UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

Luis Alberto Cuellar Hoyos

# NOn: Network Function Virtualisation Ontology Towards Semantic Service Implementation

# NOn: Uma Ontologia de Funções Virtualizadas de Rede para Implementação de Serviços Semânticos

CAMPINAS

2016

Luis Alberto Cuellar Hoyos

# NOn: Network Function Virtualisation Ontology Towards Semantic Service Implementation

# NOn: Uma Ontologia de Funções Virtualizadas de Rede para Implementação de Serviços Semânticos

Dissertation presented to the Faculty of Electrical and Computer Engineering of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Electrical Engineering, in the area of Computer Engineering.

Dissertação apresentada à Faculdade de Engenharia Elétrica e Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica, na Área de Engenharia de Computação.

Supervisor: Prof. Dr. Christian Rodolfo Esteve Rothenberg

Este exemplar corresponde à versão final da dissertação defendida pelo aluno Luis Alberto Cuellar Hoyos, e orientada pelo Prof. Dr. Christian Rodolfo Esteve Rothenberg

CAMPINAS

2016

**Agência(s) de fomento e nº(s) de processo(s):** FUNCAMP, 4881.1

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Luciana Pietrosanto Milla - CRB 8/8129

C894n

Cuellar Hoyos, Luis Alberto, 1988-
Non : network function virtualisation ontology towards semantic service implementation / Luis Alberto Cuellar Hoyos. – Campinas, SP : [s.n.], 2016.

Orientador: Christian Rodolfo Esteve Rothenberg.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Rede de computação. 2. Análise de rede. 3. Semântica. 4. Serviços na web - Semântica. 5. Ontologia. I. Esteve Rothenberg, Christian Rodolfo,1982-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações para Biblioteca Digital

**Título em outro idioma:** Non : uma ontologia de funções virtualizadas de rede para implementação de serviços semânticos
**Palavras-chave em inglês:**
Computing network
Network analysis
Semantics
Web services - Semantics
Ontology
**Área de concentração:** Engenharia de Computação
**Titulação:** Mestre em Engenharia Elétrica
**Banca examinadora:**
Christian Rodolfo Esteve Rothernberg
Oscar Mauricio Caicedo Rendón
Luciano Bernardes de Paula
**Data de defesa:** 01-07-2016
**Programa de Pós-Graduação:** Engenharia Elétrica

**COMISSÃO JULGADORA - DISSERTAÇÃO DE MESTRADO**

**Candidato**: Luis Alberto Cuellar Hoyos RA: 153844
**Data da Defesa**: 1 de julho de 2016
**Título da Tese**:
"NOn: Network Function Virtualisation Ontology Towards Semantic Service Implementation"
"NOn: Uma Ontologia de Funçoes Virtualisadas de Rede para a Implementação de Serviços Semânticos"

Prof. Christian Rodolfo Esteve Rothenberg (Presidente, FEEC/UNICAMP)
Prof. Dr. Oscar Mauricio Caicedo Rendón (FIET/UNICAUCA)
Prof. Dr. Luciano Bernardes de Paula (IFSP)

Ata de defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no processo de vida acadêmica do aluno.

*To my little brother Kike, you make every day worth it.*
*To my parents Yaneth and Luis, their wisdom and guide made me the person I'm today.*

# Acknowledgements

Christian, thanks so much for being my tutor, it was a path full of joy and hard work. Thanks for teach me and guide me on my way of becoming a better person and professional.

Thank to the Innovation Center, Ericsson Telecomunicações S.A., Brazil, for supporting this work.

Dörthe Arndt, Ruben Verborgh and fellows at the Data Science Lab, Ghent University, Belgium, thanks for the technical support on EYE and RESTdesc.

INTRIG, thanks for being my family in this two years, I learned something from each of you. We were the best group that anyone wants to have.

All my friends in Brazil, thanks for the shared moments. You people made this journey one of the best experience of my life.

