TNOVA

NETWORK FUNCTIONS AS-A-SERVICE
OVER VIRTUALISED INFRASTRUCTURES

GRANT AGREEMENT NO.: 619520

Deliverable D6.4

T-NOVA SLA & Billing

Editor Aurora Ramos (ATOS)

Contributors Aurora Ramos, Javier Melián (ATOS), Evangelos Markakis, George Alexiou (ATOS), Piyush Harsh, Manuel Pérez (ZHAW)

Version 1.0

Date December 30th, 2015

Distribution PUBLIC (PU)

Executive Summary

This document reports the results of the activities carried out in T-NOVA EU-FP7 Project "Functions as-a-Service over Virtualised Infrastructures" by Task 6.4 "SLAs and billing". Based on previous specification work in the project, the current document is delivered at the same time that the T-NOVA SLA and billing prototypes are finished (http://github.com/T-NOVA).

This report includes a summary of the main on-going related activities in the SOTA, providing a brief update from the analysis performed in the specification phase one year ago, describes the key points of T-NOVA SLA and billing frameworks, and details their different modules architecture, documentation related to their implementation (UML diagrams) and their integration with the rest of T-NOVA components (including APIs definitions). Also the results of functional verifications have been included as well as the report on requirements fulfilment.

The main T-NOVA subsystem interfacing SLA and billing frameworks is T-NOVA Orchestrator which it is expected to be finalized by end of March'2015. For this reason, final integration tests with the orchestrator will be done by that date, and therefore possible refinements may be needed in the SLA and billing implementations as well as some minor updates in the current document.

The T-NOVA SLA and billing frameworks correspond to the two commercial interactions defined in T-NOVA Marketplace:

- The Service Provider (SP) acquires Virtual Network Functions (VNFs) from the Function Providers (FPs); SLA between FPs and SP.
- The Customer acquires Network Services (NSs) provided by Service Providers based on the combination of VNFs previously purchased. SLA between SP and the Customer.

ETSI NFV requirements for SLA have been considered as input for T-NOVA SLA framework, though not a proper complete SLA business framework has been specified by ETSI so far. TMForum gives insights about metrics and SLA relations in cloud environment that has also been taken into account.

Furthermore, T-NOVA SLA framework has been developed being compliance with WS-agreement specification, as it has been identified as the most complete and extended specification for SLA procedure. All the surveyed research projects in cloud environment have followed this WS-Agreement though there is no research project in the state of the art providing SLA framework for NFV ecosystem as T-NOVA does.

After an exploratory work considering different options for billing mechanisms it has been concluded that Pay-As-You-Go is the most generic and suitable model to bill VNFs and Network Services in T-NOVA including an innovative Revenue Sharing model between Service Provider and Function Providers. FPs will benefit from the pay-as-you-earn model, an extension of pay-as-you-go in which the VNF provider will pay a percentage of the revenue received.

The T-NOVA billing framework is composed by 2 modules:

- Accounting module: it keeps a record of all the movements in the system that may have a potential impact in the billing.
- Billing module: it emits the bills based on the accounting information. The billing module being used in T-NOVA extends the generic rating-charging-billing (RCB) framework Cyclops, and whose functionalities have been extended to support the T-NOVA requirements.

All the components in the T-NOVA Marketplace (including SLA, accounting and billing) have been developed with a Software Oriented Architecture based on microservices, in which each Marketplace component has been developed separately and communicates with the others by means of RESTful APIs. This provides flexibility and scalability to the T-NOVA Marketplace in case further functionalities may want to be added in the future.

For the integration of all the different components in the Marketplace, Docker Compose has been selected; each microservice is placed in a different container, and they are integrated by means of Docker Compose file to coordinate the configuration of all the micro-services.

Table of Contents

1. INTRODUCTION 8

1.1. OBJECTIVES AND SCOPE 8
1.2. T-NOVA COMMERCIAL FRAMEWORK OVERVIEW 8
1.3. RELATION TO T-NOVA MARKETPLACE ARCHITECTURE 9
1.3.1. T-NOVA Marketplace implementation 10
1.4. DOCUMENT STRUCTURE 10

2. T-NOVA SERVICE LEVEL AGREEMENTS (SLAS) 11

2.1. STATE OF THE ART OVERVIEW FOR T-NOVA SLA 11
2.1.1. Other research projects 11
2.1.2. Standardization bodies 11
2.2. T-NOVA SLA FRAMEWORK DESIGN 15
2.2.2. SLA between SP and FP: VNF SLA 16
2.2.3. SLA between SP and customer 17
2.3. SLA MODULE ARCHITECTURE 20
2.4. T-NOVA SLA WORKFLOW 21
2.5. IMPLEMENTATION 24
2.5.1. WS-agreement 24
2.5.2. Implementation guidelines 25
2.5.3. Enforcement (assessment) 28
2.5.4. License 31
2.6. INTEGRATION 32
2.6.1. SLA module API definition 32
2.6.2. Calls to other APIs 36

3. T-NOVA BILLING AND ACCOUNTING 38

3.1. STATE OF THE ART ANALYSIS FOR BILLING IN T-NOVA 38
3.2. T-NOVA BILLING FRAMEWORK DESIGN 38
3.2.1. Billing for VNFs 38
3.2.2. Billing for Network Services 40
3.2.3. Workflow 40
3.3. ARCHITECTURE 42
3.3.1. Overview 42
3.3.2. Accounting 43
3.3.3. Billing 44
3.4. IMPLEMENTATION 47
3.4.1. Overview 47
3.4.2. Accounting 47
3.4.3. Billing 48
3.5. ACCOUNTING MODULE INTEGRATION 51
3.5.1. Accounting module API definition 51
3.5.2. Calls to other APIs 64
3.6. BILLING MODULE INTEGRATION 64
3.6.1. Billing module API definition 65

3.6.2. Calls to other APIs *70*

4. VALIDATION **71**

4.1. FUNCTIONAL VERIFICATION 71
4.2. REQUIREMENTS FULFILMENT 73
4.2.1. SLA management module requirements *73*
4.2.2. Accounting module requirements *74*
4.2.3. Billing module requirements *75*

5. CONCLUSIONS **76**

5.1. FUTURE WORK 77
5.1.1. 5G projects *78*
5.2. CONTRIBUTIONS TO STANDARDS 78

6. ANNEXES **79**

6.1. WS-AGREEMENT 79
6.1.1. Context *81*
6.1.2. Service description terms (SDT) *82*
6.1.3. Service references (SR) *82*
6.1.4. Service properties (SP) *83*
6.1.5. Guarantee terms (GT) *84*
6.1.6. Service Level Objective (SLO) *85*
6.1.7. Business Values *85*
6.2. T-NOVA SLA TEMPLATE EXAMPLE (JSON) 86
6.3. T-NOVA SLA AGREEMENT EXAMPLE (JSON) 88

7. REFERENCES **91**

8. GLOSSARY **93**

9. LIST OF ACRONYMS **95**

Index of Figures

Figure 1-1 Business T-NOVA stakeholders relationships [3] 9
Figure 1-2 T-NOVA Marketplace architecture [1] 9
Figure 2-1 SLA lifecycle 16
Figure 2-2 SLA management module in the Marketplace architecture 20
Figure 2-3 SLA management module internal architecture 20
Figure 2-4 SLA workflow sequence diagram 22
Figure 2-5 Basic structure of an SLA agreement 25
Figure 3-1 Accounting and billing workflow sequence diagram 41
Figure 3-2 Accounting and billing modules in T-NOVA architecture 42
Figure 3-3 Accounting module interfaces 43
Figure 3-4 Cyclops micro-services architecture 45
Figure 3-5 UDR micro-service architecture 45
Figure 3-6 RC micro-service architecture 46
Figure 3-7 Billing micro-service architecture 46
Figure 3-8 UML class diagram (split view) for UDR micro-service 49
Figure 3-9 UML class diagram for rc micro-service 50
Figure 3-10 UML class diagram for billing micro-service 50
Figure 3-11 Cyclops and T-Nova Marketplace Module Interactions 65

Index of Tables

Table 2-1 Summary of ETSI NFV service quality metrics [11] 12
Table 2-2 Standards for E2E Cloud SLA Management [13] 14
Table 2-3 Metrics collected by the VIM monitoring manager [26] 18
Table 2-4 Generic Networking service deployment flavours 19
Table 2-5 SLA API operation to register a provider 33
Table 2-6 SLA API operation to create a new template 33
Table 2-7 SLA API operation to update the template identified by *TemplateId* 34
Table 2-8 SLA API operation to retrieve a template identified by *templateId* 34
Table 2-9 SLA API operation to retrieve an agreement identified by *agreementId* 34
Table 2-10 SLA API operation to create a new agreement 35
Table 2-11 SLA operation to start or stop an enforcement job 36
Table 2-12 SLA API operation to retrieve information from a penalty 36
Table 3-1 Accounting module interfaces 44
Table 3-2 Accounting module information model 48
Table 3-3 Accounting API operation to get details about the client's billing model 52
Table 3-4 Accounting API operation to get the list of all active services for a user 53
Table 3-5 Accounting API operation to get the list of all VNFs purchased by a particular provider 54
Table 3-6 Accounting API operatio to get details of the revenue sharing model between SP and FP for the given VNF instance 55
Table 3-7 Accounting API operation to get the list of all sla violations for a service 56
Table 3-8 Accounting API operation to get the list of all sla-violations for a VNF 57
Table 3-9 Accounting API to get the list of all active services that use the given VNF 58
Table 3-10 Accounting API to get the list of all entries in the Accounting system or a single one if the parameter *accountId* is present 59
Table 3-11 Accounting API operation to create a new entry 60
Table 3-12 Accounting API operation to update an existing entry 60
Table 3-13 Accounting API operation to delete an existing accounting entry 61
Table 3-14 Accounting API operation to update the status of a service given its instanceId and the new status 61
Table 3-15 Accounting API operation to retrieve the list of all running VNFs the user (service provider) is using 64
Table 3-16 Billing API operation for getting user's data 66
Table 3-17 Billing API operation to query for particular resource / service id 67
Table 3-18 Billing API operation to generate bill for a particular customer 68
Table 3-19 Billing API operation to retrieves the earnings of a provider in between specified dates for all the instances 69
Table 3-20 Billing API operation to retrieves the bill of a specific user in between specified dates for all the instances 70
Table 4-1 SLA, accounting and billing verification 73
Table 4-2 SLA requirements fulfilment 74
Table 4-3 Accounting requirements fulfilment 75
Table 4-4 Billing requirements fulfilment 75

1. INTRODUCTION

1.1. Objectives and Scope

This deliverable presents the activities and results from Task 6.4, whose objective is the implementation of the T-NOVA components that constitute the SLA (Service Level Agreement) and billing framework, including the definition of:

- Mechanisms for SLAs that can support the formal definition and management of the relationships between the T-NOVA stakeholders – SLA template, negotiation, agreement.
- SLA management information for later billing and conciliation, depending on the terms and conditions gathered in the SLA and on whether this SLA has been met by all parties or not.
- Management of the monitoring information from the orchestrator whether the committed SLA has been met or not, taking the necessary actions, which can lead to simple reports or additional credits or debits in the billing account for this customer.
- Most suitable billing mechanisms for network services and VNFs in T-NOVA, including the T-NOVA accounting procedure in order to store all the information that will be needed in T-NOVA for billing purposes.

The current work relies on previous related specification and research phases in the project, including requirements elicitation, that were explained in [1] [2]. The first release of SLA, accounting and billing components is due by the end of December 2015, though it is expected that after the integration work between the marketplace and the rest of T-NOVA subsystems, a.k.a, orchestrator and function store, the SLA, accounting and billing modules could be refined based on the overall integration feedback. This will be reported in the final version of the current report that will be due by end of June 2016.

1.2. T-NOVA commercial framework overview

The T-NOVA Marketplace generic business scenario, depicted in Figure 1-1, reflects the two main commercial relationship that are in T-NOVA: one between the Service Provider (SPs) and Function Providers (FPs) to acquire standalone VNFs to compose a Network Service (NSs) and the second one between the SP and the Customer who acquire NSs.

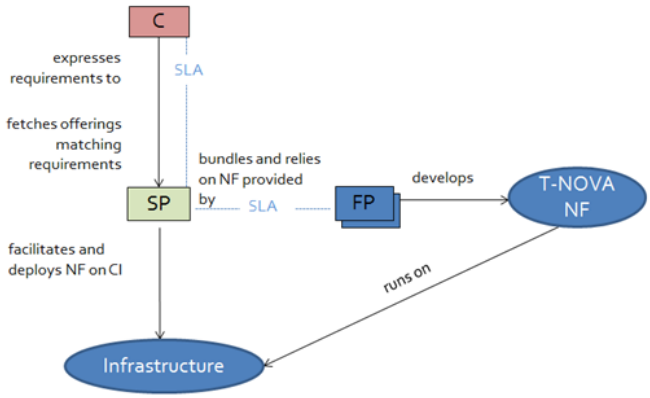


Figure 1-1 Business T-NOVA stakeholders relationships [3]

The Function Providers (FPs) that want to sell their VNFs through the T-NOVA Marketplace will enter the system providing their VNFs information: VNF metadata including technical constraints, SLAs, price, etc.

The Service Provider (SP) may then enter the system, acquire VNFs and bundle them into new NSs, including the service description, the SLA specification and its pricing. These offerings will then be exposed in the T-NOVA marketplace to the Customer. Different SLA levels and prices imply different NSs.

The Customer will be able to search for the available NS, selecting the one that better suits him/her. When the customer selects an offering the SLA agreement procedure will be initiated: between customer and SP and consequently between SP and FPs.

1.3. Relation to T-NOVA Marketplace architecture

The T-NOVA Marketplace has been designed as a distributed platform placed on top of the overall T-NOVA architecture being in charge of managing all business relationships among the T-NOVA stakeholders [1]. Figure 1-2 highlights the location and interfaces on SLA, accounting and billing modules which are the components within the scope of the current report and that will be detailed in the following sections.

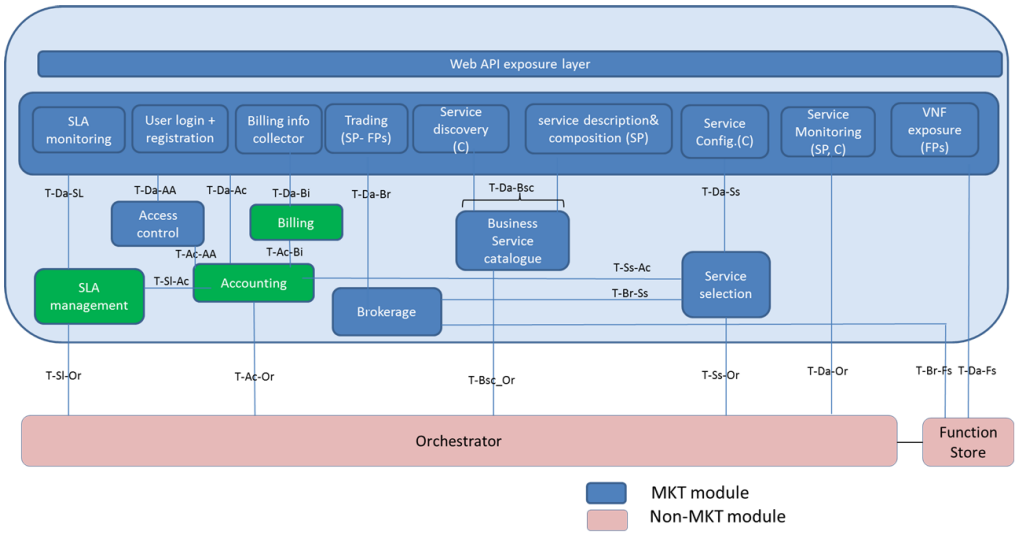


Figure 1-2 T-NOVA Marketplace architecture [1]

1.3.1. T-NOVA Marketplace implementation

As explained in [2], in order to enhance the T-NOVA Marketplace with the modularity specified in previous work [3], the T-NOVA Marketplace implementation is based on a micro services software architecture [2]. This kind of architecture provides the necessary tools for each marketplace module to run separately as a standalone service. Hence, each module can manage its own database (if needed) or share a database with other module(s) and be scaled, deployed and evolved independently.

Furthermore, by using this software architecture model, each component can be implemented separately in any technology (e.g. Java, Python, etc.) and can be more easily integrated in the overall system, which is the Marketplace.

The Marketplace's software architecture is also REST [4]-based, a set of architectural principles by which it is possible to design web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages. Therefore SLA, accounting and billing modules implementations follow this approach. Details on SLA, accounting and billing modules implementation and their request methods, following these approaches, are explained in this document.

1.4. Document structure

This document is structured as follows:

Section 2 is devoted to SLA while section 3 is devoted to accounting and billing. For each of these sections the same internal structure has been followed: firstly an overview about the main outputs of the survey done in T-NOVA Marketplace specification phase [2], including an update on the activities by the main identified relevant bodies during the last year. Then the details of the T-NOVA Marketplace SLA and accounting and billing frameworks are explained respectively. Next subsection is devoted to depicting the architecture the applicable modules, and finally the insights about their implementation and integration are explained including the collection of APIs definitions. Section 4 contains the results of the functional verification tests that have been performed for the implemented modules, as well as the report on the fulfilment of the requirements that were gather in the specification phase. The conclusions gathered from the contents of this document are provided in section 5. Annexes include insights of the WS-Agreement specification which has been followed to implement T-NOVA SLA module, as well as examples of SLA templates and agreements in JSON format.

2. T-NOVA Service Level Agreements (SLAs)

Service Level Agreements (SLAs) represent the contractual relationship between a service consumer and a service provider in order to provide a mechanism to increase trust in providers by encoding dependability commitments and ensuring the level of Quality of Service is maintained to an acceptable level. This section summaries firstly the SLA survey done applicable to T-NOVA, then the T-NOVA SLA framework is explained, and finally how the SLA management framework is implemented as part of the T-NOVA system.

In T-NOVA we are dealing with three different stakeholders: Customer, SP and FP that relate as shown in Figure 1-1, therefore we will have two different kinds of SLA contracts: between the FP and the SP and between the SP and the Customer.

2.1. State of the Art overview for T-NOVA SLA

This sub-section covers the main outputs on the surveyed performed in previous work in T-NOVA project [2], as well as an update on the activities of the more relevant initiatives related to SLA in NFV context.

2.1.1. Other research projects

Several recent research projects implementing SLA management frameworks for different environments were surveyed in [2], such as Cloud4SOA (FP7) [5], Fed4FIRE Project (FP7) [6] and the XIFI Project (FI-PPP) [7]. They were a good input to consider them when designing the T-NOVA SLA ecosystem, however, they do not address the specific particularities for the NFV business ecosystem that we address in T-NOVA.

Furthermore, automatic SLA SOTA is mainly in the scope of cloud, but T-NOVA requires a combination of network functions and cloud, Cloud not being enough. Usually, in telecommunications, SLAs can be seen as the minimum service acceptance level a customer would agree to be delivered by a communication service provider, though they are usually vague, not end-to-end and unknown to the network [8] [9]. Moreover, they are not as dynamic or as automatically managed as T-NOVA requires.

2.1.2. Standardization bodies

2.1.2.1. ETSI

At T-NOVA specification phase [1] ETSI did not define a business perspective to manage the SLA relationships among the possible stakeholders in the NFV scheme, but identified some requirements for the final network service SLA [10]:

[Req. 5] The SLA shall specify the “metrics” to define the value and variability of “stability”.

[Req. 6] The NFV shall support mechanisms to measure the following metrics and ensure that they are met per SLA:

- Maximum non-intentional packet loss rate.
- Maximum rate of non-intentional drops of stable calls or sessions (depending on the service).
- Maximum latency and delay parathion on a per-flow basis.
- Maximum time to detect and recover from faults aligned with the service continuity requirements
- Maximum failure rate of transactions that are valid and not made invalid by other transactions.

[Cont.1] The SLA shall describe the level of service continuity required.

Moreover, in [11] ETSI gives a first approach of the different service quality metrics that will influence the final service quality level that the end-user will experiment. ETSI classifies these quality metrics in four groups:

- Virtual machine service quality metrics.
- Virtual network service quality metrics.
- Technology components offered as a Service (standalone VNFs).
- Orchestration service quality metrics.

Service Metric Category	**Speed**	**Accuracy**	**Reliability**
Orchestration Step 1 (e.g., Resource Allocation, Configuration and Setup)	VM Provisioning Latency	VM Placement Policy Compliance	VM Provisioning Reliability VM Dead-on-Arrival (DOA) Ratio
Virtual Machine operation	VM Stall (event duration and frequency) VM Scheduling Latency	VM Clock Error	VM Premature Release Ratio
Virtual Network Establishment	VN Provisioning Latency	VN Diversity Compliance	VN Provisioning Reliability
Virtual Network operation	Packet Delay Packet Delay Variation (Jitter) Delivered Throughput	Packet Loss Ratio	Network Outage
Orchestration Step 2 (e.g., Resource Release)			Failed VM Release Ratio
Technology Component as a-Service -	TcaaS Service Latency		TcaaS Reliability (e.g.,defective transaction ratio) TcaaS Outage

Table 2-1 Summary of ETSI NFV service quality metrics [11]

Update on December 2015 and relation to T-NOVA

In ETSI NFV second phase, a report was published in relation to business relationships in the NFV ecosystem and further insights about suitable metrics to be part of SLAs in NFV [12]. T-NOVA had been ahead of this business SLA analysis, though we have explored here the alignment between T-NOVA approach and ETSI NFV to this respect.

