to a single server. Further more, these 'data bursts' maybe concentrated in time, thus leaving idle permanently allocated resources, which implies sub-optimal resource usage. As each sensor is unaware of the network as a whole, and the network grows from the thousands to the million of sensors, new strategies to deal with the immense flood of requests will be needed. Present use case aims to directly manipulate IoT traffic allowing the network to take full of advantages of its resources without increasing maintenance costs.

2.2.2 Executive Summary

This use case aims to show SONATA's ability to monitor, classify, and optimize IoT network traffic. A set of classification VNFs will be deployed at the edge of the network to identify each type of IoT flow. Each different flow will be classified and redirected to an IoT gateway at a local datacenter, where an appropriate NFV will perform optimizations specific to the type of flow (like transceiving

media, or batching requests). Finally the gateway will pass the optimized traffic to the target end system, both reducing the network congestion as well as the load on the central components. The SONATA network will be able to deploy new classifiers where they are needed, scale, and appropriately locate each kind of IoT Gateway to ensure compliance with SLAs, as well as upgrade components with zero downtime to the service.

2.2.3 Detailed Description

2.2.3.1 Triggering Event, Preconditions, Assumptions

The ISP that provides the internet connection service to the IoT sensor networks will have to use a NFVI to provide its service as well as a NFV MANO to manage this virtual infrastructure. SONATA service platform is required to be deployed into this NFVI and to be able to interact with the ISP's NFVI. Furthermore, IoT generated traffic will have to be injected into this ISP's network. The triggering event associated with the present use case will be the detection of IoT traffic by the SONATA service platform.

2.2.3.2 Sequence of actions

In this use case the network operator defines the SLAs for each IoT traffic type. Then, using SONATA's SDK and available catalogues, VNFs and SDN are developed to monitor and compress/batch each traffic type. In order to control the previous developed VNFs, using SONATA SDK, an IoT VNF control center is developed, the network operator deploys this IoT VNF control centre and defines its SLAs (Control centre will be responsible to deploy/undeploy VNF resources in order to make an efficient management of its resources). At the edge of the network, IoT monitoring VNFs will detect IoT generated traffic, this traffic will be redirected into IoT control centre's IoT Gateways, that will use compressing/batching VNFs to optimise the received IoT traffic. Based on traffic's type SLA, traffic will be redirected into IoT data centers, this IoT infrastructure can be deployed either on SONATA NFVI or integrated with other infrastructures. Finally, IoT VNF control centre will constantly monitor its SDN performance (monitoring QoS VNFs will allow for a constant update on the SDN performance) and deploy/undeploy resources accordingly.

2.2.3.3 Evaluation

IoT use case will be evaluated taking into consideration the IoT identifying/compressing/batching traffic operations as well as the IoT SDN optimisation. For the IoT monitoring/compressing/batching operations, IoT emulated traffic (for different traffic types) as well as user generated traffic will be inserted into the network. The percentage amount of IoT detected traffic (by type) versus the IoT generated traffic will be analysed in order to check the system's ability to identify IoT generated traffic. The compression/batching ratio (by traffic type) will also be studied in order to calculate the IoT traffic optimisation boost provided by the solution. Regarding the IoT SDN optimisation features, implementation will be validated by loading into the systems different volume data rates at different times and validating if the system is able to rescale its resources while always maintaining configured SLAs. SONATA's system must provide a scaling operation that deployed resources allows meeting pre-configured SLAs in an efficient way at all instances.

2.2.3.4 Deployment topology

The Figure 2.2 shows how the components of the use case can be deployed, with a clear separation between data and control planes. The possibility of the integration of legacy services with the

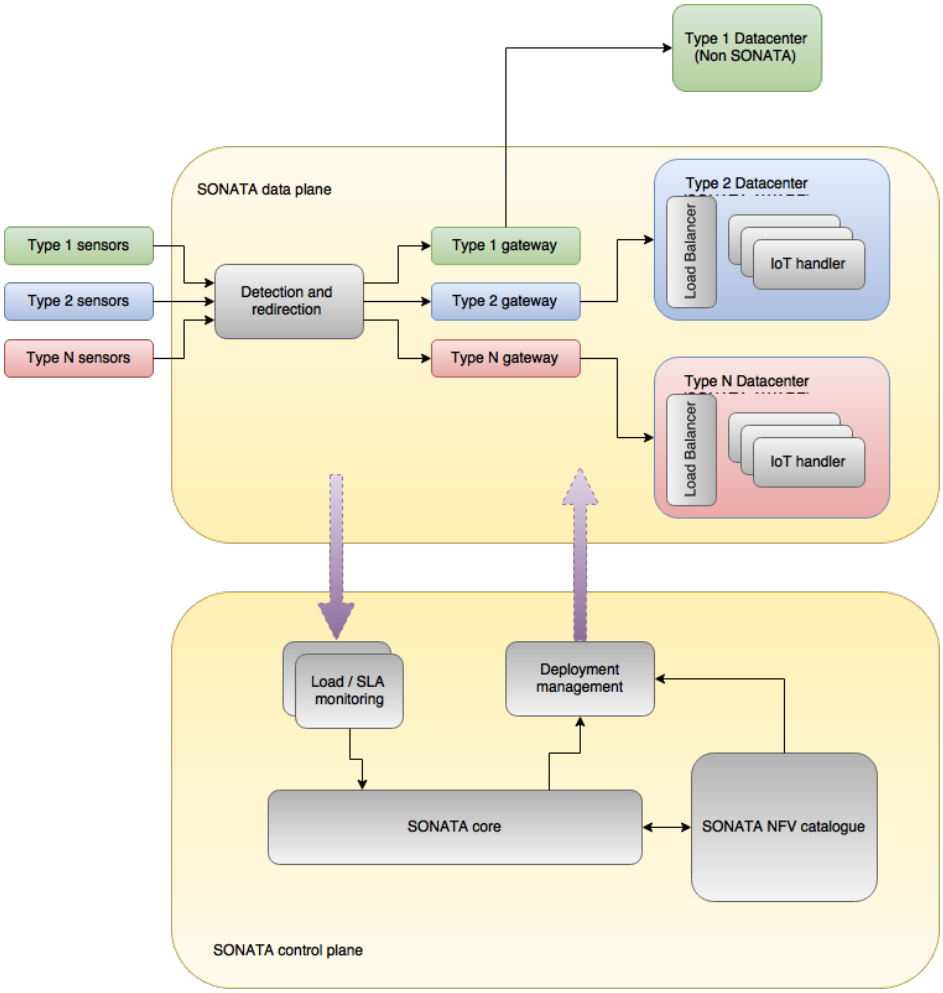

Figure 2.2: IoT deployment topology

SONATA core is taken into account as a way to ease adoption by new players. Not apparent on the image is how each monitoring VNF interacts with the data plane VNFs as the metrics to be gathered are not fully matured yet. The component deemed 'deployment management' is an amalgam of the part of the sonata MANO core that actually acts upon the deployments on the data plane, and it is kept vague on purpose until the core architecture reaches a mature level. The component deemed 'sonata core' includes all the data analysis engines as well as the decision part of the MANO engine.

2.2.3.5 Elements' scalability

The scalability design for each element on this use case becomes complicated for two reasons:

1. It is heavily dependant on the type of data produced by the sensor;
2. It may be somewhat tied to the sensor RAT (which we will try to ignore for this use case);

We choose to ignore case 2, as the scope of SONATA is the IP service network, not the radio technologies that may be used to actually communicate with the sensors as well as any gateway that may be used as a sensor<->ip transceiver.

So, two models are presented based on different assumptions:

2.2.3.5.1 Polling model - The gateway polls the sensor for measurements Each gateway can be in charge of polling a subset of the registered sensors. When the resources of a single gateway are exhausted, another gateway is deployed and takes charge of a number of sensors. How the split happens can be optimized by SONATA.

2.2.3.5.2 Event model - The sensor itself triggers the measurement In this case, a detection and redirection module (access point) must be deployed as close as possible to the sensors. This module has the sole responsibility of detecting the appropriate flows/packets, and directing them to the appropriate SDN in order to allow QoS management. In this case, the detection and redirection module can likely be one per physical access where sensors are expected, and can possibly be a shared feature where other traffic classification processes and redirections may occur.

In this case, if dealing with a less than 1ms response time network, the traffic is redirected as soon as possible to the respective datacenter.

If dealing with a high volume of less critical data, a set of gateways can be deployed and batch measurements before dispatching them to the main server. (Lets imagine a sensor that reads and sends the license plate of a car on a large crowded no stop toll - ignore the privacy issues), the gateway can batch/preprocess data from a set of sensors before sending it to the central server. In this case, the gateway can scale triggered by the load on the existing instances.

2.2.3.6 Detailed service (graph description)

On a smart cities scenario, a set of sensors may send a lot of traffic to a single server. Further more, these 'data bursts' maybe concentrated in time, thus leaving permanently allocated resources idle, which implies under optimal resource usage. As each sensor is unaware of the network as a whole, as the network grows from the thousands to the million of sensors, new strategies to deal with the immense flood of requests will be needed.

Each sensor reading (triggered or periodic) contains the following (using a generic sensor as an example)

1. Sensor identification;
2. Variable being reported (for multi variable sensors);
3. Actual reading value;

All of the remaining data will usually be overhead. As sensor data does not usually have real time constraints (stuff like parking spot sensors, or something similar, not something critical like inter vehicle communication), the measurements can be delayed for a second or two. This means the smart city operator may choose to deploy a VNF as near as possible to the sensor grid. That VNF can then batch (and maybe compact) incoming requests to the main server that usually is placed somewhere outside the operator infrastructure. This VNF can be automatically scaled according to the amount of traffic coming from the sensors in order to keep the outgoing traffic to a minimum. The scaling can be triggered by another VNF, that monitors the network for sensor traffic and chooses where and when to deploy or undeploy the batch generator VNF. When dealing with a large number of sensors, reducing the number of requests to a server is very important as the performance limit on the server is more often than not tied to the number of requests it can answer per second instead of the amount of data that it can process. This is an improvement over the existing sensor networks, as batching the requests to the server will allow a better level of service from the same amount of resources. The main clients for this feature would be cities that are implementing new IoT technologies as well as cities that are reaching the performance limits of their existing sensor networks. The VNF implementers will be the smart cities solution provider.

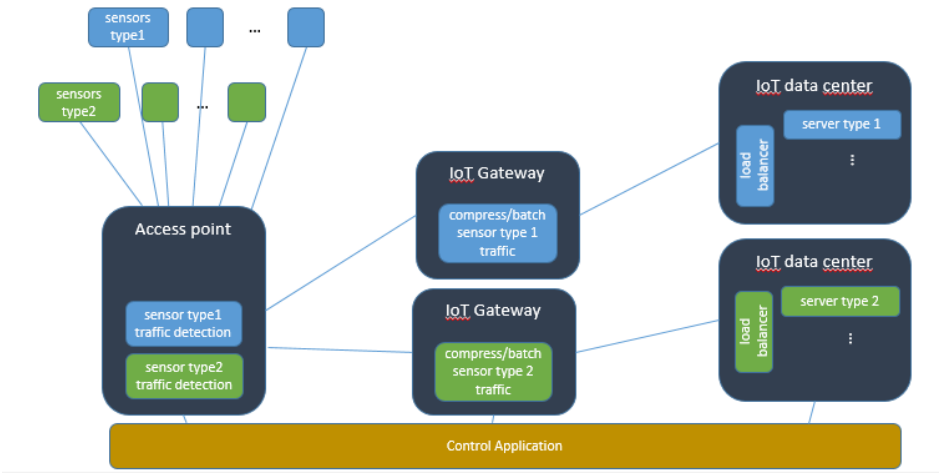

Figure 2.3: Service Graph representation

This figure shows the different area's where the SONATA framework can demonstrate its added value:

- Access Points: at edge of network: deploy traffic detection NF's where needed:
  - Traffic detection in function of sensor type;
  - Forwarding to correct IoT gateway;
- IoT gateway:
  - Placement of IoT gateways can be optimized by dedicated orchestration algorithm;

  - Traffic compression NF needs to be deployed on this node;
  - Forwarding to correct IoT data center;

- IoT data center:
  - Load balancing to different servers can illustrate elastic resource provisioning in function of the request load;

The dynamics of this service imply some sort of Control Function that is able to trigger the elastic deployment of additional NF's ,like:

- At Access Point: detect if new type of sensor is connected;
- At IoT Gateway: detect if gateway placement is still meeting specifications regarding eg. delay;
- At IoT data center: trigger load balancing of more/less server instances in function of the request load.

2.2.4 Requirements

In order to implement this use case some the following requirements must be fulfilled.

2.2.4.1 General Requirements

Requirement Name	VNF Catalogue
Description	The SDK must provide a location to store the different IoT related VNFs. It shoud be possible for the VNF developer to add update or delete VNFs from that location. Using this list the SONATA service developer can compose a complex service.
KPIs	Number of VNFs available in the catalogue
Category	Mandatory
Involved Use Case	UC IoT

Requirement Name	VNF Scaling metadata
Description	The SDK must allow definition of SLA levels for selected VNFs, other metadata should be possible to specify as well such as when and how the SONATA operator should scale the VNF, as well as the scaling strategy (up/down, in/out). This information can be used by the SONATA platform to automatically scale the IoT gateways using appropriate methodologies.
KPIs	Downtime while processing the scaling
Category	Mandatory
Involved Use Case	UC IoT

Requirement Name	VNF SLA Monitor
Description	SONATA must provide an interface to monitor VNFs SLAs and resource usage. It must highlight VNFs with high and low usage, that may need scaling or other kind of manual intervention.
KPIs	Provided metrics and data visualization
Category	Mandatory
Involved Use Case	UC IoT

Requirement Name	VNF Resource Report
Description	SONATA must provide an interface to list all resources allocated to a specific service, e.g., each IoT operator. This service allows the developer or administrator to get an overview of how the service is evolving and what datacenter resources are committed to each service.
KPIs	There is a way to list what resources are allocated to a service in runtime
Category	Mandatory
Involved Use Case	UC IoT

Requirement Name	Authorization
Description	SONATA service must limit operations based on access levels and provide means to create and manage access profiles. This will be used to define the different access levels each user will have to the system, as an example, a VNF developer should be able to deploy a VNF on the catalogue, but should not have the permission to trigger its deployment to a production network.
KPIs	Different users have a different set of permissions according to their role on the platform.
Category	Mandatory
Involved Use Case	UC IoT

Requirement Name	VNF Deployment
Description	SONATA must support placement instructions that express proximity to other entities, e.g, (i) set where the service gateways will be placed on the operator network, (ii) deploy a VNF as near as possible to a specific location, (iii) select were the VNF will be deployed.
KPIs	The same VNF can be deployed on different locations of the network in accordance with what is specified on the SONATA service description
Category	Mandatory

Requirement Name	VNF Deployment
Involved Use Case	UC IoT

Requirement Name	VNF Status Monitor
Description	SONATA should provide a high level state for each VNF, e.g., (i) deployment, (ii) operating, (iii) error.
KPIs	It is possible to get a quick overview of errors with a quick glance of a GUI displaying status data
Category	Mandatory
Involved Use Case	UC IoT

Requirement Name	IoT traffic simulator
Description	Given that there is not yet the amount of IoT traffic this use case is designed to address, there must be a way to simulate IoT sensor traffic with functions like increase or decrease traffic levels per sensor and number of sensors in order to simulate a real IoT sensor environment.
KPIs	Number of IoT traffic sensors detected by the gateway and the volume of traffic generated by each sensor Generated traffic
Category	Mandatory
Involved Use Case	UC IoT

Requirement Name	VNF integration with service
Description	Sonata must allow new VNFs to be integrated in existing services. It must allow network flow reconfiguration in order to integrate a newly deployed VNF in an existing service graph with minimum or no downtime at all.
KPIs	The amount of downtime required to reconfigure a running service graph
Category	Mandatory
Involved Use Case	UC IoT

Requirement Name	SDK VNF customization
Description	The SDK must allow the development of custom VNFs with specific algorithms to manipulate IoT traffic, like processing and batching.
KPIs	The IoT traffic generated by the sensors is correctly manipulated on the gateways
Category	Mandatory

Requirement Name	SDK VNF customization
Involved Use Case	UC IoT

2.2.4.2 Requirements Flow

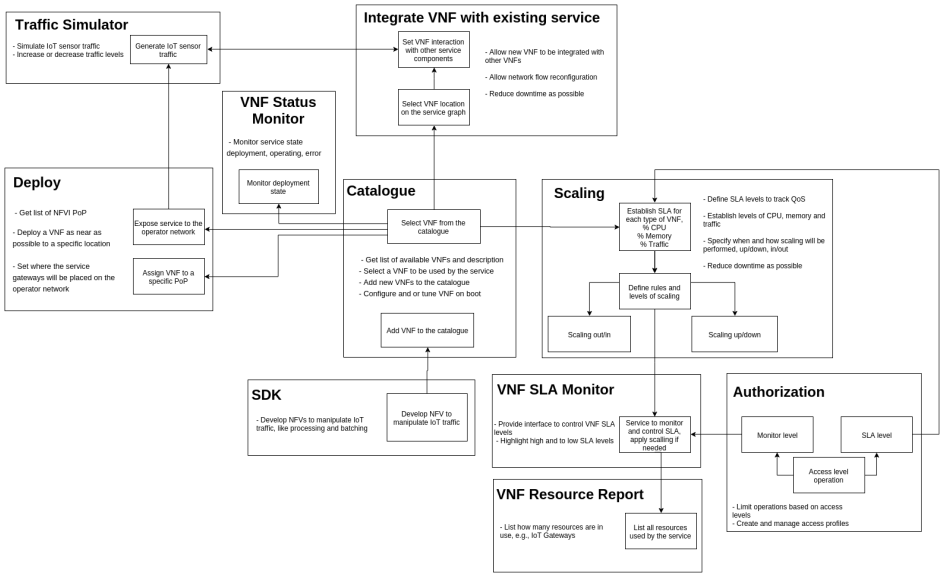

Figure 2.4: IoT use case requirements flow

2.2.4.3 Business Requirements

Requirement Name	Multiple IoT sensor vendors
Description	Framework must support traffic from different IoT sensor vendors. Traffic from each sensor should be routed through the appropriate gateway.
KPIs	Receive traffic from at least two different sensor vendors and treat it in a uniform way.
Category	Necessary
Involved Use Case	UC IoT

Requirement Name	Multiple IoT tenants
Description	Framework must support multi tenancy, i.e., The infrastructure must support multiple IoT services operating in parallel without any data meant to one operator being routed to another operator's service.
KPIs	At least two IoT services can be deployed without any conflict.
Category	Mandatory
Involved Use Case	UC IoT

2.3 Use Case: Virtual CDN

2.3.1 Rationale

Content distribution, especially video traffic is expected to be the dominant contributor to the mobile data traffic demand , therefore CDNs are being more and more present in everyday life communications, anywhere, any time and in end-user multidevice environments. Especially relevant scenarios for content delivery are massive events such as international music festivals or big sport events, such as football World Cup or the Olympic Games, in which high volume of traffic is needed maintaining an acceptable QoE for the end user.

Furthermore, CDN providers seek to exploit virtualization capabilities and leverage NFV platforms to deploy caches as VNFs (vCaches) instead of hardware appliances. The deployment of the content delivery chain based on virtual rather than physical appliances, not only achieves shifting CAPEX to OPEX, but also allows the dynamic configuration and easy upgrade of the virtualized appliances. For this purpose, vCDN has been identified by ETSI as one of the most promising use cases for NFV (UC #8). However, performance predictability, flexibility and resource efficiency are recognized by the ETSI document as main challenges for the vCDN use case. For this purpose, vCDN operators further seek mechanisms to optimize the configuration of their vCDN service. According to user traffic, operators seek to optimise vCache placement, resources and configuration (e.g. caching strategies, protocols etc.) in order to maximize end user QoE while avoiding resource overprovisioning. Optimally, this process should take place dynamically on-the-fly, using custom strategies and algorithms defined by the vCDN operator. Complementing virtual slices and NFV network services with programmability capabilities, Infrastructure providers become more competitive. In this context, SONATA aims to advance beyond static CDN virtualization, by adding dynamicity and (re)programmability.

2.3.2 Executive Summary

This use case focuses on showcasing and assessing the SONATA system capabilities in order to enhance a virtual CDN service (vCDN) with elasticity and programmability. Customers wishing to play the role of the vCDN operator use the system to dimension the service with regard to (IT and network) resources use and deploy vCaches as VNFs. Using the feedback from the service metrics, the vCDN service can be automatically re-configured according to policies set by the customer (vCDN operator). The Figure 2.5 below illustrates an overview of the vCDN Use Case.

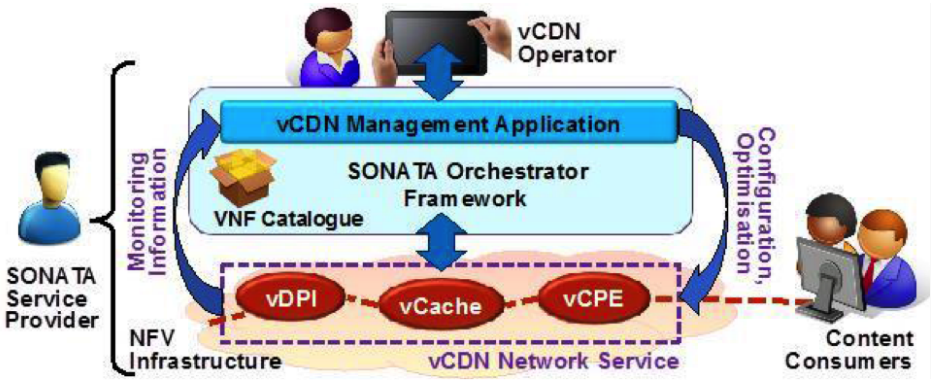

Figure 2.5: Overview of the vCDN use case

With regard to the origin of the content, this UC can be split down into two different scenarios.

The usage environment and pattern is different, yet the technological enablers and the service provisioning logic (as well as the use case sequence) are almost identical.

- Scenario 1: Distribution of highly popular content This is the "classical" vCDN scenario, where the content originates from a content provider, distributed across the vCaches and eventually delivered to a huge number of subscribers.
- Scenario 2: Edge caching and adaptation of user-generated content This scenario intends to cope with the increased load associated with user-generated content (photos, audio and video) in e.g. a flash crowd event (for example hundreds of subscribers sharing high-definition feeds from a live show). In this scenario, multiple edge VNFs are deployed in proximity to the flash crowd event which not only locally cache the user-generated content in order to facilitate its distribution, but also adapt it on the fly, i.e. reduce its resolution and/or quality (downscaling) in order to relieve the backhaul network.

2.3.3 Detailed Description

2.3.3.1 Triggering Event, Preconditions, Assumptions

A media content provider wishes to exploit the SONATA system in order to establish a vCDN virtual infrastructure to either optimize the delivery of the media content to subscribers (Scenario 1) or optimize the aggregation of user generated content (Scenario 2). A significant target audience (end users) is within network coverage and can be served by the access infrastructure of the NFVI provider.

2.3.3.2 Sequence of actions

Mapping with SONATA stakeholders (in the DoW):

1. the end users of a service (e.g., a private person) = content consumer
2. the developer of a software artefact (e.g., an NFV or a composition of NFVs into a service)= content provider = vCDN operator
3. the service platform operator (SONATA service provider), who runs a platform that manages the execution of services, and
4. the actual infrastructure operator, who often, but not necessarily, will be the same as the service platform operator

The sequence of steps below reflects the main phases of the lifecycle of the vCDN service as enabled by the SONATA system:

1. The vCDN operator enters the SONATA system and requests the vCDN service, after browsing the service catalogue
2. The vCDN operator uses the SDK and editor, to develop a vCDN Entity Executed via an Executive (EEE) (Control Application) that dynamically manages and optimizes the vCDN service. The vCDN EEE is a SONATA architecture specific term that depicts an entity provided by the service developer together with the vCDN service.

3. The EEE configures the connectivity service and reuses vCache (virtual content cache) and DPI (Deep Packet Inspection) VNFs provided by the SONATA VNF catalogue. It implements, for example, custom tailored scaling and placement logics for a particular service.

4. The vCDN operator packages the service and deploys it via the management portal; the vCDN service is deployed

5. At runtime, the vCDN EEE feedback from deployed DPI VNFs to gain awareness about the volume and type of traffic (user applications) traversing the network and sensing the content demand. This procedure also implies detecting flash crowd events, i.e. a sudden increase in traffic load in a geographical location.

6. The vCDN EEE employs optimization algorithms in order to decide on vCache VNF deployment, to re-allocate network and vCache resources and to adjust their configuration according to user traffic characteristics.

7. The vCDN operator monitors the overall performance of the service and, if needed, adjusts the policies/algorithms of the control application accordingly. In the case of flash crowd events, edge VNFs are torn down when the event is finished and the network load is resumed to normal levels.

2.3.3.3 Evaluation

For the sake of evaluation, the programmable vCDN use case can be split into two phases: i) service deployment (steps 1-4 above) and dynamic service optimization (steps 5-7 above).

For the service deployment phase, functional validation tests include the verification of the proper deployment of the service, as it has been instructed/requested by the customer, i.e. fully automated instantiation of VNFs in an optimal manner and service chaining. Metrics to be observed include service setup time as well as efficiency of the service mapping procedure. For the deployed vCDN service itself, tests include the verification of its proper operation and the acceleration of the content delivery due to caching. The assessment of the caching mechanism and strategies are out of scope since they actually depend on the cache implementation.

For the service optimization phase, tests will be performed on a service already running. One or more vCaches will be artificially loaded with high volume of requests. The control application should respond by re-scaling the vCache instance(s) and/or reconfiguring the entire vCDN service, also modifying the service function chain, if necessary. Metrics to be observed include service reconfiguration time, service downtime due to reconfiguration, as well as the gain in resource efficiency due to the dynamic adaptation. Both phases (deployment and optimization) will be evaluated in a lab-scale operational environment, in order to prove that the mechanisms supporting the SONATA service lifecycle (including the plugins developed within the Orchestration Framework) are in place and properly operating. However, since the topology and complexity of the lab testbed will be restricted, it is considered appropriate to run tests over a simulated large-scale infrastructure, in order to verify the effectiveness and efficiency of the vCDN EEE when managing tens or hundreds of VNF instances over a complex topology.

2.3.3.4 Deployment Topology

As the above Figure 2.6 illustrates, it is considered that the vCDN NS would interconnect various edge locations (NFVI PoPs) within the reach of the Telco Operator,constituting a virtual network slice embracing End Users and Media Content Providers. The vCDN EEE will be in charge

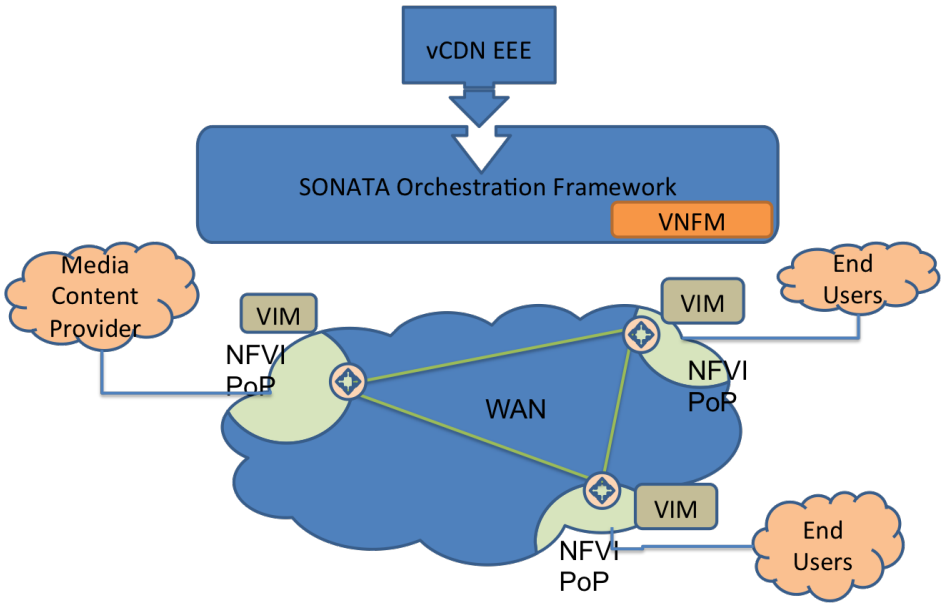

Figure 2.6: vCDN deployment topology

of the placement and operation of the VNFs composing the vCDN NS. Each NFVI-PoP (edge location where the NFV instantiation is possible) would be managed by a local instance of Virtual Infrastructure Manager. Depending on the content delivery scenario, multiple locations need to be coordinated via the vCDN EEE. At the same time the lifecycle of the VNFs will be managed by VNFM, however the VNF lifecycle could be coordinated through VNFM-like plugin (or EEE) or as part of the vCDN EEE. In this respect the SDK may provide the capability to define specific actions in the lifecycle of the VNF (eg. scaling in/out).

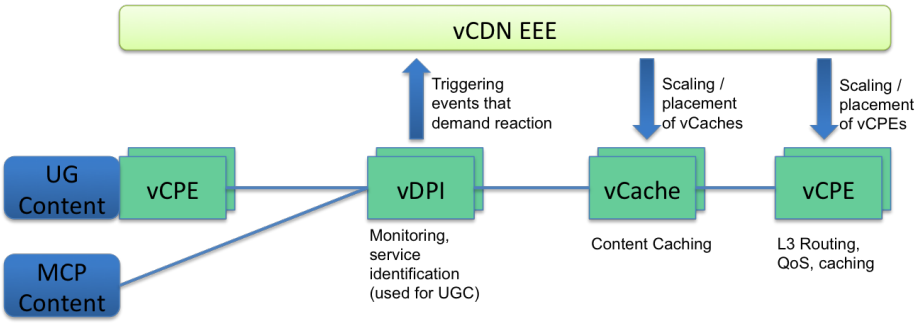

Figure 2.7: vCDN service graph

2.3.3.4.1 Service Graph The Figure 2.7 presents the Service Graph that is anticipated in order to deploy the vCDN Service through SONATA system. PNFs have been intentionally omitted in order to focus on the VNFs that are composing the e2e NS. As discussed, there exist two cases depending on who provides the content within the vCDN. The first option would be that the content is provided by the MCP who in turn is connected statically to the network through L2/L3 technologies. The traffic seeming from MCP premises is intercepted by the vDPI (which in this case could for simplicity be a shallow packet inspector). The vDPI is the VNF that will provide awareness to the vCDN EEE in order to audit the service and conduct the appropriate actions in

order to place or remove vCaches as required in order to preserve the same QoE for the consumers of the service. In the case the content is generated by the End Users acting as prosumers, the traffic will be chained through the vCPE and the vDPI. Partial caching capability for the vCPE would enable a more granular and efficient distribution of the User Generated Content (UGC) across the provisioned network.

2.3.3.4.2 Components

- vTC: a VNF capable of application classification via deep packet analysis of incoming traffic, monitoring and traffic prioritization according to user set policies.
  - SR-IOV for efficient networking
  - Storage: depends on the amount of monitoring data stored. In case none then 5GB are sufficient
  - Memory scaled according to the maximum network throughput required and to type of traffic forwarded
  - CPU: number of network ports + 1
- vCPE: VNF used for virtualisation of End User CPE
  - CPU: 1 core (no transcoding) (small number of media flows forwarded / more than 2 cores in case transcoding is employed
  - Storage: Caching of content at the vCPE requires storage provision e.g. more than 20GB
  - Memory scaled according to the maximum network throughput required and to type of traffic forwarded and the functionalities supported
- vCache: Used for content caching at the edges or in the core of the WAN network (whether this is required)
  - Increased storage requirements as well as optimised storage for achieving high IOPS.
  - Increased memory needs
  - CPU: 2 cores
- vCDN Control: implemented as a Control Plane VNF interacting with the SONATA framework. Control the placement and the scaling of the vCDN NS.
  - CPU / Memory requirements: medium
- At each PoP:
  - Network Node (Openstack) / Virtual Switch
  - Gateway / router interconnecting PoP with the WAN network.
  - VIM (comprises SDN Controller / Cloud Controler)

2.3.3.4.3 vCDN KPIs

Performance The key aspect for the vCDN would be *speed*. This translates to how fast the vCDN (most important component are the vCaches) will be delivering the files to your end users. The KPIs to look at are: latency (in ms) and throughput (in kbit/s). For each KPI the min, max, median and average should be monitored for a long-enough period and preferably during normal and/or peak traffic hours. Validation of the above requires a performance monitoring in place during this test phase so you have real data about the vCDN performance

Availability Availability is a critical factor for the vCDNs. If the content cannot be delivered reliably and quickly enough, then the user experience will be poor.

Functionality/features Another KPI of the vCDN are the functionalities that are possible for the User of the vCDN (e.g. cached content management) as well as the inherent functionalities of the vCDN (e.g. file compression).