Mayor Domingo, thanks for being my spiritual guide... there is not enough words to express the gratitude for your help.

Finally thanks to everyone that directly or indirectly has supporting me to achieve this goal!

# Abstract

Network Function Virtualization (NFV) arises as a recent technological trend in networking aiming at changing the current physical appliance model to a software-based approach to network service implementations. As today, only a set of specifications and guidelines are available which define NFV architecture views and the functional description of the main components. These specifications are meant to be read, interpreted, and implemented by human developers, thus allowing a high degree of freedom on the semantics used to develop NFV elements. As a consequence, we encounter heterogeneous manners to express the same components and a lack of common understanding across NFV domains. Moreover, interoperability among NFV components and domains is still an open challenge generally tackled by using Web Service (WS) which rely on implicit service descriptions and do not provide means to leverage common semantics. Furthermore, service integration requires costly and error-prone manual intervention along the processes of reading, interpreting and using service capabilities, resulting in a inefficient way of achieving interoperability. With the aim of addressing these practical challenges towards the realization of NFV, this thesis proposes the use of a common and convenient domain language to describe NFV components and to avoid manual intervention process through an automatic service integration by means of two approaches: NFV Ontology (NOn) and Semantic nFV Services (SnS). NOn allows describing NFV as a high level framework with reusable element descriptors following a standardized manner. SnS is the implementation of the Semantic Services approach in the NFV domain. SnS uses NOn to create explicit service descriptors, allowing smart agents from different domains with heterogeneous implementations to read, interpret, and consume NFV service capabilities. As a proof of concept for both proposals, a *Generic Client* was developed as a smart entity capable of reasoning by means of an inference engine that allows to create and consume dynamic workflows of WS. Dynamic workflows are achieved by reading the semantic services descriptions (without the need of a predefined context) and creating a plan for services consumption. As a result, the interoperability process becomes more efficient and less costly due to the automatic service integration. A total of five proof of concept use cases implementations validate the potential of the proposed NOn and SnS approaches to realize NFV.

**Keywords**: Network Function Virtualisation, NFV, Web Semantic, Semantic Services, Ontology.

# Resumo

Virtualização de Funções de Rede (NFV), surgem como uma nova tendência tecnológica em redes com o objetivo de alterar o modelo atual das implementações de serviço de rede, de uma abordagem com dispositivos fisicos para uma abordagem baseada em software. Atualmente, existem disponíveis uma série de especificações definindo a arquitetura de NFV e a descrição dos componentes principais. No entanto, as especificações são destinadas aos desenvolvedores para serem lidas, interpretadas e implementadas, permitindo assim um alto grau de liberdade na definição da semântica usada para desenvolver os elementos de NFV. Como consequência, este trabalho encontrou modos heterogêneos para expressar os mesmos componentes e uma falta de entendimento comum entre domínios. Aliás, a interoperabilidade entre diferentes componentes e domínios continua sendo um desafio aberto que geralmente é resolvido pela implementação de Serviços Web (WS), os quais são baseados em descrições implicitas e carecem dos meios para alavancar uma semântica comum. Ademais, ao fim de fazer uma integração de serviços existe uma intervenção manual de alto custo e propensa a erros, que vem unida com os processos de leitura, interpretação e implementação das funcionalidades dos serviços, resultando assim em uma maneira ineficente de atingir interoperabilidade. Com o objetivo de responder a estes desafios práticos no domínio de NFV, este trabalho propõe o uso de uma linguagem comum para descrever os componentes de NFV e evitar um processo de intervenção manual por meio de de uma integração automática do serviço por meio de duas abordagens: Ontologia de NFV (NOn) e Serviços nFV Semânticos (SNS). NOn permite descrever os componentes de alto nível e o reuso dos descritores em NFV de uma forma padronizada. SnS é a implementação de serviços semânticos no domínio NFV. SnS faz uso de NOn para criar descrições explícitas dos serviços, permitindo aque agentes inteligentes em diferentes domínios e com implementações heterogêneas consigam ler, interpretar e utilizar as capacidades de serviços de NFV. Como prova de conceito para as duas propostas foi desenvolvido um Cliente Genérico, capaz de fazer raciocínio por meio do uso de um motor de inferência que permite a criação e o consumo de fluxos dinâmicos de WS. Os fluxos dinâmicos são obtidos através da leitura das descrições dos serviços semânticos (sem a necessidade de um contexto predefinido) e da criação de um plano para consumir WS. Desse modo, tornando o processo de interoperabilidade mais eficiente e menos custoso, devido à integração automática de serviços e à redução na intervenção manual. Foram realizadas um total de cinco provas de conceitos por meio da implementação de casos de uso, que avaliaram o potencial da proposta, utilizando as abordagens NOn e SnS.