Though the roles to be played in NFV ecosystem are aligned between T-NOVA and [12], the business case explained in the latter context is slightly different in relation to the roles that are played by each stakeholder. In [12] a more cloud perspective is provided considering TMForum report on End-to-end cloud SLA management [13] and QuestForum Handbook about technical measurements [14], so the customer may have more control of the Service, which makes sense also from a business perspective. In the other hand, technical issues as automated service verification should be considered. T-NOVA on the contrary, makes the assumption that both Customer and Service Provider must closely collaborate to configure the service, this is indeed the Customer is in charge of configuration the service, but also the SP which is the one that owns the MANO and NFVI is in charge of the actual deployment based on its knowledge of the infrastructure.

The further proposed metrics by ETSI NFV in [12] are aligned with those explored in T-NOVA (section 2.2), being classified in the following groups:

- VNF Software Quality Measurements
- Function Components Offered as-a-Service Quality Measurements
- Automated Lifecycle Management Quality Measurements
- Failure Notification Quality Measurements
- Virtual Infrastructure Quality Measurements

Also a sample of SLAs is provided named as Service Level Specifications (SLSs) as the technical part of the SLA aligned with SLA specification done in T-NOVA (section 2.2):

- Key Quality Indicator (KQI)
- Threshold
- Measurement Point
- Estimator

2.1.2.2. TMForum

At the time of T-NOVA specification phase [1] writing, TMForum had not provided any study of the possible SLA relationships that can arise in NFV ecosystem specifically. However, in order to implement the T-NOVA SLA management system we looked at SLA management in different cloud environments to be adapted to the business NFV scenarios identified in T-NOVA.

TMForum [15] released in October 2014 a new version of its technical document: Enabling End-to-end Cloud SLA Management [13] which refers to concepts and considerations in multi-provider cloud environment that in T-NOVA may be applied to the study of NFV SLA management, for instance:

- Cloud metrics, fall into two major categories: business metrics (often defined within the SLA), and Technical metrics (monitoring metrics) that allow the business SLA to be met. This can be applied also for SLA NFV metrics. For instance, "response time" may be specified in the SLA, meanwhile other technical measures such as "hops" and "bandwidth" may be used to dynamically allocate resources, enabling "response time" SLAs to be met.
- Usage-based costing metrics are generally a sub-category of the business metrics and will be a major component of a Service Agreement they may or

may not be part of Service Level Agreement. Some examples of usage-based metrics are: number of users, instance minutes, storage resource capacity used bytes, CPU minutes and RAM in megabytes, etc. Cost metrics are established based on money currency per unit ("€/instance minute" for example). SLA is primarily dealing with service assurance, but usage metrics that contribute to bill calculation will not be in scope for SLA management.

TM Forum's SID (Information Framework) has a metrics interest group, which delivered in September 2013 a modeling framework for metrics, in SID release 13.5 (definition of metrics, as well as hierarchy/relationships between metrics).

Requirements	Recommendation
SLA Modelling Methodology: use a common notation that is easily understood at the business level to model the SLA attachment point	Use the notation developed in [16] as the standards for SLA roles and responsibility analysis. Use TM Forum eTOM process fragments documented in [TMF GB917] for E2E SLA process analysis and design. See examples in (intermediary role & processes) Use TM Forum information model (SID) and related entities documented in [16] for E2E SLA information model analysis and design.
Metric Model: There are various types of metrics/measurements that contribute to the overall calculation of the SLA, such as Business Metrics, Performance Metrics, and Storage Metrics etc.). A meta model is required that provides a consistent description of these metrics that likely to be developed by different organizations and SMEs.	A Metric ABE (Aggregated Business Entity) is being defined by the TM Forum Shared Information/Data Model team, this work is to define a standardized definition and entity relationships so that metrics developed by various SDO/consortia can be joined up for the end-to-end management purpose. The intention of the SID metric ABE is to support all related work in this area, such as the work done in NIST Cloud Computing Metric group and the CSMIC work.
Service Level Specification (SLS) Model: the schema for service level specification that contains all measurements that needs to be monitored for a given service.	Recommend to use SID ServiceLevelSpecification [17] as the standardized model for SLS schema development Recommend to use SID to construct Service Level Specification (SLS)
APIs: APIs to facilitate the automation and interoperability of SLA lifecycle management: SLA negotiation, activation, configuration and re-negotiation etc.	Candidates: • WS-agreement, WS-agreement negotiation • TM Forum • SMI: for data collection • SLA APIs, Catalogue management APIs (under development)

Table 2-2 Standards for E2E Cloud SLA Management [13]

Update on December 2015 and relation to T-NOVA

In the last year TMForum has released the next two reports:

- *IG1120 Virtualization Impact on SLA Management* [18]: this exploratory report provides initial thoughts on the impact of end-to-end SLA management in a fully software-defined and virtualized environment, i.e., Cloud-SDN-NFV. Its objective is to leverage knowledge and experience from the TM Forum SLAM work, and apply it to virtualized environments.
- *IG1127 End-to-end Virtualization Management: Impact on E2E Service Assurance and SLA Management for Hybrid Networks* [19]: This Application

Note brings out the challenges and impacts on end-to-end Service Assurance and SLA management in a hybrid physical/virtualized environment. This change introduces a host of actors, each responsible for part of the overall solution, and creates new SLA/OLA constructs. It also necessitates new means of operations – having SLA-linked Policy Orchestration, new RCA rules for Fault Correlation, and automated closed loop controls, thus reaping the benefits of virtualization as design principles.

TMForum still declares as future work the following items:

- SLA Management user stories and use cases for VNF package.
- Service Level Specification Template: ZOOM information model.
- Metrics for VNF and NFVI.
- Policy-model: ZOOM information model.
- Policy-based SLA Management and APIs: ZOOM Future OSS/BSS.

Therefore, T-NOVA potential contribution to TMForum can be related mostly with the SLA templates specification as part of the service information model, and metrics identified for VNFs.

2.2. T-NOVA SLA framework design

In T-NOVA there is a hierarchical SLA ecosystem, since there are two different SLAs according to the 2 different commercial relationships that exist:

1. The SLA agreed between Function Provider (FP) and Service Provider (FP).
2. The SLA between the Service Provider and its customers.

The T-NOVA SLA lifecycle will be implemented in the following steps:

1. *SLA Template Specification:* the SP and FPs follow a clear step-by-step procedure describing how to write an SLA template to provide a correct service description. (The SLA template will be a form that has the same structure as the SLA Agreement but some fields are not filled yet or might change as a result of the negotiation process).
2. *Publication and Discovery*:
   - The FP publishes the different SLA offers as part of the metadata that will be stored in the T-NOVA Function Store for each NF when uploading a VNF packaged [20]. The SP will discover the different SLA options by means of the brokerage procedure [21].
   - The SP publishes the different SLA offers for the NSs through the business service catalog for the customer to browse/compare offers.
3. *Negotiation*: agreement on SLA conditions between the customer and the SP and between the SP and the FPs.
   - For the SLA between customer and SP this will be done by the customer selecting one of the predefined offerings from the business service catalog.
   - For the SLA between SP and customer this will take place as the result of the trading process of VNFs with specific SLAs, but formalized in a second step once the service has been acquired by the customer.

4. *Resource Selection*: depending on the chosen SLA for every service, the orchestrator will allocate the resources that need to be assigned to the service in order to meet that SLA [22].
5. *Monitoring and Evaluation of the SLA*: this step will take place by comparing all the terms of the agreed SLA with the metrics provided by the orchestrator monitoring system. (These results will be available to be shown through the dashboard when the SP or customer requires it).
6. *Accounting*: this will be done invoking the charging/billing system to inform about billable items as penalties based to the result of step 5.

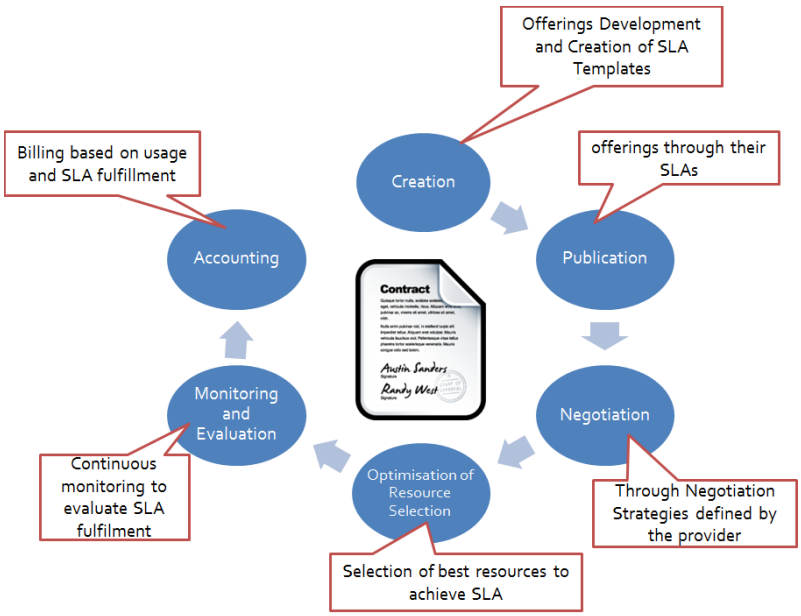


Figure 2-1 SLA lifecycle

2.2.2. SLA between SP and FP: VNF SLA

This is the SLA agreed between the SP and the different FPs that sell the VNF as part of a network service.

There could be two different approaches for the SLA associated to a VNF acquisition by the SP. On one hand each network function is a software product that the SP acquires to be deployed in his own infrastructure therefore we could think on one hand of a SLA associated to the software itself, to which we refer in 2.2.2.1. , and on the other hand having expected performance of each VNF, such as VNF downtime, number-of-subscribers, etc., to which refer in 2.2.2.2.

2.2.2.1. SLA software

What the SP purchases to the FPs are software applications with accompanied metadata and images, this is, the deployment view of the VNFs software architecture.

In software development, specific SLAs can apply to application outsourcing contracts in line with standards in software quality, and recommendations provided by neutral

organizations like CISQ, which has published numerous papers on the topic (such as Using Software Measurement in SLAs) [23].

2.2.2.2. SLA VNF specific monitoring parameters

The SLA agreed between SP and FP includes the specification of the expected performance of the VNF according to the VNF specific monitoring parameters that will be part of the VNFD in accordance with the definition for the monitoring parameters that ETSI gives for the VNFD:

Monitoring parameters, which can be tracked for a VNF can be used for specifying different deployment flavours for the VNF in a VNFD, and/or to indicate ***different levels of VNF service availability.*** *These parameters can be an aggregation of the parameters at* ***VDU level*** *e.g.,* ***memory-consumption****,* ***CPU-utilisation****,* ***bandwidth-consumption*** *etc.* ***They can be VNF specific*** *as well such as* ***calls-persecond (cps), number-of-subscribers, no-of-rules, flows-per-second, VNF downtime,*** *etc. One or more of these parameters could be influential in determining the need to scale* [24].

VNF specific metrics that will be derived for each VNF in T-NOVA use cases are detailed in [25].

2.2.3. SLA between SP and customer

Various types of metrics/measurements can contribute to the overall calculation of the technical metrics that may be part of the SLA for a Network Service. Therefore a metamodel is required to provide a consistent description of these metrics that are likely to be developed. In T-NOVA the SLA between SP and customer will be described and agreed in T-NOVA depending on the metrics that the monitoring system in T-NOVA will measure, which can depend on:

- Orchestration operation
- Virtual machine operation
- Network operation
- Monitoring metrics of the VNFs which defined their expected performance, this is the SLA agreed by the SP and FPs for all the VNFs that are part of the NS.

Therefore, in a general case, at this stage we make the assumption that the SLA between SP and a Customer will be an aggregation or combination of the SLAs agreed between the SP and FPs for the VNFs that compose the service.

A T-NOVA network service has two kinds of metrics: the ones inherited from the VNFs that compose it and metrics of the service itself that do not belong to the VNFs. At the moment of the service SLA definition, the possible metrics are presented to the service provider and the aggregated value will depend on the selected topology (m1+m2, max[m1, m2], avg[m1, m2], etc.). Only metrics of the same kind can be aggregated. The chosen formula to calculate the aggregated value will be added to the NSD so the service metric values can be generated based on the monitoring of the VNFs that are present in this formula.

The system offers also the service provider the option to add generic metrics that are not among the VNF because they depend on the infrastructure where it's deployed; metrics that are common to the network services regardless the VNFs they use, e.g. Service uptime.

The list of generic metrics to be monitored in T-NOVA relevant for the current T-NOVA use cases are collected in Table 2-3 (This list is meant to be continuously updated throughout the project in order to align with the technical capabilities and requirements of the components under development and the use cases which are implemented).

Domain	Metric	Units
VM/VNF	CPU utilisation	%
VM/VNF	No. of VCPUs	#
VM/VNF	RAM allocated	MB
VM/VNF	RAM available	MB
VM/VNF	Disk read/write rate	MB/s
VM/VNF	Network Interface in/out bitrate	Mbps
VM/VNF	Network Interface in/out packet rate	pps
VM/VNF	No. of processes	#
Compute Node	CPU utilisation	%
Compute Node	RAM available	MB
Compute Node	Disk read/write rate	MB/s
Compute Node	Network i/f in/out rate	Mbps
Storage (Volume)	Read/write rate	MB/s
Storage (Volume)	Free space	GB
Network (virtual/physical switch)	Port in/out bit rate	Mbps
Network (virtual/physical switch)	Port in/out packet rate	pps
Network (virtual/physical switch)	Port in/out drops	#

Table 2-3 Metrics collected by the VIM monitoring manager [26]

According to the Monitoring Parameters part of the NSD that ETSI has defined [24]:

The NS monitoring parameters represent those which can be tracked for this NS. These can be network service metrics that are tracked for the purpose of meeting the network

service availability contributing to SLAs (e.g. NS downtime). These can also be used for specifying ***different deployment flavours for the Network Service*** *in Network Service Descriptor, and/or to indicate* ***different levels of network service availability****. Examples include specific parameters such as* ***calls-per second (cps), number-of-subscribers, no-of-rules, flows-per second****, etc. 1 or more of these parameters could be influential in determining the need to scale-out.*

Based on the collection of parameters that can be collected at different levels in the system, we may have also as part of the SLA some SLA telco service parameters such as: delay, jitter, packet loss, etc. that are related directly with Quality of Service that the final customer will perceived when using end-to-end services.

2.2.3.1. Service deployment flavours

ETSI NFV uses the Gold, Silver, Bronze notation for the definition of a particular NS that is composed by a number of VNFs and a Connectivity Service, which we use in T-NOVA to name a group of technical parameters for the SLA specification. However that notation, as it is defined at the moment, does not correspond to any particular principle/rule common to all the possible compositions available in T-NOVA. The analogy that we can think is coming from the relevant usage of the three colour marker in networking. The table below attempts to provide a generic framework for the definition of this approach.

Flavour name	Properties
Gold	- Highest Priority Service - Scaling requirement in terms of resources are taken into account (always available) - Network traffic QoS equal to EF or at least AF1x or whatever the supported service differentiation allows - Access to the IT resources should be prioritised
Silver	- Statistical Prioritisation for the service - Guarantee the minimum requirements in terms of resources as those are specified by the NS, however able to provide additional resources in case they are available on the NFV-PoP. - Network resources could follow the established Assured Forwarding class service differentiation that has the same notion as the above for the IT resources - Access to the IT resources could be prioritised among different Silver services from multiple tenants or the same tenant (complicated)
Bronze	- Equal to a Best Effort service but with an asterisk - For IT resources No scaling is allowed - For Network resources No calling is allowed and the traffic is always mapped to Best Effort class. - The system guarantees the IT resources required for the service to be operational.

Table 2-4 Generic Networking service deployment flavours

In this way we would have a service deployment flavour that is defined independently of the kind of service, which is aligned with what is understood in the networking world for a deployment flavour, unlike the expected performance of the specific service by means of QoS parameters, that is typically part of the SLA as it has been explained in the previous section.

At the current stage we consider this as future work.

2.3. SLA module architecture

Figure 2-2 shows where the SLA management module is located within the Marketplace architecture. It has external interactions with the Dashboard, the Accounting module and the NFV Orchestrator.

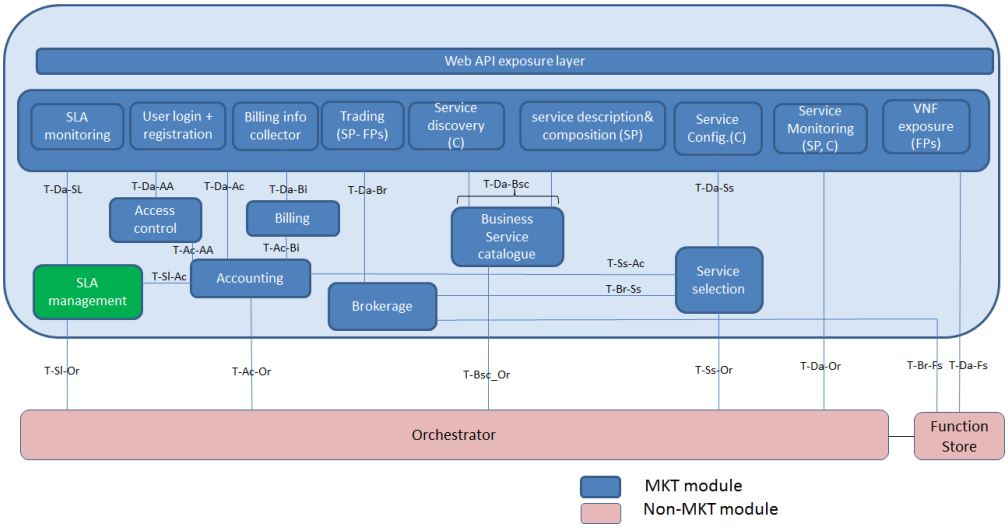


Figure 2-2 SLA management module in the Marketplace architecture

- The SLA module receives the input SLA templates from the dashboard after each VNF and service is created.
- The SLA module receives the input SLA agreements once the contracts (SP-FP and Customer-SP) have been established.
- The information of the monitoring of the services and VNFs comes from the orchestrator Service Monitoring Component which is responsible for monitoring all the service-related metrics that will be specified inside the NSD.
- The output of the SLA module is to the Accounting module, which receives the results of the SLA assessment: SLA violations and penalties of the running agreements for later proper billing.

The internal architecture of the SLA module is shown in Figure 2-3.

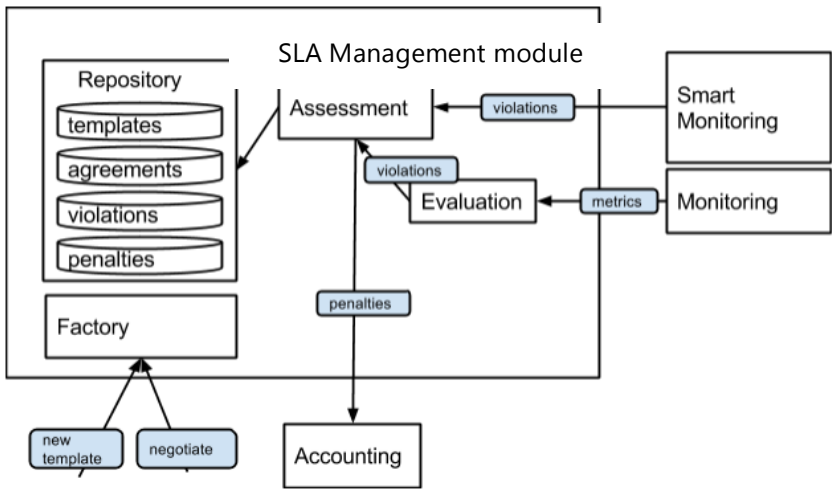


Figure 2-3 SLA management module internal architecture

The internal sub-modules have the following functions:

- Factory, that parses the templates and the agreements produced after the negotiation phase before storing them.
- Evaluation, that decides whether the values received from the monitoring constitute a SLA violation or not.
- Assesment, that calculates the penalties based on the produced violations.
- Repository, where all the templates, agreements and the information related to the violations and penalties generated from the SLA assesment are stored.

The first input of SLA management modules, SLA templates and SLA agreements after the negotiation phase. The templates as well as the agreements parsed are stored in the internal database.

After that, the second kind of input comes from the monitoring system. The SLA management module can work with the different kind of monitoring systems available:

- Simple monitoring systems that must be polled in order to retrieve the metrics or that are able to push the metrics into the SLA core once they are available.
- Smart monitoring systems that are able to evaluate the constraints, and raise the appropriate violations.

In the T-NOVA case, we collect raw data from the monitor in the Orchestrator that provides the metrics' monitoring data on request.

Once the metrics are evaluated and the SLA violations are generated as a result of this evaluation, the assessment calculates the penalties that are deducted from the violations and that's the output to the accounting system for billing purposes. The violations and penalites occurred are stored in the internal database for future consultation.

The information on how to proccess all this information is in the SLA agreements: what is considered a violation and which kind of penalty is applicable in each case.

2.4. T-NOVA SLA Workflow

As explained section 2.2 the VNFs and the Network Services (NSs) are taken as distinct products in terms of the contract, i.e. they have different agreements and are evaluated individually.

Figure 2-4 represents the workflow in the SLA process in T-NOVA Marketplace. Next the workflow is explained from each stakeholder perspective: Function Provider, Service Provider and customer.