2.3.4 Requirements

Requirement Name	VNF Catalogue
Description	The system shall offer a VNF catalogue from which the customer can select the desired VNFs
KPIs	Number of VNFs available in the catalogue
Category	Mandatory
Involved Use Case	

Requirement Name	VNF Placement
Description	The programmability framework shall allow the customer to deploy VNFs at arbitrary points into the network
KPIs	VNF deployment time
Category	Mandatory
Involved Use Case	

Requirement Name	SFC
Description	Service chaining - the programmability framework shall allow the customer to interconnect VNFs in an arbitrary graph
KPIs	Service mapping complexity and time
Category	Mandatory
Involved Use Case	

Requirement Name	VNF specific monitoring
Description	The system shall expose service and VNF metrics to the network application
KPIs	Availability of an API for VNFs capturing such metrics
Category	Mandatory
Involved Use Case	

Requirement Name	Scaling
Description	The developer should describe in the VNF Descriptor recipes for scaling his/her VNF
KPIs	Availability in the VNFD fields to define scale policies. Support the interruption of VNF operations.
Category	Mandatory
Involved Use Case	

Requirement Name	Multi NFVI orchestration
Description	SONATA Orchestrator should be able to orchestrate multiple VNF execution environments (NFVI-PoPs) located in arbitrary places in the operator network topology. The NFVI-PoPs are considered to be controlled and managed by VIMs
KPIs	Number and size of the NFVI-PoPs
Category	Mandatory
Involved Use Case	

2.4 Use case: Guaranteed, resilient and secure service delivery in Industrial Networks

2.4.1 Rationale

Industrial networks have requirements that differ somewhat from traditional carrier requirements. They are characterized predominantly by machine-to-machine rather than human-to-human or human-to-machine communications. Therefore, notions that are typical for carrier networks such as “best-effort” or “Quality of Experience” don’t make as much sense or have to be at least redefined.

Machine-to-machine communications often have very strict requirements in terms of performance. For example, a certain amount of capacity has to be guaranteed even in the face of varying background traffic. Or the end-to-end delay has to be kept below a certain fixed bound. Also, more mission-critical applications have high requirements on reliability of the network connection and packet delivery. There are also requirements on the availability of VNFs, it has to be easy to do remote VNF software upgrades, there is a high emphasis on cyber-security, strict access control for and isolation of multiple tenants of the network, and means for network security and access auditability. On the other hand, industrial networks tend to be small, predictable and more static

than carrier services. Therefore, dynamic service orchestration at runtime will be less of an issue in this context.

In summary, this use case was proposed to represent industrial network services, i.e. to complement the carrier-related use cases. While it is beyond the scope of Sonata to do research on SDN and NFV support for the above mentioned challenges, Sonata should consider industrial services in the definition of the programming model, the design of the SDK, and the design of the service execution environment. Eventually, it must be possible to model, deploy and run an industrial service (e.g. a SCADA monitoring service or an upgrade function for machine control software) entirely with the models, tools and methods which Sonata provides. This would make service development simpler, less error-prone and service execution more secure-by-design and auditable.

2.4.2 Executive Summary

While the whole spectrum of industrial networks should be supported as much as possible, for the sake of specificity, this use case elaborates on a wind park network. This scenario is explicitly taken from the 5GPPP VirtuWind project [1], with which the Sonata project officially collaborates.

A wind park operator uses Sonata to define a service that collects measurement data of the wind turbines in a local wind park. The data is forwarded to a central server via the local communications network. At the server the data is processed in real-time and commands are sent back to the wind turbines adapting certain turbine control parameters. The entire round trip time needs to be kept below a fixed delay threshold and resilience mechanisms must be in place to ensure message delivery with very high probability. An additional tenant service is defined over the local wind park communications network. For the sake of software maintenance and upgrade, it provides access to a select set of control modules in selected wind turbines to an external company. The turbine control service must not be impacted negatively by the additional service. At any point of time, the wind park operator wants to be able to see what has been accessed when by which tenant, i.e. have a complete communications trail about all tenants and applications in the wind park network.

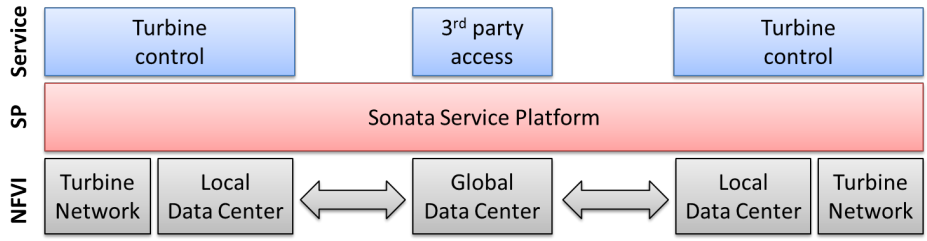

Figure 2.8: Two sample services on top of Sonata's Service Platform (SP) in a wind park network

2.4.3 Detailed Description

2.4.3.1 Triggering Event, Preconditions, Assumptions

The wind park operator has set up a communications infrastructure between a set of wind turbines. The infrastructure consists of SDN switches (for cost reasons, some switches might be pure software switches, others might be hybrid or bare metal switches) and some lightweight compute nodes in the turbines. The infrastructure is connected to a local data center which hosts the Sonata service platform. Before the use case is executed, no actual software or forwarding state is assumed to be installed in the wind park communications infrastructure.

2.4.3.2 Sequence of actions

2.4.3.2.1 Actors and their roles / involved system components The main actors involved in the use case are:

- 3rd party turbine SW providers
- wind park operator
- potentially network provider (for now, we assume self-owned network infrastructure)

The **3rd party** provider supplies software modules (and currently also hardware) for controlling the operations of the wind turbine. In the future, turbine control is expected to be executed by software-only modules that run on general-purpose hardware.

The **wind park operator** is the company that runs the wind turbines, the communication network in between (even though that may be partially leased) and the control centers with SCADA and other server software.

The **network provider** is an optional entity in this use case. In real wind park deployments, (parts of) the communication network to the wind turbines may be leased from such a provider. However, initially, we assume a local communications network which is completely self-owned by the wind park operator, as leased scenarios complicate the mechanics of management and signaling without much additional value for the requirements and architecture discussion.

The major system components that play a role in the use case are:

- SCADA servers
- AAA server
- turbine control SW (sensors, actuators)
- network management system (SDN controller)
- analytics functions

A **SCADA server** constitutes the management system for energy companies. It is comparable to a Network Management System of telecommunication carriers. The **AAA server** is responsible for user authentication and lookup of access rights. This is particularly important in the case of 3rd party access to the wind park network, e.g. for software maintenance as indicated in this use case. For managing the actual communications network, there is a **network management system** - in the context of Sonata (and VirtuWind, for that matter) this will be an SDN controller with management apps on top of it. Finally, there will be **analytics functions** which are used by the wind park operator to obtain statistics, advanced data aggregation and insights out of the 1000s of sensors attached to individual turbines.

2.4.3.2.2 Actions of actors and information flow between actors and components This use case contains two distinct services which are described below separately.

1. First Service: 3rd party access
   a) A 3rd party (i.e. the turbine control SW provider) notifies the windpark operator that a SW maintenance is due.

b) The windpark operator uses the Sonata SDK to define a simple tenant access service which consists of an access gateway and a networking connection to the involved control modules / VNFs

c) In addition to the generic tenant access service, the windpark operator specifies all tenant-specific configuration data (e.g. what exact VNFs the tenant may access, the time window for network access, networking requirements, etc.)

d) The windpark operator notifies the 3rd party (i.e. the tenant) that the tenant access service has been set up and how to connect to it.

e) The tenant uses the established tenant-specific service to access the involved VNFs to perform SW maintenance. Potentially, the access gateway VNF uses the Sonata VNFM for lifecycle management of the involved VNFs, e.g. to shut down a running VNF or install a new one with modified software.

f) Once the maintenance work has been finished or the access timing window has expired, the service is torn down.

g) For further SW maintenance purposes, the windpark operator stores the service specification plus potential configuration data in its service catalogue so that it can easily be retrieved and set up again in the future without renewed specification/configuration work.

1. Second Service: real-time turbine control service

   a) The windpark operator uses the Sonata SDK to program a control app which performs real-time control of wind turbines.

   b) Based on the control app, the operator defines a service that aggregates data from multiple sensors in a wind turbine and forwards it to the control app within a certain delay bound. The control app determines appropriate outputs and generates commands to the turbine control SW which are again forwarded in real-time to the actuators.

2.4.3.3 Evaluation

In order to satisfy this use case, Sonata's SDK must support the various annotations/primitives that have been outlined in this description. Since realistic deployment topologies are rather small, the mentioned services can be tested in an emulated (or small real) SDN-based test network. One would need a couple of sensors (or comparable synthetic traffic generators) and actuators (or synthetic traffic sinks) and a small IoT gateway VNF that could - in the easiest case - simply put all data together without further processing and forward them via the SDN network to another VNF which could be as simple as an echo function. The system has to be capable of enforcing traffic isolation, performance guarantees (incl. networking and compute performance) and VNF placement according to the indicated constraints. In fact, in the most simple case, since the services are rather static, placement could even be done manually by the windpark operator. For the first service, a simple access gateway is needed and interworking with the VNFM would need to be shown (the interworking with the AAA server is rather trivial and could be omitted). The SDK has to allow specifying basic admission control rules, i.e. which VNFs/ports/interfaces a certain tenant can access.

2.4.3.4 Deployment Topology / Assumptions

In terms of actual deployments, typical wind park networks, as depicted in Figure 2.9, would contain some tens of wind turbines, which are connected in a local wind park network via a simple topology

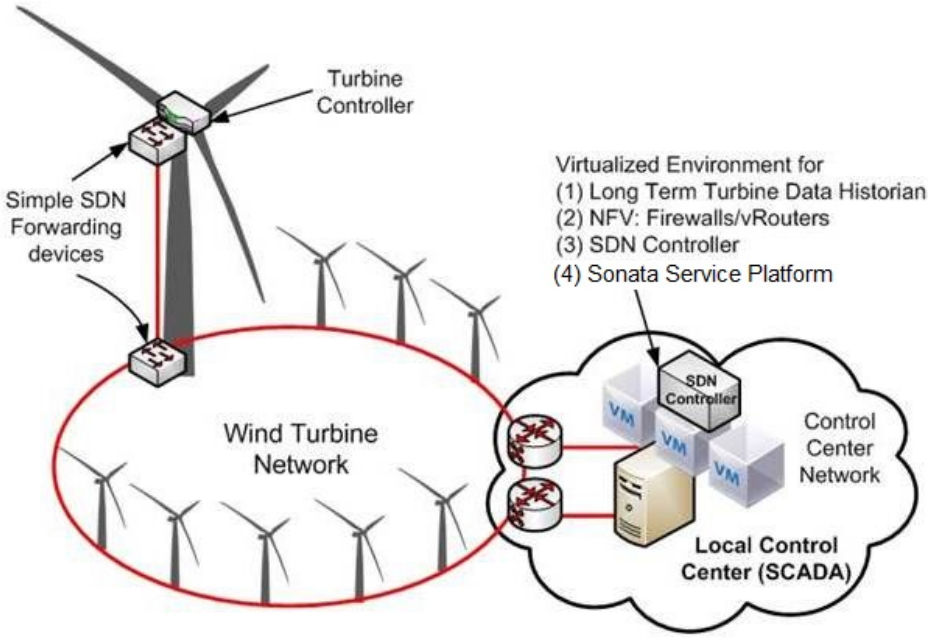

Figure 2.9: Example of a wind park network

like a ring. The local network will generally be fiber-based, if possible, which might be self-owned, but could also be (partially) leased by an external provider. In some cases, particularly in the case of offshore wind parks, connectivity will be implemented via radio links such as microwave. Today, a turbine contains so-called turbine control modules, usually located at the top of the turbine, and also some processing power, usually installed at the bottom of the turbine, to do local data aggregation or statistics collection. The future vision (as e.g. followed in 5GPPP VirtuWind) is to centralize most of the functionality of turbines and simply have plain SDN-enabled forwarding devices plus virtualized compute nodes in the turbine. Turbines contain many sensors, a single turbine can contain up to 3000 sensors, so some pre-processing of all the sensor data is done locally by up to 15 different applications. Monitoring and control traffic is sent to a small local management data center, which in turn is connected via a multi-domain (Internet) network to the global control/data center which is potentially far away, even in a different country. The network must support multiple tenants and applications which have to be strictly performance- and security-isolated. Besides various applications from the windpark operator itself, 3rd party access is also needed from time to time, e.g. for SW maintenance of control modules. Hence, there is real-time control traffic, isochronous measurement traffic, video and audio surveillance feed, sporadic file uploads, and other best-effort traffic in the network.

2.4.3.5 Service Graph / Description

2.4.3.5.1 First Service: Remote access from a 3rd party The first service considered by this use case and depicted in Figure 2.10 addresses remote access for a 3rd party SW provider (i.e. typically for his laptop or VPN client) to the wind park network. This service is provided via a dedicated access gateway that performs access control, encryption, VNFM interfacing. Access

control is needed to ensure that the 3rd party can access nothing but the module (e.g. IP address, port, Virtual Machine) that it needs to manage. Potentially, the access gateway VNF accesses the Sonata VNFM for lifecycle management of the involved VNFs, e.g. to shut down a running VNF or install a new one with modified software. Although not explicitly captured in the graph, tenant isolation is of utmost importance in this context. Red boxes and links are to indicate existing entities in the wind park network or links to such existing entities.

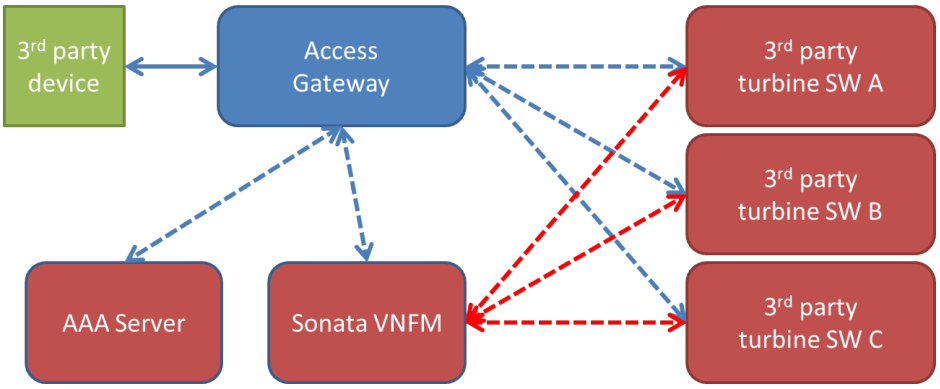

Figure 2.10: The 3rd party access service

2.4.3.5.2 Second Service: Reliable end-to-end delivery The second service in this use case is supposed to require strict end-to-end delay bounds and reliable delivery. In reality, this could be

- blade pitch angle configuration
- modifications to generator speed
- yaw rotation actions
- enabling/disabling of individual turbines or whole wind parks based on the production limit thresholds
- measuring turbine vibration level

In the deployment as depicted in Figure 2.11, the link between IoT Gateway and Control App is constrained by delay and reliability requirements, while the links between IoT Gateway and sensors/actuators are constrained by Zigbee reachability. The control app contains the intelligence needed to perform decisions based on the measured data. The purpose for a setup such as the one in this service could be that the control app tries to balance the fluctuating power generation of the various turbines with the goal to achieve an as stable overall power generation level as possible by the windpark network as a whole. This is a quite realistic example, as wind park operators have to pay significant penalties, if they produce more power than contracted, as counter-measures have to be taken elsewhere to maintain the overall power grid stability.

2.4.4 Requirements

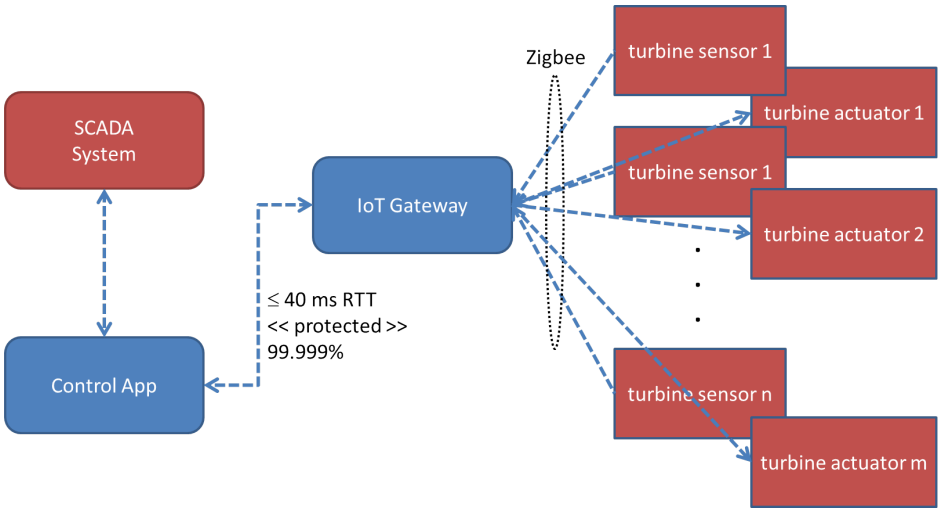

Figure 2.11: The turbine control service

Requirement Name	Integration with existing VNFs or components
Description	The programming model and service platform must allow compoments or VNFs of a new service to be integrated with existing services, VNFs or system components (such as sensors or actuators).
KPIs	support for corresponding annotations (or primitives) in the service programming model/language
Category	Mandatory
Involved Use Case	Industrial Networks

Requirement Name	Service Chaining Support Across Wide Area Networks
Description	The Service Platform must support service function chains that include service functions separated by a wide area network, e.g. across different data centers, or between the core data center and an edge cloud etc.
KPIs	west-east interfaces between service platforms or arcitectural support for service platform hierarchies
Category	Mandatory
Involved Use Case	Industrial Networks

Requirement Name	Manual Service Function Placement
Description	It should be possible to manually decide and configure where a service is to be placed. This can be very important for scenarios where the service developer knows that a Service Function has to run in a certain location / on a certain node, but is either unable to or not confident with defining placement constraints in a way that placement can be done by the orchestrator. This may be particularly the case in non-carrier verticals where experience with services may be lacking, deployment scenarios are simple, and ease of use is the primary objective.
KPIs	an API primitive towards the orchestrator that allows manual function placement
Category	Desirable
Involved Use Case	Industrial Networks

Requirement Name	Service Platform Scalability
Description	The service platform must be scalable to support a very large number of devices (e.g. sensors), a high traffic load, high dynamics etc. depending on the use case.
KPIs	support for 1000s of sensors, support for line-rate performance traffic processing in service chains across all use cases
Category	Mandatory
Involved Use Case	Industrial Networks

Requirement Name	Service Platform Customizability
Description	The service platform must be customizable to support large-scale, complex deployemnts (such as carrier networks) as well as smaller, lightweight deployments (such as enterprises or industrial networks).
KPIs	plug&play, configurable service platform architecture that can be trimmed to a minimum basic feature set
Category	Highly desirable
Involved Use Case	Industrial Networks

Requirement Name	Support for Service Templates or Cardinalities
Description	The programming model must support service templates. In other words, it must support the inclusion of **types** of nodes, or at least the notion of cardinalities in inter-node relationships, e.g. in order to define an unspecified number of nodes.

Requirement Name	Support for Service Templates or Cardinalities
KPIs	support for corresponding annotations (or primitives) in the service programming model/language
Category	Highly desirable
Involved Use Case	Industrial Networks

Requirement Name	Inter-VNF QoS constraints
Description	The programming model must support end-to-end QoS properties for inter-VNF communication, such as delay, jitter, reliability (which could be mapped to multi-path transmission by the orchestrator, the developer does not care necessarily), oversubscription
KPIs	support for corresponding annotations (or primitives) in the service programming model/language
Category	Mandatory
Involved Use Case	Industrial Networks

Requirement Name	Placement constraints for VNFs
Description	The programming model must support to specify placement constraints for VNFs, e.g. disjoint placement of active and standby VNF on physically separate machines, pinning a VNF to a specific node or node type (e.g. turbine control must run on a turbine node), hosting nodes must offer certain real-time capabilities or security isolation features, etc.
KPIs	support for corresponding annotations (or primitives) in the service programming model/language
Category	Mandatory
Involved Use Case	Industrial Networks

Requirement Name	Capability Discovery in Service Platform
Description	The service platform, notably the infrastructure abstraction layer, must support the discovery of capabilities of the physical infrastructure, e.g. support for hardware-acceleration for certain functions such as encryption or the availability of a Zigbee interface. That way, it will become possible to optimize function placement and maybe even tune applications that have access to the capability-enriched network model via the service platform's NBI.
KPIs	capability discovery primitives in the infrastructure abstraction layer and the service platform's NBI
Category	Highly desirable
Involved Use Case	Industrial Networks

Requirement Name	Isolation constraints for VNFs
Description	The programming model must support isolation constraints for VNFs. This is in terms of performance, e.g. in order to guarantee min. capacity without being preempted by concurrent services. But it is also in terms of security, e.g. in order to restrict visibility of (virtual or real) infrastructure to a particular service, or to constrain a service to specific time windows (e.g. only between 10am and 11am, or expiry one hour after first use).
KPIs	support for corresponding annotations (or primitives) in the service programming model/language
Category	Mandatory
Involved Use Case	Industrial Networks

Requirement Name	Timely alarms for SLA violation
Description	The monitoring system must supply alarms for SLA violations (or malfunctioning components) in a timely manner depending on the SLA and type of problem. This means that the failure detection, but also the service platform message bus and notification system must have real-time capabilities. E.g. VNF unavailability for real-time traffic must be signaled as fast as possible, while in the case of best-effort traffic alarm signaling can happen with modest delays. Likewise, urgency of alarms is higher for VNFs with 1000s of users compared to single-user VNFs in the general case.
KPIs	proven performance and scalability of the selected message bus system in the service platform
Category	Mandatory
Involved Use Case	Industrial Networks

2.5 Use Case: virtual Evolved Packet Core

2.5.1 Rationale

Evolved Packet Core (EPC) is the central part of an LTE mobile network and it is critical to the delivery of end-user services. It performs essential functions including subscriber tracking, mobility management and session management. These functions are provided by mainly 5 elements (HSS, MME, S-GW, P-GW and PCRF). Figure 2.12 presentes these five components of the EPC, which, along with the RAN (Radio access network), composed of the eNodeB and the UE (User Equipement), constitute the LTE architecture.

EPC is one of the key network domains that was identified by operators and vendors as a good candidate for NFV and is subject to several Proof-of-Concept demonstrations selected and approved by the ETSI NFV Working Group as a NFV implementation that delivers real-world benefits (see POCs 5,25,26,27,34,37 in [30]).

The expected benefits of a Virtualized Evolved Packet Core (vEPC) are:

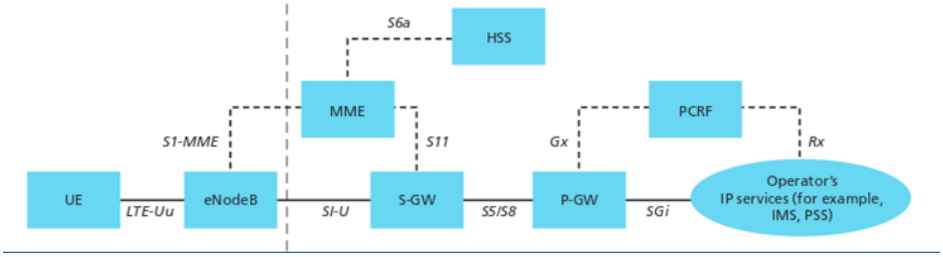

Figure 2.12: LTE architecture

- Easier customization through programmability: the core network can be more easily customized and optimized for end-user needs.
- Elasticity and cost-effectiveness to dynamically adapt to usage patterns, as opposed to over-provisioning to meet worst-case load scenarios. In situations like crowded events, emergency situations or generally situations in which there is an increased service demand, quickly deployable and elastic vEPC elements would bring increased performance, resilience, flexibility, upgradability and cost-efficiency:
- Increased resilience: it allows being more reactive and flexible in case of failures or security attacks, by quickly re-instantiating vEPC services at the best location.
- It opens the way to new Mobile Virtual Network Operator (MVNO) business models, where multiple vEPCs could run in parallel, sharing the same multiple tenants infrastructure provided by a unique telecommunications operator to multiple MVNO.
  - A vEPC is deployed as a network slice
  - Some elements might be shared by multiple tenants while other might be dedicated/isolated
  - Different levels of services can be explored:
    * The network operator enables MVNOs to deploy their own EPC (or pieces of their own EPC)
    * The network operator offers dedicated EPC services to MVNO (kind of "one-click MVNO")
- Simpler development, tests and upgrades: Fast vEPC deployment and the possibility to easily run multiple vEPCs in parallel enables hosting homogeneous development, testing, staging and production environments and performing seamless incremental upgrade (DevOps). This allows for reduced deployment costs and shorter time-to-market of new features.

It is clear that the benefits of virtualizing EPC will be demonstrated by both core parts of SONATA - the DevOps toolkit (SONATA SDK) and the service platform. The DevOps toolkit will enable the vEPC operator to easily develop and define a dynamic vEPC application that will be managed by the service platform that will monitor the application and constantly optimize its deployment.

The vEPC is a pure Telco user story, using it in SONATA will demonstrate the challenges of moving the telco from a physical world to a virtual world as part of the transformation to 5G.

2.5.2 Executive Summary

This use case focuses on showcasing and assessing the SONATA system capabilities in order to enhance a virtual EPC service with elasticity, placement optimization, resilience and DevOps methods.

The use case address MVNO wishing to operate their vEPC as a network slice on top of a virtualized infrastructure provided by NFVI providers in order to gain more flexibility.

More flexibility to continuously dimensioning the service, function of the end-users activity, to optimize placement with regards to the users' locations, to increase resilience by optimally adapting the system to failure or security attack, and to bring new maintenance workflow using DevOps principles.

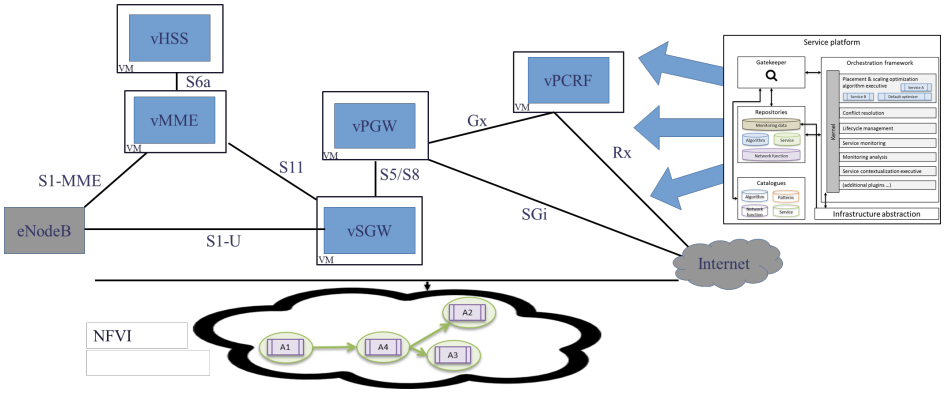

Figure 2.13: vEPC over SONATA

2.5.3 Detailed Description

2.5.3.1 Triggering Event, Preconditions, Assumptions

A mobile service provider wants to provide a mobile service without relying on its own infrastructure. Today, such a mobile service provider, also called MVNO, is sharing the Radio Access Network (RAN) of a telecom operator and, either sharing the core network (EPC) or hosting their own EPC if they need more control of their service. This use case wants to bring a new model for MVNOs, where the RAN is still shared by a telecom operator, but where the EPC is owned and operated by the MVNO but hosted in a virtualized infrastructure offered by an NFVI provider. This alternative brings a new level of flexibility, with new perspectives in terms of elasticity, scalability, placement optimization, resilience and management.

Before the use case is executed it is assumed that;

- The operator NFVI, being a shared infrastructure, is hosting active virtual network/application services for other service providers.
- There is no vEPC instance of the target MVNO in the operator NFVI.
- The target MVNO is aware of its estimated work load profile and thus wants to provision a vEPC service based on this estimation.
- The MVNO is also aware of the main application services, and hence it's operational requirements, that it wants to support over the vEPC system.
- The target MVNO has already selected a vEPC VNF software package from a selected vendor.

2.5.3.2 Sequence of actions

2.5.3.2.1 Actors and their roles / involved system components The main actors involved in the use case are:

1. The vEPC vendor
2. The vEPC service provider (SP)
3. The NFVI provider
4. Mobile device users

The **vEPC vendor** develops and provides the vEPC system software bundle as a VNF. This will include, at the minimum, the vEPC functional components (i.e., VNFC), the vEPC system descriptor (i.e., the VNFD), and the vEPC system manager (i.e., the VNFM). The VNFD will describe the functional/operational/performance requirements and /constraints/limits of the individual VNF components (VNFC), like MME, PGW, SGW, etc. It is assumed that the VNF manager (i.e., the VNFM) is part of the vEPC VNF package.

The **SP** can be an MVNO that leases resources from the NFVI provider in order to deploy, configure, and operate the vEPC system to provide end-to-end communication services to its mobile customers.

The **NFVI provider** is typically a datacenter operator that owns the physical infrastructure and hosts the vEPC system for multiple MVNOs, and ensures that the resource requirements (compute, network, storage etc.) of the MVNOs are provided as per the service agreements. It is assumed that the NFVI will have a generic VNF monitoring, management and orchestration (MANO) system as part of the service.

The **Mobile devices users** use the SP's network for both voice and data applications.

2.5.3.2.2 Action of Actory and information flow between actors and components Figure 2.14 provides an overview of the SONATA ecosystem specifying the relations between the different actors. The following sequence of minimum steps are envisaged to demonstrate parts of Sonata's SDK functions/capabilities:

1. As depicted in Figure 2.14, the vEPC vendor will offer and upload the image of the vEPC system software as a complete VNF package to SONATA's image catalog repository. An interface will be specified that will enable authorized 3rd party vendors for uploading the required software image(s).
2. The MVNO (the vEPC SP) will provide his functional/operational requirements to the SOANATA SDK, along with the estimated control/user plane (C-plane and U-plane) load profiles and RAN density (i.e., the number of eNBs that is available to the MVNO). Optionally the MVNO can also provide its preferred vEPC image in case there are multiple ones available.
3. Taking into account the load profile provided by the MVNO, the SONATA SDK will parse the VNFD to determine the vEPC VNFCs performance limits and internal affinity with each other. This will enable the SONATA SDK to determine the number of each type of VNFC (i.e., vMME, vPGW, vSGW etc) that is required to be deployed to meet the MVNO's estimated load profile. For example, a typical EPC system at a national level is composed of 10s of PGW, 100s of SGWs, and 1000s of eNBs etc.

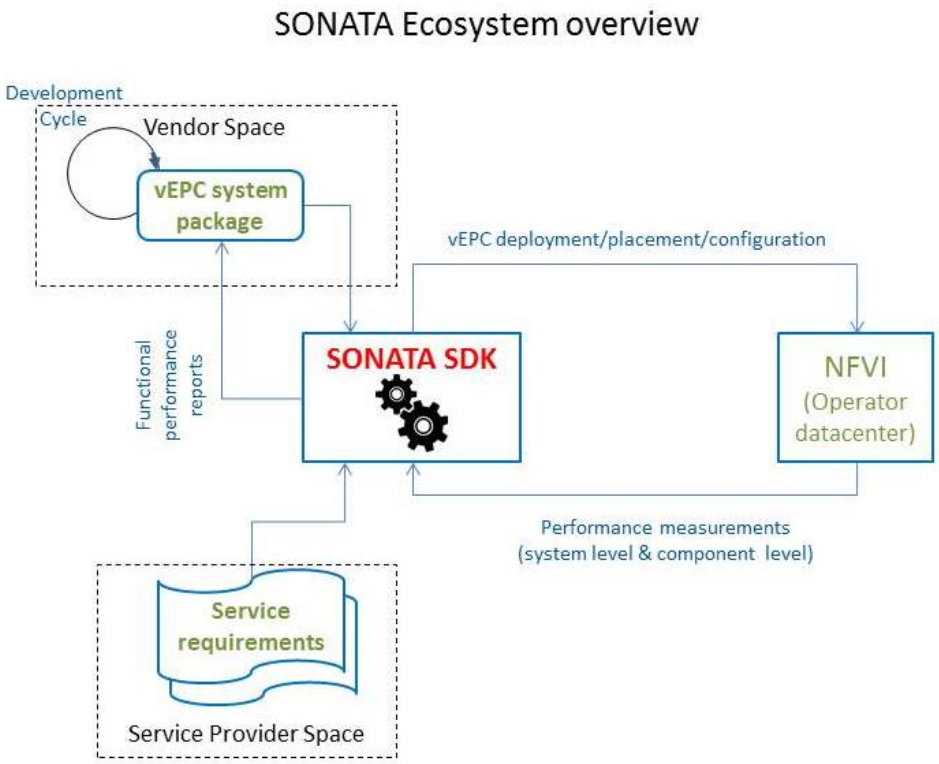

Figure 2.14: vEPC and SONATA ecosystem

4. The SOANTA SDK will also parse the VNFD in order to:
   a) Determine any (anti)affinity rules and performance requirements (e.g., bandwidth, delay, error requirements between two interconnected vEPC VNFCs) that must be taken into consideration when making deployment rules.
   b) Determine the compute, network and memory requirements for each VNFC type. This information will be used by the orchestrator to select the best possible servers for deploying the vEPC VNFCs.
5. Optionally, the SONATA SDK may also be provided with the NFVI infrastructure model and the load-delay signature of the NFVI links in order to determine the best placement strategy.
6. The SONATA SDK will create topology and configuration details and will execute the deployment of the vEPC system in the NFVI.
7. Once deployed inside the NFVI, the KPIs of the vEPC system will be monitored and fedback to the SONATA system that will then take corrective measures in case of any performance shortcomings.
8. As SONATA embodies the DevOps strategy, the information feedback from the NFVI to the SONATA system will also enable the system developer (i.e., the vendor) to quickly identify/fix any bugs that may become apparent during the system test/live operations. Besides developmental tasks, the SONATA system may dynamically change the configuration/deployment parameters in case of changing load and/or resource conditions by triggering vEPC's VNF lifecycle management actions (e.g., scale in/out, migration etc).