**Palavras-chaves**: Funçoes Virtualizadas de Rede, Ontologia, Serviços Semanticos, Web Semantica.

# List of Figures

# List of Tables

# Acronyms

**API** Application Programming Interface. 29, 32, 60, 87–89, 92

**CAPEX** Capital Expenditure. 17, 26

**CORBA** Common Object Request Broker Architecture. 18

**E2E** End-to-End. 17, 26, 27, 30

**ETSI** European Telecommunications Standard Institute. 18, 20, 27, 28, 34–38, 41, 45, 46, 48, 52, 54, 56–59, 87–90

**EYE** Euler Yet another proof Engine. 65, 83

**GUI** Graphical User Interface. 29

**HTML** HyperText Markup Language. 21, 22

**HTTP** Hypertext Transfer Protocol. 21, 22, 61, 62, 64–68, 72–74

**INDL** Infrastructure and Network Description Language. 19, 31, 42, 45

**INTRIG** Information & Networking Technologies Research & Innovation Group. 82–84

**ISG** Industry Specification Group. 18

**JSON** JavaScript Object Notation. 32, 61, 84

**MANO** Management and Orchestration. 17, 27–29, 37, 39, 46, 60, 90

**ML2** Modular Layer 2. 32

**N3** Notation 3. 23–25, 58, 60, 63, 64, 98

**NF** Network Function. 17, 26, 27, 30, 88, 91

**NFV** Network Function Virtualization. 17–20, 26–30, 32–46, 48, 50, 55, 58–61, 65, 67, 68, 74, 75, 81–83, 86, 88–91, 107–110