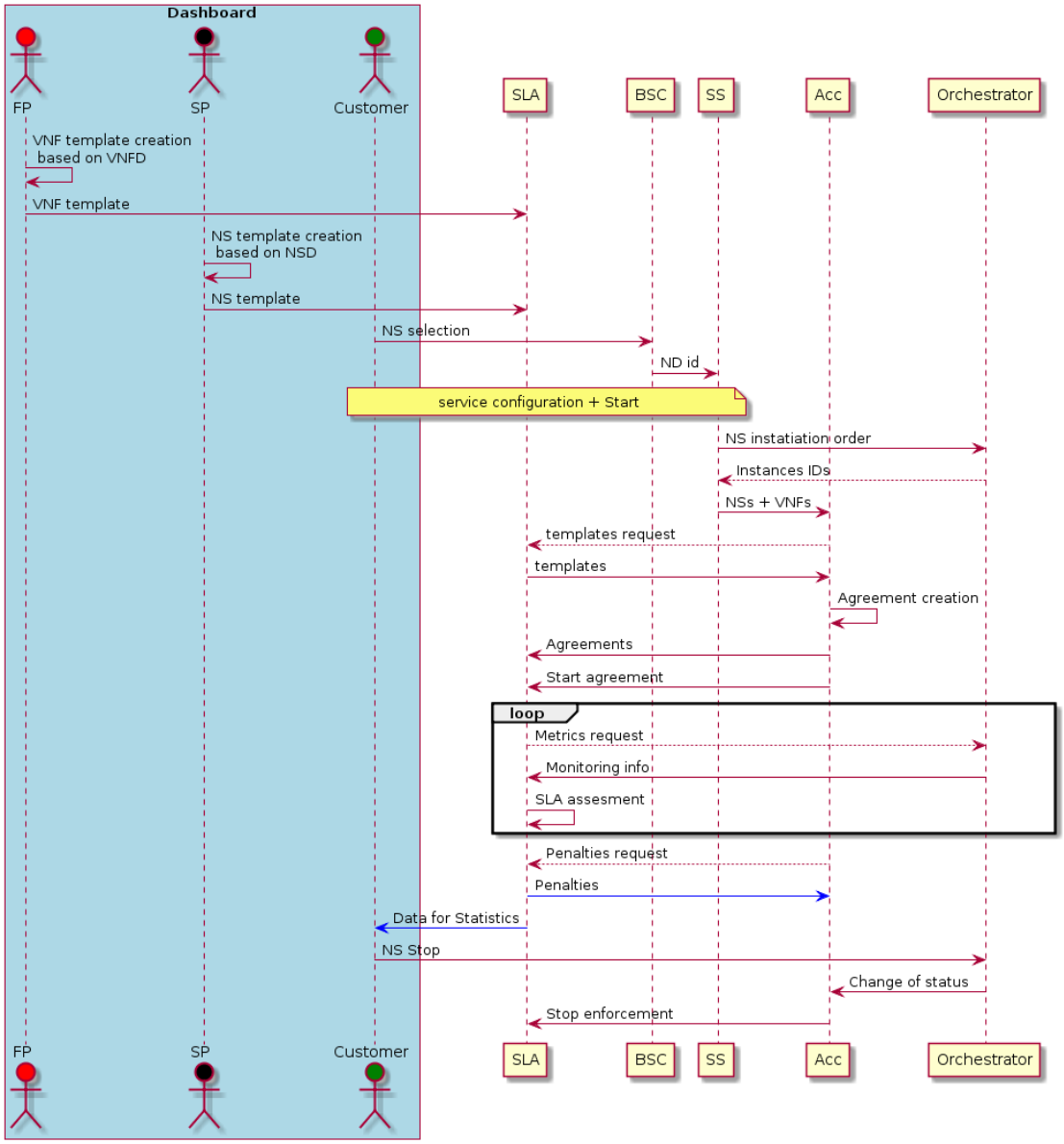


Figure 2-4 SLA workflow sequence diagram

2.4.1.1. Function Provider (FP) SLA workflow

The FP's SLA workflow starts when the FP defines a VNF in the VNF descriptor: the SLA template is then created based on this VNFD . Each VNFD may include several flavors for a single VNF. Within each flavor the following information is specified:

1. Hardware requirements (VDU (Virtual Deployment Unit)) for eachVNFC and Virtual Links connecting those VNFCs, needed to achieve the performance expressed by max o min values for certain metrics.
2. Metrics:
   - Name
   - Value

- Min. or max (whether the value is a minimum or a maximum).

3. Violations: Number of breaches within a range of time for each metric.
   - Count
   - Interval (secs)
4. Penalties: What to do if SLA is violated for each metric.
   - By default, the penalties are going to be discounts on the price the SP pays the FP for the usage of each one of his/her VNF instances.
   - Params: Example of a 10% discount in that metric for 1 day.
   - type: "discount"
   - expression: "10" <value_of_the_discount>
   - unit: "%" <unit_of_the_expression>
   - validity: "P1D" <period>

2.4.1.2. Service Provider (SP) SLA workflow

The SP's SLA workflow starts when the SP creates a network service by combining VNFs. The SLA template which of the offered SLA (SLA specification) is created based on the NSD of the recently defined service. The following information must be specified within the NSD related to the SLA:

1. Metrics are going to be part of the SLA among the metrics inherited from the VNFs with the posibility of combining the ones of the same kind from different VNFs.
2. Custom added metrics for the Service (if any)
   - E.g. Availability
3. Violations: Number of breaches within a range of time for each metric.
   - Count
   - Interval (secs)
4. Penalties: What to do if SLA is violated for each metric.
   - By default, are going to be discounts on the price the Customer must pay for the usage of the service.
   - Params: Example of a 10% discount in that metric for 1 day.
   - type: "discount"
   - expression: "10" <value_of_the_discount>
   - unit: "%" <unit_of_the_expression>
   - validity: "P1D" <period>

2.4.1.3. Customer SLA workflow

The Customer's SLA workflow starts when the Customer selects a network service for purchase and later use. This workflow is as follows:

1. The Customer selects a (Network) Service (NS) with specific SLA defined. (The same service can be offered with different SLAs).
2. The SLA agreements between the function providers of the VNFs that participate in the Service and the service provider are automatically generated by the accounting module based on the templates already created and adding some extra information:

- the begining date of the contract is added.
- the purchased VNFs (vnfId, providerId, dates, paymentMethod…) is Introduced for SLA tracking.

3. The SLA agreement between the SP and the Customer is automatically generated by the accounting module based on the templates already created and adding some extra information:
   - the begining date of the contract is added.
   - the purchased Service (serviceId, providerId, dates, period, paymentMethod…) is introduced for SLA tracking.
4. All the agreements we have just introduced are started, again by the accounting module.

While services and VNFs are running, the SLA module will collect the monitoring information for each metric that is part of an SLA from the monitoring service (Orchestrator) every minute and verify whether the agreements are meeting, storing the outcome of this process (possible penalties) in the internal database.

The SLA module is queried by the Accounting module and the Dashboard by means of its API. The Accounting module does it to gather violations and penalties information for billing purposes and the Dashboard to build statistics for the users.

This cycle will go on until the customer decides to stop the use of a service:

5. Stop & Release:

   1. Stop and Terminate the Service agreement.
   2. Stop and Terminate the agreement corresponding to each of the VNFs.
   3. Keep all the involved agreements for future statistics.

2.5. Implementation

2.5.1. WS-agreement

The SLA core is WS-Agreement [27] compliant (see more details in annex 6.1). As such, this document uses the terms used in the specification.

The WS-Agreement language defines the data types for expressing the content of an agreement. This language is defined independently from the WS-Agreement protocol and can therefore be used in a wide set of scenarios, for example with other protocol bindings. It is defined in the form of XML schema (although for T-NOVA we have translated the schema to the JSON format) and describes the data types and the structure of the Agreement document, the Agreement Template document, and the Agreement Offer document. The WS-Agreement specification defines two separate schemata, the agreement schema and the agreement state schema. The agreement schema that defines the WS-Agreement core data types and the agreement state

schema that includes the data types for the dynamic agreement monitoring, namely the agreement states, service term states and guarantee term states.

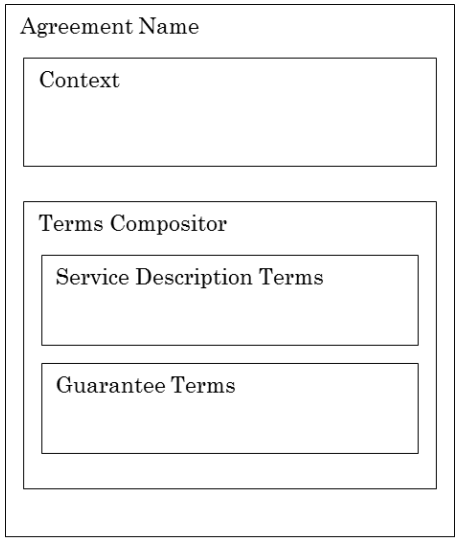


Figure 2-5 Basic structure of an SLA agreement

- An SLA agreement contains an agreement identifier, its name, an agreement context and a term compositor with a detailed description of the service to provide:Name: Optional Name
- Context: The context describes 'meta-data' of the whole Agreement, including and agreement Life-time and a template name.
- Term Compositor: This represents a scheme to compose an AND/OR/XOR relationship of the following two elements.
    - Service Description Term: Contains the information needed to instantiate or identify a service to which this agreement pertains.
    - Guarantee Term: Service Levels that the parties are agreeing to. Basically, the KQIs of the service, the SLA thresholds and the applicable penalties.

2.5.2. Implementation guidelines

The development of the SLA module is based on the following guidelines:

- A Provider offers a Service: network service or VNF.
- The service is described by ServiceDescriptionTerms with a Domain Specific Language. The ServiceDescriptionTerms are intended to define a service that has to be provisioned. This SLA module needs external provision.
- The service is represented by a Template, and the Template can be used to generate an Agreement.
- An agreement is a "document" that associates a Service and a Consumer. When the relation is in negotiation-phase, it's called an AgreementOffer. Once the agreement is accepted, it's called a Contract (this name is not used by the spec).

- A Template and an Agreement can describe some restrictions to be fulfilled by the Consumer or by the Provider.
- A violation of any restriction generates a violation.
- A violation is subdivided in breaches. A certain amount of breaches in a specified range of time constitute a violation.
- Introduced the field "requirements" inside serviceDescriptionTerm where it's described the initial requirements (typically hardware) for the service to be offered.

The WS-Agreement specifies in the context element who is the provider and the consumer, with the elements:

- AgreementInitiator (some kind of initiator identifier; OPTIONAL),
- AgreementResponder (some kind of responder identifier; OPTIONAL),
- ServiceProvider (=AgreementInitiator | AgreementResponder)

Usually, the consumer is the initiator. Although ws-agreement specifies AgreementInitiator/Responder as optional, it is recommended to specify them in the SLA template and agreement

SLA templates are created based on each VNFD and NSD. The SLA template is a draft of the contract that the involved parts will sign once the product (VNF or service) is acquired with minor modifications.

Example of an existing T-NOVA SLA template in JSON format:

```
{
    "context": {
        "agreementInitiator": null,
        "agreementResponder": "providerajax",
        "service": "TC / should an ontology be defined or this is free text
input?",
        "serviceProvider": "AgreementResponder",
        "templateId": "vnf3a2971d0-2eae-11e5-a2cb-0800200c9a66calls5k"
    },
    "name": "nombre",
    "templateId": "vnf3a2971d0-2eae-11e5-a2cb-0800200c9a66calls5k",
    "terms": {
        "allTerms": {
            "guaranteeTerms": [
                {
                    "businessValueList": {
                        "customBusinessValue": [
                            {
                                "count": 1,
                                "penalties": [
                                    {
                                        "expression": 5,
                                        "type": "discount",
                                        "unit": "%",
                                        "validity": "P1D"
                                    }
                                ]
                            }
                        ]
                    },
                    "name": "pepito",
```

```
                    "qualifyingCondition": null,
                    "serviceLevelObjetive": {
                        "kpitarget": {
                            "customServiceLevel": " { \"policies\": [ {
\"count\" : 2, \"interval\": 30 } ], \"constraint\" : \"pepito GT 0.5\" }",
                            "kpiName": "pepito"
                        }
                    },
                    "serviceScope": null
                },
                {
                    "businessValueList": {
                        "customBusinessValue": [
                            {
                                "count": 1,
                                "penalties": [
                                    {
                                        "expression": 5,
                                        "type": "discount",
                                        "unit": "%",
                                        "validity": "P1D"
                                    }
                                ]
                            }
                        ]
                    },
                    "name": "juanito",
                    "qualifyingCondition": null,
                    "serviceLevelObjetive": {
                        "kpitarget": {
                            "customServiceLevel": " { \"policies\": [ {
\"count\" : 2, \"interval\": 30 } ], \"constraint\" : \"juanito GT 0.7\"
}",
                            "kpiName": "juanito"
                        }
                    },
                    "serviceScope": null
                }
            ],
            "serviceDescriptionTerm": {
                "name": "requirements",
                "requirements": [
                    {
                        "name": "virt_mem_res_element",
                        "value": 6,
                        "unit": "GB"
                    },
                    {
                        "name": "CPU",
                        "value": 6,
                        "unit": "cores"
                    },
                    {
                        "name": "TLB size",
                        "value": 1024,
                        "unit": ""
                    },
                    {
                        "name": "storage",
```

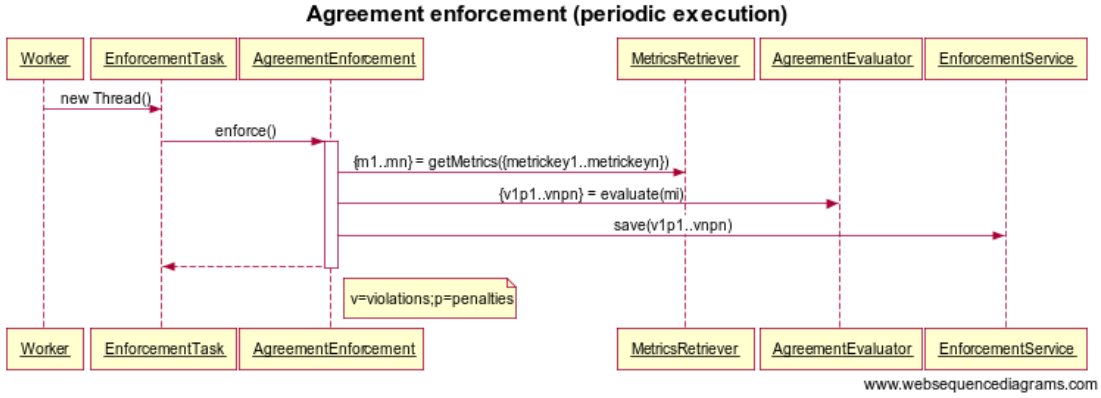
```
                        "value": 20,
                        "unit": "GB"
                    }
                ],
                "serviceName": "calls5k"
            },
            "serviceProperties": [
                {
                    "name": "MonitoredMetrics",
                    "serviceName": "default",
                    "variableSet": {
                        "variables": [
                            {
                                "location": "/monitor/pepito",
                                "metric": "xs:double",
                                "name": "pepito"
                            },
                            {
                                "location": "/monitor/juanito",
                                "metric": "xs:double",
                                "name": "juanito"
                            }
                        ]
                    }
                }
            ]
        }
    }
}
```

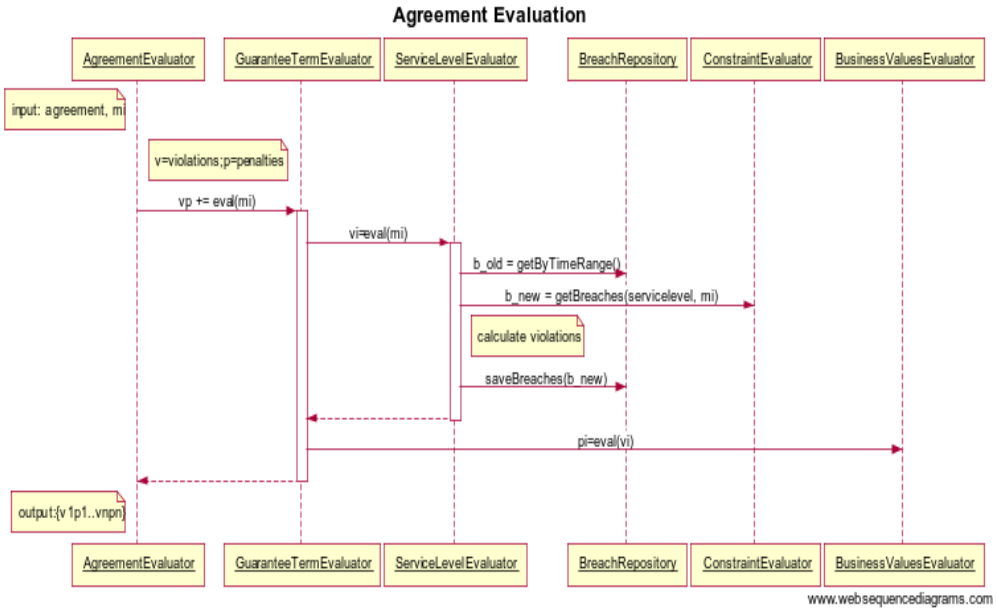

The format of the SLA agreement is exactly the same as the template but filling in the purchaser (AgreementInitiator) and modifying the information that has changed in the negotiation between the seller and the buyer.

2.5.3. Enforcement (assessment)

The enforcement is the process by which it is evaluated that the provider complies with an agreement, i.e. the measured metrics for the variables in guarantee terms fulfill the constraints. Actually, 'assessment' would be a more accurate term but we keep 'enforcement' to be compliant with the ws-agreement specification.

The enforcement process is depicted in the following sequence diagrams:





The job of each class is the following:

- AgreementEnforcement: retrieves the metrics needed to enforce an agreement (if periodic enforcement execution), calls the AgreementEvaluator and saves the detected violations and compensations in repository.
- AgreementEvaluator: calls the GuaranteeTermEvaluator for each guarantee term in the agreement.
- GuaranteeTermEvaluator: calls the ServiceLevelEvaluator, obtaining the raised violations; calls the BusinessValuesEvaluator using the violations as input.
- ServiceLevelEvaluator: calculates the violations. The implemented evaluator uses the concept of policy, where a number of violation metrics (a breach) must occur in a period of time (specified in the policy) to raise a violation. More details below.
- ConstraintEvaluator. Parses the service level constraint, and evaluates if a metric fulfills the constraint.
- MetricsRetriever. Actively queries for the last metrics of an agreement. Used in periodic execution.
  MetricsReceiver. Passively receives the last metrics of an agreement. Used in on-demand execution. The frontend for the receiver can be a REST service, a Message Queue, etc.

2.5.3.1. Metrics Retriever

The monitoring system is independent from the SLA module. For this reason, an adapter has to be implemented in order for the SLA retrieve the metrics needed to do the assessment. The way the external monitoring system communicates with the SLA module will be described later on the integration section 2.6.

2.5.3.2. Constraint evaluation

A simple constraint definition and evaluation is implemented, where a constraint can be defined by the following grammar:

```
E -> id OP VALUELIST
OP -> "GT" | "GE" | "EQ" | "LT" | "LE" | "NE" | "BETWEEN" | "IN"
VALUELIST -> "(" value "," value ")"
VALUELIST -> value
```

Only float values are allowed.

For example, the following constraints are valid:

```
responsetime LT 200
availability EQ 1
voltage BETWEEN (4.5, 5.5)
status IN (200, 204)
```

A constraint is satisfied if a metric evaluate the condition to true. Otherwise, the metric is considered as a Breach.

2.5.3.3. Policies

The policies used in C4S are an interesting feature not covered by the WS-Agreement.

A policy is compound by:

- a date interval
- a number of occurrences

The objective of a policy is specify when to raise a violation. Instead of raise a violation on every constraint breach, it's raised when a number of breaches are found within the specified interval.

WS-Agreement does not make room to define something like a policy in a ServiceLevelObjective, so we are going to define the policy in the CustomServiceLevel, along with the constraint definition. So, it's proposed to define the CustomServiceLevel like this:

```
{
  policies: [ { count : 2, interval: 120 }, { count: 2, interval: 3600 ],
  constraint: "responsetime LT 100"
}
```

So, if a service provider wants to offer policies in their SLA, they have to be compliant with this format. The constraint string is still totally domain defined.

2.5.3.4. Business rules (penalties)

A simple and generic implementation of business rules has been included in the core. Each guarantee term may have a BusinessValueList element where the penalties of not satisfying a guarantee term are defined.

The business structure defined in the ws-agreement specification is not expressive enough for our purposes, so that structure is basically ignored, and the

implementation makes use of CustomBusinessValues to define the business rules as a generalization of the standard penalties.

The recognized xml structure is:

```
<wsag:BusinessValueList>
  <wsag:Importance>xs:integer</wsag:Importance>?
  <wsag:CustomBusinessValue count="xs:integer" duration="xs:duration">
    <sla:Penalty
      type="xs:string"
      expression="xs:string"
      unit="xs:string"
      validity="xs:string"
    />*
  </wsag:CustomBusinessValue>*
</wsag:BusinessValueList>
```

Count and duration attributes are optional. If not specified, this CustomBusinessValue applies at each violation. Otherwise, it applies only if `count` violations occur in a `duration` interval of time.

The interpretation of every Penalty attribute is up to an external accounting module, but the intended meaning is:

- type: kind of penalty (e.g: discount, service, terminate)
- expression, unit: value of the penalty (e.g. discount of (50, euro), discount(100, %), service(sms))
- validity: interval of time where the penalty is applied

Each time a violation is generated, the assessment calculates if a business rule must be applied. If so, the corresponding Penalty is saved, and passed to the notification component.

The parsing of the business value list is performed in the IModelConverter. If a project wants to use a different xml structure, it can write the jaxb classes and a new BusinessValueListParser. The assignment of the parser to the model converter is done in the applicationContext.xml. The class to assign is defined in the configuration.properties file.

2.5.4. License

The SLA module code is released under the Apache License, Version 2.0 [28].

The way the SLA module is developed permits an easy integration, which will make possible the use of this module in as many contexts as possible, allowing a minor personalization.

Also the chosen kind of license helps to make the choice of using our SLA module over any other similar solution easier.

2.6. Integration

The different modules in the T-NOVA marketplace are integrated using Docker [29]. Each module is a different docker container and the inter-module communications are via REST API.

In T-NOVA, the SLA management module has interactions with different parts of the system on three different interfaces: to the Dashboard, the Accounting module and the NFV Orchestrator. How these interactions should take place in explained in detail in the next section.