When triggering actions, SONATA will also take into consideration the effect of its actions on other virtualized systems inside the NFVO. For instance, as seen from the Figure, an NFVO may host two vEPC systems from two different MVNOs (i.e., MVNO-A and MVNO-B in the figure Figure 2.15), and thus when proposing actions, such as a scale-out or migration of a vEPC component to different servers, it shall ensure that such an action shall not impact the performance of VNF instances running over the same infrastructure.

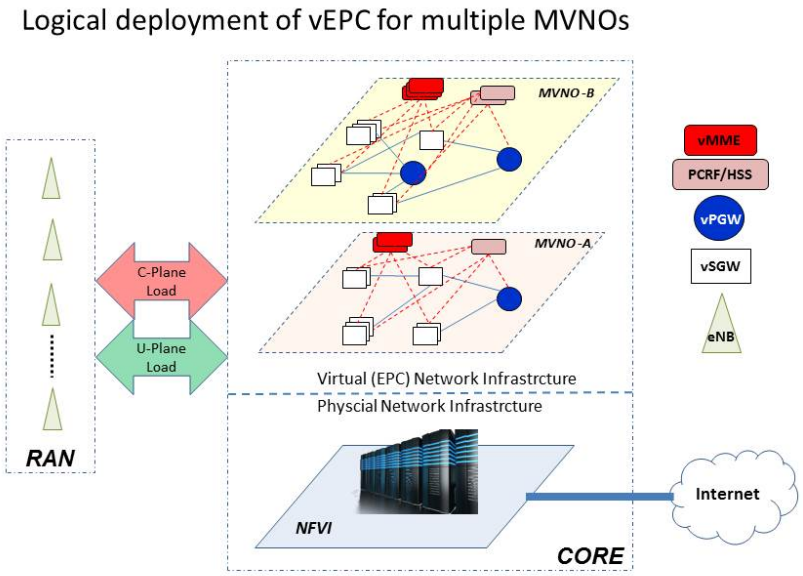

Figure 2.15: vEPC Deploy

2.5.3.3 vEPC maintenance & DevOps workflow

2.5.3.3.1 Component upgrade The DevOps approach is to incrementally and frequently update a service graph to upgrade/downgrade its components. The changes must be easily applied or roll-backed onto a running service graph instance. All the modifications must be traceable, generally by using a VCS.

Minor ongoing upgrade The following sections explain how to apply a minor upgrade on a running service graph instance. A minor upgrade is defined as a change compatible with the rest of the service graph's components. In general, the service graph is developed to ensure that this type of upgrade doesn't create a service interruption.

Partial usability upgrade The vEPC service operator can trigger a partial *usability* upgrade. This upgrade can be for example some bug fixes or security patches for a component.

1. The component (a vEPC component or a NF) developer fetches the component code source.
2. The component developer builds a new version of a component containing the fixes.

3. The component developer uploads into the Sonata's catalogues the new component as a *package* (container, VM) with a bumped version.
4. The vEPC service developer might use the Sonata SDK to check if the vEPC service graph manifest contains obsoleted components.
5. The vEPC service developer modifies the vEPC service graph manifest to replace the old component.
6. The vEPC service developer modifies the NFV rules described inside the vEPC service graph to route a subset of users to the new component.
7. The vEPC service developer bumps the version of the vEPC service graph manifest.
8. The vEPC service developer creates an up-to-date vEPC service package.
9. The vEPC service operator deploys the new up-to-date vEPC service package in the Sonata Platform.
10. The Sonata Platform computes a diff between the actual running vEPC service graph instance and the targeted vEPC service graph instance contained in the previous vEPC service package.
11. The Sonata Platform modifies the running vEPC service graph instance.
12. The vEPC service operator monitors the application and checks if any alarms is raised.
    a) Option 1: The vEPC service operator can undo his changes by:
        i. Undoing his local VCS history to revert the changes in the vEPC service graph manifest.
        ii. Re-building the previous vEPC service package.
        iii. Re-submitting the vEPC service package to the Sonata Platform.
        iv. The Sonata Platform computes a diff between the two vEPC service package then creates and applies a set of instructions to nullify the upgrade.
    b) Option 2: the vEPC service operator can trigger an automatic rolling update which will remove and replace the old component one by one.

A/B testing The previous partial upgrade process can be extended to an A/B testing workflow. In this case, the monitoring window is longer. Moreover the measures are more high level as the operator tries to detect the changes in the user behaviour.

A basic example can be:

1. The vEPC service operator chooses to insert audio ads into the end user's calls.
    a) This ads insertion feature is shipped as a service graph (some software components and a VNF):

       It intercepts VoIP packets to detect the creation of a call to reroute them inside an ads robot.
    b) The vEPC service operator customizes this ads service graph by chaining it behind a 'routing' NF.

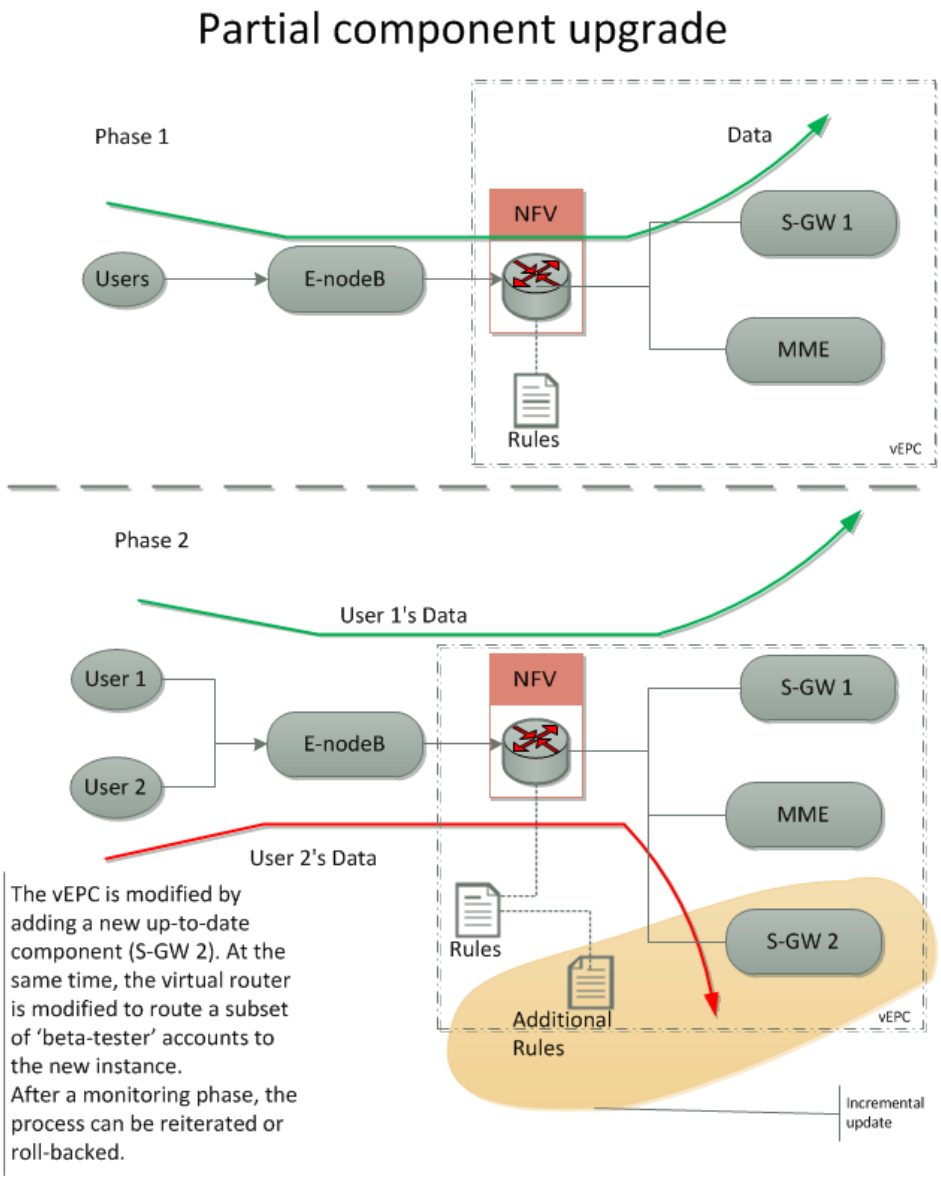

Figure 2.16: vEPC Partial_component_upgrade

c) This 'routing' NF can be controlled to not modify incoming calls or to forward them into the ads service depending of the user/rules.

2. The vEPC service operator creates rules to ensure that a constant subset of user always get the same ratio of ads (from 0% to 100%).

3. The vEPC service developer/operator apply the previous partial upgrade process to modify the running vEPC service graph instance.

4. The vEPC service operator monitors:

   a) The number and length of calls.

   b) The number of dropped calls when an ads starts playing.

   c) The number of cancelled subscriptions.

5. The vEPC service operator makes a report to the vEPC service owner to quantify the impact of the new feature.

6. The vEPC service operator uses the previoud partial upgrade process to remove the ads VNF.

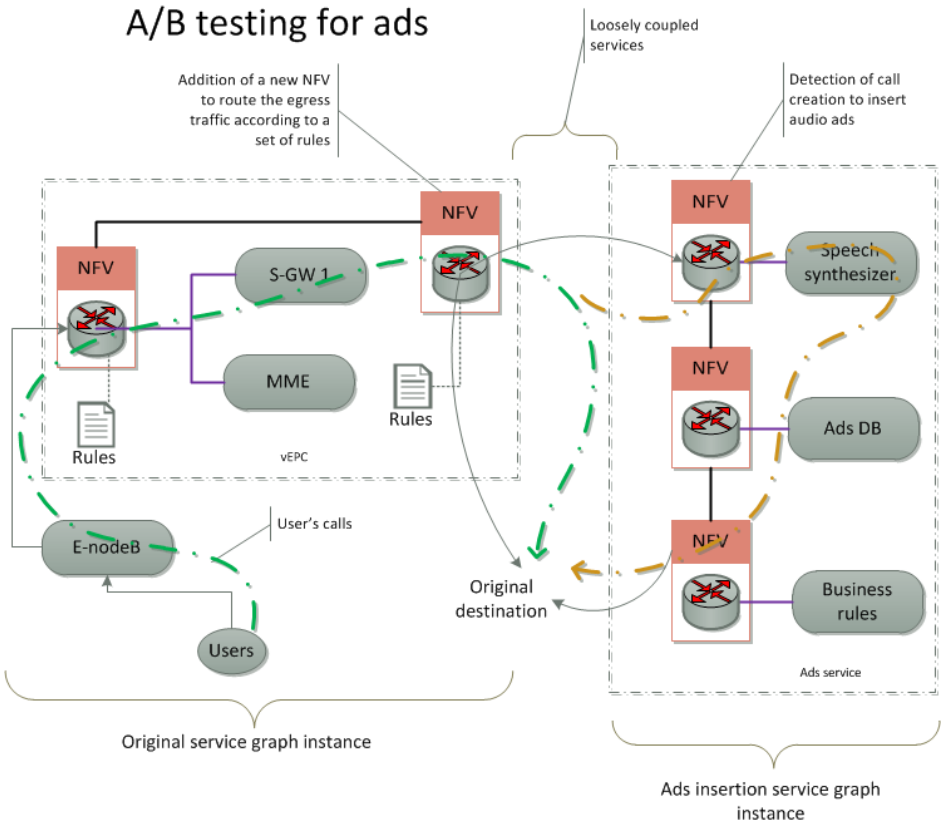

Figure 2.17: vEPC DevOps Workflow ABTesting

Rolling update The rolling update mechanism is a process to migrate a stack of identical components from one version to another. The process can be done explicitly by the vEPC service graph developer. In this case, he added extraneous components into his original service graph manifest for assuring this feature. To achieve the wanted state, the vEPC service graph developer should use the previously covered *manual* workflow.

Now, in the context of the *cloud computing*, the services are built to scale horizontally. This elasticity allows a quick adaptation to an activity spike. It does mean that this practice is a *basic* and *commonly* used operation and can impact large stacks. So this operation is generally proposed as a native instruction in frameworks and platforms.

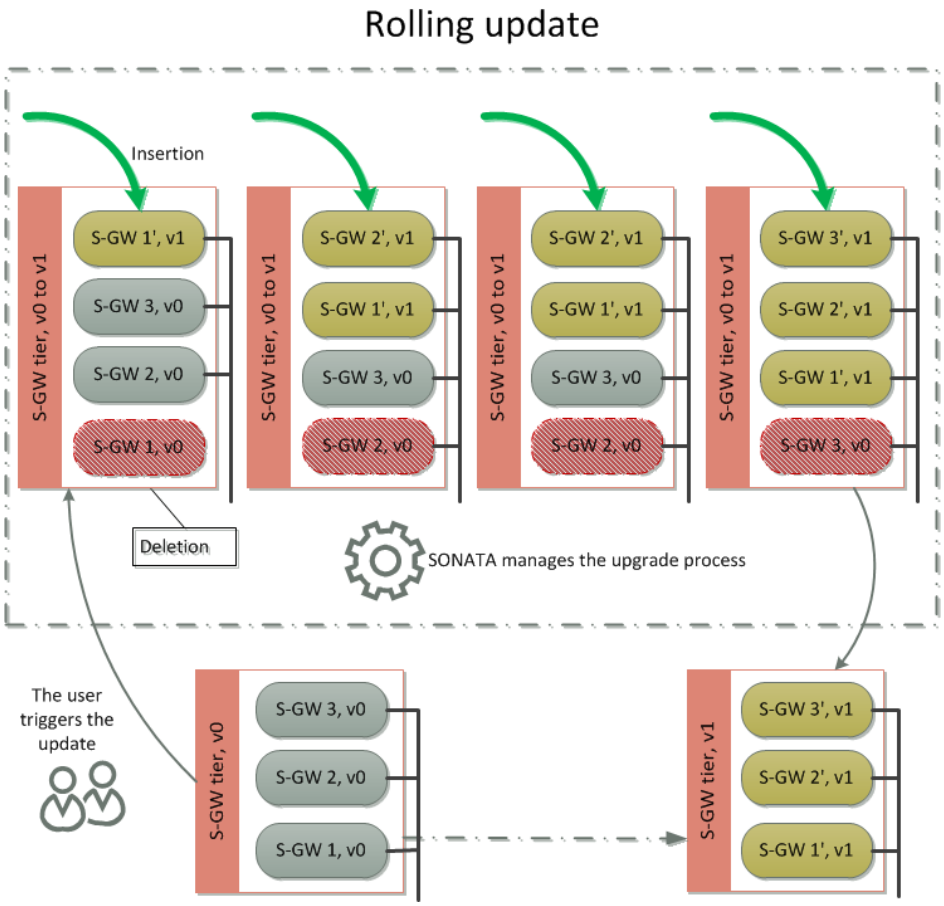

Figure 2.18: vEPC DevOps Workflow Rolling Update

Major breaking upgrade or static scaling In this case, a new vEPC component must be deployed but is incompatible with the *legacy* components. To avoid an insecure or obsolete vEPC, a complete new and up-to-date vEPC can be instantiated alongside the old one. Another scenario is when the vEPC components don't support horizontal scaling. In this case, the new vEPC is deployed with more instances.

In the vEPC context, the key point are:

- The E-node's configuration is left untouched and the migration is transparent from their point of view.

- The migration is done progressively 'on the fly'. In contrary to locking the old vEPC data to migrate it entirely to the new system.
- The users actively using the old vEPC are left untouched. The migration process can prioritize inactive users.
- If the migration goes wrong, the old system is still up and in a stable state.

The workflow is:

1. The vEPC service operator updates the service graph manifest to integrate the new vEPC 2's components.
2. The vEPC service operator can adjust the number of instances supporting the vEPC 2.
3. The vEPC service operator updates the service graph manifest to add a migration task and forward the E-node's traffic to it by changing the networking rules.
4. The vEPC service operator creates and deploys a new VEPC service graph package.
5. The Sonata Platform instantiates a the vEPC 2's components and changes the networking rules.
6. When the migration is done, the vEPC service operator modifies the service manifest: to remove the old VEPC 1's components and the migration task then to clear the networking rules.
7. The Sonata Platform applies the desired changes.

2.5.3.4 Evaluation

This use case will most probably cannot be demonstrated on SONATA's platform, as such a service is currently not available among the partners.
Nevertheless, this use case is a fundamental 5G use case, as one of the essential 5G features will be the provision of EPC services integrated as (part of) one of the *virtual network slices* that 5G networks will support for the provision of specific network services.

This use case will derive requirements for both SONATA's DevOps SDK and toolkit and SONATA's Service Platform, i.e. Sonata's DevOp's SDK and toolkit SONATA's service platform should support the various scenarios and flows that have been outlined in this description, as a characterization of one of the essential virtual network services to be supported by 5G networks from their initial deployment.

When considering demonstrations of a vEPC, the loads that should be taken into account are:

- For the MMEs:
  - 2M subscribers
  - 3K eNodeB associations
- For the SGWs and PGWs:
  - 500K bearers

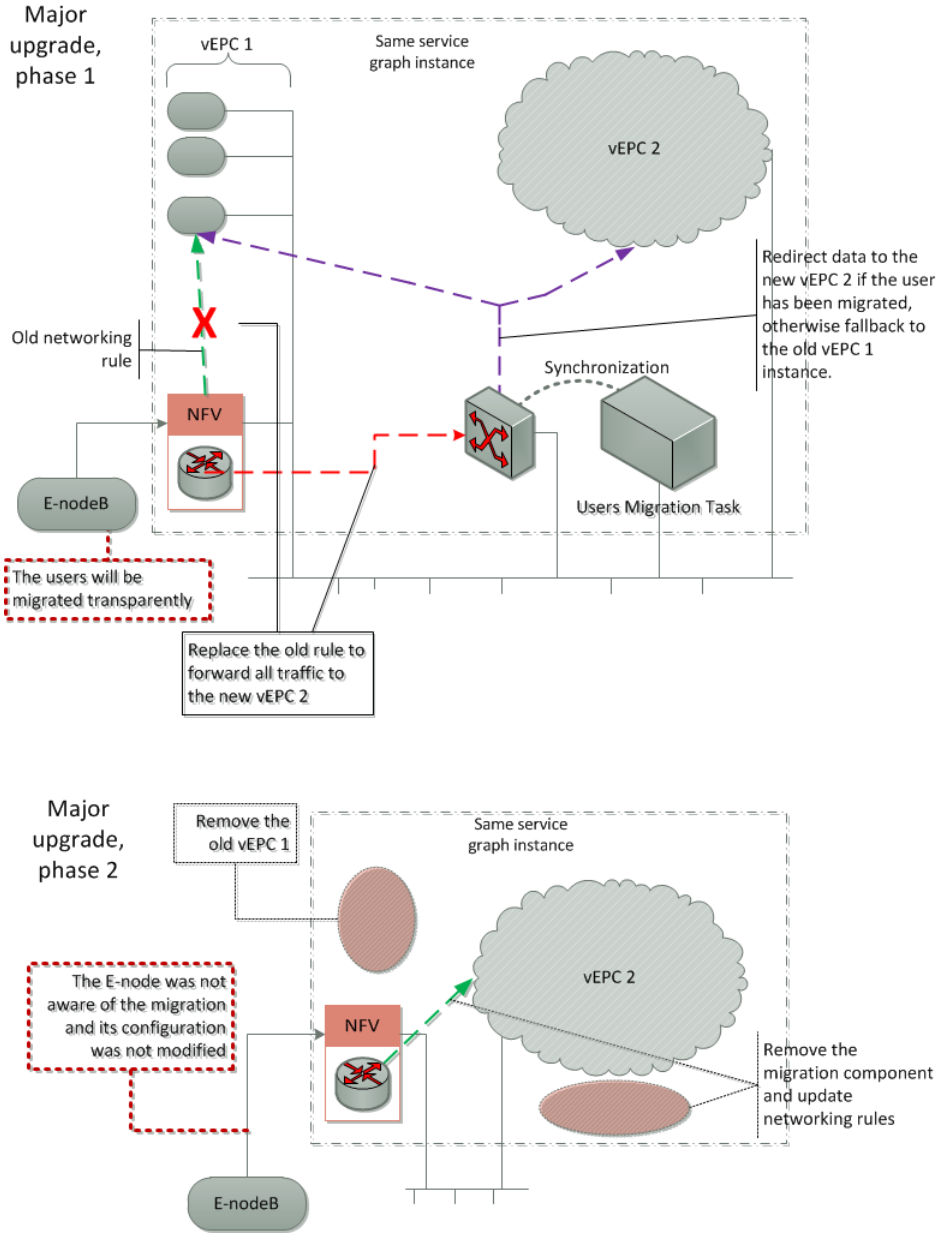

Figure 2.19: vEPC DevOps Workflow Major Upgrade

These loads are based on averaging the dimensioning operational deployments of EPC nodes of a large international operator (Telefonica).
Any demonstration of a vEPC deployment should use equivalent data, i.e keep the relations among the figures, and show that, for example, one hundredth of the capacity can be achieved by a set up that can be realistically scaled up 100-fold.

2.5.4 Requirements

Requirement Name	VNF specific monitoring
Description	The system shall expose service and VNF metrics to the network application
KPIs	Metrics accuracy and response time. Specify service SLA for the EPC service
Category	Mandatory
Involved Use Case	UC vEPC

Requirement Name	Elasticity / Scaling
Description	VNF composing the network service have to be able to scale up or down depending of the users demand. SONATA SDK should have the capability to specify different parameters (VM load, BW, latency) to be used by SONATA Orchestrator for scaling (up or down). The developer SHOULD describe in the VNF Descriptor recipes for scaling his/her VNF
KPIs	Resource usage (CPU/RAM/Bandwith), latency. Service interruption time. VNF specific monitoring metrics. Specify service SLA for the EPC service
Category	Mandatory
Involved Use Case	UC vEPC

Requirement Name	VNF Placement
Description	Orchestration policies/optimizer for VNF placement. The programmability framework shall allow the customer to deploy VNFs at arbitrary points into the network. Set where the components/ gateways will be placed on the operator network. For example, deploy a VNF as near as possible to a specific location or select were the VNF will be deployed.
KPIs	Deployment time, Cost. Specific metrics related to the service SLA/ end-user QoS.
Category	Mandatory
Involved Use Case	UC vEPC

Requirement Name	Multi NFVI orchestration
Description	SONATA Orchestrator SHOULD be able to orchestrate multiple VNF execution environments (NFVI-PoPs) located in arbitrary places in the operator network topology. The NFVI-PoPs are considered to be controlled and managed by VIMs
KPIs	Number and size of the NFVI-PoPs
Category	Highly desirable
Involved Use Case	UC vEPC

Requirement Name	Multi-tenancy
Description	Somme component can be dedicated to a tenant (hard isolation) and some other can be shared (soft isolation)
KPIs	Owner, SLA
Category	Mandatory
Involved Use Case	UC vEPC

Requirement Name	Resilience / availability
Description	Different resilience strategies must be possible. Protection, replication, restoration. A backup component must be located in a different location/infrastructure than the primary component
KPIs	VNF deployment time, availability, Cost, location, shared resources
Category	Mandatory
Involved Use Case	UC vEPC

Requirement Name	Manage update of components
Description	Sequence/ strategy of update using DevOps. Sequence for validation and migration
KPIs	Interruption time, availability
Category	Mandatory
Involved Use Case	UC vEPC

Requirement Name	Support different modes of management/control
Description	EPC can be fully managed by the operator, i.e. EPC fully deployed and managed by the customer (e.g. a MVNO). Or Hybrid were components are managed by the operator (e.g. SGW) and others by the customer (e.g. HSS). Or fully managed by the customer
KPIs	Owners, SLA by component, VNF specific monitoring metrics

Requirement Name	Support different modes of management/control
Category	Mandatory
Involved Use Case	UC vEPC

Requirement Name	Support Services with 5 nines SLA/control
Description	vEPC is a service that is operates under a 5 nines SLA, it can not allow service degradation when scaling / healing / updating / migrating
KPIs	Interruption time, availability
Category	Highly Desirable
Involved Use Case	UC vEPC

Requirement Name	Support "state-full" services
Description	vEPC is a "state-full service" - all its components are state-full, they can not loss their state when scaling / healing / updating /migrating
KPIs	Interruption time, availability
Category	Optional
Involved Use Case	UC vEPC

Requirement Name	Integration with OSS
Description	vEPC service operation involves integration with OSS system, SONATA should expose relevant APIs
KPIs	Owner specific KPI
Category	Optional
Involved Use Case	UC vEPC

2.6 Use Case: Personal Security Applications

2.6.1 Rationale

Network Operator customers are moving their requirements from an increasing bandwidth capacity to a more rich and complex service catalogue. One of the most relevant service interests from users is security, because cybercrime expansion has increased concerns and awareness about security. As a consequence customers demand more protection capacity for their network access and online presence.

The protection of Internet-connected devices, such as mobile phones and laptops, has typically been achieved through installing appropriate specific tools on each device (e.g. a personal firewall, malware analyser, parental control application). However, this raises several issues: privileged access on the device is often required, tools may use a large amount of computational resources, platform capabilities vary, and appropriate tools may be unavailable on particular systems. Moreover **users need to configure each security control on each of their devices**, resulting

in an **unsolvable puzzle** for non-technically savvy users. This results in ineffective or inconsistent protection for users who use different devices across different networks (e.g. border firewalls are available in corporate Wi-Fi environments, but the user is not protected over a radio network). In addition to this problem, 5G will imply an increase of one or even several orders of magnitude in the number of embedded devices connecting to the network, of which many contain little to no security installed. This includes sensors, controllers and appliances that fall under the broad definition of the Internet of Things (IoT). Furthermore, the inconsistent set of security controls **for a large number of devices also creates a policy management and enforcement nightmare** in an age where Bring Your Own Device (BYOD) is part of the everyday in the workplace.

Service Providers and Network Operators are providing device and pervasive access protection from Internet threats by **offloading execution of common security applications away from user devices using NFV technology** creating and Virtual Network Security functions (vNSF), like VPNs, firewalls, or parental controls services. The next steps in this **evolution consists in personalize this vNSFs per customer**, allowing to define their own rules and policies in security, triggered by different desires. This way user experiment the protection that these services offer and the customization as it was locally installed and triggered by user identity

To achieve these demands **a complex orchestration is needed in a massive 5G access** and bandwidth-demanding network. Nowadays state of the art in the area of VNF service orchestration cannot offer per user personalization of vNSFs, and manage it dynamically based on user configuration desires. **SONATA** service development framework **can develop multiples security services personalized per user** and most important be able to **orchestrate dynamically multiples VNFs**, resolve service conflicts and security demands, allowing Network service providers to monitor and respond to user demands. For example monitor a client and deploy a vNSF to mitigate and attack.

Network operators can create new business opportunities by offering highly personalized security services, based on strong DevOps principles, and with support for elastic behaviour (dynamic deployment, automatic escalation,...) of the vNSFs.

2.6.2 Executive Summary

ISP providers create and offer a security service catalogue for their customers. Users aware of security risk when using the network can search the catalogue and decide to subscribe to a security service. The Customer selects the preferred business model the service provider offers: a subscription based on time of use, volume of data, a flat rate, etc.. BSS channels triggers based on user identify on the network the orchestration, deployment and monitoring process of a set of vNSFs that combines the services required by the user. The Customer may decide to change his/her service's configuration (by applying some additional incoming filters, for example) and SONATA reconfigures the service accordingly. For example, this could result in adding and deploying a new vNSF with Parental control and the list of blocked webpages as part of the configuration.

2.6.3 Detailed Description

2.6.3.1 Triggering Event, Preconditions, Assumptions

ISP customer subscribes a Security service with a personalized configuration attached to his identity. This use case assumes that just after starting the service, and periodically, customer decide that the service requires some changes. One example can be changes in rules of a Virtual Firewall to include new services, devices or destinations. Other common situation happens when user has interest in improve the service with new features, i.e. upgrade a filtering capacity adding an

advanced parental control with categories and destinations preconfigured. In other scenarios the user do not deliberately require changes in his security service but its behavioural triggers actions by ISP and requires deploy new security network functions. This is the case of a DDoS attack or a botnet detection that could require that the ISP deploy network functions for anti-DDoS or malware eradication.

In order to executed this type UC some conditions are assumed:

- vNSF catalogue with some Network Security applications are available to respond the user needs or security events.
- The relation between the security needs and the network function is clearly identified and known using a vNSF categorization, i.e. AntiDDoS vNSF category to cover DDoS attacks or IDS vNSF category to solve botnet activity detection.
- There is a NFVI and NFV MANO [20] environment deployed on ISP network. This includes that customer channel and BSS are prepared to interact with NFV MANO.

2.6.3.2 Sequence of actions

The main actors involved in the use case are:

- ISP Subscribers. The user is a customer of an ISP with a personal vNSF demand. The role of this actors is basically demand his own security services
- Network Operator or ISP. The owner of the network or with the capacity to deliver the security Services based on user demands or own criteria. Usually DevOps units in ISP use the vNSF monitoring security reports for automatically triggering of actions when the Operation unit requires the action.
- vNSF Developer. A vendor with products that creates Security network oriented VNF. Their role is focused in sell their security products from their external catalogue adapted to the ISP and user needs.

Additionally there is a set of components involved in the UC that interact with the actors. **SONATA SDK** allows developing user service package based on user desires. It will use the presence and management of private or public catalogues of vendors or marketplace that support security-oriented VNFs. **SONATA Service platform**, with the capacity to orchestrate and manage the set of vNSF that create the customer security service and monitor the behavioural.

Figure 2.20 describes several different sequences of actions and workflows.

The first sequences of actions are related with the User service subscription (solid arrows):

- 1. Customer subscribes and configures a service. The user will use the existing ISP channels (commercial and internal procedures, through BSS/OSS) to request their needs, functionalities and configurations.
- 2. ISP defines the personalized security service based on user requeriments. Using SDK Editor and available catalogues, SDK generates the valid package for the user. User identity is linked to the service package. And delivery to service platform through gatekeeper. Most of the behaviour can be adjusted by proper configuration, but in a broad sense, including for example the definition of the chaining rules to be applied depending on the type of traffic, and that would require an active role of the orchestrator.

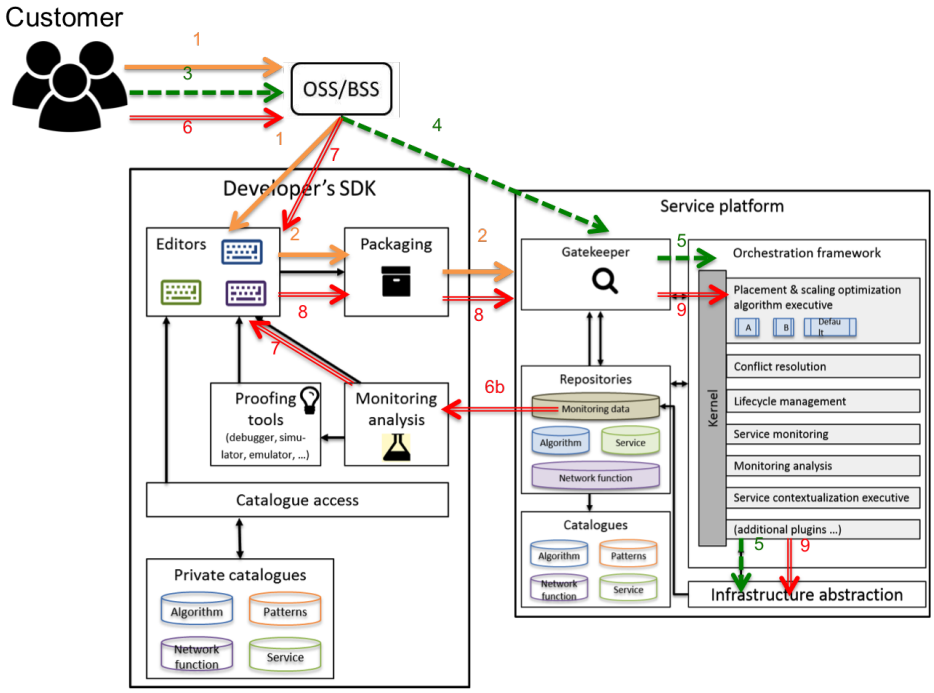

Figure 2.20: Personal security Application UC Global sequence

Next sequences involve the service delivery (dashed arrows):

- 3. User identification happens on the access to the service or the network. This information is used to identify the Personal Security services.
- 4. Network (BSS/OSS) contacts with gatekeeper to deploy the personal security service
- 5. Service platform based on identity of the user and with Network information available deploys the vNSF in the PoP associated to access node using orchestration framework and NFVI capacity.