**NFVI** NFV Infrastructure. 17, 27–29, 32, 37, 45, 60, 81, 83, 91

**NFVO** Network Function Virtualization Orchestrator. 28, 29, 37, 60, 67, 74, 81, 83, 84, 86, 88, 91

**NML** Network Modeling Language. 19, 31, 42, 45

**NO** Network Operator. 17, 26, 31, 41

**NOn** NFV Ontology. 20, 41–47, 49–62, 64, 65, 68, 75, 77–79, 81, 85, 88–91, 98–106, 111–114

**NS** Network Service. 17, 26, 27, 37

**NSD** Network Service Descriptor. 17, 27

**OPEX** Operational Expenditure. 17, 26

**OWL** Web Ontology Language. 25, 31, 51, 54

**OWL-S** Semantic markup for Web services. 19

**POX** Python-based Software-Defined Networking. 32

**RDF** Resource Description Framework. 19, 22, 23, 31, 51, 54, 58

**REST** Representational State Transfer. 20–22, 26, 29, 32, 60, 61, 65, 67, 72, 74, 86, 89–91

**RIS** Resource Information Service. 31

**RMI** Remote Method Invocation. 18, 28, 87

**RPC** Remote Procedure Call. 18

**SDK** Software Development Kit. 29

**SDN** Software Defined Networking. 32

**SLA** Service Level Agreement. 27

**SnS** Semantic nFV Services. 20, 60, 61, 65, 67, 70, 74–81, 83, 85–88, 90, 91, 115–123

**SP** Service Provider. 17, 26, 41

**SWS** Semantic Web Service. 19, 25

**TAR** Tape ARchiver. 81, 82

**UNIFY** Unifying Cloud and Carrier Networks. 30, 86–89

**URI** Universal Resource Identifier. 21, 22, 24, 32, 51, 62, 65, 66

**URL** Uniform Resource Locator. 26

**VDU** Virtual Device Unit. 44, 46, 53, 58, 64

**VIM** Virtualized Infrastructure Manager. 28, 29, 32, 37, 46, 74, 77, 79, 80, 83, 86–89

**VLD** Virtual Link Descriptor. 46

**VM** Virtual Machine. 27, 83, 88

**VNF** Virtual Function Network. 17, 27–30, 37, 38, 41–46, 48, 50, 61, 62, 64, 67, 68, 74, 81–85, 88–91

**VNF-FG** VNF Forwarding Graph. 27

**VNFC** VNF Component. 27, 28, 44, 59, 64

**VNFD** VNF Descriptor. 17, 20, 27, 28, 30, 34–36, 39, 42–48, 50, 53–59, 61–65, 67, 69–71, 73–88, 90, 107–113

**VNFM** VNF Manager. 28, 37, 83, 86, 90

**W3C** World Wide Web Consortium. 21, 25

**WS** Web Service. 18, 19, 25, 26, 28, 32, 41, 60–67, 70–72, 74, 76, 79, 81, 83, 84, 86, 88–90

**WSMO** Web Service Modeling Ontology. 19

**XML** eXtensible Markup Language. 21, 23, 30–32, 36, 51, 58

**YAML** YAML Ain't Another Markup Language. 62, 84

**YANG** Yet Another Next Generation. 30

# Contents

# 1 Introduction

Network Function Virtualization (ETSI, 2012a) emerges as a software-centric network device implementation and operational approach with the aim of avoiding typical hazards of traditional Network Services. Currently, Network Operators (NOs) and Service Providers (SPs) usually need to design, buy, link and maintain a service chain of physical appliances to deploy the Network Services (NSs). Thus, when the deployment of NS grows, the use of physical appliances increases also the Capital Expenditure (CAPEX) [1] and Operational Expenditure (OPEX)[2]. Furthermore, physical appliances have short periods of service life, which creates the need of replace the devices with low or non revenue (ETSI, 2014b). NFV addresses Network Service deployment on the virtualisation of physical appliances in a software-defined approach. Therefore, instead of having to link physical devices to create services, the software-centric approach allows to create NSs by linking Virtual Function Network over a virtualised infrastructure.

Network Function Virtualization attempts to reduce CAPEX and OPEX by making Network Functions (NFs) easier develop and less costly to maintain. At the top of a NFV Infrastructure (NFVI) sit the deployed VNFs. NFVI is composed by physical and virtualised layers of Compute, Storage, and Network nodes and in order to deploy VNFss over the NFVI relying on the Management and Orchestration (MANO) realm. To do the deployment, MANO uses a descriptor file as an input. Generally, for deployment purposes, two files are defined, one containing the operational behavior and deployment configuration of the functions (VNF Descriptor), and the other describing the End-to-End (E2E) connection of the network functions, i.e., the Network Service Descriptor (NSD).

In order to develop and deploy NFV services and Network Services[3], there is the need of creating communication among NFV architecture components, thus gaining interoperability[4]. In this work, NFV interoperability is given by two different ways. Firstly, in a local domain scenario, by linking components on a same implementation. For example, NFV MANO **A** to NFVI **A**. Secondly, in an inter-domain scenario, by linking NFV implementations from different domains. For example, NFV MANO **A** to NFVI **B**. It is important to realize that NFV interoperability is not just about NFV components but includes also the need to create communication between NFV services. Therefore, in this document the definition of interoperability covers the communication among NFV services and components, as well as local domain and

[1] Capital Expenditure is the funds spent by a company to acquire or upgrade a long-term asset.