The SLA module micro service is deployed within the Marketplace docker structure in a separate container. To be able to do so, the general *docker-compose* file needs to have a section dedicated to the SLA module and its dependencies.

Once we have the dependencies fulfilled, it's time to configure the container. A *dockerfile* (see annex) tells how to create the container and the script *DockerStart.sh* (see annex) tells how to execute it.

The SLA module relies on a MySQL database that is deployed in a different container (it's used by several modules) and we only have to create the database and the tables. This is done in the MySQL initialization file.

2.6.1. SLA module API definition

The operations supported by the SLA API are exposed to the following modules: dashboard (T-Da-Sl), accounting (T-Ac-Sl).

2.6.1.1. Dashboard interface (T-Da-Sl)

The following operations supported by the SLA API are exposed to the dashboard.

(a) Create a provider. (The uuid can be specified in the request)

URL	/providers
Type	POST
Headers	Accept: application/json Content-type: application/json
Parameters	
Response code	409: The uuid or name already exists in the database. 201: Created.
Request example	`POST /providers/ HTTP/1.1`

POST item example	`{`
	`"uuid":"fc923960-03fe-41eb-8a21-a56709f9370f",`
	`"name":"provider-prueba"`

	}

Table 2-5 SLA API operation to register a provider

(b) Create a new template.

The file might include a TemplateId or not. In case of not being included, a uuid will be assigned.

URL	/templates
Type	POST
Headers	Accept: application/json Content-type: application/json
Parameters	
Response code	▪ 409: The uuid already exists in the database. ▪ 409: The provider uuid specified in the template doesn't exist in the database. ▪ 500: Incorrect data has been suplied. ▪ 201: Created.
Request example	`POST /templates/ HTTP/1.1`
POST item example	SLA template (see annex section 2).

Table 2-6 SLA API operation to create a new template

(c) Update the template identified by *TemplateId*.

The body might include a *TemplateId* or not. In case of including a *TemplateId* in the file, it must match with the one from the url.

URL	/templates/{templateId}
Type	PUT
Headers	Accept: application/json Content-type: application/json
Parameters	▪ TemplateId: Id of the template we want to modify.
Response code	▪ 409: The *templateId* from the url doesn't match with the one from the file.. ▪ 409: Template has agreements associated. ▪ 409: Provider doesn't exist ▪ 500: Incorrect data has been supplied ▪ 200: OK
Request example	`PUT /templates/vnfvnf5gold HTTP/1.1`

PUT item example	SLA template (see annex section 2).

Table 2-7 SLA API operation to update the template identified by *TemplateId*

(d) Retrieve a template identified by *templateId*.

URL	/templates/{templatetId}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *TemplateId*: Id of the template we want to retrieve.
Response code	▪ 404: The *templateId* doesn't exist in the database. ▪ 200: OK.
Request example	`GET /templates/vnfvnf5gold HTTP/1.1`
Response example	SLA template in JSON form (see annex section 2)

Table 2-8 SLA API operation to retrieve a template identified by *templateId*

(e) Retrieve an agreement identified by *agreementId*.

URL	/agreements/{agreementId}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *agreementId*: Id of the agreement we want to retrieve.
Response code	▪ 404: The uuid doesn't exist in the database. ▪ 200: OK.
Request example	`GET /agreements/vnfidf51 HTTP/1.1`
Response example	SLA agreement in JSON form (see annex section 3)

Table 2-9 SLA API operation to retrieve an agreement identified by *agreementId*

The *AgreementId* matches the *AgreementId* attribute of *wsag:Agreement* element when the agreement is created.

2.6.1.2. Accounting interface (T-Ac-Sl)

The following operations supported by the SLA API are exposed to the accounting.

(a) Create a new agreement.

The body might include an *AgreementId* or not. In case of not being included, a *uuid* will be assigned. A disabled enforcement job is automatically created.

URL	/agreements
Type	POST
Headers	Accept: application/json Content-type: application/json
Parameters	
Response code	▪ 409: The uuid already exists in the database. ▪ 409: The provider uuid specified in the agreement doesn't exist in the database. ▪ 409: The template uuid specified in the agreement doesn't exist in the database. ▪ 500: Incorrect data has been supplied. ▪ 201: Created.
Request example	`POST /agreements/ HTTP/1.1`
POST item example	SLA agreement in JSON form (see annex section 3)

Table 2-10 SLA API operation to create a new agreement

(b) Start or stop an enforcement job.

An enforcement job is the entity which starts the assessment of the agreement guarantee terms. An agreement can be assessed only if an enforcement job, linked with it, has been previously created and started. An enforcement job is automatically created when an agreement is created, so there is no need to create one to start an assessment.

URL	/enforcements/{agreementId}/{command}
Type	PUT
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *agreementId*: Id of the agreement we want to start/stop. ▪ *command*: start | stop.
Response code	▪ 403: It was not possible to start the job. ▪ 200: OK.

Request example	`PUT /agreements/vnfidf51/start HTTP/1.1`

Table 2-11 SLA operation to start or stop an enforcement job

(c) Retrieve information from a penalty filtering by agreementId, metric name and dates.

URL	/penalties/{?agreementId,guaranteeTerm,begin,end}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *agreementId*: if specified, search the penalties of the agreement with this agreementId. ▪ *guaranteeTerm*: if specified, search the penalties of the guarantee term with this name (GuaranteeTerm[@name]), ▪ *begin*: if specified, set a lower limit of date of penalties to search, ▪ *end*: if specified, set an upper limit of date of penalties to search.
Response code	▪ 404: Erroneous data is provided in the call. ▪ 200: OK.
Request example	`GET /penalties/?agreementId=vnfidf51&guaranteeTerm =pepitovnf5&begin=2015-11-03T15:00:30CET&end=2015-11-03T17:00:30CET HTTP/1.1`
Response item example	`[{ "uuid": "5eaf2fa8-e533-4f76-8e78-cf7c3cce6b27", "agreementId": "vnfidf51", "datetime": "2015-11-03T16:10:30CET", "definition": { "type": "discount", "expression": "5", "unit": "%", "validity": "P1D" }, "violation": { "expectedValue": "pepitovnf5 GT 0.5", "actualValue": "0.16491713356365545", "kpiName": "pepitovnf5" } }]`

Table 2-12 SLA API operation to retrieve information from a penalty

2.6.2. Calls to other APIs

In order to implement T-Or-Sl it is the SLA module the one that calls the Orchestrator API for monitoring information. The format of this call is:

Request:

GET orchestrator/monitoring{?instance_type, instanceId, metric, date_begin, date_end, maxResults}

Parameters:

- *instanceType*: it can be "ns" or "vnf",
- *instanceId*: instance ID of the service or VNF,
- *metric*: name of the metric we want to obtain the monitoring data,
- *date_begin*: sets a lower limit of date of the monitoring,
- *date_end*: sets an upper limit of date of the monitoring,
- *maxResults*: if specified, limits the amount of results up to this number.

Response example:

```
HTTP/1.1 200 OK

Content-Type: application/json
[
   {
      "metricname": "packets_per_second",
      "value": 100,
      "date: "2015-01-21T18:49:00CET",
    }
]
```

3. T-NOVA BILLING AND ACCOUNTING

3.1. State of the Art analysis for Billing in T-NOVA

In [2] a detailed survey of billing models was done including internet players as Amazon [30], Google Play [31] and Apple Application Store [32] that use innovative combination of sharing revenue business models and marketplaces; also Telco API usage was considered, as in Telefónica's BlueVia [33] or Orange's Partner [34], where application developers receive a revenue share from Telco APIs usage by the final users.

On the other hand in cloud provider environments, such as Optimis project [35], different software licensing models are proposed and applied.

All these options were considered for the suitability of T-NOVA as it is explained in the following section.

3.2. T-NOVA Billing framework design

As explained in section 1.2, T-NOVA Marketplace involves two types of commercial relationships:

- The Service Provider (SP) acquires Virtual Network Functions (VNFs) from the Function Providers (FPs).
  The Customer acquires Network Services (NSs) provided by Service Providers based on the combination of VNFs previously purchased.

3.2.1. Billing for VNFs

One of the main benefits of T-NOVA approach is the opening of the telecom market to individual and SME software developers. By means of T-NOVA Marketplace, VNF developers (Function Providers, FPs) are allowed to create commercial offerings, advertising their algorithms materialized as virtual network appliances, as well as manage the whole commercial process to sell them, including SLA negotiation and billing. In order to attract as many FPs as possible and therefore maximize its impact T-NOVA will adopt a flexible go-to-market strategy including flexibility in choosing among several billing models for each particular case of VNF product that they are offering.

Based on the state of art analysis in [2] in T-NOVA four options for billing the SP for standalone VNFs have been considered:

- *Licensing*: FPs sell their VNFs to the SP using a software license. Since the Network Functions are no more than software applications, it seems logical to use this kind of purchase method. The FP concedes a license over his VNF to an SP for a specific time span, that is, during the duration of the contract the SP can make use of that VNF in unlimited Services and sell unlimited amounts of those services

to Customers. The SP will only pay once for the VNF. The leasing time may vary from a day to a lifetime. The following options are considered possible regarding the types of licensing:

- Option 1: The FP would issue licenses for several instances of each VNF to a SP. The SP will have to pay for all of them even if it is not using them, due to the fact that this use will depend on a customer purchasing a service composed by this VNF.
- Option 2: The SP will have open bar of instances of the purchased VNF and sell an infinite number of services containing that VNF.

- *Subscription*: The Customer establishes the billing period and the contract ending date will be extended automatically unless the Customer explicitly indicates it will not be extended any longer at the end of the current period.
- *Pay-As-You-Go (PAYG):* The client (Customer or SP) purchases a Service (or VNF), and he will only be charged for the time of use of that (instance of the) Service. The time counter starts once the Service is purchased. The client will make use of the service for several periods of time or for only a part of one. Usually for short periods of time, since it is on demand and therefore, more expensive. The client will be charged per hours or days and the service will be running until it's stopped by hand by the Client.
- *Pay-As-You-Earn - Revenue sharing (RS)*: The FP will get from the SP a percentage of the benefits the SP has obtained in the use of a VNF owned by the FP. I.e. FP specifies a % of participation. Once the VNF is purchased by a SP for the Service S1, the SP will pay the FP that % of the earnings for S1 during the life of S1. Payments will be done at the end of the life of S1 and at the end of every billing cycle that is agreed by the SP and the Customer at a higher level.

Either way, licensing model is neither fair nor profitable for the SP or for the FP and does not add any extra value compared to the other two methods (RS, PAYG), which are way more effective. Therefore we conclude that licensing is not suitable for T-NOVA ecosystem.

We have also come to understand that the subscription model does not contribute business wise, to a more resourceful billing system and furthermore, it is included in a PAYG model, this is, having a subscription for the use of a service or a VNF implies a closed period of use (renewable). But if the user decides to stop using the service, the remaining time until the end of the period will be charged anyway. In the best case, the subscription end of period will coincide with the end of the use and in such case; we will be talking about PAYG model.

The conclusion is that **Pay-As-You-Go** and **Revenue Sharing** are the most generic and thorough models to bill VNFs in T-NOVA and also the most beneficial for the FPs. Hence this approach is expected to provide the maximum T-NOVA impact attracting VNF developers. FPs will benefit from the **pay-as-you-earn** model, an extension of pay-as-you-go in which the VNF provider will pay a percentage of the revenue received. Bills to the FPs will be issued at the end of the life of the use of the service and at the end of every billing cycle. Therefore FPs will be able to advertise their VNFs in T-NOVA Marketplace for free, and based on the pay-as-you-earn model they will pay to the SP only when they get incomes for their VNFs.

3.2.2. Billing for Network Services

The same four billing models, as for the case of VNFs billing, were considered for network services billing: Licensing, subscription, Pay-As-You-Go and revenue sharing.

After studying all the possibilities, licensing, subscription and revenue sharing were discarded and PAYG remains as the ideal billing model for network services. The above statement is justified by the following arguments:

- Revenue sharing is unviable, due to the fact that the customer (which is the stakeholder to be billed in this case) pays for the service and does not receive any economic revenue out of it.
- Licensing method is discarded for the same reasons as for the VNFs case. One customer is unlikely to purchase more than one instance of the same service; therefore, in a real life scenario for this use case, license equals PAYG.
- Subscription: Same reasoning can be applied to this case; a customer subscribed to a service implies a closed period of use (renewable) but if the customer decides to stop using the service, the remaining time until the end of the period will be charged anyway. In the best case, the subscription end of period will coincide with the end of the use and that equals, again, to PAYG model.

3.2.3. Workflow

The workflow has been divided in four stages as it would happen in a real scenario: The VNFs creation, Service definition, Service purchase and start, Service lifecycle.

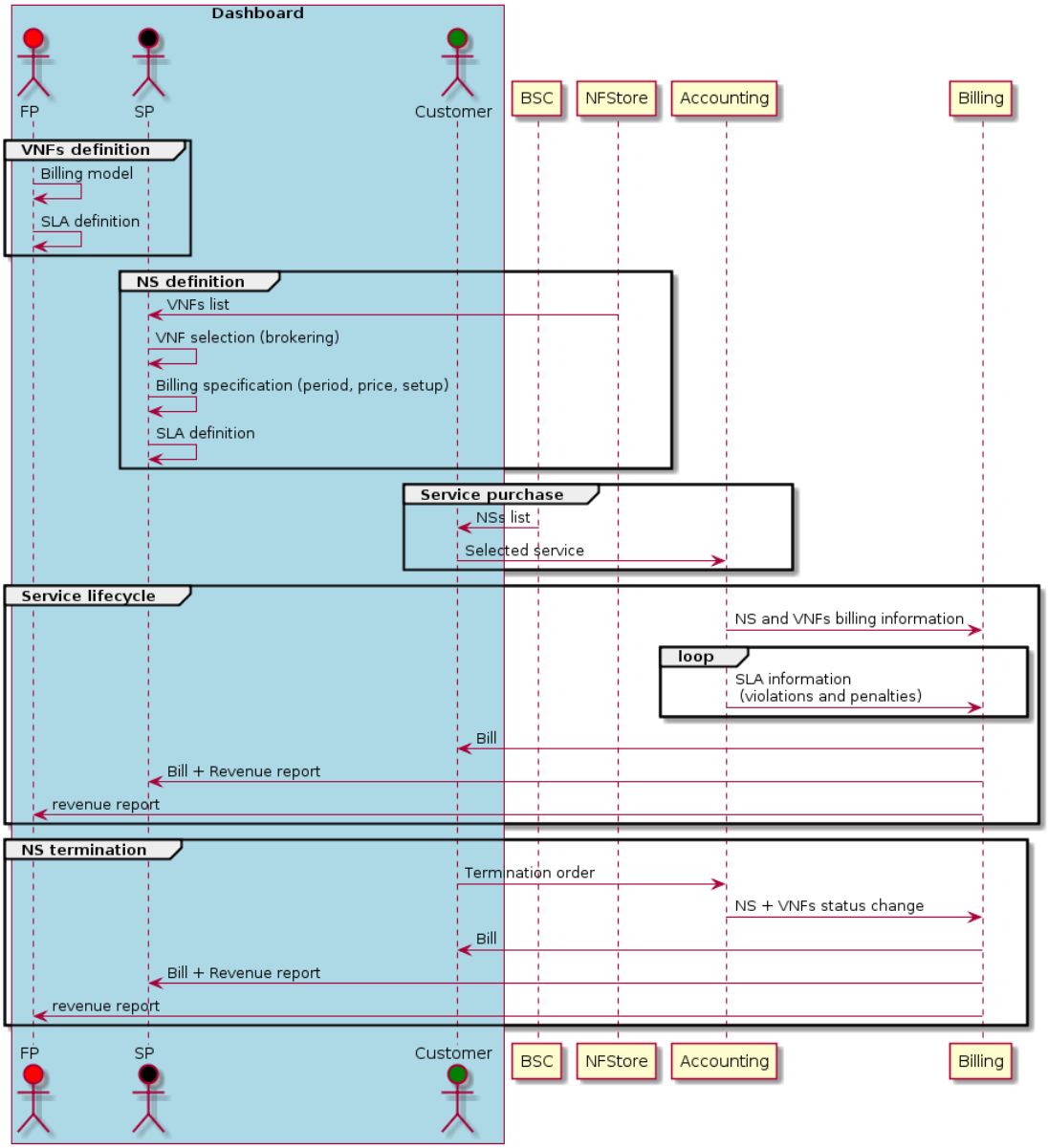


Figure 3-1 Accounting and billing workflow sequence diagram

1. The **FP** defines a new VNF from the dashboard (with different flavours).
   1.1. The FP selects the billing model: PAYG or RS:
      1.1.1. *PAYG*: The FP specifies a price per period (e.g. 1€/h). Since PAYG is indicated for short periods, the default billing cycle will be 1 month but at the previous rate.
      1.1.2. *RS*: The FP indicates only a percentage (%). The FP will receive the specified % off the earnings of the SP (in the Service where that VNF is used) for the use of one instance of the VNF.
   1.2. The FP defines the SLA: metrics to be monitored, violations and penalties and discounts associated to those, to be taken into account in the billing in case the SLA is unmet.

2. The **SP** defines a Service and for that, needs to purchase VNFs (it will be initially a reservation, as he will not be paying for them until they are used). The purchase of the VNFs is done by means of brokering process between the SP and the FPs in order to get the best VNF available in the Marketplace that suits the desires and at a minimum price.
   2.1. SP indicates the payment method for the service. *PAYG*.
      2.1.1. SP indicates a billing period, a price for that period, and a set up price. Ideal for short periods of time (less than 1 day).
   2.2. SP defines the SLA just like the FP did before.

3. The Service is purchased by a **customer**.
   3.1. Purchase orders (service and VNFs involved) are sent to the Accounting module along with billing related information: instance numbers, pricing, payment methods and timestamp for tracking purposes.

4. Service and VNFs are running
   4.1. SLA information is passed to the Billing module via the Accounting module in a loop until the customer decides to terminate the service.
   4.2. Billing module receives prices, periods, penalties, and discounts that help build a bill.
   4.3. The user can consult the bill of the services consumed (plus billing related information like earnings) which is issued every month.
5. Service termination
   5.1. SLAs are stopped and instances released.

3.3. Architecture

3.3.1. Overview

Figure 3-2 shows in green where the Accounting and Billing modules are located within the Marketplace architecture.

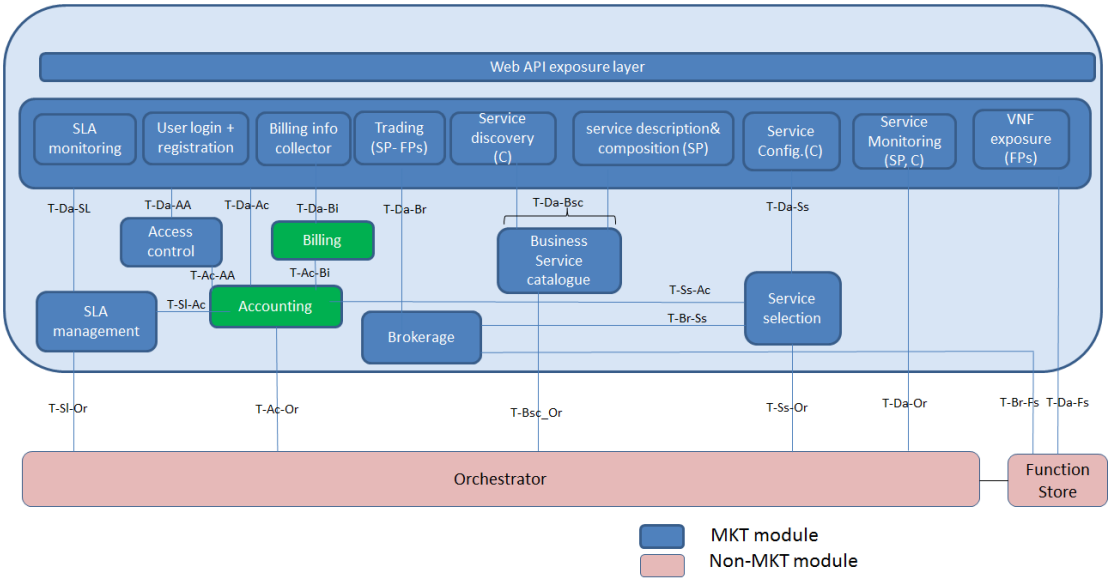


Figure 3-2 Accounting and billing modules in T-NOVA architecture

The Billing module is an external solution not custom-made for T-NOVA, therefore, an Accounting module was necessary to provide a gateway between the Billing and the rest of the system. It keeps a record of all the movements in the system that has a potential impact in the billing: when a service is instantiated, when is terminated, billing models and pricing. It also serves as a bridge between the orchestrator and the marketplace since it's the only one aware of the instance numbers and the associated SLA contracts as well as the users involved, providing the dashboard information about the running instances and their SLA.