Dynamic changes in the situation requires a personalized adaptation process (dual line arrows):

- 6. An event is produced by the user request in security service demands using BSS/OSS channel, or
- 6b. Reports and analysis from monitoring process in SONATA framework detects that it is necessary a change in the security service
- 7. The demanded changes in vNSF are processes and solved in the SDK to adapt to the new situation.
- 8. SDK define the new packages. This process can be triggered when there is the need to add a new functionality or to reconfigure the vNSF with requirements that makes substantial modifications in the vNSF or substitution of the original VNF for other. The resulted package progress to the service platform.
- 9. Orchestration component in SONATA service platform update the infrastructure elements according to the new requirements from the user.

In some situations when the demanding changes do not require deep VNFs reconfiguration that requires a SDK process, i.e. deploy a monolithic vNSF non customizable or just simple vNSF configuration changes, SDK SONATA it is not needed and the personalized adaptation process (step 7 and 8) can be managed by SDK Service platform. This allows that the global process will be short.

2.6.3.3 Evaluation

Minimum support will need a couple of personalized security applications (vNSFs) supported in an NFVI that can demonstrate incremental service capacity over existing security services. In the general case it is challenging to demonstrate that all required security services are compatible: this is an essential challenge for a proper execution of the orchestration platform, that must be able to analyze the manifests of each vNSF to ensure that compatibility.

The critical aspect to support of the use case is the personalized adaption process (red arrows), where SONATA can demonstrate the dynamic adaptation and service delivery. Two options are offered; any of them has meaning by itself.

KPIs must be studied in detail, but the more relevant are:

Quantitative: These values show a realistic estimation on real environments on an ISP network where a user request security services. The current SoTA requires long periods of provisioning and deploying cause by strong dependencies on legacy network layouts and static bundles of services no personable. The most relevants quantitative KPI that this user case expect to identify

- Time to deploy security service. Resolution of the security problem or user requirements fulfillment should be done in less that 1 hour.
- Impact in pre-existing network services of the user like browsing or OTT services. Minimum impact in traffic delay ($<$ 500 milliseconds), and Fault time (30min).
- Number of VNFs per user managed by SONATA. Not necessary in the same Node of a NFVI layer, but in distributed nodes.
  - Minimum 1 (usually default security application is identified as basic firewall)
  - Average: 2 (i.e default, traffic monitoring)
  - Maximum: 4. (i.e default, Lawful interception, AntiDDoS, traffic monitoring)
- Number of VNFs management per NFVI PoP. Using a estimation of supporting in average over 30K users per PoP and supposing that there is a service with a market penetration of 5% (approx. 1500 users), we could expect that a Infrastructure NFV orchestrator should be able to manage around 3000 VNFs. Only user demands in the configuration or security events will trigger actions on the SONATA framework so it is reasonable to expect no more than 1% of the VNFs (15) need to be altered simultaneously. A realistic value could be between (2-15 VNFs) per PoP to be managed below SONATA control.

Qualitative: User perception, and satisfaction. User awareness of the process.

2.6.3.4 Deployment Topology

Figure 2.21 shows a typical ISP network topology model. This topology includes a basic path for Internet access customer service. The topology depicts how services are deployed inside Datacenter nodes. These nodes can be centralized Datacenters with high Backbone connectivity capacity to

deal with thousands of users and host multiple services. Alternatively this nodes can be distributed in points of presence (PoP) that are requested for high bandwidth and better time response demanding services. One potential deployment on NFV architecture is focusing the model in exploit these datacenters families to deploy NFVI, or servers were VNFs can be used. 5G networks will involve probably both types and mixed ones.

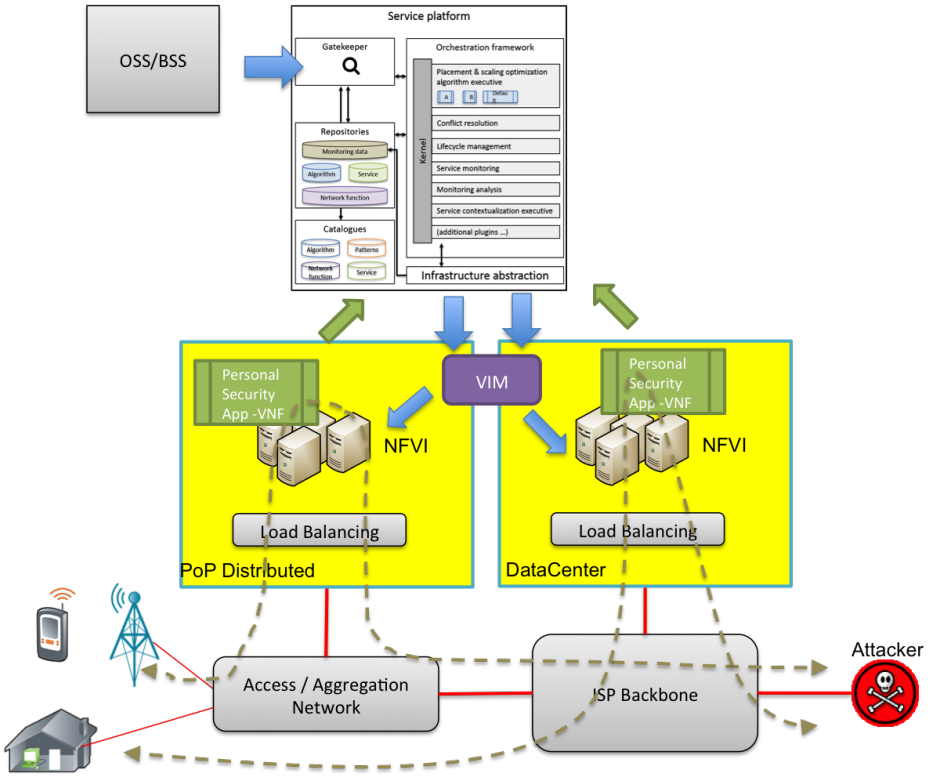

Figure 2.21: Personal security Application Deploment topology

The Nodes involved in this use case are:

- Server nodes: COTS servers with virtualization infrastructures where different personal security services are deployed. The user personal service will be allocated standalone for user. In this case the "personalization" will requires low resources in memory, storage and CPU (like an application running in a PC in user home). Alternatively if the security service can be offered aggregated, then the resources needed in a deployment will be close to commercial services with medium-high requirements (several CPU cores and several dozens of Gb of RAM).

- Network nodes: ISP works usually with dedicated nodes combined with virtual network nodes. The capacity and resource requirements are aligned with the throughput capacity in the links. A reasonable approach is that for security services demanded by a "personal service" the backbone and datacenters capacity are almost endless. Personal Security service will require bandwidth aligned with residential access bandwidth that will be higher in 5G networks starting at 1Gbps. However there is an exception in this model DoS: attacks where some users could be victims of volumetric denial of service attacks that in 5G networks means very high bandwidth demanding. SONATA architecture could help in the dynamic and quick

resource scaling response. Links and connectivity in dataplane are depicted in red in the above picture.

Scalability
The load of personal security service resources, such as intrusion detection system, malware detection, and firewall filtering, are directly attached to the user Internet service habits, and on average follows a stable pattern. These kinds of applications or VNFs are dimensioned to manage this behaviour, keeping a stable resource demand in most situations. Therefore users service demands can be considered rather static. Scalability is needed mainly in 2 potential situations:

1. the User demanding a new service capability, like filter unwanted traffic, that was previously only in traffic monitoring. Here a reasonable approximation is that the filtering, or redirecting the use traffic will require 2x resources compared with monitoring resources, specially in CPU and memory.

2. Alternative situation as mentioned before can be if the user has an anti-DoS service subscribed and received an attack where a high volume of traffic is received and load escalation is needed. In this case we can think that growing to 100x resources, is a reasonable capacity.

Monitoring
Personal security services are focused in monitoring user traffic and react. The metrics that a VNF of this family will be strongly related with logging data (security alerts like attacks, match signatures) to be exposed and bandwidth consumption strong deviation from a standard pattern, that could be identified as an attack.

2.6.4 Requirements

Requirement Name	Security VNF availability
Description	Security virtual network functions require specific capabilities that are not so common in generic VNF, like AntiDDoS or signature detection of IDS. These functionalities must be present to allow creating a valid use case. SONATA VNF Catalogue must include some Security VNFs in order to support this use case.
KPIs	Enough number of Security VNFs
Category	Mandatory
Involved Use Case	Personal Security Application

Requirement Name	Distributed NFVI
Description	ISP and Network Operators architecture requires a geographical distribution of PoP (Point of Presence) where instantiate multiples VNFs as close as possible to user or based on the service demand. One example is when a security attack happens it is preferred to react as close as possible of the source of the attack. As a consequence SONATA orchestration layer should support multiples NFVI and VIM in distributed networks.

Requirement Name	Distributed NFVI
KPIs	Support at least 2 Datacentres or PoP.
Category	Highly desirable
Involved Use Case	Personal Security Application

Requirement Name	Open interfaces towards NFV
Description	NFV components like VIM, NFVI or NFVO could be deployed with multiples providers. Indeed the number od NFV solutions is growing day by day. SONATA orchestration framework must support open or standards interfaces (southbound towards NFVI) to ensure the smooth integration of different NFV providers specially to facilitate the adoption.
KPIs	Open Interface specification or Standardization Body inclusion.
Category	Mandatory
Involved Use Case	Personal Security Application, all

Requirement Name	VNF Real-time Monitoring
Description	In order to detect and react to security incidents, VNFs will generate in real time information useful for Monitoring and response. SONATA framework must be able to collect, store, process and report in valid time windows to be useful to the ISP or the user.
KPIs	Monitoring frequency, time to process alerts.
Category	Mandatory
Involved Use Case	Personal Security Application

Requirement Name	VNF reporting to BSS/OSS and subscriber.
Description	One the most relevant use of a Personal security application is to offer personalized information. SONATA framework must support mechanism to receive, managed and progress to user, VNF's report relevant data. It could imply opening communication paths in data plane or offer a alternative channel through SONATA framework.
KPIs	Specification of a channel for user report.
Category	Highly desirable
Involved Use Case	Personal Security Application

Requirement Name	Scalability.
Description	In general there are some Security VNFs related with resource consumptions (i.e. anti-DDoS) and therefore requires dynamic scalability of resources in the SONATA framework. Also based on the concept of personal security application SONATA will require manage light VNFs but in great number, so scalability will be relevant if the number of subscribers rises considerably.
KPIs	Escalation functionality availability and provisioning times.
Category	Highly desirable
Involved Use Case	Personal Security Application, All

Requirement Name	Legacy support.
Description	Any ISP or Network Operators or corporations has today deployed security networks solutions in virtualized or bare metal appliances. The most relevant example is a Firewall device. If SONATA has the aim to offer complex solutions and integrate with existing network environment, then SONATA need to interact and manage with not only VNFs also support legacy NF.
KPIs	Integration of one legacy NF.
Category	Optional
Involved Use Case	Personal Security Application, All

Requirement Name	Availability.
Description	Users expect to have service availability similar to any other ISP service and Service providers will search for it. As a consequence fault tolerance support in configuration, and monitoring is highly recommended.
KPIs	SLA values, down times.
Category	Highly desirable
Involved Use Case	Personal Security Application

Requirement Name	Quality of service monitoring.
Description	One of the key method to detect security problems are the deterioration in the QoS. Metrics generation and degradation detection of network traffic, i.e. caused by a overloaded NFVI node or a attack, should be supported and reported.
KPIs	Traffic QoS , packet loss, delays.
Category	Highly desirable
Involved Use Case	Personal Security Application

Requirement Name	Confidentiality.
Description	All the information collected and treated by SONATA framework must keep the confidentiality and integrity principle. This must be achieving by strict authentication and authorization controls in the framework. This is especially important in this use case because the VNFs work with information per user, making it easily identifiable.
KPIs	Security controls in place.
Category	Manadatory
Involved Use Case	Personal Security Application

Requirement Name	Legal compliant.
Description	There are several legal requirements like lawful interception or data retention that could affect to user traffic. Only when these requirements apply, SONATA framework must support mechanism to accomplish it. i.e. accountability of the service.
KPIs	Legal compliance measures (only if needed).
Category	Manadatory
Involved Use Case	Personal Security Application

Requirement Name	Personalized VNF.
Description	VNF catalogue and management framework in SONATA must support the concept of “personal” in the sense that VNFs are assigned as a non-shareable resource with other users in the platform. Also Users identities in SONATA framework must allow a direct mapping between user and his VNFs .
KPIs	VNF descriptors supporting this functionality, validation of uniqueness of VNFs per use.
Category	Manadatory
Involved Use Case	Personal Security Application

Requirement Name	VNF and topology validation.
Description	Based on the principle of providing security service, SONATA service framework by itself, or using third parties, must offer a validation capacity of VNFs when it is deployed in the NFVI. This validation should cover the integrity of the VNF, user attestation and data paths.
KPIs	Attestation mechanism validation.
Category	Manadatory
Involved Use Case	Personal Security Application

Requirement Name	Security simulation tools.
Description	A common problem in security applications and service is the capability to simulate security incidents. Security simulation tools availability and integration in the SONATA framework are needed for validation and testing, i.e: DoS traffic generators, or malware traffic samples.
KPIs	Number of available security simulation tools.
Category	Highly desirable
Involved Use Case	Personal Security Application

2.7 Use Case: Separate Client and Hosting Service Providers

2.7.1 Rationale

In the current global telecommunications industry there are many different service providers. There is a tendency for any one service provider to either have little access infrastructure at all or, if they do own access infrastructure, for that access infrastructure to be concentrated in particular geographies (for example a particular country or region). This means that some level of interworking between is essential and fundamental to the telecommunications industry if end users are to enjoy an end to end service.

Interworking necessary for end to end services falls into two categories:

- **peer level interworking** where the interworking SPs interwork the same level of protocol that delivers the service to the end user;
- **client/server interworking** where only one party handles the end user service protocol (the client SP) and the other (the server SP) provides a hosting service over which the client SP can operate their end user protocol transparently (the server SP has no visibility of the end user protocol).

Both models are in widespread use in the delivery of standard telecommunications services today. The peer level model is exemplified in the Internet by BGP and in the PSTN by various C7 signalling interfaces. The client/server model is exemplified by Ethernet E-Line, E-tree, and E-LAN services as well as other leased line services.

Both models will also undoubted appear in the virtualised world of NFV and SDN. With SDN, the model should be already covered by the simple extension of the existing telecommunications client/server model outlined above. However, with NFV, a wholly new business model is possible: NFV hosting which has already been identified within the ETSI NFV ISG. The NFV infrastructure (NVFI) can be provided by one SP as a service, NFV Infrastructure as a Service (NFVIaaS), and can be used by an different SP to host VNFs and deliver network services. The NFVIaaS provider has no visibility of either the VNFs or the network services it is hosting. These are entirely within the control of the client SP.

Within this new business model, the SONATA orchestration system needs to be applicable to both the hosting SPs and the client SPs which provides the motivation for this use case.

2.7.2 Executive Summary

This use case describes the situation where there is more than one service provider involved in the establishement of an NFV network service and where both are running independent instances of

SONATA. In particular, the case is considered where one service provider is responsible for the VNFs and their composition into a network service while a different service provider is providering the NFV infrastructure on which the VNFs are hosted. The model is a 'client/server' model between service providers rather than a 'peer-peer' model.

The use case is concerned with the interaction between two instances of the SONATA system treated as a whole rather than specific interactions between components of the SONATA system. The main requirement captured by this use case relates to the business model rather than the detailed architecture of the SONATA system. The use case does not therefore detail specific functional interactions between the two instances of the use case; such details are appropriate for a later stage in the design process.

The use case requires that the SONATA orchestrator supports the following requirements.

- The SONATA system must be able to support an API which exposes the NFV Infrastructure as a service
- The SONATA system must be able to support a southbound interface which can request service elements of NFV Infrastructure as a service;
- Information held with an instance of a SONATA system must not create unnecessary duplication or other dependencies. In particular, a SONATA system offering NFV Instrastructure as a service must not hold specific information about the type of VNF or type of network services into which VNFs are composed. Similarly, a SONATA system composing network services which are hosted on NFV Infrastructure controlled by another service provider, must not hold implementation details of the infrastructure.

2.7.3 Detailed Description

2.7.3.1 Assumptions and Triggering Events

This use case is illustrated by the two use case diagrams below. The two diagrams show the same use case from two different perspectives. The first, Figure 2.22, shows the use case from the perspective of the SP offering the NFVIaaS while the second, Figure 2.23, shows the use case from the perspective of the client SP.

In the first viewpoint, the infrastructure SP from whose perspective the view of the use case is taken, will see two types of client actors. The first type of client actor will be an end user and the use case will appear identical to a situation to the situation where there is only one integrated SP. The second type of client actor is a and client SP and it is the simultaneous presence of this type of actor which defines this use case and the requirements which flow from it. While it is not a explicit requirement of this use case as more detailed functional interactions are not directly constrained by the use case and will be developed later under the architecture deliverable, we would expect that the activity which delivers the end user interactions would draw on the activity which deliveries the client SP interactions.

In the second viewpoint, the client SP which does not own NFV infrastructure will see two actors; their end users and the infrastructure owning SP supplying it with NFVI as a service. The client SP will have activity which interacts with both of these actors which would not be captured if SONATA were only scoped for an integrated SP.

2.7.3.2 Sequence of Actions and Deployment Topology

In this use case, the complete SONATA system can be regarded as a complete composite functional block as the primary intention is two instances of the complete SONATA system interworking with

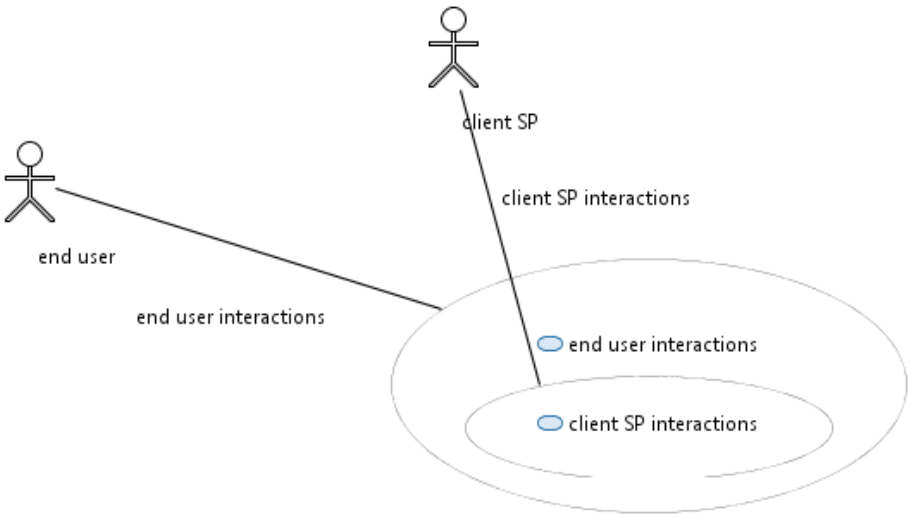

Figure 2.22: Use case from the perspective of an infrastructure owning SP

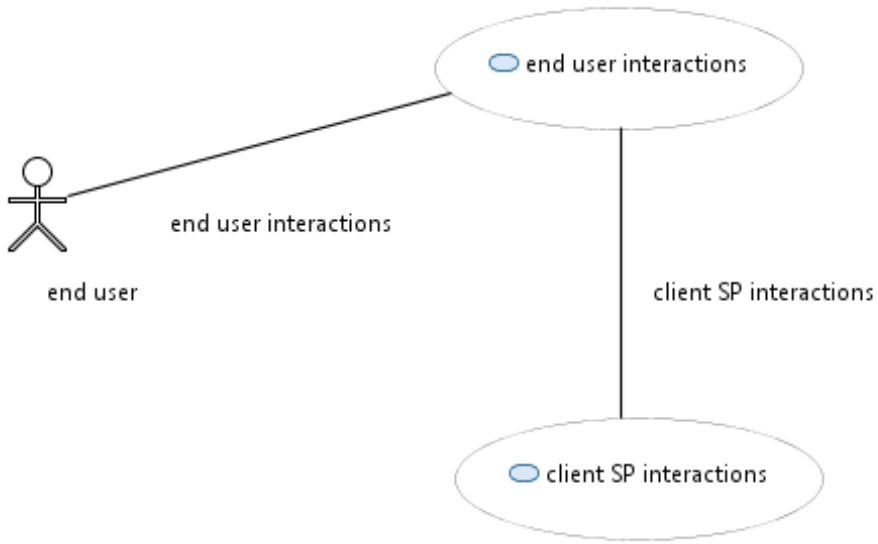

Figure 2.23: Use case from the perspective of a client SP

each other. The use case is concerned with the interaction between two instances of the SONATA system treated as a whole rather than specific interactions between components of the SONATA system. The use case does not therefore detail specific functional interactions between the two instances of the use case; such details are appropriate for a later stage in the design process in the architecture deliverable.

Figure 2.24 below shows a simple class/block definition diagram for the use case. Note this diagram and subsequent diagrams are drawn broadly using SysML conventions.

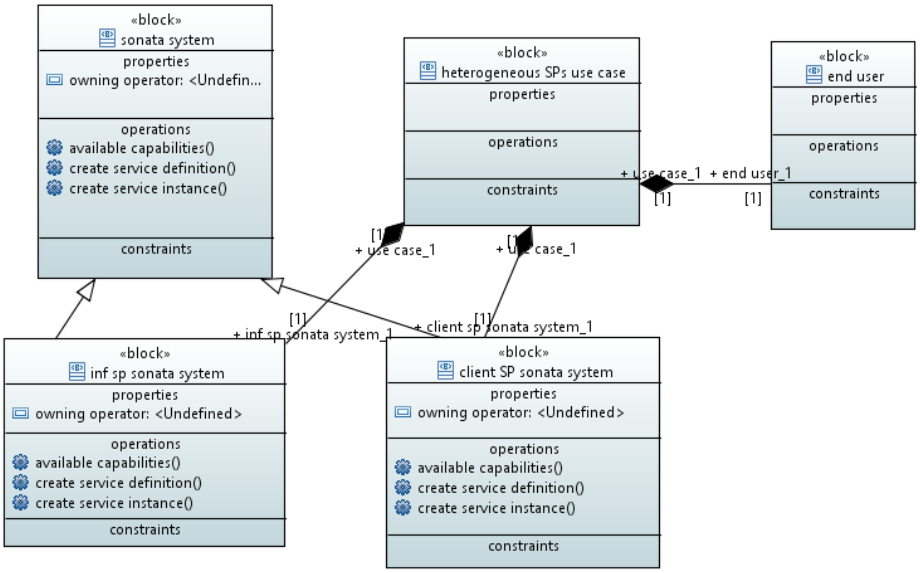

Figure 2.24: Class diagram of major functional blocks of the use case

At the very top level appropriate to this use case, we can see the overall SONATA system as a single functional block which can support three basic operations:

- respond to a request for information regarding capabilities which could be available - services, available resources, available service elements for the construction of services, capabilities to carry out service design;
- respond to a request for a new service design;
- respond to a request for a new service instance.

The diagram shows that the SONATA system can be specialised in two different types, an infrastructure SP type or a client SP type.

Finally this class diagram also shows that the formal set up of use case for testing and validation of the use case. This will consist of one end user, one client SP SONATA system and one infrastructure SP SONATA system.

The internal block diagram below, Figure Figure 2.25 shows this construction of the use case for testing and validation of the use case.

2.7.3.3 Evaluation

A UML/SysML convention is that a use case is realised by one or more UML/SysML activities. This means that the test and evaluation of a use case can a verification that the appropriate inter-

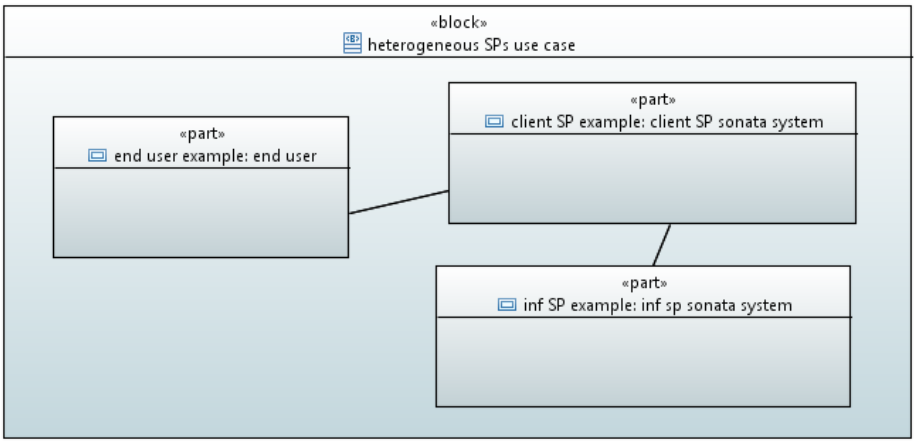

Figure 2.25: Functional block diagram for the use case

actions of the SONATA implementation described by appropriate UML/SysML activities deliver the appropriate interactions between the actors.

In this use case, both the client SP and the infrastructure SP must have activities corresponding to each of the main operations supports by the SONATA system, that is

- an activity to respond to requests for information about capabilities
- an activity to respond to requests for new service designs
- an activity to respond to requests for service instances

2.7.3.3.1 Evaluation of the SONATA System of the Client SP The figures below give a high level view of these activities for the client SP in response to an end user request. They show how the client SP SONATA system must be able to test whether an interaction is needed with the infrastructure SP in order to meet the request, make that interaction, evaluate a response using that information, and only then respond to the end user.

The high level activity diagram for the response to a request for capability information by an end user is shown below in Figure Figure 2.26.

The high level activity diagram for the response to a request for a new service design by an end user is shown below in Figure Figure 2.27.

The high level activity diagram for the response to a request for a new service instance by an end user is shown below in Figure Figure 2.28.

2.7.3.3.2 Evaluation of the SONATA System of the Infrastructure SP The infrastructure SP must also have activities which can respond to requests from the client SP, for example for a response to a request for capability information is should below in Figure Figure 2.29.

The other activity diagrams are essentially the same for each of the activities.

2.7.4 Requirements

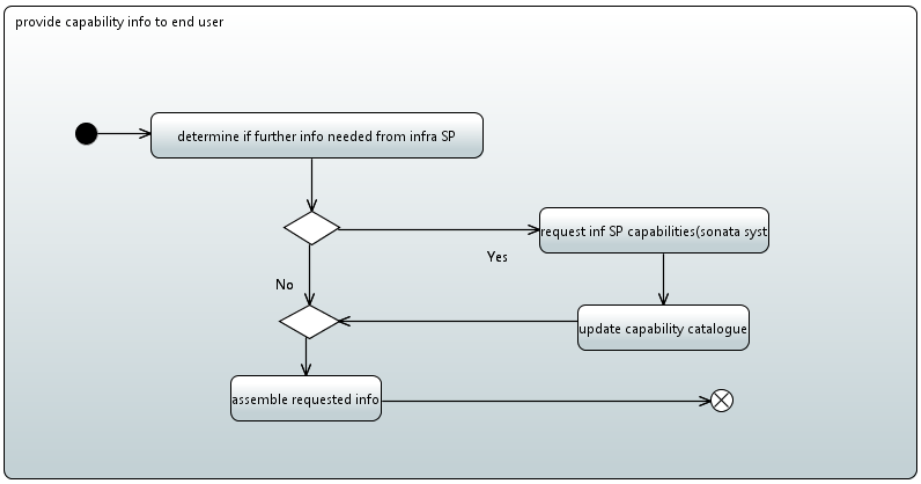

Figure 2.26: Activity diagram for the client SP response to a capability information request

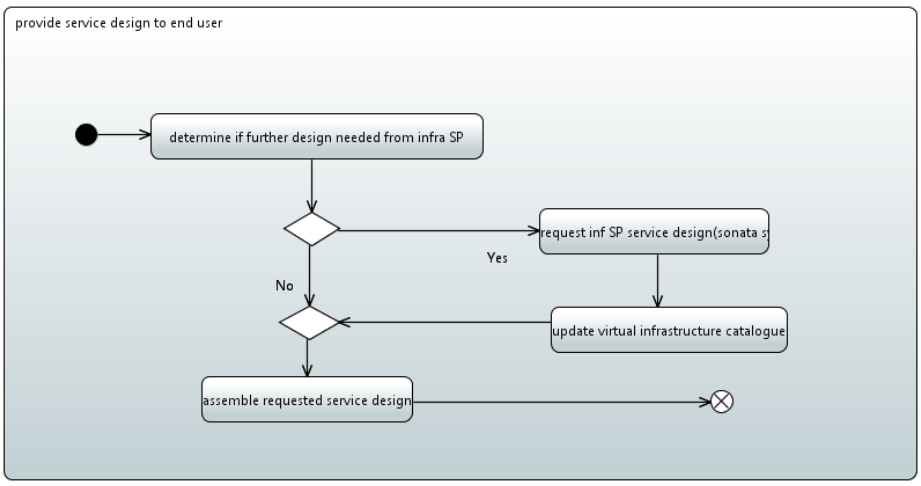

Figure 2.27: Activity diagram for the client SP response to a new service design request

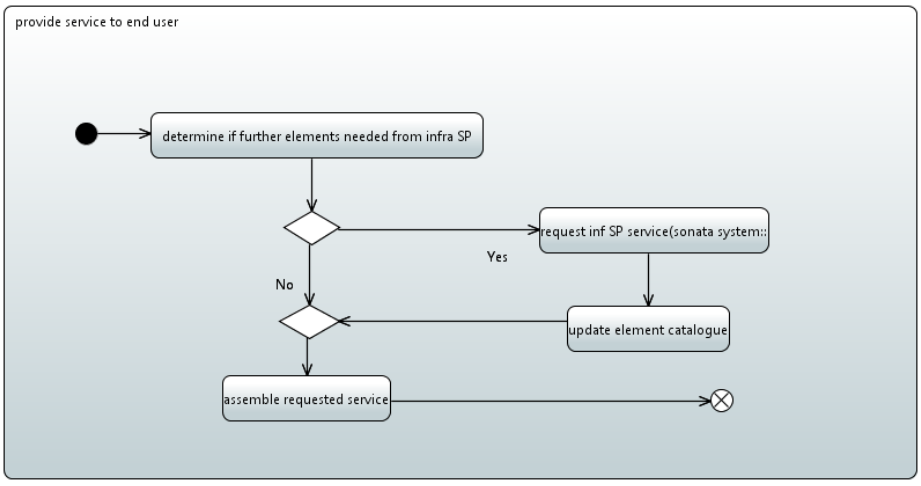

Figure 2.28: Activity diagram for the client SP response to a new service design request

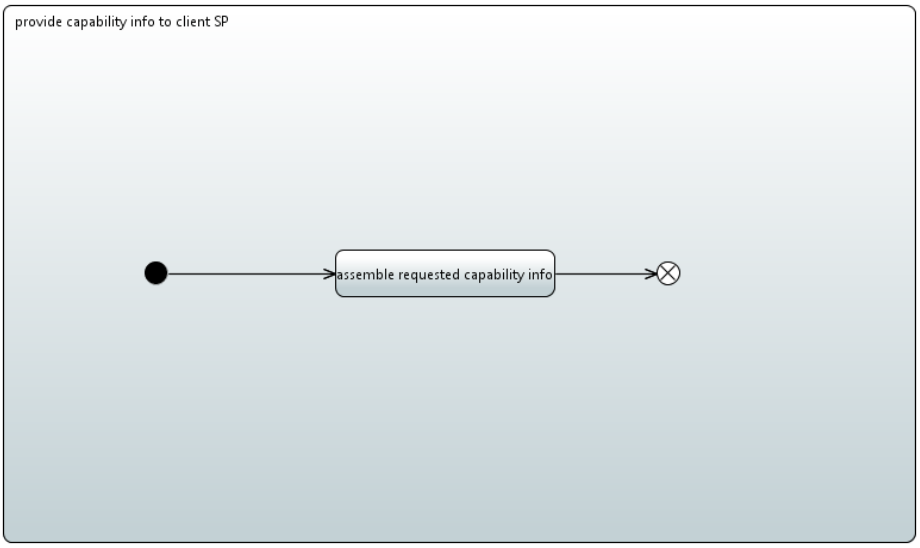

Figure 2.29: Activity diagram for the hosting SP

Requirement Name	NFVI Northbound API
Description	The SONATA system must be able to support an API which exposes the NFV Infrastructure as a service
KPIs	Successful management of the lifecycle of NFVI service elements (VMs and VNs) Sufficient abstract service parameters to meet the deployment constraints of the NS/VNFs of all other use cases.
Category	Mandatory
Involved Use Case	ALL

Requirement Name	Southbound Plugin to use NFVI API
Description	The SONATA system must be able to support a southbound interface which can request elements of NFV Infrastructure as a service
KPIs	Successful activity of NFVI host service elements (VMs and VNs) with the constraints required by the NS/VNFs.
Category	Mandatory
Involved Use Case	ALL

Requirement Name	No unnecessary information duplication when NS/VNFs and the NFVI are operatored by different SPs
Description	Information held by an instance of a SONATA system must not create unnecessary duplication or other dependencies. In particular, a SONATA system offering NFV Instrastructure as a service must not hold specific information about the type of VNFs or network services into which the VNFs are composed. Similarly, a SONATA system of a client SP composing network services hosted on NFV Instrastructure controlled by another service provider must not hold implementation details of the infrastructure.
KPIs	Suiable abstracted set of NFVI service parameters which can meet the deployment constraints of all of NS/VNFs from all other use cases. Minimal and abstract interactions across the NFVI as a Service API during NFVI service element lifecycle.
Category	Mandatory
Involved Use Case	ALL

3 Limitations and challenges in current service platforms design and operation

3.1 Limitations in terms of programming models

Programmability is a term that is currently being used in many different ways and on various abstraction levels. On the lowest level, there is what could be called *switch* programmability. While OpenFlow [25] be regarded a standardized programming model for switches, it is really more of a protocol, whereas programmability would typically entail the notion of abstract programming interfaces (APIs). Recently, programming languages have been proposed that allow things like packet parsing and processing (in other words: "programming" of the switch forwarding behavior) in much more intuitive terms. The most prominent examples of such languages are Frenetic [10] and P4 [3]. While Frenetic's syntax is closer to python, P4 has a C-style language. Both come with a compiler that not only generates the proper rules in the switches, but also takes care of conflict resolutions, flow rule (de-)aggregation and race conditions in the face of multiple independent programming commands.