3.3.2. Accounting

The accounting module is in charge of registering all the business relationships and events (subscriptions, SLA evaluations and usage) that will be needed for billing, being the the intermediate component between the billing module and the rest of the system:

- Provides the billing module all the information it needs.
- Keeps a track of the instances (services and VNFs)

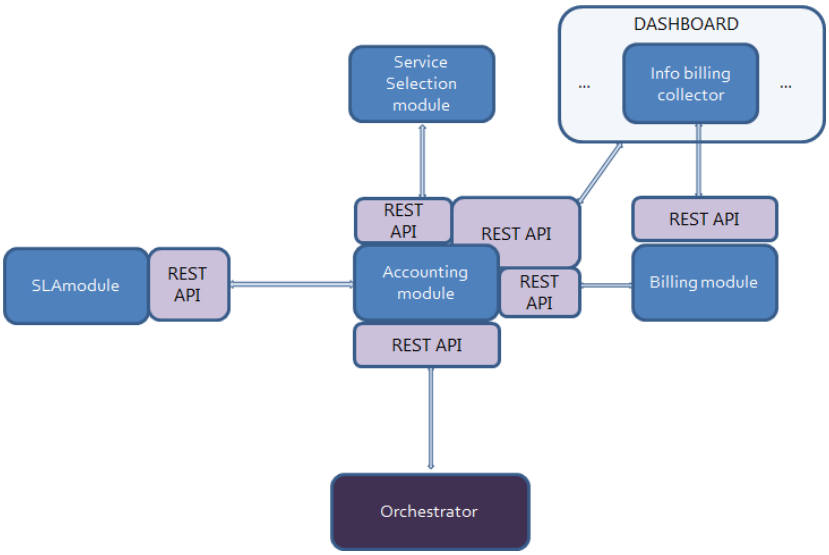


Figure 3-3 Accounting module interfaces

The Accounting module is created as a microservice for the marketplace and has interactions with several other microservices and entities:

Service Selection	Service selection introduces the starting information in the accounting once there is a confirmation from the Orchestrator that the service has been successfully instantiated. Stakeholders Ids, Instance Id, product Id, product type, SLA agreement, updated pricing information.
SLA module	Accounting creates the agreements in the SLA module based on the pre-created templates. Extracts SLA related information on billing module request: Penalties and violations.

Billing	Sends notification of the relevant billing events to the billing module: service start/stop. Sends (on request) all the SLA information that is needed for an appropriate billing.
Orchestrator	Receives notification of any change in the status of a service and modifies it accordingly.
AA	All communications need to be secure. For the marketplace communications we use JWT and by means of the AA we can validate each request.
Dashboard	Sends the list of running instances and VNFs to be shown and monitored in the dashboard. Sends (on request) SLA information to complement the service/VNF monitoring.

Table 3-1 Accounting module interfaces

3.3.3. Billing

Billing forms an integral part of the T-Nova Marketplace. Not only enables the service provider to charge and bill the customers for consumption of the services offered through the marketplace, but also aids in sustaining the FP ecosystem by enabling the revenue sharing between the SP and the FPs. The billing module being used in this project extends the generic rating-charging-billing (RCB) framework Cyclops [36], the development of which started in FP7 project Mobile Cloud Networking [37], and whose functionalities are being extended to support the marketplace and T-NOVA requirements.

The billing module (from here on referred as Cyclops) design is influenced by the micro-services design pattern wherein functionally independent pieces are created as a service with clear REST API interfaces to allow inter-service data exchange and communication. The modules by design are part of a distributed deployment wherein each service can be deployed on the same host or on separate hosts depending on the operational optimization goals. Having a micro-service design also allows for maximum reuse of the modules.

Cyclops' prominent micro-services are:

- Usage data records generation service
- Rating and charging micro-service
- Billing (bill generation and management) micro-service
- Messaging service
- Authentication and authorization module

The linkages and relationship among these micro-services are shown below in figure xx. The figure is created using the functional modelling concepts (FMC).

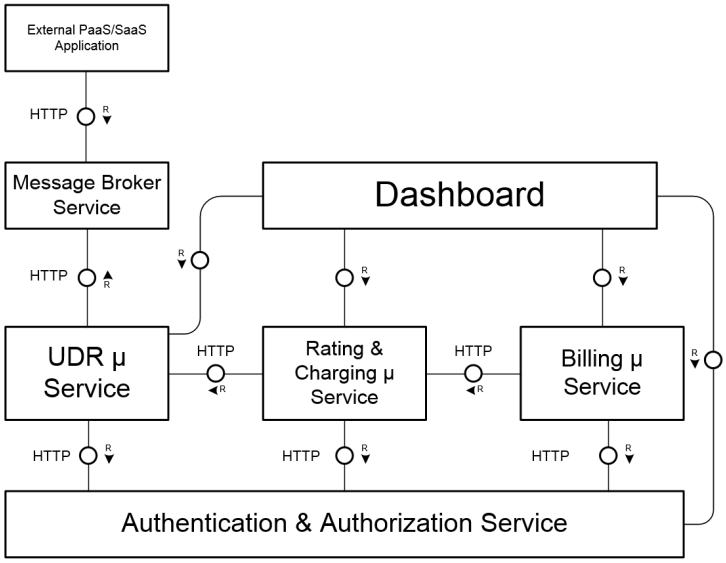


Figure 3-4 Cyclops micro-services architecture

3.3.3.1. Usage data record (UDR) generation micro-service

Usage data record micro-service is responsible for the collection of usage data from various sources. For OpenStack clouds, it has a built in driver, which depending on the configured monitoring intervals collects relevant metered data from Ceilometer service. For other services, the data points are processed from the messaging service queues. In T-NOVA, the accounting module sends relevant billing events for all services to the Cyclops messaging service.

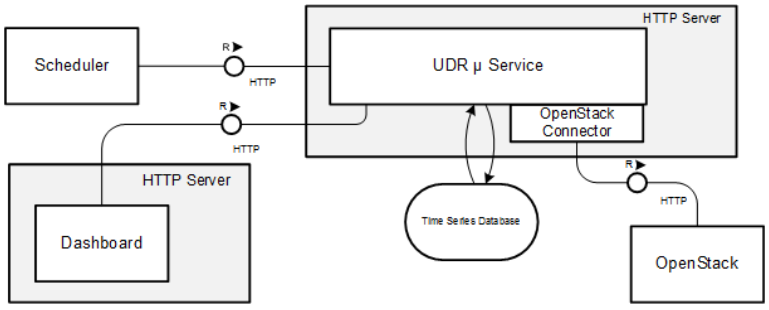


Figure 3-5 UDR micro-service architecture

UDR micro-service also constructs the usage records for every user periodically. And it provides data query API for other services. The data stored in UDR allows for rich data visualization and analytics.

3.3.3.2. Rating, charging (RC) micro-service

The RC micro-service is responsible for determining the correct rate for any resource type depending on the business rules, or billing model. In T-NOVA, for each service, the Accounting module keeps track of agreed billing and revenue sharing models for each customer. Cyclops RC service utilizes the Accounting APIs to get the models and

generates the rate data accordingly. It also processes the udr-records and transforms into charge-records (CDRs) periodically for each user.

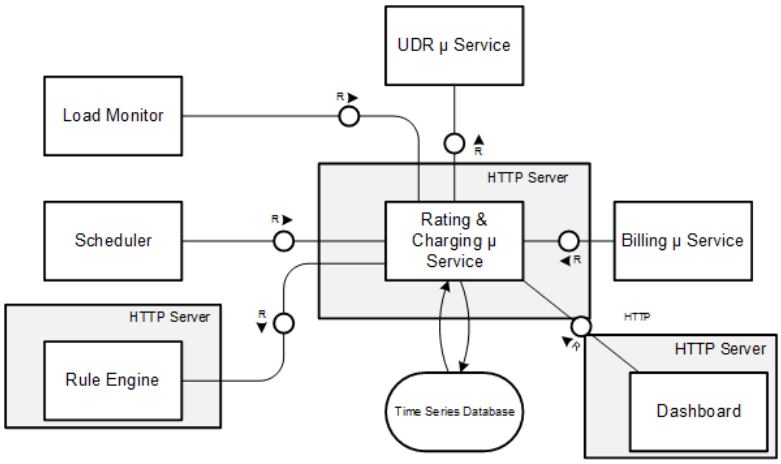


Figure 3-6 RC micro-service architecture

3.3.3.3. Billing micro-service

The billing micro-service in Cyclops aggregates all the CDRs for a given duration and generates the bill data. It also processes exceptions such as SLA violations and any promotional rules such as coupons and discounts before generating the bill.

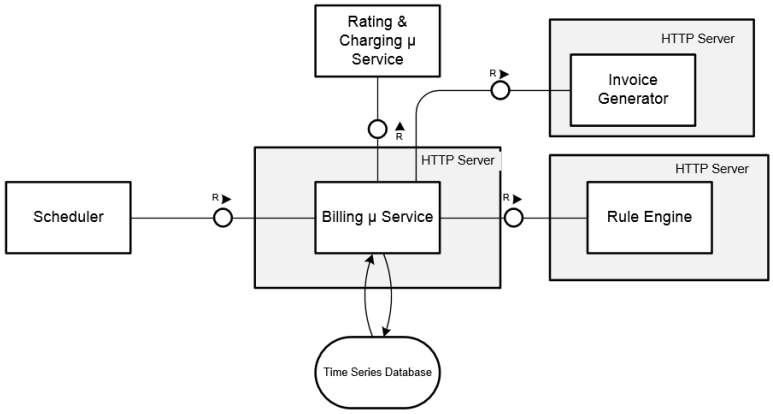


Figure 3-7 Billing micro-service architecture

3.3.3.4. Cyclops messaging service

Cyclops messaging allows external (non-natively supported) services to send relevant billing data into the framework for further processing into a unified billing strategy. The framework is capable of creating separate messaging queues for different data sources. The Accounting module sends the billing events to one such queue.

3.3.3.5. Authentication and authorization module

The micro-services authorize communication with each other through the auth-n/z module. T-Nova Gatekeeper service provides authorization and authentication service to Cyclops.

3.4. Implementation

3.4.1. Overview

The accounting module and the billing one are separate microservices. The accounting is developed entirely in python and the billing module in java. Both expose a RESTful API for intermarketplace communication.

3.4.2. Accounting

The accounting module is created to satisfy the needs of the billing module and to facilitate its integration with the rest of the T-NOVA system. Due to this, there wasn't any existing similar software in the market we could use, so a custom made one had to be developed.

The Accounting module implements a RESTful API, following the guidelines in the Marketplace

3.4.2.1. Information model

Based on the requirements, the information the Accounting module has to store to be able to provide suitable results is shown in the next table:

Id	Internal id of the Accounting module
productType	It can be a "ns" or "vnf".
instanceId	Id of the instance provided by the Orchestrator once the product is instantiated.
productId	Id of the product in the original store (NFStore or BSC).
agreementId	Id of the SLA contract in the SLA module.
relatives	In case the entry represents a function, this field would be the instance number of its father service. In case it's a service, it would represent the list of instance numbers of the VNFs that compose the service.
flavour	Flavour picked by the client among the offered ones in the original descriptor.
startDate	Timestamp of the product instantiation.
lastBillDate	Date and time when the last bill was generated.
providerId	Provider that offers the product.
clientId	Client that makes use of the product.
status	Current status of the instance in the system ("running" | "stopped")
billingModel	PAYG or RS
period	Base period for the billing.

priceUnit	Currency (typically "EUR")
periodCost	Price per period.
setupCost	Price of setting up the service or VNF.
renew	In case the subscription billing model is introduced, this field represents whether the customer wants to renew the service at the end of the period or not.
dateCreated	Date and time the entry was created in the Accounting module.
dateModified	Date and time the entry was modified. Used to know when there is a change in the status.

Table 3-2 Accounting module information model

This information is serialized in JSON format and the Accounting module offers different serializes for the external modules to extract the information they consider enough and not having to deal with the whole set.

3.4.2.2. License

The code of the module is released under Apache 2 license.

3.4.3. Billing

After a thorough review of relevant technologies and billing platforms available for reuse in the T-NOVA project, it was found that no suitable open-source framework had support for multi-actor scenarios which is envisioned in this project. The closest framework with necessary features was Cyclops, which was selected for further feature-extension and use in the T-Nova marketplace. The results of technology options evaluated can be found in [2].

3.4.3.1. UML Class diagrams of micro-services

The UML class diagrams below shows the implementation details of each micro-services. The framework is majorly implemented in Java as web-services which are deployed in tomcat containers.

(a) UDR micro-service class diagram

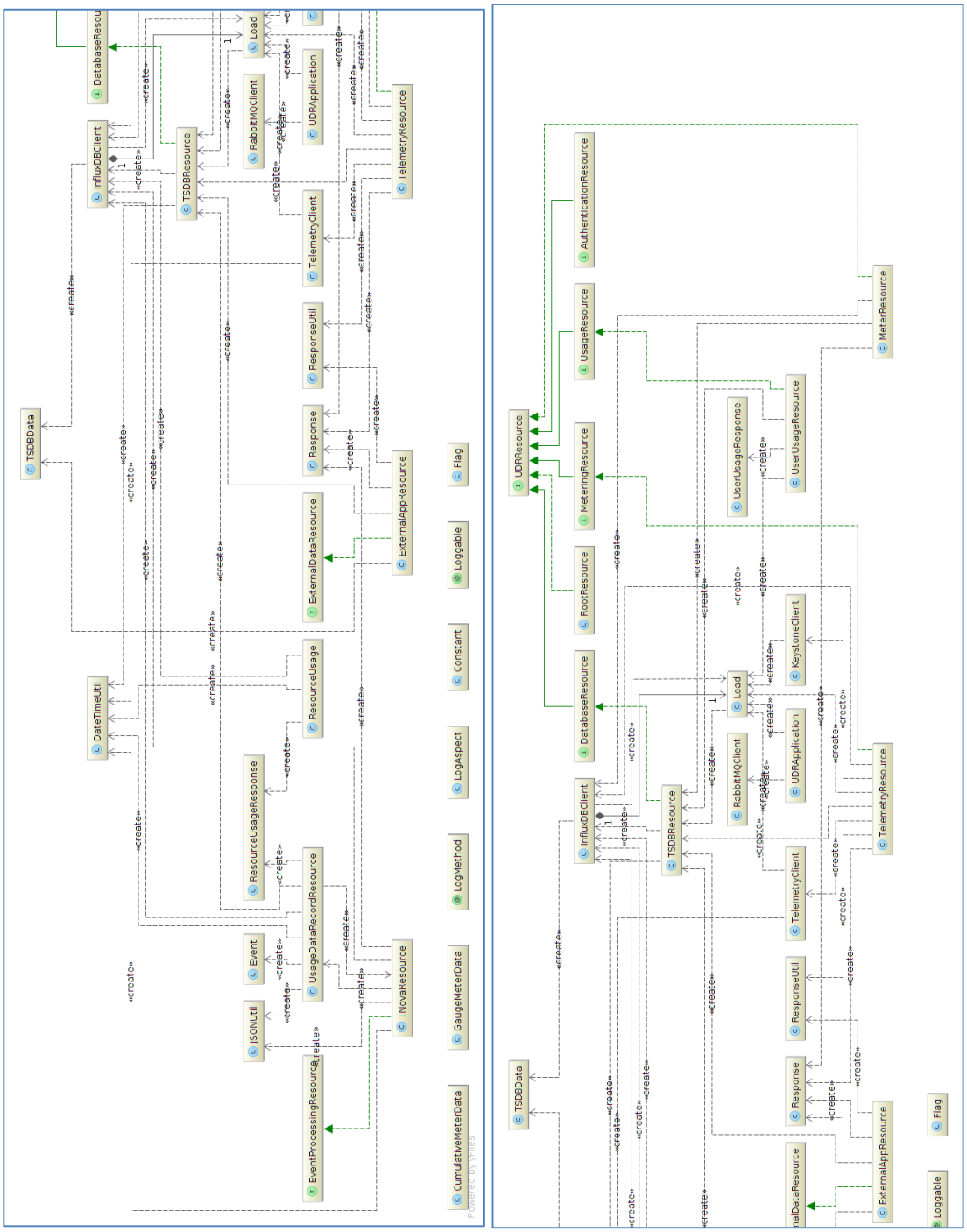


Figure 3-8 UML class diagram (split view) for UDR micro-service

The figure above shows the UML class diagram for the classes that are part of UDR micro-service. The figure is a split view representation as the single image was too big for including in this document.

(b) RC micro-service class diagram

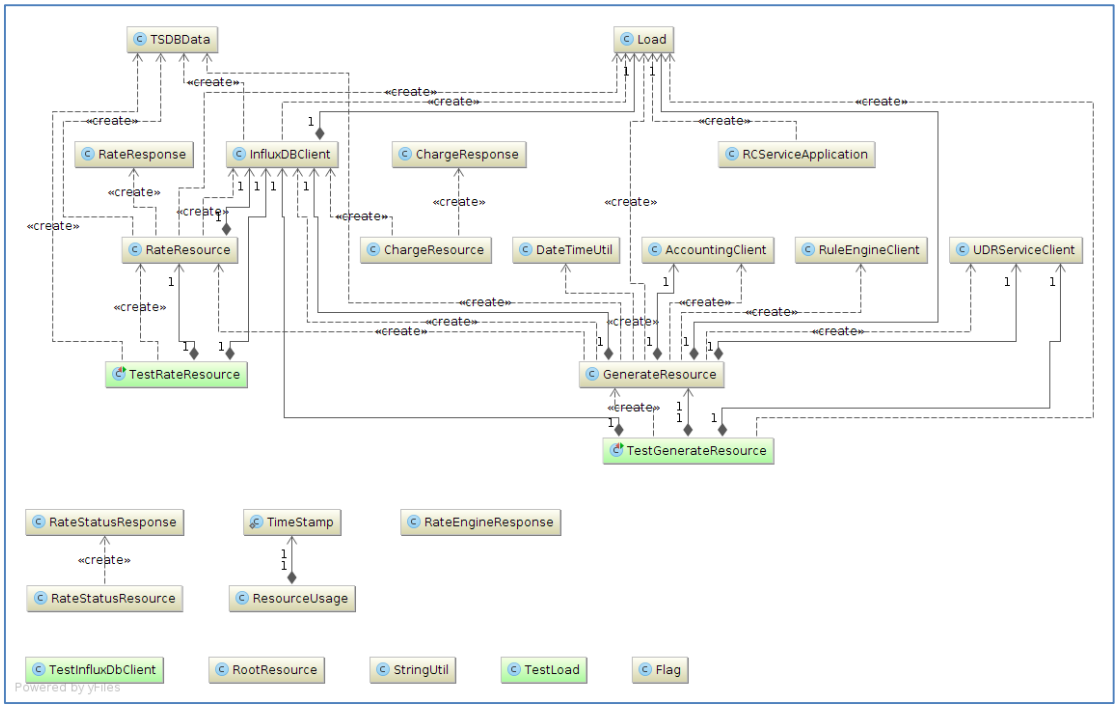


Figure 3-9 UML class diagram for rc micro-service

Figure 3-9 shows the implemented class UML diagram for the rating and charging (rc) micro-service.

(c) billing micro-service class diagram

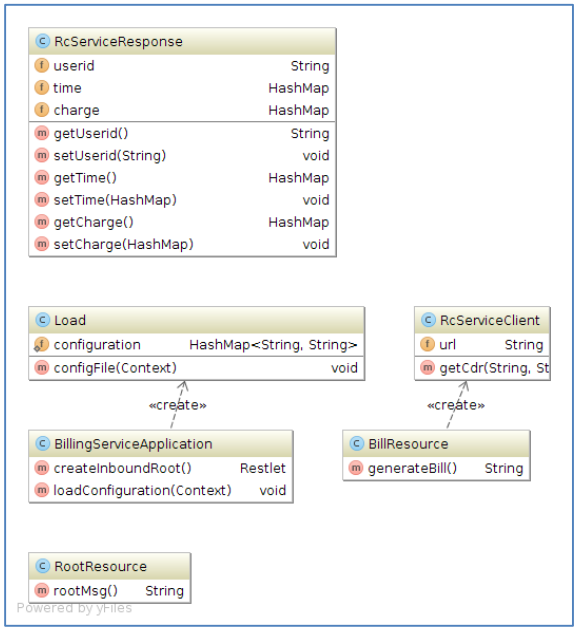


Figure 3-10 UML class diagram for billing micro-service

Figure 3-10 shows the UML class diagram for the billing micro-service. As can be seen from the complexity of the micro-services, the UDR micro-service is most complex as it has to ensure timely and correct data collection, as well as recovery workflows in case of collection failures. Furthermore, correct persistence in the time-series database ensures the later stages of the charging and bill generation works without errors.

In contrast, the billing micro-service which currently only does data aggregation and transformation into bill data is least complicate of all the core micro-services in Cyclops framework.

3.5. Accounting module Integration

The Accounting module has interfaces to 3 other components of T-NOVA: Billing module, Service Selection module, and Orchestrator. Each one with a set of REST API calls.

The Accounting module microservice is deployed within the Marketplace docker structure in a separate container. To be able to do so, the general *docker-compose* file needs to have a section dedicated to the Accounting module and its dependencies. In this case, the Accounting relies on a MySQL database that is deployed in a different container (it is used by several modules) and we only have to create the database and the tables. This is done in the MySQL initialisation file (see annex).

Once we have the dependencies fulfilled, it's time to configure the container. A *dockerfile* tells how to create the container and the script *DockerStart.sh* tells how to execute it.

3.5.1. Accounting module API definition

The operations supported by the Accounting API are exposed to the following modules: billing (T-Bi-Ac), service selection (T-Ac-SS), dashboard (T-Da-Ac), SLA (T-Ac-Sl), orchestrator (T-Or-Ac).

3.5.1.1. Billing interface (T-Bi-Ac)

The following operations supported by the Accounting API are exposed to the billing.

(a) Details about the client's chosen billing model and specs. for the queried service instance id. It returns a single element.

URL	/service-billing-model/?clientId={clientId}&instanceId={serviceInstanceId}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *clientId*: Id of the user we want the billing model details of. ▪ *instanceId*: String that univocally identifies a service instance.
Response	▪ 200: With empty results when erroneous data is provided in the

code	call. ▪ 200: OK.
Request example	`GET /service-billing-model/?clientId=c1&instanceId=id02 HTTP/1.1`

Response example	<pre>{
	"startDate": "2015-06-10T00:00:00Z",
	"lastBillDate": "2015-06-10T00:00:00Z",
	"billingModel": "PAYG",
	"period": "P1D",
	"priceUnit": "EUR",
	"periodCost": 1.5,
	"setupCost": 2.0
	}</pre>

Table 3-3 Accounting API operation to get details about the client's billing model

(b) Retrieve the list of all active services the user is using.

URL	/ service-instance-list/?clientId={clientId}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *clientId*: Id of the user from whom we will get the list of purchased and active services (optional).
Response code	▪ 200: With empty results when erroneous data is provided in the call. ▪ 200: OK.
Request example	`GET /service-instance-list/?clientId=c1 HTTP/1.1`

Response example	<pre>[
	{
	"id": 11,
	"instanceId": "id19",
	"productId": "s2",
	"agreementId": "vnf3a2971d0-2eae-11e5-a2cb-0800200c9a66calls5k",
	"relatives": "id04",
	"productType": "ns",</pre>

```
        "flavour": null,
        "startDate": "2015-06-10T00:00:00Z",
        "lastBillDate": "2015-06-10T00:00:00Z",
        "providerId": "p1",
        "clientId": "c1",
        "status": "running",
        "billingModel": "PAYG",
        "period": "P1D",
        "priceUnit": "EUR",
        "periodCost": 1.5,
        "setupCost": 2.0,
        "renew": false,
        "dateCreated": "2015-07-28T14:36:14Z",
        "dateModified": "2015-10-08T09:21:28Z"
    }
]
```


Table 3-4 Accounting API operation to get the list of all active services for a user

(c) Retrieve the list of all VNFs purchased by a particular provider (client).

URL	/vnf-list/?clientId={client_id}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *clientId*: Id of the user for whom we will get all the purchased VNFs.
Response code	▪ 200: With empty results when erroneous data is provided in the call. ▪ 200: OK.
Request example	`GET /vnf-list/?clientId=p1 HTTP/1.1`
Response element example	`{ "id": 1, "instanceId": "id01", "productId": "vnf1", "agreementId": "s1vnf2_4", "relatives": "id02",`

	"productType": "vnf", "flavour": null, "startDate": "2015-06-10T00:00:00Z", "lastBillDate": "2015-06-10T00:00:00Z", "providerId": "f1", "clientId": "p1", "status": "stopped", "billingModel": "PAYG", "period": "P1D", "priceUnit": "EUR", "periodCost": 1.5, "setupCost": 2.0, "renew": true, "dateCreated": "2015-06-11T13:29:16Z", "dateModified": "2015-11-03T10:53:40Z" }

Table 3-5 Accounting API operation to get the list of all VNFs purchased by a particular provider

(d) Details of the revenue sharing model between SP and FP for the given VNF instance.

URL	/vnf-billing-model/?spId={user_id}&instanceId={VNF__instance_id}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *spId*: Id service provider that purchased the VNF. ▪ *instanceId*: Id of the VNF instance we need the billing model of.
Response code	▪ 200: With empty results when erroneous data is provided in the call. ▪ 200: OK.
Request example	`GET /vnf-billing-model/?spId=p1&vnfId=vnf2 HTTP/1.1`
Response element example	{ "startDate": "2015-06-16T00:00:00Z", "lastBillDate": "2015-06-25T00:00:00Z", "billingModel": "PAYG",

	"period": "P1D", "priceUnit": "EUR", "periodCost": 1.0, "setupCost": 1.0 }

Table 3-6 Accounting API operatio to get details of the revenue sharing model between SP and FP for the given VNF instance

(e) List of all sla-violations for a given service instance for the queried time window and a give metric name.

URL	/sla/service-violation/?instanceId={service_instance_id}&metric={metric_name}&start={time-date}&end={time-date}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *instanceId*: Id of the service instance we want to obtain the SLA violations from. ▪ *metric*: Metric name, for filtering purposes. (optional) ▪ *start*: Starting date of the SLA violations time frame. (optional) ▪ *end*: Ending date of the SLA violations time frame. (optional)
Response code	▪ 200: With empty results when erroneous data is provided in the call. ▪ 500: There is a connection problem between the Accounting and the SLA modules. ▪ 200: OK.
Request example	`GET /sla/service-violation?instanceId=ids101&metric=juanitoservice6&start=2015-11-03T15:00:30CET&end=2015-11-03T17:00:30CET HTTP/1.1`

Response element example	{
	"agreementId": "serviceids101",
	"definition": {
	"expression": "5",
	"type": "discount",
	"validity": "P1D",
	"unit": "%"
	},
	"uuid": "8e440f6b-b5e0-4231-acf8-7ebc9d1d5e66",
	"violation": {
	"kpiName": "juanitoservice6",

	"actualValue": "0.43127626083203485", "expectedValue": "juanitoservice6 GT 0.7" }, "datetime": "2015-11-03T16:10:30CET" }

Table 3-7 Accounting API operation to get the list of all sla violations for a service

(f) List of all sla-violations for a given VNF instance for the queried time window and a give metric name.

URL	/sla/vnf-violation/?instanceId={vnf_instance_id}&metric={metric_name}&start={time-date}&end={time-date}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *instanceId*: Id of the VNF instance we want to obtain the SLA violations from. ▪ *metric*: Metric name, for filtering purposes. (optional) ▪ *start*: Starting date of the SLA violations time frame. (optional) ▪ *end*: Ending date of the SLA violations time frame. (optional)
Response code	▪ 200: With empty results when erroneous data is provided in the call. ▪ 500: There is a connection problem between the Accounting and the SLA modules. ▪ 200: OK.
Request example	`GET /sla/vnf-violation?instanceId=idf51&start=2015-11-03T15:00:30CET&end=2015-11-03T17:00:30CET&metric=pepitovnf5 HTTP/1.1`

Response element example	{
	"agreementId": "vnfidf51",
	"definition": {
	"expression": "5",
	"type": "discount",
	"validity": "P1D",
	"unit": "%"
	},
	"uuid": "5eaf2fa8-e533-4f76-8e78-cf7c3cce6b27",
	"violation": {
	"kpiName": "pepitovnf5",

	"actualValue": "0.16491713356365545", "expectedValue": "pepitovnf5 GT 0.5" }, "datetime": "2015-11-03T16:10:30CET" }

Table 3-8 Accounting API operation to get the list of all sla-violations for a VNF

(g) List of all active services that use the given VNF.

URL	/service-list/?vnfId={vnf_id}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *vnfId*: Id of the VNF we want to know in how many services it has been used (optional).
Response code	▪ 200: With empty results when erroneous data is provided in the call. ▪ 200: OK.
Request example	`GET /service-list/?vnfId=vnf5 HTTP/1.1`
Response element example	{ "id": 25, "instanceId": "ids100", "productId": "s5", "agreementId": "s1vnf2_4", "relatives": "idf50", "productType": "ns", "flavour": null, "startDate": "2015-10-08T07:09:19Z", "lastBillDate": "2015-10-08T07:09:19Z", "providerId": "p5", "clientId": "c5", "status": "running", "billingModel": "PAYG", "period": "P1D", "priceUnit": "EUR", "periodCost": 1,

	"setupCost": 2, "renew": false, "dateCreated": "2015-10-08T07:09:19Z", "dateModified": "2015-10-08T07:09:19Z" }

Table 3-9 Accounting API to get the list of all active services that use the given VNF

3.5.1.2. Service Selection interface (T-Ac-SS)

The following operations supported by the Accounting API are exposed to the service selection module.

(a) List of all entries in the Accounting system or a single one if the parameter *accountId* is present.

URL	/accounts/{accountId}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *accountId*: Id of the accounting entity we want to retrieve (optional).
Response code	▪ 200: With empty results when erroneous data is provided in the call. ▪ 404 - Not found: when the provided accountId does not exist in the database ▪ 200: OK.
Request example	`GET /accounts/11 HTTP/1.1`
Response element example	{ "id": 11, "instanceId": "id19", "productId": "s2", "agreementId": "vnf3a2971d0-2eae-11e5-a2cb-0800200c9a66call s5k", "relatives": "id04", "productType": "ns", "flavour": null, "startDate": "2015-06-10T00:00:00Z", "lastBillDate": "2015-06-10T00:00:00Z",

	<pre>"providerId": "p1",
	"clientId": "c1",
	"status": "running",
	"billingModel": "PAYG",
	"period": "P1D",
	"priceUnit": "EUR",
	"periodCost": 1.5,
	"setupCost": 2,
	"renew": false,
	"dateCreated": "2015-07-28T14:36:14Z",
	"dateModified": "2015-10-08T09:21:28Z"
	}</pre>

Table 3-10 Accounting API to get the list of all entries in the Accounting system or a single one if the parameter *accountId* is present

(b) Create a new accounting entry.

URL	/accounts/
Type	POST
Headers	Accept: application/json Content-type: application/json
Parameters	
Response code	▪ 400 - Bad request: when the body in the request is not well formed or when there is a problem with the SLA agreement creation. ▪ 408 - Request Timeout: when there is a problem with the status message queue. ▪ 201: Created.
Request example	<pre>POST /accounts/ HTTP/1.1</pre>

POST item example	<pre>{
	"instanceId": "id19",
	"productId": "s2",
	"agreementId": "vnf3a2971d0-2eae-11e5-a2cb-0800200c9a66cal ls5k",
	"relatives": "id04",
	"productType": "ns",
	"flavour": “silver”,
	"providerId": "p1",</pre>

	<pre>"clientId": "c1",
	"status": "running",
	"billingModel": "PAYG",
	"period": "P1D",
	"priceUnit": "EUR",
	"periodCost": 1.5,
	"setupCost": 2,
	"renew": false
	}</pre>

Table 3-11 Accounting API operation to create a new entry

(c) Update an existing accounting entry.

The content in the body will overwrite the content of the resource. The *dateModified* field will be updated with the current time.

URL	/accounts/{accountId}
Type	PUT
Headers	Accept: application/json Content-type: application/json
Parameters	▪ accountId: Id of the accounting entity we want to update.
Response code	▪ 400 - Bad request: when the body in the request is not well formed. ▪ 404 - Not found: when the provided accountId does not exist in the database. ▪ 200: OK.
Request example	<pre>PUT /accounts/11 HTTP/1.1</pre>

PUT item example	<pre>{
	"flavour": “gold”
	}</pre>

Table 3-12 Accounting API operation to update an existing entry

(d) Delete an existing accounting entry.

URL	/accounts/{accountId}
Type	DELETE
Headers	Accept: application/json Content-type: application/json

Parameters	▪ accountId: Id of the accounting entity we want to delete.
Response code	▪ 400 - Bad request: when the body in the request is not well formed. ▪ 404 - Not found: when the provided accountId does not exist in the database. ▪ 204: No content (OK).
Request example	`DELETE /accounts/11 HTTP/1.1`

Table 3-13 Accounting API operation to delete an existing accounting entry

3.5.1.3. Orchestrator interface (T-Or-Ac)

The following operations supported by the Accounting API are exposed to the orchestrator.

(a) Update the status of a service given its instanceId and the new status.

The status of the involved functions will be updated automatically to the new one.

URL	/servicestatus/{ns_instance}/{new_status}/
Type	PUT
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *ns_instance*: Id service instance we want to change the status. ▪ *new_status*: name of the status.
Response code	▪ 400 - Bad request: when the body in the request is not well formed. ▪ 404 - Not found: when the provided accountId does not exist in the database. ▪ 408 - Request Timeout: when there is a problem with the status message queue. ▪ 200: OK.
Request example	`PUT /servicestatus/id09/stopped/ HTTP/1.1`

Table 3-14 Accounting API operation to update the status of a service given its instanceId and the new status

3.5.1.4. Dashboard interface (T-Da-Ac)

The following operations supported by the Accounting API are exposed to the Dashboard.

(a) Retrieve SLA related information to show in the dashboard given the userId and the wheter you want to retrieve VNFs or network services.

URL	/sla-info/?clientId={clientId}&kind={ns|vnf}

Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *clientId*: Id of the user that is using the network service or the VNF. ▪ *kind*: It can take 2 values: ns | vnf.
Response code	▪ 400 - Bad request: when the body in the request is not well formed. ▪ 500: There is a connection problem between the Accounting and the SLA modules. ▪ 200: OK.
Request example	`GET /sla-info/?clientId=c1&kind=ns HTTP/1.1`
Response example	`[ { "productId": "service6", "productType": "ns", "clientId": "c1", "providerId": "p6", "SLAPenalties": 35, "agreementId": "serviceids101", "dateCreated": "2015-10-08T07:31:37Z", "dateTerminated": "2015-12-15T17:26:45.071444" } ]`

(b) Retrieves the list of all running services the user (customer) is using.

URL	/servicelist/{userId}/
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *userId*: Id of the user for whom we want to retrieve the service list.
Response code	▪ 400 - Bad request: when the body in the request is not well formed. ▪ 200: OK.
Request example	`GET /servicelist/c1 HTTP/1.1`

Response example	[{ "id": 2, "instanceId": "id02", "productId": "s1", "agreementId": "s1vnf2_4", "relatives": "id01, id03", "productType": "ns", "flavour": null, "startDate": "2015-06-11T00:00:00Z", "lastBillDate": "2015-06-11T00:00:00Z", "providerId": "p1", "clientId": "c1", "status": "running", "billingModel": "PAYG", "period": "P1D", "priceUnit": "EUR", "periodCost": 1.0, "setupCost": 1.0, "renew": true, "dateCreated": "2015-06-11T13:29:16Z", "dateModified": "2015-12-10T09:29:41Z" },]

(c) Retrieves the list of all running VNFs the user (service provider) is using.

URL	/vnflist/{userId}/
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *userId*: Id of the user for whom we want to retrieve the service list.
Response code	▪ 400 - Bad request: when the body in the request is not well formed. ▪ 200: OK.
Request	`GET /vnflist/p5 HTTP/1.1`

example	

Response example	[
	{
	"id": 24,
	"instanceId": "idf50",
	"productId": "vnf5",
	"agreementId": "s1vnf2_4",
	"relatives": "ids100",
	"productType": "vnf",
	"flavour": null,
	"startDate": "2015-10-08T07:07:43Z",
	"lastBillDate": "2015-10-08T07:07:43Z",
	"providerId": "f5",
	"clientId": "p5",
	"status": "running",
	"billingModel": "PAYG",
	"period": "P1D",
	"priceUnit": "EUR",
	"periodCost": 1.0,
	"setupCost": 2.0,
	"renew": false,
	"dateCreated": "2015-10-08T07:07:43Z",
	"dateModified": "2015-10-08T07:07:43Z"
	}
	]

Table 3-15 Accounting API operation to retrieve the list of all running VNFs the user (service provider) is using

3.5.2. Calls to other APIs

In order to implement T-Ac-SI it is the accounting the one that calls the SLA API according to the definition provided in section 2.6.1.2.

3.6. Billing module integration

The Cyclops framework gets the billing events from the Accounting module. Accounting module also provides the billing and revenue sharing models between

various actors. In turn, Cyclops allows for generation of billing and revenue sharing reports for any desired time-period. The interaction between Cyclops and T-NOVA Marketplace modules are shown in the Figure 3-11:

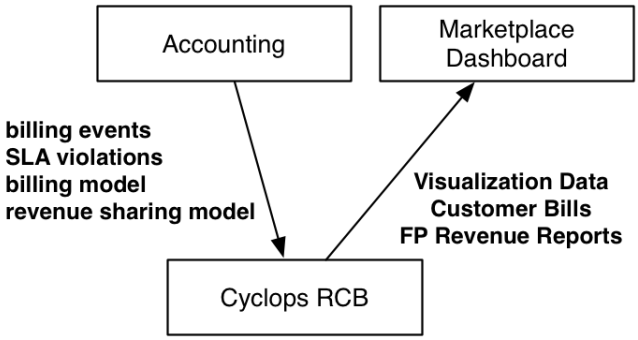


Figure 3-11 Cyclops and T-Nova Marketplace Module Interactions

The two prominent interfaces between Cyclops framework and external T-NOVA modules are:

- Cyclops-Dashboard.
- Cyclops-Accounting

The billing microservice is deployed within the Marketplace docker structure in a separate container. To be able to do so, the general docker-compose file needs to have a section dedicated to the billing module and its dependencies.

Once we have the dependencies fulfilled, it's time to configure the container. A *dockerfile* tells how to create the container and the script *DockerStart.sh* tells how to execute it.

3.6.1. Billing module API definition

The APIs that are relevant for integration with other modules of T-NOVA marketplace are shown here. The definitions for the internal APIs can be found in the framework's GitHub WiKi page. The Cyclops framework is still undergoing significant changes to incorporate all the T-NOVA requirements. The APIs described below are generally stable, but minor modification in the light of implementation experiences in the future can be expected. In such a case, the final API descriptions will be made available in WP7 deliverables, and also as part of the code release documents.

3.6.1.1. Dashboard interface (T-Da-Bi).

The operations supported by the billing API are exposed to the dashboard (T-Da-Bi).

(a) Usage query API for getting user's data

URL	`http://localhost:8080/udr/usage/users/{user_id}`
Type	GET
Headers	x-auth-token : String

Parameters	from : Date to : Date
Response Code	200 : Success
Request	None

Response	{
	"userid": "49588f5cea984040bc05d871eff67d2f",
	"time": {
	"to": "2015-01-12 01:10:00",
	"from": "2015-01-12 01:01:00"
	},
	"usage": {
	"openstack": [
	{
	"name": "cpu_util",
	"columns": [
	"time",
	"sequence_number",
	"avg"
	],
	"points": [
	[
	1421024460734,
	124666640001,
	74.31932
	],
	[
	1421024460734,
	124666550001,
	0.7899716
	]
	]
	}
	]
	}
	}

Table 3-16 Billing API operation for getting user's data

(b) Usage query API for particular resource / service id

URL	http://localhost:8080/udr/usage/resources/{resource_id}
Type	GET
Headers	x-auth-token : String
Parameters	from : Date to : Date
Response Code	200 : Success
Request	None

Response	{
	"resourceid": "49588f5cea984040bc05d871eff67d2f",
	"time": {
	"to": "2015-01-12 01:10:00",

	"from": "2015-01-12 01:01:00" }, "column": ["time", "mean", "userid"], "usage": [[0, 0, "46fe4a610a8b44948a5b61427b0b5ecd"], [0, 0, "49588f5cea984040bc05d871eff67d2f"], [0, 2.950836399999999, "99909daae8924e7a9b96cd964e9d64e3"],] }

Table 3-17 Billing API operation to query for particular resource / service id

(c) Bill generation API for a particular customer

URL	http://localhost:8080/billing/invoice
Type	GET
Headers	x-auth-token : String
Parameters	Customerid: String from : Date to : Date
Response Code	200 : Success
Request	None

Response	{
	"time": {
	"to": "2015-06-15 23:59",
	"from": "2015-06-15 00:00"
	},
	"charge": {
	"columns": [
	"time",
	"sequence_number",
	"userid",
	"usage",
	"price",
	"resource"
	],
	"points": [

	[1434361731726, 413986240001, "f83aa92bc3c64a3497b334cc712b0491", 5, 15.84, "service-id-aaab-hg1562711-ahsbba"], [1434361731726, 413986230001, "f83aa92bc3c64a3497b334cc712b0491", 37, 124.4, "service-id-aaac-hg1562711-ahsbbs"]] } }

Table 3-18 Billing API operation to generate bill for a particular customer

(d) Retrieves the earnings of a provider in between specified dates for all the instances.

Note: The "totalViolations" field represents the total amount of violations that the instance has done in this period of time as also represents all the discounts already applied.

URL	/billing/revenue?provider={provider}&from={from}&to={to}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *provider*: Id of the provider who we want to generate the revenue report for. The provider can be either a Sprovider or a FProvider ▪ *from*: first date of the bill in format: yyyy-MM-dd hh:mm:ss ▪ to: last date of the bill in format: yyyy-MM-dd hh:mm:ss
Response code	▪ 200: With empty results when erroneous data is provided in the call. ▪ 200: OK.
Request example	GET/billing/revenue?provider=p1&from=2016-01-18%2009:34:00&to=2016-01-18%2009:42:59 HTTP/1.1
Response example	{ "provider": "p1", "from": "2016-01-18 09:34:00", "to": "2016-01-18 09:42:59", "revenues": [{ "time": "2016-01-18T09:35:44Z", "instanceId": "id02", "provider": "p1", "price": 0.0024363425925925924,

	"priceUnit": "EUR",
	"discountValue": 0,
	"finalPrice": 0.0024363425925925924,
	"totalViolations": 0
	},
	{
	"time": "2016-01-18T09:40:43Z",
	"instanceId": "id02",
	"provider": "p1",
	"price": 0.0024363425925925924,
	"priceUnit": "EUR",
	"discountValue": 0,
	"finalPrice": 0.0024363425925925924,
	"totalViolations": 0
	}
	]
	}

Table 3-19 Billing API operation to retrieves the earnings of a provider in between specified dates for all the instances

(e) Retrieves the bill of a specific user in between specified dates for all the instances.

Note: The "totalViolations" field represents the total ammount of violations that the instance has done in this period of time as also represents all the discounts already applied.

URL	/billing/bill?userId={userId}&from={from}&to={to}
Type	GET
Headers	Accept: application/json Content-type: application/json
Parameters	▪ *userId*: Id of the user we want to generate the bill for. The user can either be a Customer or a SProvider ▪ *from*: first date of the bill in format: yyyy-MM-dd hh:mm:ss ▪ to: last date of the bill in format: yyyy-MM-dd hh:mm:ss
Response code	▪ 200: With empty results when erroneous data is provided in the call. ▪ 200: OK.
Request example	GET /billing/bill?userId=p1&from=2016-01-18%2009:34:00&to=2016-01-18%2009:37:59 HTTP/1.1
Response example	{ "userId": "p1", "from": "2016-01-18 09:34:00", "to": "2016-01-18 09:37:59", "charges": [{ "time": "2016-01-18T09:35:43Z", "instanceId": "id01", "provider": "f1", "price": 0.0029484953703703704,

	"priceUnit": "EUR",
	"discountValue": 0,
	"finalPrice": 0.0029484953703703704,
	"totalViolations": 0
	},
	{
	"time": "2016-01-18T09:35:44Z",
	"instanceId": "id03",
	"provider": "f1",
	"price": 0.0029484953703703704,
	"priceUnit": "EUR",
	"discountValue": 0,
	"finalPrice": 0.0029484953703703704,
	"totalViolations": 0
	}
	]
	}

Table 3-20 Billing API operation to retrieves the bill of a specific user in between specified dates for all the instances

(f) Accounting interface

The accounting module is the main component that pushes all billing relevant data into Cyclops framework for further processing using the Cyclops Messaging service. The billing relevant events pertaining to a given service instance could be –

a) service-running,
b) service-stopped,
c) service-suspended,
d) service-resumed

Accounting module also enables Cyclops to get the agreed billing model and revenue-sharing model data. Furthermore, it also makes the SLA violations and the penalty model available to Cyclops for factoring in before the bills are generated.