On the *networking* level, different approaches have been proposed to program the virtual topology setup and end-to-end treatment of flows or applications. Prominent examples are group-based or intent-based interfaces. In the latter case, applications are envisaged to be able to give rather declarative (instead of imperative) commands about what the networking behavior should look like rather than how exactly this will be implemented. An intent-based command could, for example, be "all finance staff need priority access to the SAP system". An interpreter would then translate "finance staff" into a set of ports, "SAP system" into a set of IP addresses and "priority access" into a bandwidth reservation scheme or DiffServ QoS policy. Virtual Tenant Network (VTN) [17] and comparable approaches also work on the same level of abstraction (namely the network level, not the switch level), but have a little different scope, which is defining the virtual topology for a tenant together with the associated policies.

On the *cloud* level, there are various languages (such as CIMI [9]) or widely used tools (such as OpenStack [39]) for "programming" the cloud management domain. Cross-platform tools such as Terraform [15] come with their own APIs to allow cloud services across multiple cloud instances, potentially with different client management systems, e.g. Amazon AWS [2] and OpenStack.

The major contribution of SONATA is expected to be on the *service* programmability level.

3.1.1 Existing approaches and their limitations

A well-known service level abstraction is known as the Application-Based Network Operations (ABNO) architecture for multi-technology network operation that is oriented to the application needs such as capacity/connectivity, resource reservation, network and function virtualization, and reliability. The approach is to organize and extend existing technologies in a toolbox for information gathering, path computation, topology abstraction, traffic engineering and provisioning/reserving of resources. The key element of the toolbox, namely the Path Computation Engine (PCE), is further extended to provide policy enforcement capabilities for ABNO.

The most active discussion in terms of service programming is currently around service function chaining, or, in more general terms, service function composition. The idea is to compose more complex services out of individual service functions, such as CG-NAT, intrusion-detection, firewalls, access control, en/decryption, video optimizers, TCP optimizers, policers, HTTP header enrichment, parental control, malware protection, etc. Essentially, this implies services to be "programmed" in terms of graphs. Nodes represent the service functions and edges represent the communication relationships and sequential ordering of the service functions such as to produce the overall service. Nodes and edges in the service graph can be enriched with annotations of various kinds. Nodes, i.e. service functions, for example, can have annotations pertaining to their resource requirements (number of CPU cores, amount of memory and storage), or to other properties such as availability requirements constraining their placement in the physical infrastructure. Likewise, edges can have annotations describing their requirements such as the SLA (necessary capacity, or the maximum delay, etc.) for inter-function communication. Also, resilience of communication links can be described in abstracted terms which could then be mapped by service execution environment to 1+1 configurations, for example.

One such graph-based service specification language is defined by ETSI NFV [21], where this is called the Network Function Forwarding Graph. Another, more enriched variant is defined in the EU project Unify [12], where, in addition to the actual service functions and their dependencies, also monitoring points can be specified. A specification model that goes further than pure service graphs is called service templates. Here, the idea is to specify *types* of service functions, i.e. a form of wildcards, in the service graph. A typical multi-tier web application, for example, might have a web server, a business logic and a database component. However, instead of specifying an exact database model in the service graph, the idea is to provide an abstract node representing any SQL database, for instance. The service execution system can then decide at runtime which specific SQL database to select for a specific instance of the service graph, i.e. an actual running service. Note, however, that such generalities may have implications for the business logic component: it is one thing for the service developer to specify a specific database node and ship a VM image with a pre-installed database together with the service description, it is a different thing to connect to an existing SQL database (the orchestrator is free to do so by virtue of the template) where the proper tables may still need to be set up.

Another extension of plain service graphs are scaling relationships. This captures two possible approaches. One is that a scaling relationship may specify how many service functions can be supported, e.g. by a single load balancer. For service instantiation, this means that if more service functions than that are needed, the existing load balancer instance has to be scaled out. Another approach is to use such scaling properties to express dependencies of sensitivity to load increase. For example, when the load increases and one service function needs to be scaled up by a factor of 2, an adjacent service function may need to be scaled out by a factor of 3, as it is more sensitive to the same load increase. Such experiences and dependencies can be specified as part of the service graph to help the orchestrator optimize resource utilization and service experience.

The existing service graph models still have limitations. There are many properties that would be very helpful for service deployment and orchestration which cannot be captured in current models. One example are security requirements. An orchestrator may choose to deploy a certain service function as a Docker [19] module or as a virtual machine. Docker does not offer the same level of isolation as a virtual machine, so security-wise, it does make a difference how exactly a service function is virtualized. However, the service developer does not want to care about details of specific virtualization technologies. He might be a real-time media expert (e.g. in case of video delivery services), so such virtualization details are something outside his domain knowledge and rightfully so. Still, he will know about the security requirements of his service, so a proper

service programming abstraction needs to capture them in a way that makes sense for the service developer and not for the orchestrator. This is just one example to motivate the purpose of proper abstraction: it provides a meaningful separation of concerns and bridges knowledge domains by means of compilers or mapping models.

3.1.2 SONATA's approach to overcome these limitations

SONATA will advance the state of the art of service programmability in two major respects. First, it will work on evolved service graph models that can capture much richer information, not only in the domain of isolation, but also in terms of DevOps. The goal is to provide - as part of the service specification - instructions for the service platform that enable it to perform service-specific monitoring, collect statistics as well as debugging and profiling information. Also, service developers shall be able to convey service-specific orchestration algorithms which will be executed and regulated by the overall orchestration control loop.

Second, SONATA will strive to provide better programming support for service functions that make use of multiple libraries, most of which may already be available in the service platform.

3.2 Limitations in service programming tools and IDEs

3.2.1 Introduction

The software defined network (SDN) approach has introduced a new way of network programmability. As a result, a variety of network programming support tools and high-level languages, like Frenetic [11] or Pyretic [40] showed up to support the development of network applications. Furthermore, network programmability allows using debugger-like tools or model checking approaches to verify the correct operation of an entire network. It also creates the need to have integrated development tools and a unified execution environment to create, run, and exchange network applications in a vendor-independent fashion. These topics are partly covered by the FP7 project NetIDE [13]. However, with the advent of NFV, additional programmability and thus complexity, is introduced and new requirements for network development support arise. This starts by the development of single network functions and their configuration. Most of the available tool support for these tasks comes from the cloud computing community providing tools to create virtual machine (VM) images and install software in them, like Vagrant [16], Ansible [18], Puppet [23] and Chef [44]. Nevertheless, none of these tools provides support to implement, test, and provision entire network services composed of a number of network functions controlled by a custom, potentially complex control logic. These limitations and the missing tool support have to be addressed in order to exploit the full potential of NFV, like reduced time-to-market and the use of DevOps development methodologies.

3.2.2 Existing service programming tools and IDEs

Service programming tools enhance the service platform and exist as separate executables or entities, outside of the platform. Service programming tools usually consist of the following functionalities:

- **Editing tools** provide support in the write-up of the code/programming language. Usually this involves forms of auto-completion and/or syntax coloring.
- **Debugging tools** help the programmer to find errors within the written service program. This might be further decomposed into the following:

  - **Proofing tools** enable to validate either syntactical or semantic properties of written service programs at programming time. This might involve parsing and/or model checking functionality.
  - **Runtime debugging tools** enable the programmer to perform fine-grained checks of the running service program in a controlled deployment environment. This usually involves the functionality to inspect state of the program or to execute parts of the program step by step.
  - **Post mortem debugging tools** enable the programmer to investigate traces after a service program failed, crashed, or gave erroneous behavior.

- **Profiling tools** enable the programmer to estimate resource and performance characteristics of a given service program.

- **Configuration Management tools** enable the programmer to define execution environments for eg. VNFs. Not only launching them but also ensuring the necessary dependencies are installed and up to date, modifying the state of their configuration at deployment time. Configuration Management is the epitome of automation, it forms the bedrock of achieving supportable, repeatable and consistent services.

- **Deployment tools** enable the programmer to deploy an app or VNF in an isolated environment (eg. as container or VM) onto the infrastructure. They can also be seen as packaging tools for a discrete VNF, not a complete service. They can be used together with a configuration management tool to further configure the deployed environment. From a service programming perspective, these tools help to program the service as a linked network of several VNFs, each running in its own deployment environment. It can help to reliably reproduce working applications across the infrastructure as well as find bugs and do high quality Quality Assurance.

Below analysis lists some examples which are representative for the main tools categorization described in the beginning of this section. We only give a very brief description to sketch its use in the context of service programming.

- **Ansible/Chef/Puppet** We list these 3 tools together because they all serve the same purpose from a service programming point of view: Configuration Management. They were all built with that very goal in mind: to make it much easier to configure and maintain dozens, hundreds, or even thousands of servers. The Table 3.1 lists their main characteristics. Architecture related it is worth noticing that Chef/Puppet are client/server based, meaning that a specific server needs to be running and agents on every node that need to be configures. Ansible is push based, and does not require agents running on the nodes, the configuration script is pushed via SSH.

Table 3.1: Configuration Management tools

CM tool	Dependency	Configuration scripting	Architecture
Chef	Ruby	Ruby	Client/Server
Puppet	Ruby	Ruby/Puppet-specific	Client/Server
Ansible	Python	YAML	Push

- **Vagrant** falls into the deployment tool category, with a limited scope: a way to use Oracle's Virtual Box or VMWare Fusion on a developer workstation for the purpose of creating disposable and shareable development environments. So at its core, Vagrant is a simple wrapper around Virtualbox/VMware. Vagrant is a configuration management tool for VMs, one feature is eg. taking a snapshot of a configured VM environment and then deploying that snapshot elsewhere (via only an configuration file, not the complete VM image).

- **Docker** [19] / **Flockport** [7] are both wrapping container technology (LXC linux containers). They are deployment tools, open source virtualization platforms that virtualize individual apps, rather than the entire operating system (like VMs). LXC is a container technology which gives you lightweight Linux containers and Docker is a single application virtualization engine based on Linux containers. With LXC you get a container that behaves like a lightweight virtual machine and you can use it as such. With Docker you get a container that can run only one application at a time. While they might appear similar they are 2 different things. We'll not go further into details here, but in the context of this section it is highlighted that, like VMs, these container technologies offer a way to package and deploy applications in a cloud environment, making them fit as part of a complete service and configurable by configuration management tools.

- **Cbench** [51] / **OFlops** [41] are some benchmark tools available, specifically designed for the profiling of OpenFlow switches and controllers: OFlops(OpenFLow Operations Per Second) is a standalone controller that benchmarks various aspects of an OpenFlow switch. Oflops implements a modular framework for adding and running implementation-agnostic tests to quantify an switch's performance. Cbench (controller benchmarker) is a program for testing OpenFlow controllers by generating packet-in events for new flows. Cbench emulates a bunch of switches which connect to a controller, send packet-in messages, and watch for flow-mods to get pushed down. These tools are only covering a very specific subset of all the related aspects of service deployment.

- **Click** [22] is a software framework for building NF's in a modular way by combining elements, which perform simple operations on network packets, into a graph-like structure defining the packet's processing flows and thus implementing a more advanced NF. This modularity offers unique possibilities for (de)composing VNF's, which can be difficult to achieve using other frameworks for VNF programming. The very fine-grained modularity which can be achieved by implementing VNF's using these discrete elements, the easy re-use and development of new elements, are key aspects making the Click framework very suitable as a tool to deploy VNF's in a decomposed way. A higher-level, Click-specific, language is used to connect elements into a graph-like structure, packets then flow along the graph's edges. This structure can be edited with any standard text editor, no specific proofing tool is available. As alternative to a textr editor, a GUI tool called Clicky, is available to build VNFs by combining and linking discrete Click elements. Click elements can be combined into typical NFs like: Ip Router, NAT, DNS proxy, BRAS ... Debugging can be done using Handlers: Each element can easily install any number of handlers, which are access points for user interaction. This lightweight mechanism is most appropriate for modifications local to an element, such as changing a maximum queue length, adding/deleting routes in table, exporting statistics and other element information. A lightweight VM called ClickOS, enables the deployment of Click scripts as an isolated VM environment.

- **ONOS** [32] is marketed as a network operating system, and can be seen as an extended Openflow controller. As an SDN controller, it has a southbound interface (it could be Open-

Flow, or it could be a legacy protocol like NetConf) connecting to the network's data plane (switches, routers). ONOS acts as an overlay on the total network, with features as high availability and scalability. ONOS provides innovative northbound abstractions that simplify the creation, deployment, and operation of configuration, management and control applications. The global network view and application intent framework (link policies between hosts) are two examples. Network policies are inserted as a high-level app, which can be seen as a plugin into the controlling network operating system. ONOS translates the high-level network policies to low-level datapath configurations (eg. via Openflow). The developer can program network policies via a java-based app, so any java editor (eg. IntelliJ) can be used as editing and proofing tool. The app is then packaged, sent and plugged into the ONOS server.

Above description shows that basic functionality to deploy services is available, but spread over a plethora of tools, an ecosystem which is in full development.

for example, the tools available in the service programming context are usually coming from either the domain of cloud platforms or from a more networking/SDN perspective. Cloud platforms are usually subdivided into three subcategories: Infrastructure as a Service (Iaas), Platform as a Service (PaaS), or Application as a Service (AaaS). The first category provides virtualized infrastructure as a service, usually virtual machines and a certain form of interconnection network (usually LAN-type connectivity, or layer 3 and above). PaaS provides a software infrastructure on top of IaaS in order to support application developers to write services (e.g., pre-installing database, or web service framework such as Ruby on Rails). In AaaS, end-user applications themselves are provided, such as Customer Relationship Management solutions. The strict boundaries between IaaS, PaaS and AaaS, however have been challenged through the rise of hybrid services (e.g., services which provide containers rather than VMs).

Programming frameworks stemming from the networking field also split into a the category of platforms focusing on programming data plane functionalities, i.e., high-speed processing of data packets (cf. Click) vs. programming platforms for the control of networks (cf. ONOS). The latter either relates to traditional distributed control functionality in support of routing and signaling within networks, or with respect to logically centralized Software-Defined Networking platforms. These frameworks allow a service developer to program data-plane logic, which can later be deployed by the service platform as part of a service. Therefore they can be useful tools in a wider context of service development. However, each of these frameworks has its own programming model, so combining them inside a single SDK would be a cumbersome task. This illustrates the difficulty of combining existing tools into an easy to use service development environment. Without drowning into too many specific details of existing frameworks.

In order to support or implement any existing tools or frameworks, a modular architecture approach would be recommended. That way, a specific editor or debugging tool could be plugged in or packaged into the SDK when needed. As the SONATA project evolves, it will become more clear which of the above described tools will become more relevant and how monitoring data can be made available. But it is clear that profiling tools will support the SONATA ambition of extending QoS and SLA management for a network service. Runtime debugging tools on the other hand, will help to implement innovative features related to service-level debugging, but will require additional functional functionalities such as for example test packet generation or more specialized monitoring.

Ideally, a service platform should incorporate as much as possible of the above mentioned features and tools.

With respect to the state of the art regarding service programming tools, there are two main departing points: i) community (open-source) software projects or ii) research projects. A more exhaustive list of existing projects will be provided at later stages in the project. In this context, we

will focus on a first assessment of each of the big categories. A service platform should incorporate as much as possible of the above mentioned features. In the next sections we elaborate on the existing service platforms, how they implement these tool functionalities and what is missing.

3.2.2.1 Cloud Service Orchestration community projects

Cloud Service Orchestration projects are usually clones of the IaaS and PaaS platforms of the big cloud providers such as Google, Microsoft and Amazon. In the IaaS space, OpenStack, Cloudstack and Kubernetes are typical examples, Eucalyptus provides an open source software for building private and hybrid clouds compatible with Amazon Web Services (AWS) APIs. In the PaaS space, Juju Charms, OpenTOSCA, OpenStack+Heat and Cloudify are well-established projects. Cloud service orchestration software is usually not tuned for addressing *bump in the wire* packet-handling related functionalities, but focuses on endpoint functionality such as web servers or databases. Most of them have a GUI which usually translates to structured document formats such as XML, JSON or YAML conform the TOSCA data model. Syntactic proofing functionality is usually provided by YAML schema validators. Semantic proofing functionality is in general not available, although Juju Charms provides some static testing functionality. Runtime debugging functionality is largely unexisting, or limited to the investigation of logfiles potentially supported by a more extended monitoring platform (e.g. OpenStack Ceilometer). For post mortem failure analysis, again the only functionality platforms provide is based on log file inspection. Profiling functionality is very limited, although Juju charms and Kubernetes provide some profiling support. The OpenMANO project is not focused on the general cloud service orchestration problems, but in the specific application of cloud orchestration to provide NFV-based network services. Therefore, it provides a basic cloud orchestration module, *openvim*, that takes direct advantage of EPA (Enhanced Platform Awareness) and supports smart placement and optimized infrastructure network configuration for NFV scenarios. Currently, openMANO does not provide SDK tools apart from a REST API and CLI, but evolving an OpenMANO SDK based on them within the timeframe of SONATA seems a feasible task.

Later on in the project, SONATA will need to adopt a data model, describing the services and then package them for deployment. Current platforms have their way of describing service graphs, and recipies for deploying the included VNFs and linking them together. However, further steps need to be taken on how to implement and deploy also service specific logic such as scaling or placement algorithms. Also the feedback and analysis of monitoring data or logfiles has room for improvement. This will enable more advanced debugging feature, moving the service deployment from a trial-and-error method to a more controlled and predictable performance.

Let's take a closer look at some of the mentioned community projects:

- **JuJu** [49] is an open source service modelling tool, part of Ubuntu's cloud portfolio. Puppet and Chef are great tools for configuring servers and keeping them consistent across a network. Juju works a layer above that by focusing on the service the application delivers, regardless of the machine on which it runs. Services are described using JuJu Charms. These can be written in any language that can run on Ubuntu, e.g. Python, Ruby, PHP, or even a simple bash script. Some charms even reuse existing Chef or Ansible cookbooks. A catalog or store is available where Charms for many applications can be downloaded. A web GUI allows for easy configuring and editing of the Charms.

- **OpenTOSCA** [46] provides an open source ecosystem for the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) standard which was developed at the University of Stuttgart. As the name indicates, TOSCA has two parts (a) the topology, describing the components and relations of the application, and (b) the orchestration, explicitly

modelling management aspects of the application. The OpenTOSCA ecosystem consists out of three parts (i) OpenTOSCA Container, a TOSCA runtime environment (ii) Winery, a graphical modelling TOSCA tool and (iii) Vinothek, a self-service portal for the applications available in the container.

- **OpenStack + Heat**, Heat [33] 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. A Heat template describes the infrastructure for a cloud application in a text file that is readable and writable by humans, and can be checked into version control, diffed, etc'. Heat primarily manages infrastructure, but the templates integrate well with software configuration management tools such as Puppet and Chef. The Heat team is working on providing even better integration between infrastructure and software. A service is deployed using a Heat Orchestration Template (HOT) [34]. HOT templates are written as structured YAML text files.

- **Cloudify** [47] is an open source TOSCA-based cloud orchestration software platform written in Python. Built on a YAML DSL (Domain Specific Language) configuration files called "blueprints" define the application's configurations, services and their tier dependencies. These blueprint files describe the execution plans for the lifecycle of the application for installing, starting, terminating, orchestrating and monitoring the application stack. Cloudify uses the blueprint as input that describes the deployment plan and is responsible for executing it on the cloud environments. The blueprint also employs cloud driver configuration files as well, to describe machines and their images for the chosen cloud, making it possible to manage the infrastructure as code. For each component it describes the location of your binaries, installation and monitoring configurations. By creating an abstraction layer that isolates the code from the underlying infrastructure, Cloudify is able to support many cloud environments (Openstack, AWS, VMWare, Docker ...). It comes with a GUI to edit blueprints and monitor deployed services.

The above described platforms offer ways of packaging and automating deployment of services. However, real ways of implementing service dynamics are still lacking. Any logic that automatically scales running services, taking into account things like service specific state migration between nodes, is not available or still immature. The service programmer has no control over the orchestration, without knowing detailed specification about the infrastructure, so high-level orchestration algorithms are not definable by the service programmer. Debugging or development tools are also not yet available on a service level. Editors for generating service graphs and a specific service graph syntax, together with deployment recipes, are the most worked-out tools in the described projects.

3.2.2.2 Existing research projects

Service platforms and more specific flexible service programmability is also an aspect in several other research projects (eg. FP7 or 5G-PPP EU research projects), each having its own focus. Below we give an overview of the related research projects we know of. To our best knowledge, we list for each project any features or tools related to service programming or development environments . Limitations or gaps will be highlighted. This analysis allows us to identify the points where SONATA will bring added value.

3.2.2.2.1 UNIFY The FP7 UNIFY project [12] which is currently ongoing has two work packages related to an SDK for NFV-based service programming. WP3 mainly focuses on the data and

programming models and workflows for service programming, while WP4 focuses on the DevOps concept as well as the monitoring platform in support of it. A couple of tools are currently developed. The ESCAPE orchestration emulator [45] is basically a Mininet-enabled environment which has been extended with several abstraction layers and functionalities in support of the UNIFY programming framework. These are implemented in the Python POX SDN control framework, and are able to orchestrate containers in Mininet with Click NFs, OpenStack domains as well as x86 nodes optimized for packet processing. The SDK support is limited to standard Python development functionality at the service level, or Click development tools (e.g., Clicky, or Click script DSL syntax coloring) for NF development. In WP4 three categories of tools are currently being developed: monitoring functions, observability points and SDN measurement and verification tools. In the first category tools are available for packet loss monitoring, OpenFlow rule verification (AutoTPG), control channel inspection (Watchpointer), or verification tools of Network Function Forwarding Graph model instances. Observability point tools include a query language for retrieval of monitoring data, or a multi-component debugging tool extending EMACS with a DSL for rapid network debugging (Epoxide). The last category involves tools for loss monitoring of aggregated flows.

Limitations In short, in the UNIFY framework, a service is defined by a YANG-based model of a forwarding graph. This file describes all VNFs, the links between them and needed monitoring points. The orchestrator then needs to map this service graph onto the available infrastructure.UNIFY focuses less on the service layer part of the platform and more on the orchestrator. The mapping algorithm in the orchestrator is however under control of the service platform operator. UNIFY provides no means for a service developer to add any specific orchestration related logic. The placement of VNFs can eg. be influenced by providing extra specifications, such as link delay, in the service graph, which pose extra boundary constraints for the orchestration algorithm. this requires of course that the orchestration algorithm takes these extra parameters into account, which means extra complexity for the already hard-to-compute mapping algorithm. Some services might need to take very specific parameters into account, which would require cumbersome adaptations in the orchestrator, making it a less modular architecure.

There is currently no complete development environment available in UNIFY which combines all of the described tools. Epoxide [37] is service debugging tool but only covers a small subset of service related programming and debugging. It uses a specific Click-inspired language to start basic monitoring tools (eg. ping, iperf, ovs-dpctl, traceroute, tcpdump ...) and filters their output for analysis. The parameters to be monitored and analysis features are currently limited. Many more extensions need to be implemented. Its modular implementation in Emacs might provide a steep learning curve but offers a lightweight environment. The debug actions are not automated and need to be manually started. Any automated troubleshooting is not yet available, although it could be possible using the current framework. The integrated prototype is currently heavily under development, and is mainly an orchestration tool called ESCAPE. The tools described above are available in UNIFY as algorithms or freestanding tools, but they are not yet integrated within the ESCAPE platform. No IDE is available to extend the usability of the UNIFY platform and develop new services. As mentioned, it is not possible to incorporate service specific placement or scaling logic into any orchestration functionality. Any service driven dynamics, such as elastic scalability, is to be incorporated inside a control VNF itself, running as a dedicated VNF inside the service on the infrastructure. From this control VNF, the orchestrator is requested to change the running service. Any data monitored by the control VNF, stays inside the infrastructure layer. Therefore it is more difficult in UNIFY to export any monitoring data for use by profiling or analysis tools. This is a different view compared to SONATA, where these custom tailored entities stay running

on the service platform level, where they are easier available for export /import from an SDK environment.

3.2.2.2.2 NetIDE The NetIDE project [13] which hast started early 2014 and is ongoing until the end of 2016 tries to create an integrated development environment for portable network applications. The main goal of the project is enabling vendor-independent SDN application execution and support for the end-to-end development lifecycle of network applications. To do so, three major components are developed. The first one is called *network engine* and is responsible to execute controller independent network applications. The second component is an application repository which provides predefined network applications which can be reused by developers. The last component is an Eclipse based integrated development environment which interacts with the network engine for full DevOps support [6] [5]. They plan to provide extended tool support for SDN application development, including garbage collector, resource management, model checking and profiling tools. The current IDE release integrates Mininet as lightweight test environment. The general setup of these three components is very similar to SONATA's approach having on *service platform*, *catalogues*, and *SDK*.

Limitations The focus of NetIDE is providing an integrated toolchain to support SDN application development. It does not provide solutions for network function development nor for the implementation of complex network services. There is no interaction with orchestration components since the SDN controller applications are executed as part of a controller framework, like OpenDaylight. NetIDE may provide useful inputs to the networking part of SONATA.

3.2.2.2.3 T-Nova T-NOVA [38] aims to introduce and promote a novel Marketplace for NFV, introducing new business cases and considerably expanding market opportunities by attracting new entrants to the networking market. Within T-NOVA, the function development, exchange and trading processes take place in a multi-actor ecosystem, allowing software Function Developers to export their products into the global networking marketplace. Activities and outcomes of the T-NOVA project are relevant to SONATA mainly with respect to the service orchestration platform. But T-NOVA GUI for Service Providers and its interface with the T-NOVA orchestrator and the T-NOVA Function Store can be seen as a relevant starting point for SONATA to build the SDK for service composition.

Limitations T-NOVA focuses on a marketplace for VNFs, similar to the catalogues in the SONATA framework. However, the process how a developer should edit and add services or VNFs to this marketplace is less specified. The project lacks any specific development tools for debugging or profiling a VNF. Also it is not handled how a service can contain any orchestration specific functions such related to placement.

3.2.3 SONATA's approach to overcome these limitations

As explained above, existing programming tools for network services are either developed in an SDN or NFV perspective. In an SDN context, they provide higher-level programming languages or abstractions to program the network, install link policies and checking their correctness. The NFV inspired tools provide means to ship, install and configure VM images or containers. SONATA's ambition is to combine those two, providing a toolbox that addresses all aspects of a complete service deployment. Where a service is composed out of several network functions, combined with a custom, potentially complex logic. Therefore, the SONATA development toolbox will need to

support the development and verification of custom-tailored control entities, such as placement and scaling logic. These are included in the service package and modify the orchestration behavior for a particular service. They can be programmed and added by the service developer. Previously described existing tools, platforms or projects do not provide anything related to structured profiling methods, targeted at the performance of the complete service and not a small part of it. It is left entirely up to the programmer to specify the HW (CPU, memory) or networking (delay, bandwidth) requirements for the VNFs to be deployed in the service. This includes benchmarking the VNFs against different workloads. The SONATA platform aims to alleviate this task by providing access to monitoring data inside the SDK. Furthermore, the SDK will provide the necessary tools aimed at the devolpment and debugging of the high-level service dynamics, as described previously in this section.

These are innovations that is not yet found in existing service platforms and therefore no particular SDK tools are yet available. To extend the usability and effectiveness of the SONATA service platform, service programming toolbox or the SDK will provide following functionalities and easy the work of the service developers by exposing programming models and abstractions:

- To support the developer in reusing existing NFVs, services, or decision logics, the SDK foresees **catalogue** access. it can interface with, query, or browse existing catalogues, e.g., a private one of the developer, those of the target service platform, or even public marketplaces (like developed by T-Nova or Murano), possibly handling licensing and payment issues. (This is akin to dynamically installing libraries in scripting languages from public repositories).

- Provide a **package** that can be sent to the service platform for deployment and describes and encapsulates all the service components in a service model: VNFs, network links ,SLA, scaling and placement logic, certificates for authorization (eg. monitor data access, catalog access, service user download/edit rights).

- Provide a set of **editors** for the various service components. Specifically the service composition needs to be defined and stored, using an agreed model and syntax. Next to this, also the service specific logic which is implemented using custom-tailored control entities, needs to be edited and packaged in an agreed format so it is deployable by the service platform. They define eg. custom placement onto the available infrastructure, scaling rules with specific decompositions or state migration rules.

- Support functions exist in the SDK, in particular, **monitoring analysis tools** and **profiling tools** (e.g., debuggers or emulators for service compositions). These tools will work on the level of composed services, enabling a developer to understand whether a service bottleneck is, e.g., computation-bound or communication-bound, or performing what-if analysis (e.g., would it help to start up another VM?), debugging placement and scaling logics, analysis monitoring results, and others. These tools closely interact with DevOps support functions in the service platform, which make monitoring/analysis data available. Best practices how to use these tools will develop over time in the project, using experience gained from developing our own proof of concepts for the use cases. In SONATA, the service programming tools are part of an SDK. This SDK is a framework, running separately from the service platform itself, and communicating with the actual service platform via a single dedicated interface or gateway. Through this gateway, the developer can access service catalogues, add service packages or access monitor data, as authenticated by the service platform operator. Moreover, SONATA will provide an SDK which is deployable as a single environment. The environment will encompass a complete toolbox capable of composing, editing, debugging and verifying a service.

3.3 Limitations in terms of current service platforms and orchestration frameworks

Network function virtualization introduces new ways to deploy and manage network services. Its main focus has, so far, been on reducing cost compared to traditional network services achieved by on-demand resource allocation, like in IaaS clouds. As a result, existing NFV platforms are fairly simple and lack support for service developers and service operators who want to manage their services with these systems. Another downside of existing platforms is their inflexible orchestration functionalities caused by orchestration approaches that try to manage different kinds of network services with one fixed orchestration system.

3.3.1 Existing approaches and their limitations

There is a wide variety of existing orchestration and management support tools. Starting with basic cloud managers, like OpenStack Nova or OpenNebula, which provide only basic functionalities to deploy single virtualized compute resources. On these compute resources network functions can be executed but the management tools lack the notion of composed services and forwarding chains. As a result, these tools can only be considered as infrastructure management tools controlled by a more specialized orchestration system to build an NFV service platform.

Other cloud management solutions, like OpenStack Heat, Terraform, and Ubuntu's JuJu, enable service developers to specify services composed of several interconnected virtual machines which can then be deployed on existing cloud infrastructures. However, these tools do not focus on the special needs of NFV, like network forwarding rules. None of them allows the service developer to influence the placement of the deployed services in distributed cloud environments.The most notable approaches and projects which directly focus on NFV orchestration and service platform development can be divided into tree categories. The first consists of the research projects such as FP7 T-NOVA and FP7 UNIFY; the second category consist of open source solutions like Open-MANO [48], OpenStack with its various projects, OpenBaton [8]; the last category is a set of common trends from a first generation of developing commercial NFVO solutions that are seeing initial marketing in vendors' NFV transition portfolio. These approaches and projects and their limitations are discussed in the following.

3.3.1.1 T-NOVA

T-NOVA (Network Function as a Service over Virtualized infrastructures) [38] is an FP7 project that implements a management/orchestration platform for the automated provisioning, configuration, monitoring and optimization of Network Functions-as-a-Service (NFaaS) over virtualized Network/IT infrastructures. It also exploits and extends Software Defined Networking platforms for managing the network infrastructure.

Moreover, in order to facilitate the involvement of diverse actors in the NFV scene and attract new market entrants, T-NOVA includes a "NFV Marketplace" on top of the orchestration layer, in which network services and network functions by several developers can be published and brokered/traded. Via the Marketplace, the Service Provider can compose services by using those Functions, and its customers can browse and select the services and virtual appliances, as well as negotiate the associated SLAs and be charged under various billing models. Activities and outcomes of the T-NOVA project are relevant to SONATA mainly with respect to the service orchestration platform. Moreover, T-NOVA GUI for Service Providers and its interface with the T-NOVA orchestrator and the T-NOVA Function Store can be seen as a relevant starting point for SONATA to build the SDK for service composition.

Similarly to SONATA, the T-NOVA NFV orchestration platform, TeNOR, is the core component of the T-NOVA architecture framework. Its primary role is to manage, deploy and monitor all network services and virtual network functions lifecycle. It includes the implementation of an algorithm to map services jointly into the network and cloud infrastructure, choosing the optimal resources to deploy the service and scales and migrate according to the monitoring data received.

Another point of relevance is the Infrastructure Virtualisation and Management (IVM) layer of T-NOVA, which provides the infrastructure substrate to be managed by the Orchestrator. The T-NOVA IVM could essentially provide an indicative reference infrastructure to be also managed by the SONATA Orchestrator platform. In line with current trends in NFV infrastructure, the T-NOVA IVM is based on an integration of OpenStack and ODL for the realization of the NFVI-PoPs, in which OpenStack Cloud Controller and ODL comprise the VIM for the management of the infrastructure. The exposed interfaces are based on open APIs and are implemented with REST API allowing interactions with Orchestration frameworks.

3.3.1.1.1 Limitations DevOps is not supported in T-NOVA orchestration platform, so the level of automation is limited. In other words, T-NOVA services are "on-boarded" once and in a static manner. T-NOVA offers the Service Provider a graphical interface to select (purchase) available VNFs and to combine them in order to build the forwarding graph (VNFFG) to be included in the descriptor (NSD). From that point on, iterations are not foreseen; the Network Service advertised is considered mature and final. If new developments are to be accommodated, the whole procedure needs to be re-started, i.e. the NS needs to be deleted, all instances removed and re-created from the beginning with the new VNFs.

Another limitation of the T-NOVA system is the lack of VNF validation and certification before on-boarding a VNF to the system. In T-NOVA, verification, Function Provider (FP) certification and validation as well as (maybe) workload characterization are processes to take place off-line before on-boarding and they are not automated. T-NOVA also lacks a formal service verification process. Only specific semantic validations are done based on the NSD and VNFD: for example, a NS monitoring parameter must be composed by VNF provided monitoring parameters (these have to be described on the VNF Descriptor).

Finally, NS intelligence and programmability is also limited in T-NOVA. In specific, the T-NOVA Orchestrator does not provide any capabilities for network service programmability, apart from a northbound API which allows basic service operations (deployment, start/stop, monitoring etc.) In other words, it is not possible for the customer to apply arbitrary behavioral logic for his/her network service, which would dictate e.g. rescaling based on custom rules, VNF migration and re-placement, service reconfiguration etc. Also, it is not possible for a customer to dynamically compose a service their selves, but only to choose from a catalogue of pre-composed NSs.

3.3.1.2 Unify

The FP7 UNIFY project considers the entire network, from home networks up to data centre, as a "unified production environment" supporting virtualization,programmability and automation and guarantee a high level of agility for network operations and for deploying new, secure and quality services, seamlessly instantiatable across the entire infrastructure. UNIFY develops an automated, dynamic service creation platform. A service abstraction model and a global orchestrator, with novel optimization algorithms, enables the automatic optimal placement of networking, computing and storage components across the infrastructure. New management technologies based on experience from DCs, called Service Provider DevOps, are developed and integrated into the orchestration architecture to cope with the dynamicity of services. The applicability of a universal node based on commodity hardware will be evaluated in order to support both network functions and traditional

data centre workloads, with an investigation of the need of hardware acceleration.UNIFY provides a range of tools addressing the above needs. The integrated prototype is currently being further extended, and is mainly an orchestration tool called *ESCAPE*. This tool is a POX-based orchestrator environment which has been extended to address the orchestration and devops challenges as indicated above. Currently there are control adaptors available for Mininet, an OpenStack domain, and for an x86-based node supporting optimized packet-handling for a range of NFs exploiting DPDK. The UNIFY service programming model enables for service-level placement on virtualized infrastructure and is well-supported by ESCAPE. Service-driven dynamics is also possible through a Cf-Or interface using the same programming model.

3.3.1.2.1 Limitations The ESCAPE tool is currently heavily under development and does not yet provide fully featured service platform. The VNF catalog which is currently implemented is heavily focused on Click Modular Router NFs, NFs based on other technologies such as VMs or Docker are not well-integrated. Although the UNIFY architecture and programming model supports KPI formulation and enforcement through decomposition rules, this is currently not supported within the ESCAPE environment, as there is no automated process available to translate KPI's into required configuration of required resources, NFs and monitoring functionality. This might be implemented in the future, but will be limited to particular infrastructure domains.Verification algorithms in order to validate Virtual Network Function-Forwarding Graphs are available in UNIFY, but they are not integrated within the ESCAPE platform. In addition, the NF and service life-cycle management is rather limited and currently service platform-driven placement is only optimized at deployment time or at scheduled re-optimization time instances rather than enabling continuous re-optimization.

3.3.1.3 OpenMANO

OpenMANO [48] is an open source project initiated by Telefonica that aims to provide a practical implementation of the reference architecture for NFV management and orchestration proposed by ETSI NFV ISG. The project's first release was in early 2015 and it is currently under active development. The OpenMANO framework consists of three major components: *openvim*, *openmano*, and *openmano-gui* all available under Apache 2.0 license. The first component is not directly related to the orchestration task and focuses on building a virtual infrastructure manager (VIM) that is optimized for VNF high and predictable performance. Although it is comparable to other VIMs, like OpenStack it includes control over SDN with plugins (floodlight, OpenDaylight) aiming for high performance dataplane connectivity. It offers a CLI tool and a northbound API used by the orchestration component *openmano* to allocate resources from the underlying infrastructure, this includes the creation, deletion and management of images, flavours, instances and networks. Openvim provides a lightweight design that does not requirer additional agents to be installed on the managed compute nodes.

The orchestration component itself can either be controlled by a web-based interface (*openmano-gui*) or by a command line interface (CLI) through its northbound API. OpenMANO's orchestrator is able to manage entire service chains that are called *network scenarios* and correspond to ETSI NFV's *network services* at once. These *network scenarios* consist of several interconnected VNFs and are specified by the service developer with easy YAML/JSON descriptors. It offers a basic life-cycle of VNF or scenarios (define/start/stop/undefine). This goes beyond what simple cloud management solutions, like OpenStack, can handle. The easy to install framework includes both, catalogues for predefined VNFs and entire network services including support to express EPA (Enhanced Platform Awareness) requirements.

3.3.1.3.1 Limitations OpenMANO offers limited flexibility and does not provide a plugin concept to easily exchange components and functionalities. Currently it does not provide automated scaling functionality and there is no way to influence placement decisions by a service developer. This results in limited control over deployed services and adaptations of the orchestration functionality would be necessary. The project does not provide interfaces for the integration of service development tools, like feedback channels for detailed monitoring data to be accessed by service developers. This limits the current system functionalities to orchestration and management tasks, but for providing an end-to-end solution that supports the entire development cycle of network services a covering or extension would be needed.

3.3.1.4 OpenBaton

OpenBaton [8] was released in October 2015 under Apache 2.0 license and was previously part of the www.opensdncore.org OpenSDNCore project. It was started by Frauenhofer FOKUS and aims to provide a NFVO framework that is fully compatible with the ETSI NFV ISG specifications. It uses OpenStack as underling VIM and provides a plugin mechanism to support additional VIM types. The same mechanism is provided to integrate either the default virtual network function manager (VNFM) or a VNFM provided by a third party. These VNFMs can communicate with OpenBaton by using a message queue system or a RESTful JSON interface.OpenBaton uses the ETSI NFV description format to specify VNFs and network services consisting of multiple VNFs. It can mange the end-to-end deployment of these services across multiple data center instances (NFV PoPs) and provides basic slicing support for multi-tenant environments. The system is implemented in Java and provides a web-based dashboard and a command line interface (CLI) for user interactions.

3.3.1.4.1 Limitations The current version does still focus on providing the basic network service provisioning and management functionalities and there is no support for autoscaling or fault management at the moment. OpenBaton does not offer build-in VNF monitoring functionalities to directly support the service optimization process. It does not offer an SDK component for integrated support of a DevOps-like service development methodology but it offers an SDK for VNFM development which is based on a Java library. The NFVO component does not provide functionalities to directly influence the placement decisions for a particular service which is a clear limitation compared to SONATA's service platform.

3.3.1.5 OpenStack

OpenStack is an open-source cloud computing platform for public and private clouds. It is built out of a series of interrelated projects that deliver a cloud infrastructure solution. It is one of the leading cloud platforms governments and big-name companies around the world use, including major carriers, such as Comcast and AT&T. OpenStack is managed by the OpenStack Foundation, a non-profit, vendor-neutral, multi-stakeholder effort to help build and promote the OpenStack platformwhich oversees both development and community-building around the project. While OpenStack in 2010 was made up of two companies, the OpenStack Foundation in 2015 numbers well over 100 members. OpenStack's APIs are defector a standard for IaaS APIs for both private and public cloud and it is the most conmanly IaaS used by both enterprises and telcos. The OpenStack's projects that are most relevant for SONATA are:

- **Tacker** - OpenStack's Tacker project aims on developing a general-purpose orchestrator and VNF manager for OpenStack that is compatible to the MANO design of ETSI reference

architecture. The goal is to support the end-to-end orchestration and management of network services composed of several VNFs deployed on multiple OpenStack instances. Tacker uses TOSCA's NFV profile schema to describe VNFs and services. As default, it uses the OpenStack Heat component to interact with the underlying VIMs by translating parts of the TOSCA definition to the Heat specific template language. The project provides a management driver framework that can be used to inject initial configurations to VNFs and to update configurations during operation. This framework provides an extendable design so that vendors can include their own management and configuration tools.

- **Heat** - OpenStack's Heat project is the main project in the OpenStack Orchestration program. It implements an engine that supports launching of multiple composite cloud applications based on templates in the form of text files that can be treated like code. The Heat template describes the infrastructure for a cloud application in a text file that is readable and writable by humans, and can be checked into version control, diffed, etc'. Infrastructure resources that can be described include: servers, floating ips, volumes, security groups, users, etc. Heat also provides an autoscaling service.

- **Murano** - OpenStack's Murano project is an application catalog, enabling application developers and cloud administrators to publish various cloud-ready applications in a browsable categorized catalog. The key goal of the Murano project is to provide UI and API which allows to compose and deploy composite environments on the Application abstraction level and then manage their lifecycle.

- **Mistral** - OpenStack's Mistral project is a workflow service - any process can be described as a set of tasks and task relations , once this description is upload to Mistral, Mistral takes care of the state management, correct execution order, parallelism, synchronization and high availability. Mistral also provides flexible task scheduling so processes can run according to a specified schedule (instead of running it immediately).

- **Congress** - OpenStack's Congress project provides policy as a service across any collection of cloud services in order to offer governance and compliance for dynamic infrastructures. Congress aims to provide an extensible open-source framework for governance and regulatory compliance across any cloud services (e.g. application, network, compute and storage) within a dynamic infrastructure. It is a cloud service whose sole responsibility is policy enforcement.

3.3.1.5.1 Limitations The various projects are currently under development and their components are not yet finalized. Most of the projects address the needs of the VIM layer, while SONATA is focused on the higher levels built on top of it. The projects currently support only OpenStack as underlying infrastructure (and not any VIM).

3.3.1.6 Developing NFVO Commercial Solutions

A first generation of NFV orchestrators (NFVOs) are appearing in the NFV transition marketing portfolios of major vendors and telecom solution providers. Based on ETSI reference architecture, these initial NFVO solutions will include both resource and service orchestration functionality. There is a clear proprietary ecosystem trend in the majority of these initial solutions that often link to larger NFV end-to-end portfolios with, for example, supporting VNF catalogs and VNFM components. However, several solutions are advertised as vendor agnostic in the context of their interactions with underlying VNFMs and VIMs as they follow a common architecture, and also recognize standards such as YANG service/resource models and TOSCA deployment templates.

3.3.1.6.1 Limitations These first-generation commercial solution designs provide orchestration capabilities but other components to create a full service platform environment are either missing or proprietary for the vendor's commercial strategy. Flexibility in terms of an open plug-in approach is a key aspect missing, which makes it hard to adapt to new services and bespoke requirements for adopters. The extent of built-in DevOps compatibility that recognizes the extended value chain of software networks in not clear, in particular the need for a specialized workflow between platform operators and service developers of different entities, and integrated within supporting development tools.

3.3.2 SONATA's approach to overcome these limitations

SONATA's service platform will provide additional flexibility on two levels, to overcome the highlighted limitations of existing orchestration solutions.

The first level addresses the flexibility of the orchestrator as such, in which the microservice-based architecture of the T-NOVA Orchestrator will be enhanced by a pluggable architecture design consisting of several loosely coupled components that can be easily replaced by the platform operator to change the behavior of the orchestration system. The second level of flexibility addresses the need of service developers to customize management functionalities for their particular services. This issue will be solved by allowing service developers to bundle custom-tailored control entities together with their services which are then used to modify the behavior of the orchestration system for a particular service. A typical example for this are custom scaling and placement logics. This feature goes beyond the customization capabilities of the presented orchestration frameworks and gives service developers a novel degree of freedom compared to custom applications which interact with the northbound interfaces of the existing orchestration solutions, like OpenStack or OpenMANO.

In addition to these innovations in the orchestration layer of the system, SONATA will also tackle the limitations of service developer support in existing solutions. This will be done by providing an SDK which interacts with SONATA's service platform and supports the development process with direct feedback from running services, bringing the DevOps methodology to NFV. In T-NOVA, Function Providers had to develop their VNFs from scratch, since no reuse of other and existing VNFs was explicitly put in their flow of development. SONATA will improve this by developing the SDK which provides these flows, integrates monitoring and accelerates deployment, thus effectively supporting DevOps.

3.4 Limitations in terms of current service monitoring solutions

The concept of Network Function Virtualization is likely to become the future of telecommunications industry, as it allows network operators and service providers to gain greater and more flexible control on their network elements and services, respectively. Furthermore, NFV provides enhanced management capabilities by offering flexibility and agility. Alongside the emerging concept of NFV comes Software Defined Networking technology that promises to simplify network management tasks by separating the control plane (controller) from the data plane (switches) as well as Cloud Computing paradigm, offering virtualized resources for service deployment and execution.

One of the main issues that have to be tackled in this new era of NFV/SDN is network and service monitoring, requiring accurate and timely statistics not only on network but also on computation and storage resources involved at different aggregation levels. This necessity stems from the fact that virtualized network functions as well as other supporting functions are deployed, in the majority of the cases, as Virtual Machines or containers (e.g. Docker) in virtualized (cloud) environments.

In this perspective, traditional monitoring tools, frameworks and platforms (e.g. SNMP, NetFlow, Zabbix, etc) fall short in satisfying monitoring requirements arising from all involved stakeholders and including data from several technologies.

3.4.1 Existing approaches and their limitations

This section provides insight about numerous state-of-the-art service monitoring solutions relevant to the one to be developed and implemented in the SONATA service platform.

For each solution, advantages and disadvantages are discussed.

3.4.1.1 OpenFlow-enabled Switch Monitoring

OpenFlow has emerged as the de facto standard for communication between the controller and the switches in an SDN-based infrastructure. Apart from offering flow control and communication interfaces, OpenFlow provides a flow level and aggregated statistics collection mechanism from the data plane, exposing respective, high level interfaces. Thus, network administrators and service providers can use this high level API to monitor network status. In order to keep the switch design simple, the OpenFlow monitoring collection is following the pull-based paradigm, where the controller queries the switches periodically or upon request.

In particular, in an NFV/SDN network, the controller is a single point of contact to retrieve monitoring information and thus its continous monitoring along with the monitoring of the underlying infrastructure consisting of several switches is mandatory.

This section provides information on the state-of-the-art monitoring tools utilized for collecting network monitoring data from OpenFlow-enabled networks (including controller and switches) along with their limitations.

3.4.1.2 CBench

Controller Benchmarker (CBench) is a program, developed by Stanford University [50], for testing OpenFlow controllers by generating packet-in events for new flows. CBench emulates a bunch of switches which connect to a controller, send packet-in messages, and watch for flow-mods to get pushed down.

As its description implies, CBench is simply a program [51], emulating switches just taking advantage of packet-in functionality provided by OpenFlow and thus it could only be considered as a plugin to the broader scope of SONATA service monitoring solution.

3.4.1.3 Lattice

The Lattice Monitoring Framework, developed by University College London [24], provides functionality to add powerful and flexible monitoring facilities to systems. Lattice has a minimal runtime footprint and is not intrusive, so as not to adversely affect the performance of the system itself or any running applications. The monitoring can be built up of various components provided by the framework, so creating a bespoke monitoring sub-system.

The framework provides data souces, data consumers, and a control strategy. In a large distributed system there may be hundreds or thousands of measurement probes which can generate data. It would not be effective to have all of these probes sending data all of the time, so a mechanism is needed that controls and manages the relevant probes.

Lattice framework consists of the following elements:

- **Producers and Consumers**

The monitoring system itself is designed around the concept of producers and consumers. That is there are producers of monitoring data, which collect data from probes in the system, and there are consumers of monitoring data, which read the monitoring data. The producers and the consumers are connected via a network which can distribute the measurements collected. The collection of the data and the distribution of data are dealt with by different elements of the monitoring system so that it is possible to change the distribution framework without changing all the producers and consumers. For example, the distribution framework can change over time, say from IP multicast, to an event bus, or a publish / subscribe framework. This should not affect too many other parts of the system.

- **Data Sources and Probes**

In many systems probes are used to collect data for system management. In this regard, Lattice will follow suit. However, to increase the power and flexibility of the monitoring we introduce the concept of a data source. A data source represents an interaction and control point within the system that encapsulates one or more probes. A probe sends a well defined set of attributes and values to the consumers. This can be done by transmitting the data out at a predefined interval, or transmitting when some change has occured.

The measurement data itself is sent via a distribution framework. These measurements are encoded to be a small as possible in order to maximise the network utilization. Consequently, the measurement meta-data is not transmitted each time, but is kept separately in an information model. This information model can be updated at key points in the lifecycle of a probe and can be accessed as required by consumers.

- **Distribution Framework**

In order to distribute the measurements collected by the monitoring system, it is necessary to use a mechanism that fits well into a distributed architecture such as the management overlay. We need a mechanism that allows for multiple submitters and multiple receivers of data without having vast numbers of network connections. For example, having many TCP connections from each producer to all of the consumers of the data for that producer would create a combinatorial explosion of connections. Solutions to this include IP multicast, Event Service Bus, or publish/subscribe mechanism. In each of these, a producer of data only needs to send one copy of a measurement onto the network, and each of the consumers will be able to collect the same packet of data concurrently from the network.

3.4.1.4 OpenNetMon

OpenNetMon is an open-source software implementation to monitor per-flow metrics (throughput, delay, packet loss) in OpenFlow networks, developed by Delft University of Technology [4].

OpenNetMon is a POX OpenFlow controller module monitoring per-flow QoS metrics to enable fine-grained traffic engineering. By polling flow source and destination switches at an adaptive rate, OpenNetMon obtains accurate results while minimizing the network and switch cpu overhead, by determining the right polling algorithm and frequency.

The per-flow throughput and packet loss is derived from the queried flow counters (inherently supported by OpenFlow), while delay is measured by injecting probe packets directly into switch data planes, traveling the same paths (nodes, links, buffers) and thus determining a realistic end-to-end delay for each flow.

OpenNetMon consists of two components implemented in POX OpenFlow controller. The forwarding component is responsible for the reservation and installation of paths, while the monitoring

component is responsible for the actual monitoring. Both components rely on the POX Discovery module to learn network topology and updates.

Although OpenNetMon provide a valuable tool for monitoring network metrics, it cannot be considered as an integrated solution, since it lacks both processing and storing capabilities for monitoring data which is a mandatory requirement for SONATA monitoring platform. However, the polling algorithms implemented in OpenNetMon might be used by SONATA to reduce monitoring traffic overhead in the network.

3.4.1.5 PayLess

PayLess is a network monitoring framework for SDN, developed by University of Waterloo [31], offering a number of advantages towards developing network management applications on top of the SDN controller. In particular:

- Provides an abstract view of the network and a uniform way to request statistics about the resources.
- it is developed as a collection of pluggable components, whose interactions are abstracted by well-defined interfaces.

Thus, PayLess provides a flexible RESTful API for flow statistics collection at different aggregation levels. It uses an adaptive statistics collection algorithm that delivers highly accurate information in real-time, without incurring significant network overhead. In this respect, a service can express high level primitives in its own context to be monitored and get the collected data from the PayLess data store.

Although relevant to SONATA service monitoring platform, PayLess deals only with network monitoring metrics and it is unlikely that extensions of this framework can include monitoring data from other sources. However, its modular design (built on the concept of pluggable components) and the availability of RESTful APIs could be considered during the SONATA service monitoring platform design phase.

3.4.2 Cloud Computing Resources Monitoring

In principle, monitoring is a fundamental part of every cloud computing infrastructure in order to expose monitoring services and data, either to the network operators, monitoring their physical/virtual resources and thus ensure their operational health and availability, or to the service providers, monitoring their allocated resources to evaluate performance and validate SLAs.

Up to now, state-of-the-art monitoring solutions on cloud environments mainly target homogeneous, single-entity administered cloud infrastructures. But, the trend is moving towards a federated service cloud architecture consisting of heterogeneous approaches in terms of monitoring tools. Given that SONATA cloud computing environments will be mostly based on OpenStack, this section provides information on monitoring tools commonly used for this purpose.

3.4.2.1 OpenStack Ceilometer

Ceilometer [35] aims to become the infrastructure to collect measurements within OpenStack so that no two agents would need to be written to collect the same data. Its primary targets are monitoring and metering, but the framework should be easily expandable to collect for other needs. To that effect, Ceilometer should be able to share collected data with a variety of consumers.

Ceilometer aims to deliver a unique point of contact for billing systems to acquire all counters they need to establish customer billing, across all current and future OpenStack components. The delivery of counters must be traceable and auditable. The counters must be easily extensible to support new projects, and agents doing data collections should be independent of the overall system.

Taking into consideration that most of the SONATA partners are using OpenStack as their cloud computing environment, Ceilometer could be considered as a reasonable choice for adoption in SONATA. However, Ceilometer is not in the maturity level that will allow SONATA to easily integrate it in its service monitoring platform [36].

3.4.2.2 Nagios

Nagios is the de facto industry standard for monitoring IT infrastructures that empowers organizations to identify and resolve IT infrastructure problems before they affect critical business processes. Nagios offers more than 3,000 plugins, spanning from system metrics, network protocols, applications, databases, services and servers. Moreover, Nagios provides means for reporting and alerting. It is based on an open source software license under the terms of the GNU General Public License version 2, as published by the Free Software Foundation. A detailed list of available plugins can be found in [28]. Although Nagios was originally designed to run under Linux, it shall work under most other machines.

By using plugins and add-ons, Nagios offers the capability to monitor host resources and processes (CPU load, memory usage, disk utilization, etc) as well as network services (SMTP, POP3, HTTP, etc.) on Windows and Linux/Unix machines. As a high-level view example, monitoring Windows-based host resources requires installation of the NSClient++ add-on on the host machine and check_nt plugin on the Nagios monitoring server. Similarly, Linux/Unix-based host resources can be monitored by installing the NRPE add-on, allowing to execute plugins on remote Linux/Unix hosts.

Unlike many other monitoring systems, Nagios does not include any internal mechanisms for monitoring hosts and services on the network. Instead, it relies on external programs (called plugins) to perform the measurements. Plugins are compiled executables or scripts (Perl scripts, shell scripts, etc.) that can be run from a command line to check the status or a host or service, while the results are used by Nagios to determine the current status of hosts and services on the network.The advantage of this approach is that one is free to monitor anything given that the process can be automated. On the other hand, the disadvantage relies on the fact that Nagios has absolutely no idea about what it is monitored; it just tracks changes in the state of the monitored resources or services.

Nagios supports add-ons that allow the environment to monitor a large infrastructure. More specifically, it provides two different options for setting up a distributed monitoring environment:

- In case of relatively large installations that require off-loading checks of the monitoring server to other machines, Nagios provides an add-on that faces the problem of scalability and complexity of distributed Nagios setups, following a master/slave configuration approach. In this respect, the Nagios server hands out jobs to slave nodes, while the master server contains all the configuration and check definitions [29].

- A different approach is followed in [27] and [26], where several Nagios servers that monitor a portion of the entire infrastructure are set-up, managed by a central dashboard that allows checking quickly the status of resources and services belonging to different portions of the infrastructure from a single server. In contrast to the previous distribution approach, within these add-ons the configuration is handled on the distributed (child) servers.

3.4.3 Docker containers Monitoring

As Docker is used for larger deployments it becomes more important to get visibility into the status and health of docker environments. The Docker client inherently supports a few basic functions for monitoring what containers are running, retrieving basic statistics about behavior and performance, and generating a running view of container resource consumption akin to the "top" command in Linux. As Docker's own APIs for monitoring has started to become mature, so has the latest wave of container-monitoring technologies that leverages them and provide valuable solutions in this respect. The following tools are most representative for monitoring Docker containers.

3.4.3.1 cAdvisor

cAdvisor [14] is a utility that itself is crated in a Docker container and generates quick and useful information on the basic behaviors of running containers. Thus, cAdvisor is a useful tool that is trivially easy to setup, it saves time from performing ssh into the server to look at resource consumption and also produces graphs. In addition, the pressure gauges provide a quick overview of when a cluster needs additional resources. Furthermore, unlike other tools, cAdvisor is open source and apart from allocating some processing resources there is no additional cost of running cAdvisor. However, it has it limitations. It can only monitor one docker host and hence in the case of a multi-node deployment, the stats will be disjoint and spread though out the cluster.

3.4.3.2 Scout

Scout [42] addresses several of the limitations present in cAdvisor. Scout is a hosted monitoring service which can aggregate metrics from many hosts and containers and present the data over longer time-scales. It can also create alerts based on those metrics. Another advantage of using Scout over cAdvisor is that it has a large set of plugins which can pull in other data about several deployments in addition to docker information. This makes Scout a monitoring system that caters for various resources in a system.

One drawback of Scout is that it does not present detailed information about individual containers on each host, like cAdvisor can. This is problematic, if heterogeneous containers are running on the same server. Another drawback of Scout is that it is not open-source and comes at a cost.

3.4.3.3 Sensu

Scout provides centralized monitoring and alerting, however it is a hosted services that can get expensive for large deployments. If one needs a self-hosted, centralized metrics service, the Sensu open source monitoring framework [43] may be considered as viable solution. Unfortunately Sensu does not have any docker support out of the box. However, using the plugin system one can configure support for both container metrics as well as status checks. In addition Sensu is able to aggregate the values of several hosts in one place and raise alerts over those checks, although alerting functionality is not as advanced as DataDog or Scout.

However, the big drawback of Sensu is the difficulty of deployment that it is preventive for adoption in SONATA.

3.4.4 SONATA's approach to overcome these limitations

SONATA service monitoring system must address a number of issues. In particular:

- It must seamlessly integrate and homogenize data coming from different sources, tools and environments. Thus, one of the primary objectives is to capitalize on existing monitoring

solutions and develop the appropriate adapters that will resolve any platform-specific issues, offering a unified approach.

- Furthermore, the proposed SONATA platform must also provide means for the proper operations on these data (such as aggregation, filtering, etc) as well as both real-time and historical data. In this perspective, the monitoring solution to be developed and implemented must provide mechanisms for retrieving such data in an easy way.
- Moreover, visualization tools must be in place to facilitate user needs in a friendly manner.

As a conclusion, SONATA consortium will evaluate the applicability of existing state-of-the-art service monitoring solutions and develop additional components or functionality to fulfil architecture and use case driven requirements.

4 SONATA Requirements

The requirements of the SONATA framework will pave the way for the design and development of the SONATA architecture. This section describes the complete and full specifications of the requirements for the SONATA Architecture identified so far. Additional requirements , if any, will be taken into account during the SONATA architecture iteration process. These requirements are derived from 6 diverse use cases, detailed in Section 2, hence making SONATA relevant for a wide range of ICT applications and services in perspective of 5G networks. The requirements are divided in to three main groups, including Business, Functional and Non-Functional requirements. The Functional requirements are further classified in to Service Programming, SDK, Service Platform and Service Monitoring requirements.

Table 4.1 presents a complete list of all the requirements identified for each use case for the SONATA framework. As one can observe, several of the listed requirements, derived from different use cases, are similar in scope and purpose and hence can be consolidated into a single requirement. Therefore, an exhaustive iteration for refining and consolidating the requirements was performed. The number indicated in the Table 4.1 for each requirement refers to the consolidated requirement number in the subsequent subsections. The rest of this section describes the requirements derived after the requirement consolidation process.

Table 4.1: Mapping SONATA requirements to use cases

No.	Requirement	IoT	vCDN	Ind Net	vEPC	PSA	SCHProv
	Business Requirements						
1	Catalogue Specs Mappings	4.1.1	4.1.1	4.1.1	4.1.1	4.1.1	4.1.1
2	Instances Mappings	4.1.2	4.1.2	4.1.2	4.1.2	4.1.2	4.1.2
3	Usage of Service Platform	4.1.3	4.1.3	4.1.3	4.1.3	4.1.3	4.1.3
4	Audit Service Chain Changes	4.1.4	4.1.4	4.1.4	4.1.4	4.1.4	4.1.4
5	Isolated/Reserved Vs Sharing Services	4.1.5	4.1.5	4.1.5	4.1.5	4.1.5	4.1.5
6	Definition of Policies of Service for User	4.1.6	4.1.6	4.1.6	4.1.6	4.1.6	4.1.6
7	Monitoring Service Usage	4.1.7	4.1.7	4.1.7		4.1.7	
8	Service Activation Management Operations	4.1.8	4.1.8	4.1.8	4.1.8	4.1.8	4.1.8
9	Service Management Operations - Configuration	4.1.9	4.1.9	4.1.9	4.1.9	4.1.9	4.1.9

No.	Requirement	IoT	vCDN	Ind Net	vEPC	PSA	SCHProv
10	Service Infrastructure Management Operations	4.1.10	4.1.10	4.1.10	4.1.10	4.1.10	4.1.10
11	Legal Compliant	4.1.11	4.1.11	4.1.11	4.1.11	4.1.11	4.1.11
12	Multiple IoT Vendors	4.1.12					
13	Multiple IoT Tenants	4.1.13					
14	Multi-Tenancy				4.1.13		
15	Support Different Modes of Management/Control				4.1.14		
	Functional Requirements – Service Programming						
16	VNF Catalogue	4.2.1.1	4.2.1.1	4.2.1.1	4.2.1.1	4.2.1.1	4.2.1.1
17	VNF Placement	4.2.1.7	4.2.1.7	4.2.1.7	4.2.1.7	4.2.1.7	4.2.1.7
18	Service Chaining (SFC)	4.2.1.2	4.2.1.2	4.2.1.2	4.2.1.2	4.2.1.2	4.2.1.2
19	VNF Scaling		4.2.1.3		4.2.1.3		
20	Inegration with Existing VNFs / Components			4.2.1.4			
21	Support for Service Templates/Cardinalities			4.2.1.5			
22	Inter-VNF QoS constraints			4.2.1.6			
23	Placement Constraints for VNFs			4.2.1.7			
24	Isolation constraints for VNF			4.2.1.8			
25	Security VNF Availability					4.2.1.9	
26	Personalized VNF					4.2.1.10	
27	Manage Update of Components				4.2.1.11		
	Functional Requirements – SDK						
28	SDK edition		4.2.2.1				
29	SONATA Reliability		4.2.2.2				
30	SONATA DevOps		4.2.2.3				
31	On-Demand Runtime Control		4.2.2.4				
32	Security Simulation Tools					4.2.2.5	
33	VNF Deployment	4.2.1.7					
	Functional Requirements – Service Platform						
34	Support for Integration with existing VNFs/Components			4.2.3.1			

No.	Requirement	IoT	vCDN	Ind Net	vEPC	PSA	SCHProv
35	Service Chaining Support Across WANs			4.2.3.2			
36	Manual Service Function Placement			4.2.3.3			
37	Capability Discovery in Service Platform			4.2.3.4			
38	SONATA Multitenancy		4.2.3.5				
39	SONATA DevOps		4.2.3.6				
40	Resource Infrastructure Mapping		4.2.3.7				
41	Application Deployment		4.2.3.8				
42	Ongoing Services Scale Up\Down		4.2.3.9				
43	Multi NFVI Orchestration		4.2.3.10				
44	Analytics plugin		4.2.3.11				
45	NS management northbound interface		4.2.3.12				
46	Distributed NFVI	4.2.3.13	4.2.3.13	4.2.3.13	4.2.3.13	4.2.3.13	4.2.3.13
47	Open Interfaces Toward NFV	4.2.3.14	4.2.3.14	4.2.3.14	4.2.3.14	4.2.3.14	4.2.3.14
48	Legacy Support	4.2.3.15	4.2.3.15	4.2.3.15	4.2.3.15	4.2.3.15	4.2.3.15
49	VNF Catalogue	4.2.1.1					
50	VNF Resource Report	4.2.3.16					
51	Authorization	4.2.3.17					
52	Traffic Simulator	4.2.3.18					
53	VNF Integration with Service	4.2.3.19					
54	SDK	4.2.2.1					
55	VNF Placement				4.2.1.7		
56	NFVI Northbound API	4.2.3.12	4.2.3.12	4.2.3.12	4.2.3.12	4.2.3.12	4.2.3.12
57	Southbound Plugin to use NFVI API	4.2.3.14	4.2.3.14	4.2.3.14	4.2.3.14	4.2.3.14	4.2.3.14
58	No Information Duplication for NS/VNFs with Heterogeneous SPs	4.2.3.20	4.2.3.20	4.2.3.20	4.2.3.20	4.2.3.20	4.2.3.20
	Functional Requirements – Service Monitoring						
59	Timely alarms for SLA violation			4.2.4.1			
60	Statistics Interface		4.2.4.2				
61	VNF Specific Monitoring		4.2.4.3		4.2.4.3		
62	VNF Real-time Monitoring					4.2.4.4	

No.	Requirement	IoT	vCDN	Ind Net	vEPC	PSA	SCHProv
63	VNF Reporting to BSS/OSS & Subscriber					4.2.4.5	
64	Quality of Service Monitoring					4.2.4.6	
65	VNF and Topology Validation					4.2.4.7	
66	VNF Scaling	4.2.1.4					
67	VNF SLA Monitor	4.2.4.8					
68	VNF Status Monitor	4.2.4.9					
	Non-Functional Requirements						
69	Service Platform Scalability			4.3.1			
70	Service Platform Customizability			4.3.2			
71	SONATA System Interface		4.3.3				
72	SONATA System Update		4.3.4				
73	SONATA Security		4.3.5				
74	SONATA platform high availability		4.3.6				
75	Authentication	4.3.7	4.3.7	4.3.7	4.3.7	4.3.7	4.3.7
76	Scalability	4.3.1	4.3.1		4.3.1	4.3.1	4.3.1
77	Availability					4.3.6	
78	Confidentiality					4.3.8	
79	Multi NFVI Orchestration				4.3.9		
80	Resilience / Availability					4.3.6	
81	Support Services with 5 nines SLA/Ctrl				4.3.10		
82	Support state-full services				4.3.11		
83	Integration with OSS				4.3.12		

4.1 SONATA's Business requirements

This subsection discusses SONATA's business requirements. Figure 4.6 provides an overview of how business requirements are mapped to SONATA architecture components.