3.6.2. Calls to other APIs

As part of the T-Bi-Ac the billing module calls the Accounting API according to the definition provided in section 3.5.1.1.

4. VALIDATION

4.1. Functional verification

To test the functionality of the SLA management, accounting and billing modules we have created some sample VNFs and Network Services with real data to make it as close to a real scenario as possible, trying to cover all the functionalities with one example. Table 4-1 collects the results for this verification tests.

#	Functionality	Action	Call from	Request to	Verification	Ok?
1	Providers registration	Register FP, SP and customer	Dashboard	SLA mgmt.	Verify the introduced providers are in the SLA management module database	Yes
2	VNF Templates introduction	Create VNF template based on the VNFD and send it to the SLA management module	Dashboard	SLA mgmt.	Check the SLA module database for the template and the logs to see there has not been errors in the process	Yes
3	Service templates introduction	Create NS template based on the NSD and send it to the SLA management module	Service selection	SLA mgmt.	Check the SLA module database for the template and the logs to see there has not been errors in the process	Yes
4	VNFs and NSs tracking	Create an entry for the purchased service and for each of the involved VNFs	Service selection	Accounting	Verify the service and all the VNFs are present in the accounting database with all the necessary information.	Yes
5	FP-SP agreement introduction	Fill up the fields in the VNF template that have changed during the negotiation process and create the agreement	Accounting	SLA mgmt.	Check the SLA module database for the agreement and the logs to see there has not been errors in the process	Yes
6	SP-Customer agreement introduction	Fill up the fields in the NS template that have changed during the negotiation process and create the agreement	Accounting	SLA mgmt.	Check the SLA module database for the agreement and the logs to see there has not been errors in the process	Yes
7	Start the agreements	Once received the service instantiation order, the SLA agreements corresponding to	Accounting	SLA mgmt.	Request the list of the running agreements from the SLA module and check the ones we recently started are on the list	Yes

		the purchase of the NS and the VNFs within need to be started as well				
9	"Start" event introduction	Introduce the events corresponding the start of the NS and VNFs in the billing event queue	Accounting	Billing	Request the list of events in the billing queue and verify the events we have just created are present	Yes
9	Generate SLA violations	Feed the SLA management module with random monitoring data that should generate a number of SLA violations	SLA mgmt.	SLA mgmt.	Request the list of violations for the example agreements we have created and see if there are any SLA penalties.	Yes
10	Stop the service	Simulate a call from the orchestrator that request the stop of the service instance	Orchestrator	Accounting	Verify in the accounting module the status of the service (and the involved VNFs) is no longer "running" and it's "stopped" now	Yes
11	"Stop" event introduction	Introduce the events corresponding the stop of the NS and VNFs in the billing event queue	Accounting	Billing	Request the list of events in the billing queue and verify the events we have just created are present	Yes
12	Stop the agreements	Stop the assessment of the SLA agreements involved	Accounting	SLA mgmt.	Request the list of the running agreements from the SLA module and check the ones we just stopped are not on the list	Yes
13	Request SLA penalties	Request a bill generation for some user	Billing	Accounting	Request for the list of all SLA violations within the queried time frame, verification is the discounts/penalties reflected in the bill/revenue-sharing-report generated for the user/function developer.	Yes
14	Request bill	Request a bill generation for some user for a given time frame	Dashboard	Billing	The response contains the amount due for the customer. The charge data records can be verified in the InfluxDB series.	Yes
15	Request revenues report	Make a call to the billing module with the function provider ID and the time frame	Dashboard	Billing	The response contains the revenue share report along with any penalties for SLA violations. The periodic reports can be	Yes

					verified in the InfluxDB series.	
16	Show SLA statistics	Request SLA statistics for a given NS or VNF, running or stopped	Dashboard	Accounting	See the charts drawn on the screen	Yes

Table 4-1 SLA, accounting and billing verification

Further validation tests will be performed in T-NOVA under WP7 (Pilot integration and field trials).

4.2. Requirements fulfilment

Following the successful execution of the aforementioned functional tests, Table 4-2, Table 4-3 and Table 4-4 explain how the implemented and tested SLA, accounting and billing frameworks respectively fulfill the requirements which were set in the specification phase [1].

4.2.1. SLA management module requirements

Req. id	Requirement Name	Requirement Description	Met	Implementation
SLA.1	SLA information customer-SP storage	The SLA management module SHALL store all the SLA agreements between a customer and the SP for each service.	YES	SLA module stores in a MySQL database all the agreements that are introduced by the Customers (as agreement initiators) automatically at the moment of the purchase by means of the accounting module.
SLA.2	SLA information SP-FPs storage	The SLA management module SHALL store all the SLA agreements between the SP and the FPs for each VNF.	YES	SLA module stores in a MySQL database all the agreements that are introduced by the Service provider (as agreement initiator) automatically at the moment of the purchase by means of the accounting module.
SLA.3	SLA – orchestrator interface	The SLA management module SHALL be connected to the orchestrator to let it know about the agreed SLA for each service. (When the SLA is not fulfilled the orchestrator will have to initiate the applicable action, e.g. rescaling)	NO	Due to implementation and execution efficiency, these values are not provided by the SLA module but are extracted from the VNFD and the NSD directly.
SLA.4	SLA fulfilment information storage (from the orchestrator)	The SLA management module SHALL store all the information about SLA fulfilment for eventual compensations or penalties for later billing.	YES	All information related to the SLA fulfilment generated by the SLA module is stored in the internal database and it's not deleted even if the service is no longer in use.

SLA.5	SLA – accounting/billing interface	The SLA management module SHALL be connected the accounting/billing system to let it know about eventual compensations or penalties when the SLA has not been fulfilled for a specific service.	YES	The SLA module is queried about possible penalties and associated discounts by the accounting module by means of the exposed REST API.
SLA.6	SLA visualisation by customer and SP	The SLA management module SHALL be connected to the Dashboard to allow a customer and SP to visualize SLA fulfilment information when requested.	YES	Due to the fact that the dashboard is not aware of the agreement IDs, this request will have to be done through the accounting module.
SLA.7	SLA procedure mechanisms	The SLA management module SHALL provide mechanisms to get an agreement presented and agreed between the parties	YES	The SLA module exposes a REST API for this matter.

Table 4-2 SLA requirements fulfilment

4.2.2. Accounting module requirements

Req. id	Requirement Name	Requirement Description	Met	Implementation
Ac.1	Accounting notification - VNF starts	The accounting system SHALL know if a VNF or network service starts correctly.	YES	The accounting is notified by the orchestrator (through the service selection module) when a service or a VNF starts by means of a REST API call.
Ac.2	Resources usage for billing	The accounting system SHALL store all the information about resources usage by each service for later billing purposes.	NO	Due to technical project decisions, the use of resources doesn't constitute a billable item, therefore, no information about them is stored in the accounting module.
Ac.3	Price information for billing	The accounting system SHALL store the information about prices agreed by each customer for later billing purposes.	YES	All the information related to the pricing for the usage of a service or VNF is stored in the accounting database and is sent to the billing module on request.
Ac.4	SLA billable items	The accounting system SHALL be aware of the information about SLA fulfilment for billing compensations or penalties.	YES	The accounting system queries the SLA module for penalties and compensations due to SLA non-fulfilment on billing module request.
Ac.5	Bill cycle	The accounting module SHALL be able to provide the billing related information for any given period for each customer,	YES	The accounting module is able to provide billing information for any given period as the dates of the service and VNF lifecycle events (start, stop) are stored and a REST API is set for this purpose.

		and SP.		
Ac.6	Components relationships	The accounting system SHALL store the necessary information of the service and VNF instances, agreements, providers and customers.	YES	Every entry in the accounting system contains details of the client and provider involved in a purchase, the SLA they agreed on and the dates of the events occurred during the service or VNF life. All this information is available on a REST API interface.

Table 4-3 Accounting requirements fulfilment

4.2.3. Billing module requirements

Req. id	Requirement Name	Requirement Description	Met	Implementation
Bil.1	Price information for customer billing	The billing module SHALL receive the information about prices agreed by each customer for each service.	YES	The billing module receives all the billing related information from the accounting module (where it's stored) by means of a REST API request.
Bil.2	Price information for SP billing	The billing module SHALL receive the information about prices agreed by each SP for each VNF.	YES	The billing module receives all the billing related information from the accounting module (where it's stored) by means of a REST API request.
Bil.3	Bill issuing	The billing module SHALL issue bills when the customer's bill cycle finishes or service pay-as-you-go finishes and stores them within the customer profile.	YES	The billing module is ready to issue bills on every required period but due to system conveniences, the bills will be issued every 1st day of the month and will contain the usage of all the services and VNFs in the last month.
Bil.4	Billing-accounting interface	The billing SHOULD receive all the information needed for billing from the accounting module.	YES	The billing module receives all the billing related information from the accounting module (where it's stored) along with the SLA unfulfilments, the lifecycle events (to know whether a service or a VNF has been stopped for a period of time) and the users involved in the transaction by means of a REST API request.
Bil.5	Billing-User management interface	The billing module SHALL get the specific user related information along with the information received from the accounting system for billing purposes.	YES	The billing module retrieves the information from a specific user from the UMAA module and the billing related information from the accounting module to compose a bill.
Bil.6	Billing-Dashboard interface	The billing module SHOULD provide information (statistics and graphs) to the dashboard.	YES	Information related to the cost, the usage of a service or VNF and the fulfilment of the SLAs are provided to the dashboard on request.

Table 4-4 Billing requirements fulfilment

5. CONCLUSIONS

This document reports the outputs of T-NOVA Task 6.4 – SLAs and billing, which aimed to design and implement the all the necessary components in the T-NOVA SLA and billing frameworks.

Based on the State of Art survey performed in relation to the SLA (standardization bodies and other projects) and based on the requirements elicitation in the specification phase [1], the following main decisions has been taken to implement T-NOVA SLA framework which have been detailed in this report:

- The SLA T-NOVA framework is characterized by two SLA levels, corresponding to the two commercial iteractions in T-NOVA Marketplace:
  - Between FPs-SP and, SLA associated to standalone VNFs.
  - Between SP and the Customer, associated to Network Services (NSs).
- In relation to standardization bodies, ETSI NFV requirements for SLA have been considered as input for T-NOVA SLA framework, though not a proper complete SLA business framework has been specified by ETSI so far. TMForum gives some insights about metrics and SLA relations in cloud environment that has also been taken into account. However, T-NOVA specification work was ahead of these two standardization bodies so a potential contribution from T-NOVA was submitted to ETSI and will be submitted to TMForum in the following months.
- T-NOVA SLA framework is WS-agreement compliance, as it has been identified as the most complete and extended specification for SLA procedure. All the surveyed research projects in cloud environment have followed this WS-Agreement though there is no research project in the state of the art providing SLA framework for NFV ecosystem as T-NOVA does.
- The SLA management module implemented in T-NOVA is an evolution of an open source component used previously for cloud environment, being adapted to NFV as well as to the two different SLA levels considered in T-NOVA.

Based on the State of Art survey performed in relation to the billing options (internet commercial players, telco providers and other projects) and based on the requirements elicitation in the specification phase [1], the following main decisions has been taken to implement T-NOVA billing framework which have been detailed in this report:

- Billing in T-NOVA is done for 2 different products according to the 2 commercial relationships in T-NOVA ecosystem:
  - The Service Provider (SP) acquires Virtual Network Functions (VNFs) from the Function Providers (FPs).
  - The Customer acquires Network Services (NSs) provided by Service Providers based on the combination of VNFs previously purchased.
- After an exploratory work considering different options for billing mechanisms it has been concluded that Pay-As-You-Go is the most generic and suitable model to bill VNFs and Network Services in T-NOVA including an innovative Revenue Sharing model between Service Provider and Function Providers. FPs will benefit

from the pay-as-you-earn model, an extension of pay-as-you-go in which the VNF provider will pay a percentage of the revenue received.

- The T-NOVA billing framework has been designed to be composed by 2 modules:
  - Accounting module: it keeps a record of all the movements in the system that may have a potential impact in the billing: when a service is instantiated, when is terminated, billing models and pricing. It also serves as a bridge between the orchestrator and the marketplace getting the information about the running instances and their SLA. It has been developed from scratch for T-NOVA requirements.
  - Billing module: it emits the bills based on the accounting information. The billing module being used in T-NOVA extends the generic rating-charging-billing (RCB) framework Cyclops [36], the development of which started in FP7 MCN project [37], and whose functionalities have been extended to support the T-NOVA requirements.

All the components in the T-NOVA Marketplace (including SLA, accounting and billing) have been developed with a Software Oriented Architecture based on microservices, in which each Marketplace component has been developed separately (UML diagrams in this report) and communicates with the others by means of RESTful APIs (documented in this report). This provides flexibility and scalability to the T-NOVA Marketplace in case further functionalities may want to be added in the future and also this also facilitated the development process by different developers in T-NOVA.

For the integration of all the different components in the Marketplace Docker has been selected; each microservice relies in a different container, and they are integrated by means of Docker file that coordinates the integration.

It has been included in this report also some basic functional verification tests that have been performed in order to validate the developments and check the requirements fulfilments. Further validation tests will be done in the project within the specific WP for that purpose.

The T-NOVA SLA, accounting and billing modules prototypes will be uploaded to http://github.com/T-NOVA.

5.1. Future work

We have identified two main items that can be considered as future work for the SLA and billing frameworks in T-NOVA:

- ETSI NFV approach use the Gold, Silver, Bronze notation for the definition of a particular NS that is composed by a number of VNFs and a Connectivity Service, which we use in T-NOVA to name a group of technical parameters for the SLA specification. However that notation, as it is defined at the moment, does not correspond to any particular principle/rule common to all the possible compositions available in T-NOVA. The analogy that we can think is coming from the relevant usage of the three color marker in networking. This will allow us as

future work to reach more accurate notion of the whole QoS offered for Network Services that can be assured as part of the SLA, which is still a topic that need further research in the SOTA. This issue comes due to QoS for Network Services in NFV can be affected by many different factors: VNF software, compute, storage and networking QoS.

- The possible second identified future work will be for the whole T-NOVA system to consider offering the customer the possibility to suspend and resume a service. This will impact consequently in the billing procedure and how this will be considered in case the resources may be reserved or not for a specific paused service.

Year 2016 in T-NOVA project will be devoted to the system integration and testing of all its components, e.g. with the T-NOVA Orchestrator and Virtualized Infrastructure Management. We do expect that system integration may detect some gaps or need of fine tuning the interfaces. Moreover, testing the whole T-NOVA system can identify some non-functional aspect that could suggest refining some part of the current work. Also the complete implementation of T-NOVA Orchestrator, main component interfacing T-NOVA Marketplace, is expected to be finalized by end of March. Then, finally integration tests between Marketplace and Orchestrator should be done, therefore refinements on the Marketplace could be needed later on, for example in relation to the interfaces between monitoring system and SLA and accounting modules.

5.1.1. 5G projects

T-NOVA SLA management module has been identified to be potentially extended to multi Service Provider environment within 5GEx project [38]. This project aims to build a sandbox to extend software networks in a multi-domain/operator environment. 5GEx does not aim to implement a full marketplace, even billing aspects are not in its scope, but SLA management issues should be part of the multidomain orchestrator that 5GEx aims to build.

5.2. Contributions to standards

In relation to standardization bodies: ETSI NFV requirements for SLA has been considered as input for T-NOVA SLA framework, though not a proper complete SLA business framework has been specified by ETSI so far. TMForum gives some insights about metrics and SLA relations in cloud environment that has also been taken into account. However, T-NOVA specification work was ahead of these to standardization bodies so a potential contribution was submitted to ETSI and will be submitted to TMForum in the following months.

6. ANNEXES

6.1. WS-Agreement

WS-Agreement specifies an xml structure to define agreements and templates, and two layers interface of web services for operation. [39] contains a summary of the specification.

The XML representation of an agreement or a template has the following structure:

```
<wsag:Agreement AgreementId="xs:string">
    <wsag:Name>xs:string</wsag:Name> ?
    <wsag:Context>
        wsag:AgreementContextType
    </wsag:Context>
    <wsag:Terms>
        wsag:TermCompositorType
    </wsag:Terms>
</wsag:Agreement>
```

The following describes the attributes and tags listed in the schema outlined above:

- `/wsag:Agreement`

This is the outermost document tag which encapsulates the entire agreement. An agreement contains an agreement context and a collection of agreement terms.

- `/wsag:Agreement/@AgreementId`

This is a mandatory identifier of this particular version of the agreement. It must be unique between Agreement Initiator and Agreement Responder. Through the effect of extended negotiation mechanisms not defined in this specification, different agreement documents MAY be regarded semantically as updated versions of an existing agreement relationship, potentially having the same Name and being exposed by the same Endpoint Reference. This id attribute helps agreement responder and consumer uniquely identify the version currently in force. If an agreement instance document is modified during the lifecycle of an Agreement resource, the identifier MUST also be replaced with a new, unique identifier.

- `/wsag:Terms`

The terms of an agreement comprises one or more service definition terms, and zero or more guarantee terms grouped using logical grouping operators.

The following is an example of an agreement:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:Agreement xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement"
```

```
  AgreementId="sample-agreement">

  <wsag:Name>Sample Agreement</wsag:Name>
  <wsag:Context>
    <wsag:AgreementInitiator>client-prueba</wsag:AgreementInitiator>
    <wsag:AgreementResponder>f4c993580-03fe-41eb-
8a21</wsag:AgreementResponder>
    <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
    <wsag:ExpirationTime>2014-03-07T12:00:00</wsag:ExpirationTime>
    <wsag:TemplateId>template02</wsag:TemplateId>
    <sla:Service xmlns:sla="http://sla.atos.eu">sample-
service</sla:Service>
  </wsag:Context>
  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceDescriptionTerms Name="SDT" ServiceName="ServiceName"/>
      <wsag:ServiceProperties Name="ServiceProperties"
ServiceName="ServiceName">
        <wsag:VariableSet>
          <wsag:Variable Name="availability" Metric="xs:double">
            <wsag:Location>metric1</wsag:Location>
          </wsag:Variable>
        </wsag:VariableSet>
      </wsag:ServiceProperties>
      <wsag:GuaranteeTerm Name="GT-availability">
        <wsag:ServiceScope ServiceName="ServiceName"/>
        <wsag:ServiceLevelObjective>
          <wsag:KPITarget>
            <wsag:KPIName>AVAILABILITY</wsag:KPIName>
            <wsag:CustomServiceLevel>
              {"constraint" : "availability BETWEEN (0.99, 1)"}
            </wsag:CustomServiceLevel>
          </wsag:KPITarget>
        </wsag:ServiceLevelObjective>
      </wsag:GuaranteeTerm>
    </wsag:All>
  </wsag:Terms>
</wsag:Agreement>
```

A template has basically the same structure, and the intention of templates is to serve as base for new agreements. So, a procedure to create a new agreement for a template could be:

1. Retrieve the template for a service. The template could have "prefilled" the context element (excepting the consumer), the service properties, and the guarantee terms.
2. Build an agreement xml using the template as a base, and filling the rest of needed elements.
3. Initiate the negotiation.

The step 2 is domain-dependant, and it is recommended to add a domain factory that encapsulates this workflow, but having a simpler interface. For example:

1. Retrieve the template for a service, and extract the properties and boundaries for this service.

2. Send the (consumer, templateId, properties, boundaries) in json format to the domain service.
3. The domain service retrieves the template, builds the agreement and initiates the negotiation.

To create templates for a service, it is recommended a similar procedure.

The SLA core implements a mechanism to facilitate this kind of factory, with the use of IParsers. The project can provide a translator from a simple format to WS-Agreement, so the inputs of agreements and templates to the SLA core are in this simple format. This helps to reduce the complexity of WS-Agreement format.

The default implementation only allows a wsag:All term.

The default implementation does not handle the CreationContraints elements. It should be handled in the suggested domain layer.

6.1.1. Context

The context describes some metadata about the agreement/template.

The specification is:

```
<wsag:Context xs:anyAttribute>
    <wsag:AgreementInitiator>xs:anyType</wsag:AgreementInitiator> ?
    <wsag:AgreementResponder>xs:anyType</wsag:AgreementResponder> ?
    <wsag:ServiceProvider>wsag:AgreementRoleType</wsag:ServiceProvider>
    <wsag:ExpirationTime>xs:DateTime</wsag:ExpirationTime> ?
    <wsag:TemplateId>xs:string</wsag:TemplateId> ?
    <wsag:TemplateName>xs:string</wsag:TemplateName> ?
    <xs:any/> *
</wsag:Context>
```

- `/wsag:Context/wsag:AgreementInitiator`

This optional element identifies the initiator of the agreement creation request.

- `/wsag:Context/wsag:AgreementResponder`

This optional element identifies the agreement responder, i.e. the entity that responds to the agreement creation request.

- `/wsag:Context/wsag:ServiceProvider`

This element identifies the service provider and is either `AgreementInitiator` or `AgreementResponder`. The default is `AgreementResponder`.

- `/wsag:Context/wsag:TemplateId`

If a template was used to create an offer, the TemplateId in the Context must be set.

- `/wsag:Context/wsag:TemplateName`

The template name must be included in an offer if the offer is based on a template

- The default implementation handles sla:Service element (WS-Agreement allows this kind of extensions), to identify the service provided in the agreement, as the WS-Agreement allows several provider services to be in the agreement.
- The attribute ServiceName is present in the rest of elements in the agreement. The value of this attribute specifies an individual service of the several ones that may be inside an agreement/template, but is intended to only have meaning inside the agreement. As the ServiceName does not identify a service as known externally, the sla:Service element should be used for this matter.
- In the case of only one ServiceName per agreement, the ServiceName value is a do not care value; it can have the same value as the sla:Service element, or have a fixed value. It is a domain task to specify this.

6.1.2. Service description terms (SDT)

The Service Description Term describes the offered service. Its main purpose is to describe the type of service to be provisioned in the case that this provision is made in the SLA-system itself.

The definition is:

```
<wsag:ServiceDescriptionTerm
    wsag:Name="xs:string" wsag:ServiceName="xs:string">

    <xs:any> ... </xs:any>
</wsag:ServiceDescriptionTerm> +
```

The default implementation does not handle the service description terms, and as such, the service must be provisioned externally.

The implementer may provide additional features handling the Service Description Terms. For example, the SDT can be filled by the system with needed information about the allocated resources, and only known after the allocation (e.g. IP).

6.1.3. Service references (SR)

A service reference points to a service. So, if the service provided in the agreement is an external service, it may be referenced here. This way, the url/identifier/whatever associated with a ServiceName attribute can be known. Refer to page 20 of the spec for more details.

The definition is:

```
<wsag:ServiceReference
    wsag:Name="xs:string" wsag:ServiceName="xs:string">

    <xs:any> ... </xs:any>
```

```
</wsag:ServiceReference> +
```

The default implementation does not handle the service references.

6.1.4. Service properties (SP)

ServiceProperties are used to define measurable and exposed properties associated with a service, such as response time and throughput.