4.1.1 Catalogues Specification Mappings

This requirement is present on the business interaction between the service platform provider (SONATA owner) and the service provider. SONATA platform should establish and manage the mappings between the CustomerFacingService and/or Products specifications (owned by the service

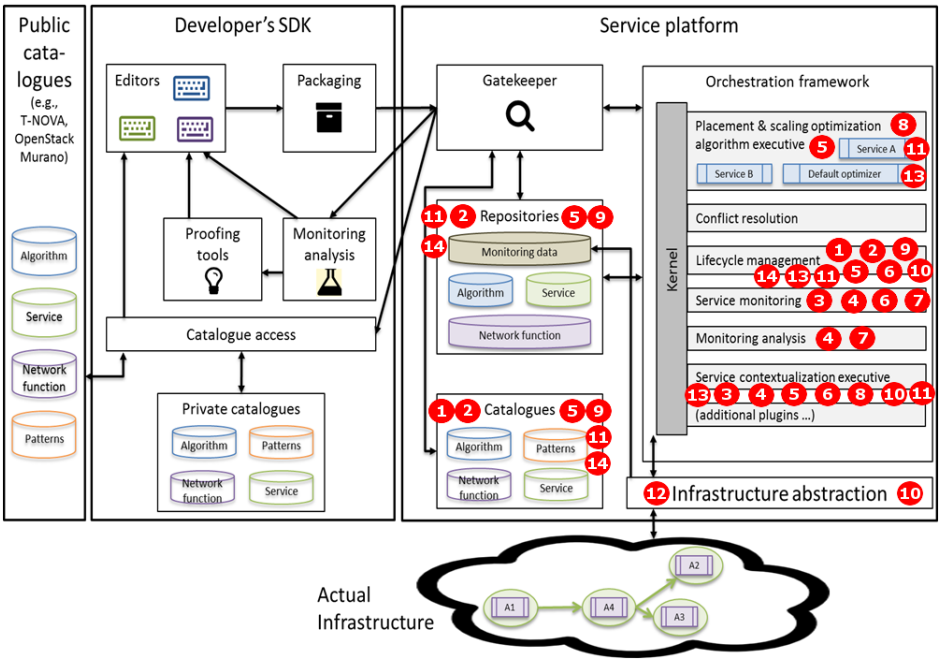

Figure 4.1: Mapping Business Requirements To Architecture

provider business area) and the Network Service specifications built over the SONATA service platform to know which upper specification entities (service/product) are affected by a concrete network service specification.

Possible components impacted by this requisite are Catalogues and Orchestration framework (lifecycle management plugin).

- KPI: Number of specification mappings.
- Category: **MANDATORY**.
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.1.2 Instances Mappings

This requirement is present on the business interaction between the service platform provider (SONATA owner) and the service provider. SONATA platform should establish the mappings between the CustomerFacingService or Products instances (owned by the service provider) and the Network Services instances built over the service platform to know which upper instance entities (service/product) are affected by a concrete network service instance.

Possible components impacted by this requisite are Catalogues, Repositories and Orchestration framework (lifecycle management plugin).

- KPI: Number of Instances mappings.
- Category: **MANDATORY**.
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.1.3 Usage of Service Platform

This requirement is present on the business interaction between the service platform provider (SONATA owner) and the service provider. SONATA platform should monitoring the usage of the components in the service platform (Network Services and/or VNF) by one service provided for the service provider to the end users for mediation purposes. In the use case of vCDN for example the number of vCaches used to support the contents in a specific region.

Possible components impacted by this requisite: Orchestration framework (service monitoring plugin, service contextualization executive).

- KPI: NetworkService usage events, VNF usage events.
- Category: HIGHLY DESIRABLE.
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.1.4 Audit Service Chain Changes

This requirement is present on the business interaction between the service platform provider (SONATA owner) and the service provider. SONATA platform should store details about the changes on the components used for a specific service chain for later auditing purposes.

Possible components impacted by this requisite:Orchestration framework (service monitoring and monitoring analysis plugins).

- KPI: service chain components status change events.
- Category: DESIRABLE.
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.1.5 Isolated or Reserved vs Sharing Services

This requirement is present on the business interaction between the service platform provider (SONATA owner) and the service user.SONATA platform should support the creation, depending on the type of user, of isolated/shared services to build different kind of products/services (e.g. basic/premium services). For the isolated network services also a reservation of resources could be allowed to guarantee the feasibility of these services always that is being required.

Possible components impacted by this requisite: Catalogues, Repositories and Orchestration framework (placement & scaling optimization algorithm executive, lifecycle management and service contextualization executive plugins).

- KPI: operations for reserve/free resources for Network Services.
- Category: DESIRABLE.
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.1.6 Definition of Policies of Service for User

This requirement is present on the business interaction between the service platform provider (SONATA owner) and the service user. SONATA platform must allow the definition of a set of policies for each network service to manage the use of the service by the end user (user profiling, SLA, access control).

Possible components impacted by this requisite: Orchestration framework (lifecycle management, service monitoring and service contextualization executive).

- KPI: Network service SLA(definition and thresholds), NetworkService ACLs, Network Service usage reaching thresholds events.
- Category: HIGHLY DESIRABLE.
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.1.7 Monitoring Service Usage

This requirement is present on the business interaction between the service platform provider (SONATA owner) and the service user. SONATA platform should support the monitoring of service usage by the user for mediation purposes.

Possible components impacted by this requisite: Orchestration framework (service monitoring and monitor analysis plugins).

- KPI: NetworkService Usage.
- Category: **MANDATORY**.
- Involved Use Case: IoT, vEPC, Industrial Networks, Personal Security Application.

4.1.8 Service Activation Management Operations

his requirement is present on the business interaction between the service platform provider (SONATA owner) and the service user. SONATA platform should offer operations for the management of the service from the business perspective needs (e.g. launch a service in a specific date/interval - weekend offers, service deactivation, non-payment management). A particular customer's product might need several interactions with the activation/deactivation processes because it is a composed service.

Possible components impacted by this requisite: Orchestration framework (placement & scaling optimization algorithm executive, service contextualization executive plugins).

- KPI: activation operations supported for network services.
- Category: HIGHLY DESIRABLE.
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.1.9 Service Management Operations - Configuration

This requirement is present on the business interaction between the service platform provider (SONATA owner) and the service user. SONATA platform should offer a management interface for configure the service with user/business requirements in a running environment (by example restart a device or change the capabilities of a component). An end user request generates a reconfiguration of the service and this must be done online (Customer Care or Self Service interaction)

Possible components impacted by this requisite: Catalogues, Repositories and Orchestration framework (lifecycle management plugin).

- KPI:configuration operations supported for network services,time to perform operations,number of operations allowed by second.
- Category: **MANDATORY**.
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.1.10 Service Infrastructure Management Operations

This requirement is present on the business interaction between the service platform provider (SONATA owner) and the virtual infrastructure provider(s). The SONATA platform should manage the partner definitions with infrastructure providers to use the virtual resources over several SLAs and include restrictions to the services.

Possible components impacted by this requisite: Orchestration framework (lifecycle management and service contextualization executive plugins) and Infrastructure abstraction.

- KPI: NFVI SLAs, events of reaching thresholds of virtual infrastructure, quantity of resources of each type,elasticity of each resource.
- Category: DESIRABLE.
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.1.11 Legal Compliance

There are several legal requirements like lawful interception or data retention that could affect to user traffic. Only when these requirements apply, SONATA framework must support mechanism to accomplish it. i.e. accountability of the service.

Possible components impacted by this requisite: Catalogues, Repositories and Orchestration Framework (Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins).

- KPI: Legal Compliance Measures (Only if needed)
- Category: ”’MANDATORY ”’
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers

4.1.12 Multiple IoT Sensor Vendors

Framework must support traffic from different IoT sensor vendors.

Possible components impacted by this requisite: Infrastructure abstraction.

- KPI: Receive traffic from at least two different sensor vendors and treat it in a uniform way.
- Category: DESIREABLE
- Involved Use Case: IoT

4.1.13 Multi-tenancy

The SONATA framework must support multi tenancy, i.e., the infrastructure must support multiple services. The framework should allow to dedicate some component to a particular tenant (hard isolation) and also allow to share some components (soft isolation) among different tenants. For example, SONATA should support multiple IoT or vEPC services.

Framework must support multi tenancy, i.e., The infrastructure must support multiple IoT services.

Possible components impacted by this requisite: Orchestration framework (Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins).

- KPI: Owner, SLA, At least two IoT services without any conflict
- Category: **MANDATORY**
- Involved Use Case: vEPC and IoT

4.1.14 Support Different Modes of Management/Control

EPC can be fully managed by the operator, i.e. EPC fully deployed and managed by the customer (e.g. a MVNO). Or Hybrid were components are managed by the operator (e.g. SGW) and others by the customer (e.g. HSS). Or fully managed by the customer.

Possible components impacted by this requisite: Catalogues, Repositories and Orchestration framework (Lifecycle management plugin).

- KPI: Owners, SLA by component, VNF specific monitoring metrics
- Category: **MANDATORY**
- Involved Use Case: vEPC

4.2 SONATA's Functional Requirements

4.2.1 Service programming requirements

This section discusses SONATA's service programming functional requirements. Figure 4.6 shows how these requirements are mapped to SONATA architecture components.

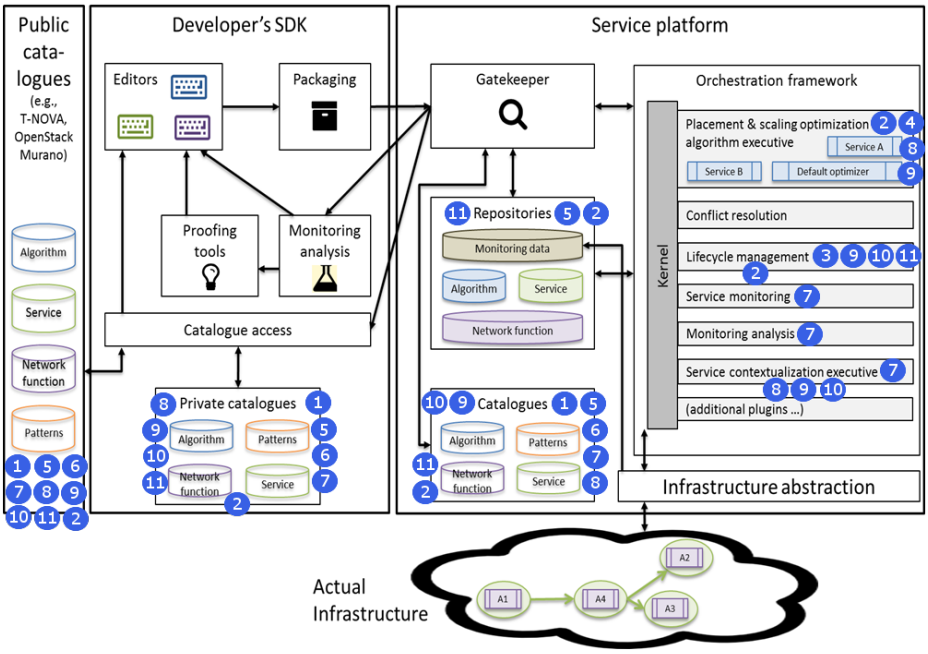

Figure 4.2: Mapping Service Programming Functional Requirements To Architecture

4.2.1.1 VNF Catalogue

The system shall offer a VNF catalogue, i.e., a list of the available VNFs, with information and detailed description of the services and VNFs available for the developer to use, indicating their specific performance. It should also enable the developer to configure VNFs and add new ones.

Possible components impacted by this requisite: Public, Private and Service Platform Catalogues.

- KPI: Number of VNFs available in the catalogue
- Category: **MANDATORY**
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.2.1.2 Manage Update of Components

Sequence/ strategy of update using DevOps. Sequence for validation and migration.

Possible components impacted by this requisite: Public, Private and Service Platform Catalogues, Repositories and Orchestration framework (Lifecycle management and Placement & scaling optimization algorithm executive plugins).

- KPI:Interruption time, availability
- Category: **MANDATORY**
- Involved Use Case: vEPC

4.2.1.3 Service Chaining (SFC)

Service chaining - the programmability framework shall allow the customer to interconnect VNFs in an arbitrary graph.

Possible components impacted by this requisite: Orchestration framework (Lifecycle management).

- KPI: Service mapping complexity and time
- Category: **MANDATORY**
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.2.1.4 VNF Scaling

The programming model must support scalability of a VNF depending on the user demand. The developer should describe the recipes for scaling his/her VNF in the VNF Descriptor that is to specify when and how the operator can scale the VNF, Scaling up/down, Scaling out/in while specifying different parameters (VM load, Bandwidth, latency). Moreover, it should allow to define configurable SLA levels for selected VNF, such as reduce or eliminate downtime when scaling a VNF.

Possible components impacted by this requisite: Orchestration framework (Placement & scaling optimization algorithm executive).

- KPI: Resource usage (CPU/RAM/Bandwidth), availability in the VNFD fields to define scale policies, support the interumption of VNF operations, downtime while processing the scaling.
- Category: **MANDATORY**
- Involved Use Case: vCDN, vEPC, IoT

4.2.1.5 Integration with Existing VNFs or Components

The programming model must allow compoments or VNFs of a new service to be integrated with existing services, VNFs or system components (such as sensors or actuators).

It should also allow network flow reconfiguration as well as reduce or eliminate downtime when re-configuring the service.

Possible components impacted by this requisite: Public, Private and Service Platform Catalogues, and Repositories.

- KPI: support for corresponding annotations (or primitives) in the service programming model/language, the amount of downtime required to reconfigure a running service graph.
- Category: **MANDATORY**
- Involved Use Case: Industrial Networks, IoT

4.2.1.6 Support for Service Templates or Cardinalities

The programming model must support service templates. In other words, it must support the inclusion of **types** of nodes, or at least the notion of cardinalities in inter-node relationships, e.g. in order to define an unspecified number of nodes.

Possible components impacted by this requisite: Public, Private and Service Platform Catalogues.

- KPI: support for corresponding annotations (or primitives) in the service programming model / language
- Category: HIGHLY DESIREABLE
- Involved Use Case: Industrial Networks

4.2.1.7 Inter-VNF QoS Constraints

The programming model must support end-to-end QoS properties for inter-VNF communication, such as delay, jitter, reliability (which could be mapped to multi-path transmission by the orchestrator, the developer does not care necessarily), oversubscription.

Possible components impacted by this requisite: Public, Private and Service Platform Catalogues, and Orchestration framework (service contextualization executive, service monitoring and monitoring analysis plugins).

- KPI: support for corresponding annotations (or primitives) in the service programming model / language
- Category: **MANDATORY**
- Involved Use Case: Industrial Networks

4.2.1.8 Placement Constraints for VNFs

In order to enable flexible orchestration policies, the programming model must support optimization for VNF placement. That is, the programming model must support to specify placement constraints for VNFs, e.g. disjoint placement of active and standby VNF on physically separate machines, pinning a VNF to a specific node or node type (e.g. turbine control must run on a turbine node). The hosting nodes must offer certain real-time capabilities or security isolation features, etc. In other words, the programmability framework shall allow the customer to deploy VNFs at arbitrary points into the network. Set where the components/ gateways will be placed on the operator network. For example, deploy a VNF as near as possible to a specific location or select were the VNF will be deployed.

Possible components impacted by this requisite: Public, Private and Service Platform Catalogues, and Orchestration framework (Placement & scaling optimization algorithm executive and service contextualization executive plugins).

- KPI: support for corresponding annotations (or primitives) in the service programming model / language, Deployment time, Cost. Specific metrics related to the service SLA/ end-user QoS
- Category: **MANDATORY**
- Involved Use Case: Industrial Networks, vEPC, IoT

4.2.1.9 Isolation Constraints for VNFs

The programming model must support isolation constraints for VNFs. This is in terms of performance, e.g. in order to guarantee min. capacity without being preempted by concurrent services. But it is also in terms of security, e.g. in order to restrict visibility of (virtual or real) infrastructure to a particular service, or to constrain a service to specific time windows (e.g. only between 10am and 11am, or expiry one hour after first use).

Possible components impacted by this requisite: Public, Private and Service Platform Catalogues, and Orchestration framework (Placement & scaling optimization algorithm executive, Lifecycle management and service contextualization executive plugins).

- KPI: support for corresponding annotations (or primitives) in the service programming model / language
- Category: **MANDATORY**
- Involved Use Case: Industrial Networks

4.2.1.10 Security VNF Availability

Security virtual network functions require specific capabilities that are not so common in generic VNF, like AntiDDoS or signature detection of IDS. These functionalities must be present to allow creating a valid use case. SONATA VNF Catalogue must include some Security VNFs in order to support this use case.

Possible components impacted by this requisite: Public, Private and Service Platform Catalogues, and Orchestration framework (Lifecycle management and service contextualization executive plugins).

- KPI: Enough number of Security VNFs
- Category: **MANDATORY**
- Involved Use Case: Personal Security Application

4.2.1.11 Personalized VNF

VNF catalogue and management framework in SONATA must support the concept of "personal" in the sense that VNFs are assigned as a non-shareable resource with other users in the platform. Also Users identities in SONATA framework must allow a direct mapping between user and his VNFs.

Possible components impacted by this requisite: Public, Private and Service Platform Catalogues, Repositories and Orchestration framework (Lifecycle management plugin).

- KPI: VNF descriptors supporting this functionality, validation of uniqueness of VNFs per use.
- Category: **MANDATORY**
- Involved Use Case: Personal Security Application

4.2.2 SDK related requirements

This section discusses SONATA SDK's functional requirements. Figure 4.6 depicts how these requirements are mapped to SONATA architecture components.

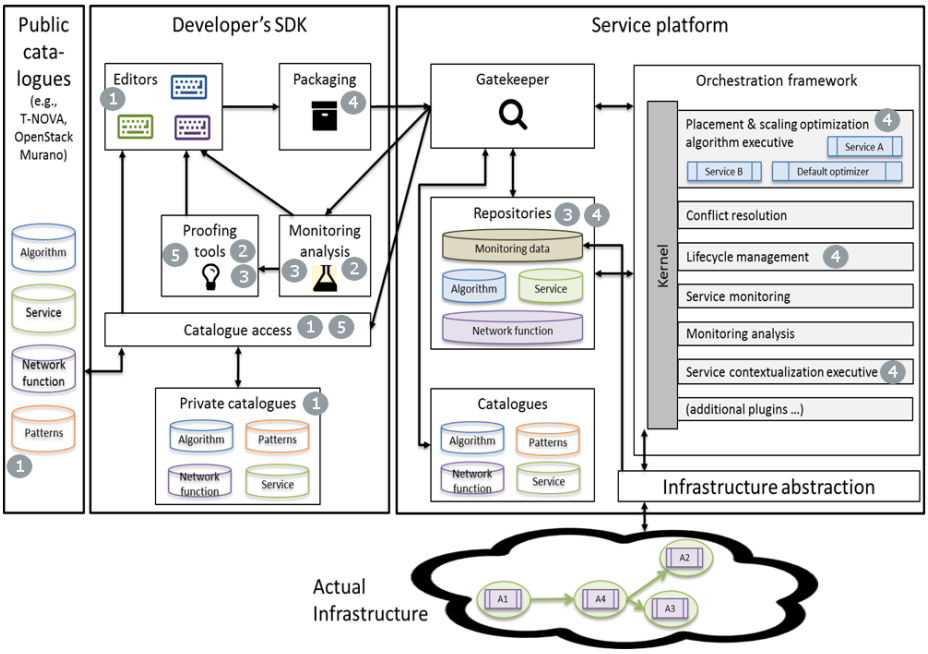

Figure 4.3: Mapping SDK Functional Requirements To Architecture

4.2.2.1 SDK Edition

SONATA SDK editor shall provide mechanisms to compose services with other services or VNFs in order to create an application which will control and manage the service from end to end. For example, management and control of a vCDN service and manipulation of IoT traffic, like processing and batching.

Possible components impacted by this requisite: Editors, Catalogue access and Public and Private Catalogues.

- KPI: Manipulate IoT traffic
- Category: **MANDATORY**
- Involved Use Case: vCDN and IoT

4.2.2.2 SONATA Reliability

SONATA shall provide mechanisms to easily build services with desired reliability, ensuring the possibility of using backup or redundant VNF.

Possible components impacted by this requisite: Proofing tools and Monitoring analysis.

- KPI: Reliability Options
- Category: **MANDATORY**
- Involved Use Case: vCDN

4.2.2.3 SONATA DevOps

SONATA shall provide tools for direct DevOps communication and feedback between services running, raised problems, catalogue, packages released and placement graphs.

Possible components impacted by this requisite: Proofing tools, Monitoring analysis and Repositories.

- KPI: DevOps tools & Instruments
- Category: **MANDATORY**
- Involved Use Case: vCDN

4.2.2.4 On-Demand Runtime Control

SONATA system shall allow operators to include some scaling/lifecycling modifications during service runtime, which should not lead to end instances and restart them, but manually scaling/modifying parameters in ongoing VNFs due to specifical previsions.

Possible components impacted by this requisite: Packaging, Repositories and Orchestration framework(Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins).

- KPI: Runtime possible modifications
- Category: **MANDATORY**
- Involved Use Case: vCDN

4.2.2.5 Security Simulation Tools

A common problem in security applications and service is the capability to simulate security incidents. Security simulation tools availability and integration in the SONATA framework are needed for validation and testing, i.e: DoS traffic generators, or malware traffic samples.

Possible components impacted by this requisite: Proofing tools and Catalogue access.

- KPI: Number of available security simulation tools
- Category: HIGHLY DESIREABLE
- Involved Use Case: Personal Security Application

4.2.3 Service platform requirements

This section discusses SONATA's service platform functional requirements. Figure 4.6 maps these requirements to SONATA architecture components.

4.2.3.1 Support for Integration with Existing VNFs or Components

The service platform must allow compoments or VNFs of a new service to be integrated with existing services, VNFs or system components (such as sensors or actuators).

Possible components impacted by this requisite: Repositories, Catalogues, Orchestration framework (Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins) and Infrastructure abstraction.

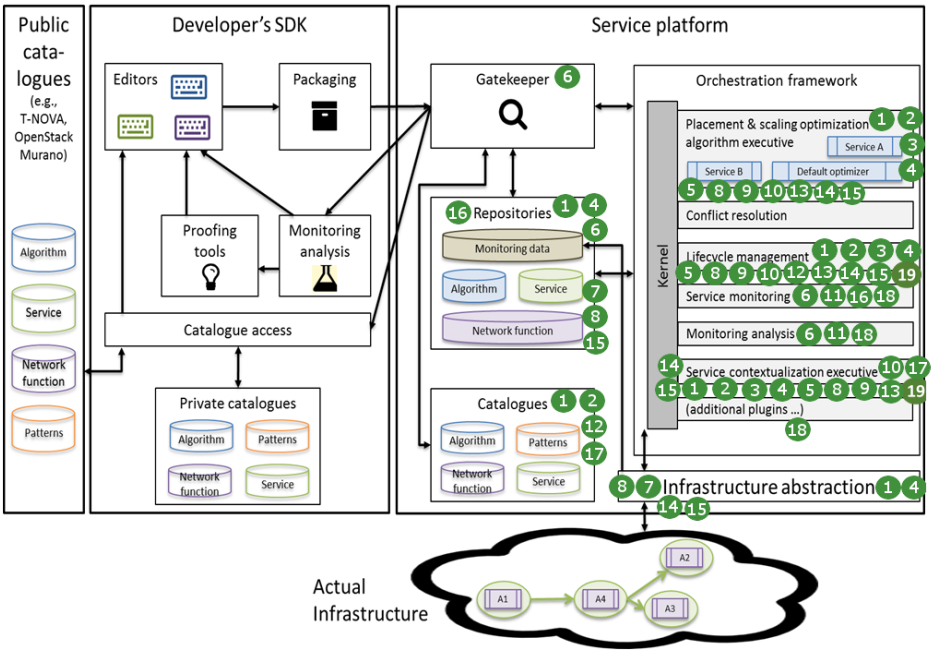

Figure 4.4: Mapping Service Platform Functional Requirements To Architecture

- KPI: support for corresponding annotations (or primitives) in the service programming model/language
- Category: **MANDATORY**
- Involved Use Case: Industrial Networks.

4.2.3.2 Service Chaining Support Across Wide Area Networks

The Service Platform must support service function chains that include service functions separated by a wide area network, e.g. across different data centers, or between the core data center and an edge cloud etc.

Possible components impacted by this requisite: Repositories and Orchestration framework (Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins).

- KPI: west-east interfaces between service platforms or arcitectural support for service platform hierarchies
- Category: **MANDATORY**
- Involved Use Case: Industrial Networks

4.2.3.3 Manual Service Function Placement

It should be possible to manually decide and configure where a service is to be placed. This can be very important for scenarios where the service developer knows that a Service Function has to run in a certain location / on a certain node, but is either unable to or not confident with defining placement constraints in a way that placement can be done by the orchestrator. This may be particularly the case in non-carrier verticals where experience with services may be lacking, deployment scenarios are simple, and ease of use is the primary objective.

Possible components impacted by this requisite: Orchestration framework (Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins).

- KPI: an API primitive towards the orchestrator that allows manual function placement
- Category: DESIRABLE
- Involved Use Case: Industrial Networks

4.2.3.4 Capability Discovery in Service Platform

The service platform, notably the infrastructure abstraction layer, must support the discovery of capabilities of the physical infrastructure, e.g. support for hardware-acceleration for certain functions such as encryption or the availability of a Zigbee interface. That way, it will become possible to optimize function placement and maybe even tune applications that have access to the capability-enriched network model via the service platform's NBI.

Possible components impacted by this requisite: Repositories, Orchestration framework(Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins) and Infrastructure abstraction.

- KPI: capability discovery primitives in the infrastructure abstraction layer and the service platform's NBI
- Category: HIGHLY DESIRABLE
- Involved Use Case: Industrial Networks

4.2.3.5 SONATA Multitenancy

SONATA shall be able to provide shared components to different tenants at the same time.

Possible components impacted by this requisite: Orchestration framework(Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins).

- KPI: Number of tenants
- Category: **MANDATORY**
- Involved Use Case: vCDN

4.2.3.6 SONATA DevOps

SONATA shall provide tools for direct DevOps communication and feedback between services running, raised problems, catalogue, packages released and placement graphs.

Possible components impacted by this requisite: Gatekeeper, Repositories and Orchestration framework(Service monitoring and Monitoring analysis plugins).

- KPI: DevOps tools and instruments
- Category: **MANDATORY**
- Involved Use Case: vCDN

4.2.3.7 Resource Infrastructure Mapping

SONATA service platform shall map the vCaches, vDPIs and vCPEs involved on the Control Application to deploy into specific infrastructure available (network, storage and computing).

Possible components impacted by this requisite: Repositories and Infrastructure abstraction.

- KPI: Resource allocation and time
- Category: **MANDATORY**
- Involved Use Case: vCDN

4.2.3.8 Application Deployment

SONATA system shall package the developed application and deploy it over the mapped resources, launching the vCDN instances.

Possible components impacted by this requisite: Repositories, Infrastructure abstraction and Orchestration framework (Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins).

- KPI: Time
- Category: **MANDATORY**
- Involved Use Case: vCDN

4.2.3.9 Ongoing Services Scale Up/Down

SONATA service platform shall provide interfaces to request new resources or reduce the existing ones, so the control application can ask for scaling up/down depending on its implemented optimization algorithms.

Possible components impacted by this requisite: Orchestration framework(Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins).

- KPI: Interruption, overloading
- Category: **MANDATORY**
- Involved Use Case: vCDN

4.2.3.10 Multi NFVI Orchestration

SONATA Orchestrator should be able to orchestrate multiple VNF execution environments (NFVI-PoPs) located in arbitrary places in the operator network topology. The NFVI-PoPs are considered to be controlled and managed by VIMs.

Possible components impacted by this requisite: Orchestration framework(Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins).

- KPI: Number and size of the NFVI-PoPs
- Category: **MANDATORY**
- Involved Use Case: vCDN and vEPC

4.2.3.11 Analytics Plugin

SONATA Orchestrator SHOULD be able to perform some analytic operations over the data that provide the services/functions. Dedicated analytic components must be offered to guarantee real time operations.

Possible components impacted by this requisite: Orchestration framework(Service monitoring and Monitoring analysis plugins).

- KPI: Time to perform analytics in real time operations
- Category: **MANDATORY**
- Involved Use Case: vCDN

4.2.3.12 NS Management Northbound API / Interface

The SONATA platform must be able to support an API which exposes the NFV Infrastructure as a service. For example, SONATA must have a common set of operations needed to manage the NS from the B/OSS, typical are feasibility, reserve, activation, deactivation, termination (other like design, reserve and provision actions have no sense at NFV) in an API or interfaces at northbound to be used/invoked by the B/OSS.

Possible components impacted by this requisite: Catalogues and Orchestration framework(Lifecycle management plugin).

- KPI: Set of operations allowed, successful management of the lifecycle of NFVI service elements (VMs and VNs), sufficient abstract service parameters to meet the deployment constraints of the NS/VNFs of all other use cases.
- Category: **MANDATORY**
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.2.3.13 Distributed NFVI

ISP and Network Operators architecture requires a geographical distribution of PoP (Point of Presence) where instantiate multiples VNFs as close as possible to user or based on the service demand. One example is when a security attack happens it is preferred to react as close as possible of the source of the attack. As a consequence SONATA orchestration layer should support multiples NFVI and VIM in distributed networks.

Possible components impacted by this requisite: Orchestration framework(Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins).

- KPI: Support at least 2 Datacentres or PoPs
- Category: HIGHLY DESIRABLE
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.2.3.14 Open Interfaces Towards NFV

The Sonata platform must be able to support a southbound interface which can request elements of NFV Infrastructure as a service. NFV components like VIM, NFVI or NFVO could be deployed with multiples providers. Indeed the number od NFV solutions is growing day by day. SONATA orchestration framework must support open or standards interfaces (southbound towards NFVI) to ensure the smooth integration of different NFV providers specially to facilitate the adoption.

Possible components impacted by this requisite: Orchestration framework(Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins) and Infrastructure abstraction.

- KPI: Open Interface specification or Standardization Body inclusion, successful activity of NFVI host service elements (VMs and VNs) with the constraints required by the NS/VNFs.
- Category: **MANDATORY**
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.2.3.15 Legacy Support

Any ISP or Network Operators or corporations has today deployed security networks solutions in virtualized or bare metal appliances. The most relevant example is a Firewall device. If SONATA has the aim to offer complex solutions and integrate with existing network environment, then SONATA need to interact and manage with not only VNFs also support legacy NF.

Possible components impacted by this requisite: Catalogues, Repositories and Orchestration Framework(Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins).

- KPI: Integration of one legacy NF.
- Category: OPTIONAL
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.2.3.16 VNF Resource Report

SONATA must provide an interface to list all resources allocated to a specific service, e.g., each IoT operator. This service allows the developer or administrator to get an overview of how the service is evolving and what datacenter resources are committed to each service.

Possible components impacted by this requisite: Repositories and Orchestration Framework(Service monitoring plugin).

- KPI: List how many resources are in use by each service.
- Category: **MANDATORY**
- Involved Use Case: IoT

4.2.3.17 Authorization

SONATA service must limit operations based on access levels and provide means to create and manage access profiles.

Possible components impacted by this requisite: Catalogues and Orchestration Framework(Service contextualization executive plugin).

- KPI: Access and block functionalities regarding access level
- Category: **MANDATORY**
- Involved Use Case: IoT

4.2.3.18 Traffic Simulator

Given that there is not yet the amount of IoT traffic the IoT use case is designed to address, there must be a way to simulate traffic, with functions like increase or decrease traffic levels per sensor and number of sensors in order to simulate a real IoT sensor environment.

Possible components impacted by this requisite: Orchestration Framework(Monitor analysis, Service monitoring and an additional plugin).

- KPI: Number of IoT traffic sensors, Generated traffic
- Category: **MANDATORY**
- Involved Use Case: IoT

4.2.3.19 No Information Duplication for NS/VNFs with Heterogeneous SPs

Information held by an instance of a Sonata system must not create unnecessary duplication or other dependencies. In particular, a Sonata system offering NFV Instrastructure as a service must not hold specific information about the type of VNFs or network services into which the VNFs are composed. Similarly, a Sonata system of a client SP composing network services hosted on NFV Instrastructure controlled by another service provider must not hold implementation details of the infrastructure.

Possible components impacted by this requisite: Orchestration Framework(Lifecycle management and Service contextualization executive plugins).

- KPI: Suiable abstracted set of NFVI service parameters which can meet the deployment constraints of all of NS/VNFs from all other use cases, minimal and abstract interactions across the NFVI as a Service API during NFVI service element lifecycle.
- Category: **MANDATORY**
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.2.4 Service monitoring requirements

This section discusses SONATA's service monitoring functional requirements. It discusses how requirements are mapped to architecture components (highlighted in Figure 4.6).

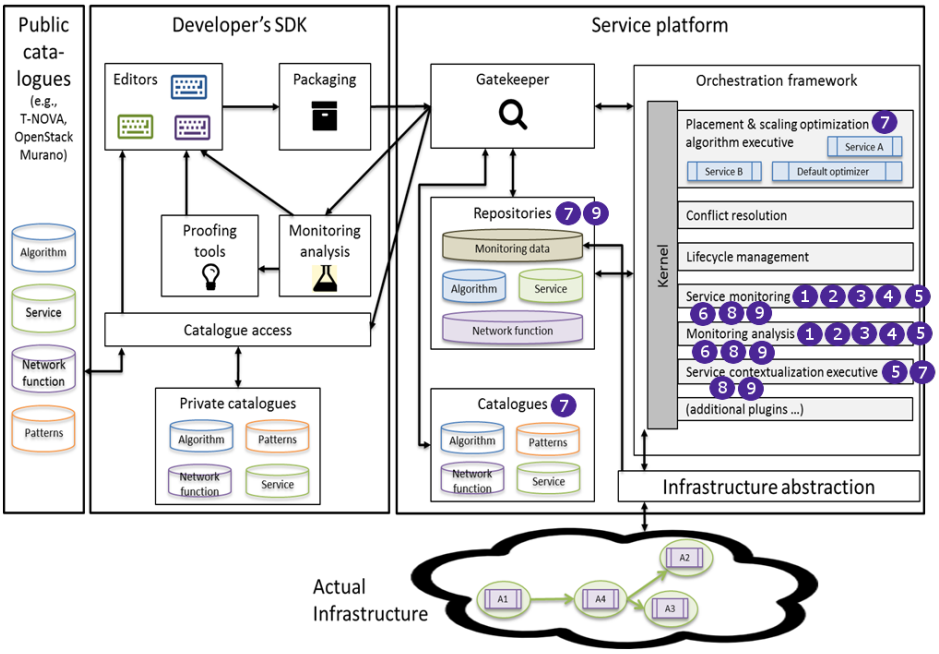

Figure 4.5: Mapping Service Monitoring Functional Requirements To Architecture

4.2.4.1 Timely Alarms for SLA Violation

The monitoring system must supply alarms for SLA violations (or malfunctioning components) in a timely manner depending on the SLA and type of problem. This means that the failure detection, but also the service platform message bus and notification system must have real-time capabilities. E.g. VNF unavailability for real-time traffic must be signaled as fast as possible, while in the case of best-effort traffic alarm signaling can happen with modest delays. Likewise, urgency of alarms is higher for VNFs with 1000s of users compared to single-user VNFs in the general case.

Possible components impacted by this requisite: Orchestration Framework(Monitor analysis and Service monitoring plugins).

- KPI: proven performance and scalability of the selected message bus system in the service platform
- Category: **MANDATORY**
- Involved Use Case: Industrial Networks

4.2.4.2 Statistics Interface

SONATA should have a statistical view on the main dashboard, so resources information, typical VNF locations, number of scaling operations achieved, etc can be post-analyzed and then studied by operators in order to modify service parameters and graphs in the future.

Possible components impacted by this requisite: Orchestration Framework(Monitor analysis and Service monitoring plugins).

- KPI: Statistic elements
- Category: OPTIONAL
- Involved Use Case: vCDN

4.2.4.3 VNF Specific Monitoring

The system shall expose service and VNF metrics to the network application, which collects them so levels of consumption can be monitored and are visible to the operator.

Possible components impacted by this requisite: Orchestration Framework(Monitor analysis and Service monitoring plugins).

- KPI: Availability of an API for VNFs capturing such metrics. Metrics accuracy and response time, e.g., specify service SLA for the EPC service.
- Category: **MANDATORY**
- Involved Use Case: vCDN, vEPC

4.2.4.4 VNF Real-time Monitoring

In order to detect and react to security incidents, VNFs will generate in real time information useful for Monitoring and response. SONATA framework must be able to collect, store, process and report in valid time windows to be useful to the ISP or the user.

Possible components impacted by this requisite: Orchestration Framework(Monitor analysis and Service monitoring plugins).

- KPI: Monitoring frequency, time to process alerts.
- Category: **MANDATORY**
- Involved Use Case: Personal Security Application

4.2.4.5 VNF Reporting to BSS/OSS and Subscriber

One the most relevant use of a Personal security application is to offer personalized information. SONATA framework must support mechanism to receive, managed and progress to user, VNF's report relevant data. It could imply opening communication paths in data plane or offer a alternative channel through SONATA framework.

Possible components impacted by this requisite: Orchestration Framework(Monitor analysis, Service monitoring and Service contextualization executive plugins).

- KPI: Specification of a channel for user report.
- Category: HIGHLY DESIRABLE
- Involved Use Case: Personal Security Application

4.2.4.6 Quality of Service Monitoring

One of the key method to detect security problems are the deterioration in the QoS. Metrics generation and degradation detection of network traffic, i.e. caused by a overloaded NFVI node or a attack, should be supported and reported.

Possible components impacted by this requisite: Orchestration Framework(Monitor analysis and Service monitoring plugins).

- KPI: Traffic QoS , packet loss, delays.
- Category: HIGHLY DESIRABLE
- Involved Use Case: Personal Security Application

4.2.4.7 VNF and Topology Validation

Based on the principle of providing security service, SONATA service framework by itself, or using third parties, must offer a validation capacity of VNFs when it is deployed in the NFVI. This validation should cover the integrity of the VNF, user attestation and data paths.

Possible components impacted by this requisite: Catalogues, Repositories and Orchestration Framework(Placement & scaling optimization algorithm executive and Service contextualization executive plugins).

- KPI: Attestation mechanism validation
- Category: **MANDATORY**
- Involved Use Case: Personal Security Applilcation

4.2.4.8 VNF SLA Monitor

While the platform does offer some dergree of autonomic abilities, there is still the possibility that a human can detect errors or potential points for optimization that the framework cannot. Hence, SONATA must provide an interface to monitor VNFs SLAs and resource usage. It must highlight VNFs with high and low usage, that may need scaling or other kind of manual intervention.

Possible components impacted by this requisite: Orchestration Framework(Monitor analysis, Service monitoring and Service contextualization executive plugins).

- KPI: Provided metrics and data visualization
- Category: **MANDATORY**
- Involved Use Case: IoT

4.2.4.9 VNF Status Monitor

Each VNF has a high level lifecycle, SONATA should allow the monitoring of that lifecycle in order to get an overview of the current service state. e.g., (i) deployment, (ii) operating, (iii) error.

Possible components impacted by this requisite: Repositories and Orchestration Framework(Monitor analysis, Service monitoring and Service contextualization executive plugins).

- KPI: Feedback detail provided
- Category: **MANDATORY**
- Involved Use Case: IoT

4.3 SONATA's Non-functional requirements

This section discusses SONATA's non-functional requirements. Figure 4.6 represents how non-functional requirements are mapped to SONATA architecture components.

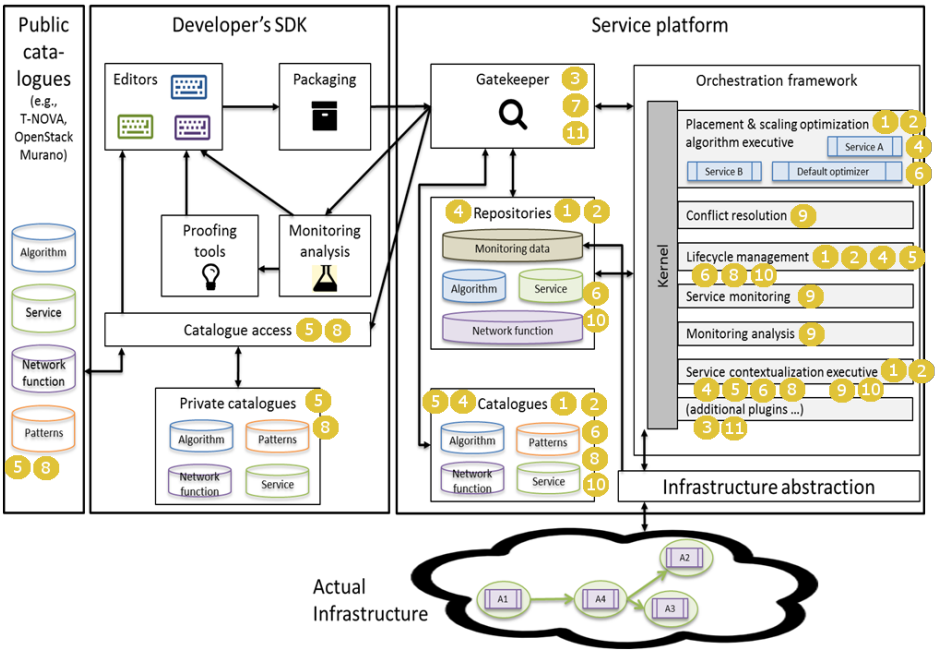

Figure 4.6: Mapping Non Functional Requirements To Architecture

4.3.1 Service Platform Scalability

The service platform must be scalable to support a very large number of devices (e.g. sensors), a high traffic load, high dynamics etc. depending on the use case. For example, there are some Security VNFs related with resource consumptions (i.e. anti-DDoS) and therefore requires dynamic scalability of resources in the SONATA framework. Also based on the concept of personal security application SONATA will require manage light VNFs but in great number, so scalability will be relevant if the number of subscribers rises considerably.

Possible components impacted by this requisite: Catalogues, Repositories and Orchestration framework(Placement & scaling optimization algorithm executive, Service contextualization executive and Lifecycle management plugins).

- KPI: support for 1000s of sensors, support for line-rate performance traffic processing in service chains across all use cases, Escalation functionality availability and provisioning times
- Category: **MANDATORY**
- Involved Use Case: Industrial Networks and Personal Security Application

4.3.2 Service Platform Customizability

The service platform must be customizable to support large-scale, complex deployemnts (such as carrier networks) as well as smaller, lightweight deployments (such as enterprises or industrial networks).

Possible components impacted by this requisite: Catalogues, Repositories and Orchestration framework(Placement & scaling optimization algorithm executive, Service contextualization executive and Lifecycle management plugins).

- KPI: plug & play, configurable service platform architecture that can be trimmed to a minimum basic feature set

- Category: HIGHLY DESIRABLE
- Involved Use Case: Industrial Networks

4.3.3 SONATA System Interface

SONATA should be operated over an intuitive platform with graphical and CL interfaces on a dashboard, and tools which allow operators to run services in a smooth way, not needing a comprehensive deep knowledge of SONATA system.

Possible components impacted by this requisite: GateKeeper and Orchestration framework (Aditional plugin - Management interface).

- KPI:Use facility, using time
- Category: HIGHLY DESIRABLE
- Involved Use Case: vCDN

4.3.4 SONATA System Update

SONATA system should be able to be updated in future releases, for both SDK and Service Platform integrated systems, and it should provide mechanisms to deal with inter-releases problems.

Possible components impacted by this requisite: Repositories, Catalogues and Orchestration framework(Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins).

- KPI: Updates and releases interoperability
- Category: OPTIONAL
- Involved Use Case: vCDN

4.3.5 SONATA Security

SONATA shall provide mechanisms that assure operator services are not accessed or read by other entity external to SONATA. At the same time, it shall review that VNF which are used are also secure in these terms.

Possible components impacted by this requisite: Public, Private and Service Platform Catalogues, SDK Catalogue access and Orchestration framework(Lifecycle management and Service contextualization executive plugins).

- KPI: Security issues
- Category: **MANDATORY**
- Involved Use Case: vCDN

4.3.6 SONATA Platform High Availability/Resilience

It is clear that the orchestration platform takes a very important role in the deployment and operation of the NFV/SDN environment. The central vision of the platorm and its plugins increment a lot the needs for a high availability platform where we must consider resilience aspects like: protection, clusterization, load balancing, session synchonization, replication, restoration, unique points of failure elements, standalone behaivour of services without service platform access. For example, a backup component must be located in a different location/infrastructure than the primary component.

Furthermore, the users expect to have service availability similar to any other ISP service and Service providers will search for it. As a consequence fault tolerance support in configuration, and monitoring is highly recommended.

Possible components impacted by this requisite: Repositories, Catalogues and Orchestration framework(Placement & scaling optimization algorithm executive, Lifecycle management and Service contextualization executive plugins).

- KPI: Time of downtime allowed, SLA values, down times, VNF deployment time, availability, Cost, location, shared resources;
- Category: **MANDATORY**
- Involved Use Case: vCDN, Personal Security Application, and vEPC

4.3.7 Authentication

SONATA system should provide mechanisms for authentication/authorization, so operators can access the platform.

Possible components impacted by this requisite: GateKeeper.

- KPI: Access
- Category: OPTIONAL
- Involved Use Case: vCDN, IoT, vEPC, Industrial Networks, Personal Security Application, Separate Client and Hosting Service Providers.

4.3.8 Confidentiality

All the information collected and treated by SONATA framework must keep the confidentiality and integrity principle. This must be achieving by strict authentication and authorization controls in the framework. This is especially important in this use case because the VNFs work with information per user, making it easily identifiable.

Possible components impacted by this requisite: Public, Private and Service Platform Catalogues, SDK Catalogue access and Orchestration framework(Lifecycle management and Service contextualization executive plugins).

- KPI: Security controls in place
- Category: **MANDATORY**
- Involved Use Case: Personal Security Application

4.3.9 Support Services with 5 nines SLA/control

vEPC is a service that is operates under a 5 nines SLA, it can not allow service degradation when scaling / healing / updating / migrating.

Possible components impacted by this requisite: Orchestration Framework(Monitor analysis, Service monitoring, Service contextualization executive and Conflict resolution).

- KPI: Interruption time, availability
- Category: HIGHLY DESIRABLE
- Involved Use Case: vEPC

4.3.10 Support "State-Full" Services

vEPC is a service that is operates under a 5 nines SLA, it can not allow service degradation when scaling / healing / updating / migrating.

Possible components impacted by this requisite: Repositories, Catalogues and Orchestration framework(Lifecycle management and Service contextualization executive plugins).

- KPI: Interruption time, availability
- Category: OPTIONAL
- Involved Use Case: vEPC

4.3.11 Integration with OSS

vEPC service operation involves integration with OSS system, SONATA should expose relevant APIs.

Possible components impacted by this requisite: GateKeeper and Orchestration framework(Aditional plugin - Management interface).

- KPI: Owner specific KPI
- Category: OPTIONAL
- Involved Use Case: vEPC

5 Conclusion and Recommendations to the Architecture Design

5.1 Main requirements

This document describes the 6 use cases identified by the consortium to guide the key requirements for the SONATA framework. They have been selected to cover a wide array of current and future ICT services where NFV orchestration could create significant benefits:

- Virtualizing a **Content Delivery Network (vCDN)** is an ideal case to explore a multitude of placement and elasticity strategies since caching is a stateless and easily deployable service, applicable in both telecommunications and IT domains.
- The **Internet of Things** use case targets the promising market of Smart Cities, where the number of devices raises scalability challenges in terms of monitoring, classification and optimization.
- In the area of mobile networks, the concept of **Virtual Evolved Packet Core (vEPC)** has been approved by the ETSI NFV Working Group as prone to deliver significant benefits such as elasticity, placement optimization and resilience.
- The **Industrial Networks** use case, illustrated through a network of wind turbines, represents a domain with specific requirements in terms of performance and reliability.
- The **personal security** use case aims at solving the complexity of protecting users online by deploying security services according to consistent policies no matter the device and the access network.
- The last use case covers the **interworking between separate Client and Hosting Service Providers** to deliver an end-to-end service.

Such a wide spectrum of use cases resulted in a vast set of requirements for SONATA covering most of the industry needs. In order to target a realistic subset, section 4 establishes different requirement categories: mandatory, (highly) desirable or optional. This conclusion summarizes the most important requirements the SONATA architecture will need to address.

The main mandatory requirement is a **service platform** with a **NFV orchestrator** providing the following features:

- It must be **scalable** and **highly available**, since it is a critical asset.
- It needs to provide a **VNF catalog** and support also **legacy NFs**.
- It needs to perform **VNF placement** according to multiple criteria:

  - It must take into account **placement constraints** defined using the programmability framework in the development phase. The placement may either be restricted to a specific location or through constraints such as hardware requirements, cardinality, logical distance, disjoint groups, etc. Such constraints may also be targeted at ensuring high availability, by ensuring redundant or backup VNFs.
  - It must **optimize VNF placement** with respect to the available resources while meeting the specified objectives in terms of performance or inter-VNF QoS constraints such as delay, jitter and reliability. The infrastructure abstraction layer must support the discovery of capabilities of the physical infrastructure in order to expose them to the logic responsible of the VNF placement.
  - It must support **service function chaining**: given an arbitrary graph of interconnected VNFs described using the programmability framework, the service platform must be able to deploy it.
  - It must support placement in **distributed NFV infrastructures** separated by wide area networks. Indeed, the orchestrator must not be restricted to one NFVI environment but needs to be able to orchestrate multiple NFVI-POPs.

- It needs to enable **multi-tenancy** with isolation between different tenants.
- It must allow creating and managing **access profiles** to limit operations based on access levels. It must be able to restrict information to the users, following specific infrastructure policies.
- It must support **VNF scaling** to adapt to service load. NFV developers must be able to specify when and how the operator can scale the VNF according to different parameters (VM load, bandwidth, latency...).

Tooling to support all the steps of the VNF lifecycle must be provided by the SDK and the service platform. This includes:

- describing NFVs and services as well as their non-functional properties such as desired reliability strategy
- composing services with other services or VNFs
- packaging the developed VNF
- deploying it over the mapped resources
- launching the VNF instance
- configuring the service
- managing the VNF runtime lifecycle: updates, migration and scaling
- providing service usage information for the service user
- providing real-time monitoring information such as triggers for scaling or SLA violation alarms
- providing reporting information of relevant data over a time window

5.2 State of the art and limitations

Section 4 gathered a state of the art of the existing solutions related to SONATA. None of the reviewed technologies addressed all of the requirements derived from SONATA's use cases. In particular:

- In term of semantics, none of the existing description models satisfies SONATA's objectives with respect to VNF placement constraints. For example, the ABNO and ETSI NFV are unable to convey constraints on security or isolation. Besides, they don't support any custom control logic for placement or elasticity, e.g. through scripts in a Domain-Specific Language (DSL). Moreover, they don't provide means to describe monitoring flows, which is necessary to connect monitoring tools at runtime. Yet, the ETSI NFV is a good starting point for an enhanced model.

- The outlined service programming tools are too specialized. Some focus on VNF creation, a few on cloud appliances and some on SDN management. None of the solutions satisfies the requirement of having an "Open Interface towards NFV" or supporting "Service Chaining". In addition, none of them can provide the user experience devised in the "SDK edition" requirement. In particular, it isn't possible to create a custom control logic component. Nevertheless, Cloudify, UNIFY or T-NOVA might prove to be a good foundation for SONATA.

- The reviewed service platforms aren't complete enough. They lack the ability to use monitoring or logging information for a control loop. But more importantly, no solution supports custom control logic or autonomous scaling therefore the "VNF Scaling" requirement is not covered. Even so, T-NOVA might prove to be a good starting point.

- With respect to existing service monitoring solutions, some of the ones evaluated can be good candidates for SONATA monitoring system. However, all of them lack adaptation and integration with respect to the service lifecycle and support of different technologies and tools that will be achieved in SONATA.

5.3 Recommendations

Taking into account the main requirements from the use cases and the current state-of-the-art, the key recommendations to ensure that SONATA's architecture brings valuable innovation to the community are the following:

- SONATA has the ambition to combine the SDN and the NFV perspectives. Existing solutions address each of them either separately or provide loose integration between the two. SONATA must provide a complete integration of both to address all the aspects of service lifecycle.

- SONATA must propose evolutions to existing service graph models describing non-functional aspects such as security and isolation. It must also include instructions for monitoring and profiling. SONATA should support this advanced model with a DevOps approach.

- SONATA must provide a SDK capable of building VNFs. Those functions will serve as the base components of the service graph. The specificities of implementing VNFs, i.e., the network function applications, are out of the scope of the project and SONATA should be able to use VNFs created with existing frameworks.

- SONATA must support custom control logic. This implies the possibility of building service-specific management components. With them, custom orchestration or scaling algorithms can be injected into the deployed services. To do so, such components must interact with the SONATA service platform at runtime. It requires a plugin-ready platform architecture and a stable API for communication.

- SONATA should validate the service graphs and the generated VNFs. This verification step will detect errors and inconsistencies into services. Existing tools should be reused for individual components, and SONATA must go further by validating complete services. Monitoring analysis tools should complement these validations. Again, these features should be seamlessly integrated with SONATA's DevOps experience. For instance, it should be simple to map monitoring streams to the service graph.

- One of the main elements in the SONATA's architecture is the catalogue. It should be able to store VNFs, services and control logic. The catalogue can be present in several locations and have different access levels: public, private or shared. An authentication layer and an authorization mechanism should protect this access while guaranteeing licenses.

- Two key factors create a DevOps experience. First the developer describes its requirements then a platform realizes them. This philosophy reinforces the split of SONATA into two subsystems. The SDK must build, validate and package services and their related components. The platform must receive a description and transform this manifest into a runtime instance. SONATA's SDK should be "locally" deployable and linkable to many platforms. SONATA's platform should be distributed and easily scalable.

5.4 The challenge

In the end, SONATA's challenge is to build on top of the achievements of existing initiatives to propose an overarching approach. Various technologies tackle many complex aspects in the cloud, SDN and VNF domains, but convergence has not yet taken place. SONATA will be the vector to bring all those worlds into a new common era.

6 Acronyms

- **5GPPP** 5G Infrastructure Public Private Partnership
- **AAA** Authentication Authorization and Accounting
- **AaaS** Application as a Service
- **ABNO** Application-Based Network Operations
- **API** Application Programming Interface
- **AWS** Amazon Web Services
- **BSS** Business Support System
- **BW** Bandwidth
- **BYOD** Bring Your Own Device
- **CapEx** Capital Expenditure
- **CBENCH** Controller Benchmarker
- **CDN** Content Distribution Network
- **CIMI** Cloud Infrastructure Management Interface
- **CLI** Command Line Interface
- **CPE** Customer-Premises Equipment
- **DDoS** Distributed Denial of Service
- **DPI** Deep Packet Inspection
- **DSL** Domain Specific Language
- **EEE** Entity Executed via an Executive
- **EPA** Enhanced Platform Awareness
- **EPC** Evolved Packet Core
- **ETSI** European Telecommunications Standards Institute
- **FP** Function Provider
- **HSS** Home Subscriber Server
- **IaaS** Infrastructure as a Service

- **IoT** Internet of Things
- **ISG** Industry Specification Group
- **IVM** Infrastructure Virtualization and Management
- **KPI** Key Point Indicator
- **LTE** Long-Term Evolution
- **LXC** Linux Container
- **MANO** NFV Management and Orchestration
- **MCP** Media Content Provider
- **MME** Mobility Management Entity
- **MVNO** Mobile Virtual Network Operator
- **NBI** NorthBound Interface
- **NF** Network Function
- **NFaaS** NF as a Service
- **NFV** Network Function Virtualization
- **NFVI** Network Function Virtualization Infrastructure
- **NFVIaaS** NFVI as a Service
- **NFVO** Network Function Virtualization Orchestrator
- **NSD** Network Service Descriptors
- **NSF** Network Security Function
- **OpEx** Operational Expenditure
- **OFlops** OpenFlow Operations per second
- **OSS** Operations Support Systems
- **PaaS** Platform as a Service
- **PCE** Path Computation Engine
- **PCRF** Policy and Charging Rule Function
- **PDN** Packet Data Network
- **PGW** Packet Gateway
- **PoC** Proof of Concept
- **PoP** Point of Presence
- **RAN** Radio Access Network

- **QoS** Quality of Service
- **SCADA** Supervisory Control and Data Acquisition
- **SFC** Service Chaining
- **SDK** Software Development Kit
- **SGW** Serving Gateway
- **SLA** Service Level Agreement
- **SNMP** Simple Network Management Protocol
- **SP** Service Provider
- **TOSCA** Topology and Orchestration Specification for Cloud Applications
- **UE** User Equipment
- **UGC** User Generated Content
- **vCache** Virtual Cache
- **vCDN** Virtual CDN
- **vEPC** Virtual EPC
- **vMME** Virtual MME
- **vNSF** Virtual NSF
- **vPGW** Virtual PGW
- **VPN** Virtual Private Network
- **vSGW** Virtual SGW
- **VIM** Virtual Infrastructure Manager
- **VNF** Virtual Network Function
- **VNFC** Virtual Network Function Component
- **VNFFG** VNF Forwarding Graph
- **VNFD** Virtual Network Function Descriptor
- **VNFM** Virtual Network Function Manager
- **VTN** Virtual Tenant Network
- **WAN** Wireless Area Network

A Bibliography

[1] Horizon 2020. VirtuWind – Virtual and programmable industrial network prototype deployed in operational wind park. Website, July 2015. Online at [https://5g-ppp.eu/virtuwind/].

[2] Amazon. AWs: Amazon Web Services. Website, July 2006. Online at [http://aws.amazon.com/].

[3] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming Protocol – independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[4] TU Delft. Accurate and precise OpenFlow Monitoring Pox module. Website, October 2015. Online at [https://github.com/TUDelftNAS/SDN-OpenNetMon].

[5] R Doriguzzi-Corin, E Salvadori, Aranda Gutierrez, C Stritzke, A Leckey, K Phemius, E Rojas, and C Guerrero. NetIde: Removing vendor lock-in in Sdn. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–2. IEEE, 2015.

[6] Federico M Facca, Elio Salvadori, Holger Karl, Diego R López, Pedro Andrés ArandGutiérrez, Dragan Kostic, and Roberto Riggio. NetIde: First steps towards an integrated development environment for portable network apps. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pages 105–110. IEEE, 2013.

[7] Flockport. Flockport: Say hello to simple containers! Website, September 2014. Online at [https://www.flockport.com/].

[8] Frauenhofer Fokus. OpenBaton. Website, October 2015. Online at [http://openbaton.github.io/].

[9] Distributed Management Task Force. CImi: Cloud Infrastructure Management Interface. Website, October 2010. Online at [http://www.dmtf.org/standards/cmwg/].

[10] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: A network Programming Language. *SIGPLAN Not.*, 46(9):279–291, September 2011.

[11] Nate Foster, Rob Harrison, Michael J Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: A network programming language. In *ACM SIGPLAN Notices*, volume 46, pages 279–291. ACM, 2011.

[12] FP-7. Unify – Unifying cloud and carrier networks. Website, November 2013. Online at [https://www.fp7-unify.eu/].

[13] FP-7. NetIde. Website, October 2015. Online at [http://www.netide.eu/].

[14] Google. cAdvisor. Website, October 2015. Online at [https://github.com/google/cadvisor].

[15] HashiCorp. Terraform: Infrastructure as code, July 2014, howpublished=Website. Online at [https://terraform.io/].

[16] HashiCorp. Vagrant: Development environments made easy. Website, October 2015. Online at [https://www.vagrantup.com/].

[17] Venkatrangan Govindarajan Hideyuki Tai, Masashi Kudo. VTn: Virtual Tenant Network. Website, June 2013. Online at [https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):Main].

[18] Ansible Inc. Ansible: Configuration Management. Website, October 2015. Online at [http://www.ansible.com/].

[19] Docker Inc. Docker: An open platform for distributed applications, August 2013, howpublished=Website. Online at [http://www.docker.com/].

[20] ETSI European Telecommunications Standards Institute. Website, December. Online at [http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf.

[21] ETSI European Telecommunications Standards Institute. Website, November. Online at [http://www.etsi.org/technologies-clusters/technologies/nfv/ http://www.etsi.org/technologies-clusters/technologies/nfv/.

[22] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[23] Puppet Labs. Puppet: Application orchestration. Website, October 2015. Online at [https://puppetlabs.com/].

[24] University College London. Lattice Monitoring. Website, October 2015. Online at [http://clayfour.ee.ucl.ac.uk/ikms/index.html].

[25] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, April 2008.

[26] Nagios. Nagios Distributed Monitoring. Website, October 2015. Online at [https://exchange.nagios.org/directory/Addons/Distributed-Monitoring].

[27] Nagios. Nagios Fusion Description. Website, October 2015. Online at [https://www.nagios.com/products/nagios-fusion/].

[28] Nagios. Nagios Overview. Website, 2015. Online at [https://exchange.nagios.org/directory/Plugins].

[29] Nagios. NagiosXi Description. Website, October 2015. Online at [http://library.nagios.com/library/products/nagiosxi/documentation/308-using-dnx-with-nagios].

[30] ETSI NFV. ETsi Nfv Pocs Overview. Website, 2014/5. Online at [http://nfvwiki.etsi.org/index.php?title=PoCs_Overview].

[31] University of Waterloo. Website, October. Online at [http://clayfour.ee.ucl.ac.uk/ikms/index.html https://github.com/srcvirus/floodlight.

[32] ONOS. ONos: Sdn Os with scalability, high availability, high performance, and right abstractions. Website, October 2015. Online at [http://onosproject.org/].

[33] Openstack. HEat - OpenStack Orchestration. Website. Online at [https://wiki.openstack.org/wiki/Heat].

[34] Openstack. Heat Orchestration Template (Hot) Guide. Website. Online at [http://docs.openstack.org/developer/heat/template_guide/hot_guide.html].

[35] OpenStack. OpenStack Ceilometer Architecture. Website, October 2015. Online at [http://docs.openstack.org/developer/ceilometer/architecture.html].

[36] OpenStack. OpenStack Ceilometer Developer. Website, October 2015. Online at [http://docs.openstack.org/developer/ceilometer/].

[37] István Pelle, Tamás Lévai, Felicián Németh, and András Gulyás. One tool to rule them all: A modular troubleshooting framework for Sdn (and other) networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 24. ACM, 2015.

[38] T-NOVA project. T-nOva- Network Functions as-a-Service over Virtualised Infrastructures. Website, 2015. Online at [http://www.t-nova.eu/].

[39] The OpenStack Project. OpenStack: The Open Source Cloud Operating System. Website, July 2012. Online at [http://www.openstack.org/].

[40] Joshua Reich, Christopher Monsanto, Nate Foster, Jennifer Rexford, and David Walker. Modular Sdn programming with Pyretic. *Technical Report of USENIX*, 2013.

[41] Charalampos Rotsos, Nadi Sarrar, Steve Uhlig, Rob Sherwood, and Andrew W Moore. OFloPs: An open framework for OpenFlow switch evaluation. In *Passive and Active Measurement*, pages 85–95. Springer, 2012.

[42] ScoutApp. Scout Application. Website, October 2015. Online at [https://scoutapp.com/].

[43] Sensu. Sensu – Monitoring for todays infrastructure. Website, October 2015. Online at [https://sensuapp.org/].

[44] Chef Software. Chef: Automation for Web-Scale It. Website, October 2015. Online at [https://www.chef.io/].

[45] Balázs Sonkoly, János Czentye, Robert Szabo, Dávid Jocha, János Elek, Sahel Sahhaf, Wouter Tavernier, and Fulvio Risso. Multi-domain service orchestration over networks and clouds: a unified approach. In *Proceedings of the 2015 ACM conference on SIGCOMM*. ACM, 2015.

[46] Universität Stuttgart. OpenTosCa - Open Source TosCa Ecosystem. Website, 2015. Online at [http://www.iaas.uni-stuttgart.de/OpenTOSCA/].

[47] GigaSpaces Technologies. Cloudify - Cloud Orchestration & Automation Made Easy. Website, 2014. Online at [http://getcloudify.org/].

[48] Telefonica. OpenManO. Website, October 2015. Online at [http://www.tid.es/long-term-innovation/network-innovation/telefonica-nfv-reference-lab/openmano].

[49] Canonical Ltd. Ubuntu. Juju: Model, build and scale your environments on any cloud. Website, 2015. Online at [https://jujucharms.com/].

[50] Stanford University. CBench. Website, October 2015. Online at [http://archive.openflow.org/wk/index.php/Oflops].

[51] Stanford University. CBenchFloodLight. Website, October 2015. Online at [https://floodlight.atlassian.net/wiki/pages/viewpage.action?pageId=1343657].