```
<wsag:ServiceProperties
    wsag:Name="xs:string" wsag:ServiceName="xs:string">

    <wsag:VariableSet>
        <wsag:Variable wsag:Name="xs:string" wsag:Metric="xs:URI">
            <wsag:Location>xs:string</wsag:Location>
        </wsag:Variable> *
    </wsag:VariableSet>
</wsag:ServiceProperties> +
```

The service properties are a set of variables that are used in the guarantee terms constraints. So, for example, if a constraint is : *uptime > 90*, there can be two service properties: ActualUptime and DesiredUptime. And the constraint will be ActualUptime > DesiredUptime.

The default implementation does not use the service properties this way. It does not use the thresholds as service properties; only the actual metric.

The following is a sample of a service property being valid in the default implementation:

```
<wsag:Variable Name="Uptime" Metric="xs:double">
    <wsag:Location>service-ping/Uptime</wsag:Location>
</wsag:Variable>
```

- The name of the variable is used in the Guarantee Terms.
- The optional metric attribute refers to a schema type that the value of the variable must fulfil.
- The location is defined in the spec as "the value of this element is a structural reference to a field of arbitrary granularity in the service terms - including fields within the domain-specific service descriptions". According to WSAJ Guarantee Evaluation Example [39], this is interpreted as the place where to find the actual value of the metric, referencing to an element in the SDT with, e.g., xpath.

In the default implementation, as the SDTs are not handled, the location is ignored.

Alternative implementations may interpret the location as the "abstract location of the metric". So, the location can be used if the monitoring module expects a name different than the metric name to return measures.

6.1.5. Guarantee terms (GT)

The guarantee terms hold the constraints that are being enforced in the service exposed in this agreement.

The definition is:

```
<wsag:GuaranteeTerm Name="xs:string" Obligated="wsag:ServiceRoleType">
    <wsag:ServiceScope ServiceName="xs:string">
        xs:any ?
    </wsag:ServiceScope> *
    <wsag:QualifyingCondition> xs:anyType </wsag:QualifyingCondition> ?
    <wsag:ServiceLevelObjective>
        ...
    </wsag:ServiceLevelObjective>
    <wsag:BusinessValueList>
        ...
    </wsag:BusinessValueList>
</wsag:GuaranteeTerm>
```

- `/wsag: GuaranteeTerm/@wsag:Name`

The mandatory name attribute (of type xs:string) represents the name given to a guarantee. Since an Agreement MAY encompass multiple GuaranteeTerms each term SHOULD be given a unique name.

- `/wsag:GuaranteeTerm/@wsag:Obligated`

This attribute defines, which party enters the obligation to the guarantee term. The wsag:ServiceRoleType can be either `ServiceConsumer` or `ServiceProvider`. The default implementation does take this attribute into account, and always consider it as ServiceProvider.

- `/wsag:GuaranteeTerm/wsag:ServiceScope`

A guarantee term can have one or more service scopes. A service scope describes to what service element specifically a guarantee term applies. It contains a ServiceName attribute and any other XML structure describing a substructure of a service to which the scope applies. For example, a performance guarantee might only apply to one operation of a Web service at a particular end point.

- `/wsag:GuaranteeTerm/wsag:ServiceScope/@ServiceName`

The name of a service to which the guarantee term refers. A guarantee term service scope applies to exactly one service.

An example of guarantee term is:

```
<wsag:GuaranteeTerm Name="GT-ResponseTime">
    <wsag:ServiceScope ServiceName="service-ping"/>
    <wsag:ServiceLevelObjective>
```

```
        <wsag:KPITarget>
            <wsag:KPIName>Uptime</wsag:KPIName>
            <wsag:CustomServiceLevel>
                    {"constraint" : "Uptime BETWEEN (90, 100)"}
            </wsag:CustomServiceLevel>
        </wsag:KPITarget>
    </wsag:ServiceLevelObjective>
</wsag:GuaranteeTerm>
```

6.1.6. Service Level Objective (SLO)

The SLO in an assertion over the service attributes and/or external factors as date, time.

The definition is:

```
<wsag:ServiceLevelObjective>
    <wsag:KPITarget>
        <wsag:KPIName>xs:string</wsag:KPIName>
        <wsag:CustomServiceLevel>xs:any</wsag:CustomServiceLevel>
    </wsag:KPITarget>
    |
    <wsag:CustomServiceLevel> xs:anyType </wsag:CustomServiceLevel>
</wsag:ServiceLevelObjective>
```

KpiName is a name given to the constraint, The sample uses the same name as the service property used in the constraint. This makes more sense when using thresholds as service properties. This value is used as the attribute kpiName of any violation of this GT.

The CustomServiceLevel is not specified by WS-Agreement, and a simple default implementation is provided. See ConstraintEvaluator section in the developer guide.

Although there are three ways to define an SLO in WS-Agreement, the one supported in the SLA core is shown in the previous example.

6.1.7. Business Values

Associated with each Service Level Objective is a Business Value List that contains multiple business values, each expressing a different value aspect of the objective.

The definition is:

```
<wsag:BusinessValueList>
    <wsag:Importance> xs:integer </wsag:Importance> ?
    <wsag:Penalty>
        <wsag:AssessmentInterval>
            <wsag:TimeInterval>xs:duration</wsag:TimeInterval> |
            <wsag:Count>xs:positiveInteger</wsag:Count>
        </wsag:AssessmentInterval>
        <wsag:ValueUnit>xs:string</wsag:ValueUnit>?
        <wsag:ValueExpression>xs:anyType</wsag:ValueExpr>
    </wsag:Penalty> *
    <wsag:Preference>
        <wsag:ServiceTermReference>xs:string </wsag:ServiceTermReference> *
        <wsag:Utility>xs:float</wsag:Utility> *
    </wsag:Preference> ?
```

```
    <wsag:CustomBusinessValue>xs:anyType</wsag:CustomBusinessValue> *
</wsag:BusinessValueList>
```

For example:

```
<wsag:GuaranteeTerm Name="GT-ResponseTime">
    <wsag:ServiceScope ServiceName="service-ping"/>
    <wsag:ServiceLevelObjective>...</wsag:ServiceLevelObjective>
    <wsag:BusinessValueList>
        <wsag:Importante>3</wsag:Importante>
        <wsag:Penalty>
            <wsag:AssessmentInterval>
                <wsag:Count>100</wsag:Count>
            </wsag:AssessmentInterval>
            <wsag:ValueUnit>EUR</wsag:ValueUnit>
            <wsag:ValueExpression>10</wsag:ValueExpression>
        </wsag:Penalty>
    </wsag:BusinessValueList>
</wsag:GuaranteeTerm>
```

The concept behind this is that a violation of a GT can involve a business penalty. On the other hand, a fulfilled GT can involve a business reward.

6.2. T-NOVA SLA template example (JSON)

The following template represents the SLA definition of and VNF flavour in T-NOVA:

```
{
    "context": {
        "agreementInitiator": null,
        "agreementResponder": "f5",
        "service": "TC / should an ontology be defined or this is free text
input?",
        "serviceProvider": "AgreementResponder",
        "templateId": "vnfvnf5gold"
    },
    "name": "vnf5gold",
    "templateId": "vnfvnf5gold",
    "terms": {
        "allTerms": {
            "guaranteeTerms": [
                {
                    "businessValueList": {
                        "customBusinessValue": [
                            {
                                "count": 1,
                                "penalties": [
                                    {
                                        "expression": 5,
                                        "type": "discount",
                                        "unit": "%",
                                        "validity": "P1D"
                                    }
                                ]
                            }
```

```
                        ]
                    },
                    "name": "pepitovnf5",
                    "qualifyingCondition": null,
                    "serviceLevelObjetive": {
                        "kpitarget": {
                            "customServiceLevel": " { \"policies\": [ {
\"count\" : 2, \"interval\": 30 } ], \"constraint\" : \"pepitovnf5 GT 0.5\"
}",
                            "kpiName": "pepitovnf5"
                        }
                    },
                    "serviceScope": null
                },
                {
                    "businessValueList": {
                        "customBusinessValue": [
                            {
                                "count": 1,
                                "penalties": [
                                    {
                                        "expression": 5,
                                        "type": "discount",
                                        "unit": "%",
                                        "validity": "P1D"
                                    }
                                ]
                            }
                        ]
                    },
                    "name": "juanitovnf5",
                    "qualifyingCondition": null,
                    "serviceLevelObjetive": {
                        "kpitarget": {
                            "customServiceLevel": " { \"policies\": [ {
\"count\" : 2, \"interval\": 30 } ], \"constraint\" : \"juanitovnf5 GT
0.7\" }",
                            "kpiName": "juanitovnf5"
                        }
                    },
                    "serviceScope": null
                }
            ],
            "serviceDescriptionTerm": {
                "name": "requirements",
                "requirements": [
                    {
                        "name": "virt_mem_res_element",
                        "value": 6,
                        "unit": "GB"
                    },
                    {
                        "name": "CPU",
                        "value": 6,
                        "unit": "cores"
                    },
                    {
                        "name": "TLB size",
                        "value": 1024,
```

```
                        "unit": ""
                    },
                    {
                        "name": "storage",
                        "value": 20,
                        "unit": "GB"
                    }
                ],
                "serviceName": "calls5k"
            },
            "serviceProperties": [
                {
                    "name": "MonitoredMetrics",
                    "serviceName": "default",
                    "variableSet": {
                        "variables": [
                            {
                                "location": "/monitor/pepitovnf5",
                                "metric": "xs:double",
                                "name": "pepitovnf5"
                            },
                            {
                                "location": "/monitor/juanitovnf5",
                                "metric": "xs:double",
                                "name": "juanitovnf5"
                            }
                        ]
                    }
                }
            ]
        }
    }
}
```

6.3. T-NOVA SLA agreement example (JSON)

The following agreement represents the SLA between a function provider (f5) and a service provider (p6) using the previous template as a base:

```
{
    "context": {
        "agreementInitiator": "p6",
        "agreementResponder": "f5",
        "service": "TC / should an ontology be defined or this is free text
input?",
        "serviceProvider": "AgreementResponder",
        "templateId": "vnfvnf5gold"
    },
    "name": "vnf5gold",
    "agreementId": "vnfidf51",
    "terms": {
        "allTerms": {
            "guaranteeTerms": [
                {
                    "businessValueList": {
                        "customBusinessValue": [
                            {
```

```
                                "count": 1,
                                "penalties": [
                                    {
                                        "expression": 5,
                                        "type": "discount",
                                        "unit": "%",
                                        "validity": "P1D"
                                    }
                                ]
                            }
                        ]
                    },
                    "name": "pepitovnf5",
                    "qualifyingCondition": null,
                    "serviceLevelObjetive": {
                        "kpitarget": {
                            "customServiceLevel": " { \"policies\": [ {
\"count\" : 2, \"interval\": 30 } ], \"constraint\" : \"pepitovnf5 GT 0.5\"
}",
                            "kpiName": "pepitovnf5"
                        }
                    },
                    "serviceScope": null
                },
                {
                    "businessValueList": {
                        "customBusinessValue": [
                            {
                                "count": 1,
                                "penalties": [
                                    {
                                        "expression": 5,
                                        "type": "discount",
                                        "unit": "%",
                                        "validity": "P1D"
                                    }
                                ]
                            }
                        ]
                    },
                    "name": "juanitovnf5",
                    "qualifyingCondition": null,
                    "serviceLevelObjetive": {
                        "kpitarget": {
                            "customServiceLevel": " { \"policies\": [ {
\"count\" : 2, \"interval\": 30 } ], \"constraint\" : \"juanitovnf5 GT
0.7\" }",
                            "kpiName": "juanitovnf5"
                        }
                    },
                    "serviceScope": null
                }
            ],
            "serviceDescriptionTerm": {
                "name": "requirements",
                "requirements": [
                    {
                        "name": "virt_mem_res_element",
                        "value": 6,
```

```
                        "unit": "GB"
                    },
                    {
                        "name": "CPU",
                        "value": 6,
                        "unit": "cores"
                    },
                    {
                        "name": "TLB size",
                        "value": 1024,
                        "unit": ""
                    },
                    {
                        "name": "storage",
                        "value": 20,
                        "unit": "GB"
                    }
                ],
                "serviceName": "calls5k"
            },
            "serviceProperties": [
                {
                    "name": "MonitoredMetrics",
                    "serviceName": "default",
                    "variableSet": {
                        "variables": [
                            {
                                "location": "/monitor/pepitovnf5",
                                "metric": "xs:double",
                                "name": "pepitovnf5"
                            },
                            {
                                "location": "/monitor/juanitovnf5",
                                "metric": "xs:double",
                                "name": "juanitovnf5"
                            }
                        ]
                    }
                }
            ]
        }
    }
}
```

7. REFERENCES

[1] D2.42 - Specification of Network Function Framework and T-NOVA Marketplace. T-NOVA project.

[2] D6.01 - Interim report on T-NOVA Marketplace implementation. T-NOVA project.

[3] Deliverable D2.1 – System Use Cases and Requirements. T- NOVA project.

[4] RESTFul APIs: http://www.django-rest-framework.org/.

[5] CLOUD4SOA project: http://cordis.europa.eu/fp7/ict/ssai/docs/call5-cloud4soa.pdf.

[6] FED4FIRE project: http://www.fed4fire.eu/.

[7] XIFI project: https://www.fi-xifi.eu/home.html.

[8] Vodafone: http://www.vodafone.co.uk/about-this-site/terms-and-conditions/mobile-broadband-via-the-phone/index.htm.

[9] Orange: http://clientes.orange.es/soporte_y_ayuda/pdf/1051479_CCGG_PD.pdf.

[10] ETSI GS NFV 004 Network Function Virtualization. Requirements.

[11] ETSI GS INF 010Network Function Virtualization. Service Quality Metrics.

[12] ETSI GS NFV REL005 V<0.1.4> Network Functions Virtualisation (NFV); Assurance; Report on Quality Accountability Framework. October 2015.

[13] TMFORUM Enabling End-to-End Cloud SLA Management. October 2014.

[14] QuestForum, "TL 9000 Measurements Handbook", release 5.0, July 2012.

[15] TM Forum, "TM Forum WebSite," http://www.tmforum.org.

[16] TMFORUM SLA Management Handbook.

[17] TMForum Information Framework. http://www.tmforum.org/InformationFramework/1684/Home.html.

[18] TMForum IG1120 Virtualization Impact on SLA Management - April 2015.

[19] TMForum IG1127 End-to-end Virtualization Management: Impact on E2E Service Assurance and SLA Management for Hybrid Networks R15.0.0 - May 2015.

[20] D5.01 - Interim Report on Network Functions and associated framework. T-NOVA project.

[21] D6.2 - Brokerage module. T-NOVA project.

[22] D3.01 - Interim Report on Orchestration Platform Implemenation. T-NOVA project.

[23] CISQ. Consortium for IT Software Quality. http://www.it-cisq.org.

[24] ETSI NFV ISG, "NFV-MAN 001 NFV Management and Orchestration," July 2014. [Online]. Available: http://docbox.etsi.org/ISG/NFV/Open/Latest_Drafts/NFV-MAN001v061 %20management%20and%20orchestration.pdf.

[25] D4.41 - Monitoring and Maintenance – Interim. T-NOVA project.

[26] D4.01 - Interim Report on Infrastructure Virtualisation and Management. T-NOVA project.

[27] WebService-Agreement specification. http://wsag4j.sourceforge.net/site/wsag/overview.html.

[28] Apache license, http://www.apache.org/licenses/LICENSE-2.0.

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

[30] Amazon DevPay: http://aws.amazon.com/devpay/.

[31] Google Play. https://play.google.com/store.

[32] Apple App Store (requires iTunes): http://www.itunes.com/appstore/.

[33] BlueVia website: http://www.bluevia.com/.

[34] Orange Partner website: http://www.orangepartner.com.

[35] OPTIMIS Project http://cordis.europa.eu/fp7/ict/ssai/docs/call5-optimis.pdf.

[36] Cyclops, http://icclab.github.io/Cyclops/.

[37] MCN project Mobile Cloud Networking - http://www.mobile-cloud-networking.eu/site/.

[38] 5GEx project - 5G Exchange - https://5g-ppp.eu/5gex/.

[39] WSAG Guarantee Evaluation Example, https://packcs-e0.scai.fraunhofer.de/wsag4j/server/guarantee_evaluation_example.html.

8. GLOSSARY

Name	Description
Access Control Module	Component in the marketplace that administers security managing and enabling access authorization/control for the different T-NOVA stakeholders considering their roles and permissions.
Accounting Module	Component in the marketplace that stores all the information needed for later billing for each user: usage resources for the different services, SLAs evaluations, etc.
Billing Module	Component in the marketplace that produces the bills based on the information stored in the accounting module
Business Service Catalog	Catalog in the marketplace that stores all the available offerings.
Brokerage Module	Component in the marketplace that enables trading of VNFs, facilitating the auctioning between Function Providers.
T-NOVA Customer (customer)	Stakeholder that aims to acquire T-NOVA Network Services.
Dashboard	Graphical User Interface (GUI) for the stakeholders to interact with the system. In T-NOVA it has 3 different views: SP dashboard, FP dashboard and customer dashboard.
Function provider	Software developer that offer VNFs in the marketplace to be sold.
Function store (NF Store)	The T-NOVA repository holding the images and the metadata of all available VNFs/VNFCs
NFV Infrastructure	The totality of all hardware and software components which build up the environment in which VNFs are deployed
Marketplace	The set of all tools and modules which facilitate the interactions among the T-NOVA actors, including service request, offering and provision, trading, service status presentation and configuration, SLA management and billing
NS Catalogue	The Orchestrator entity which provides a repository of all the descriptors related to available T-NOVA services
Offering	Each Network Service available in the marketplace together with a SLA level and price. It is created by the Service Provider and store in the Business Service Catalogue to advertise the services to the customer.
Orchestrator	The highest-level infrastructure management entity which orchestrates network and IT management entities in order to

	compose and provision an end-to-end T-NOVA service.
Service Provider	Stakeholder that offer Network Services through the marketplace creating offerings in the business service catalogue. To create the network services the SP acquires VNFs from the Function Providers. The VNF are deployed over the T-NOVA infrastructure.
SLA Management Module	Component in the marketplace that establishes and stores the SLAs among all the involved parties and checking if the SLAs have been fulfilled or not will inform the accounting system for the pertinent billable items.
SLA template	The SLA template is a form has the same structure as the SLA Agreement but some fields are not filled yet or might change, like the providers or the final price.
Stakeholder	Each of the kind of actors that can use T-NOVA system: SP, FPs, customers.
T-NOVA Network Service ("service")	A network connectivity service enriched with in-network VNFs, as provided by the T-NOVA architecture.
T-NOVA Operator	The T-NOVA system administrator that owing the T-NOVA infrastructure controls the activity of all the T-NOVA users.
VNF catalogue	The Orchestrator entity which provides a repository with the descriptors of all available VNF Packages.
VNF	A virtualised (pure software-based) version of a network function

9. LIST OF ACRONYMS

Acronym	Explanation
API	Application Programming Interface
BSS	Business Support System
CPU	Central Processing Unit
DoW	Description of Work
GUI	Graphical User Interface
ETSI	European Telecommunication Standard Institute
EU	End User
FP	Function Provider
ISG	Industry Specification Group
ISP	Internet Service Provider
IT	Information Technology
KPI	Key Performance Indicator
MANO	Management and Orchestration
NFaaS	Network Functions-as-a-Service
NF	Network Function
NFC	Network Function Component
NFV	Network Functions Virtualisation
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
NS	Network Service
OSS	Operational Support System
QoS	Quality of Service
RTT	Round trip time
SaaS	Software-as-a-Service
SDN	Software-Defined Networking
SDO	Standards Development Organisation
SID	Shared Information/Data model
SLA	Service Level Agreement
SP	Service Provider

UC	Use Case
VM	Virtual Machine
VNF	Virtual Network Function
VNFaaS	Virtual Network Function as a Service
VNFD	Virtual Network Function Descriptor
WP	Work Package