[2] Operating Expense is the cost continuously spent to maintain the production of a product or service to keep a business.

[3] NFV service makes reference to those services developed to implement the NFV architecture, such as resource allocation or VNF instantiation. Hence, NFV services are different from Network Service

[4] On NFV interoperability is defined by reference points or communication interfaces.

inter-domain scenarios.

## 1.1 Problem Description

With the aim of achieving homogeneity on NFV implementations, the European Telecommunications Standard Institute (ETSI) (ETSI, 1988) has produced a series of specifications describing business and deployment aspects. Currently, NFV is on the second phase of work and the ETSI NFV Industry Specification Group (ISG), has proposed a series of challenges to focus in this phase (ETSI, 2014a). Two of them are to achieve NFV interoperability and to make an interface specification (ETSI, 2012b)(ETSI, 2014b). Due the ETSI plans to achieve and ensure interoperability between different implementations trough the definition of interface specifications, both challenges are linked together. Hence, specifications are used as a guideline for developers to develop and implement interfaces. However, these specifications are meant to be interpreted by humans, for this reason software agents are not able to follow this specifications. Thus, generating problems inherent to interface integration, affecting NFV implementations in a negative manner in a path for achieving interoperability. Furthermore, software integration process comes along with resource and time costs. Those costs are associated with the difficulty level in the integration process, with costs increasing in proportion to the integration complexity. In our work, we identified two of the root problems that turn integration a difficult task: (*i*) the lack of common understanding and semantics to express and describe interfaces, and (*ii*) the need of manual intervention[5] to consume and use interfaces and their capabilities.

Interoperability on NFV implementations can be achieved by using different wrapping technologies such Remote Procedure Call (RPC), JAVA Remote Method Invocation (RMI) or Common Object Request Broker Architecture (CORBA), however those technologies have dependence and communication boundaries such programming language, operating system, communication protocols or data structure, adding some restrictions to the integration process. At the crossroads, Web Service[6] is a technology with the goal of removing the mentioned boundaries. Furthermore, geographical location issues are removed, allowing distributed architectures to locate components and services around the world[7].

WS technology provides implicit descriptions to define their capabilities and follows a client/server architecture. Due to the implicit descriptions there must be manual intervention to define, interpret, and consume service capabilities. Furthermore, there is not a common understanding on a domain language to create service descriptions (variables and methods are defined arbitrarily). Hence, software agents can read but not interpret service capabilities,

[5] Manual intervention refers to any task that in order to be accomplish need the intervention of a human.

[6] W3C Definition: Software system designed to support inter-operable machine-to-machine interaction over a network. It has an interface described in a machine-processable format.

[7] WS is the technology generally used for the distributed architectures, this work assumes its implementation as the default mechanism used by WS in order to gain interoperability.

therefore manual intervention is needed to do so. Furthermore, in some cases there is the need of creating brokers or middleware to interpret service descriptions and capabilities for one implementation to another –increasing the integration cost. This work assumes that problems mentioned above for WS are equivalent to the problems previously mentioned on NFV interface integration.

## 1.2 Approach and Research Objectives

In 2001, in the seminal work on the *The Semantic Web* (BERNERS-LEE *et al.*, 2001), Tim Barners Lee states that the current Web and the Semantic Web are not two different concepts, both are meant to be complementary technologies [8]. Semantic Web, tries to change the manner of how the Web works today, going from a human interpretation towards a machine interpretation by relying in the use of Resource Description Framework (RDF) (RDF, 2014) and an ontological representations of real world. Semantic Web attempts to create common knowledge and share it across the Web, thus creating a homogeneous understanding of specific concepts. In the other hand, in an effort to reduce manual intervention the Semantic Web Services (SWSs) technology was created. SWS technology born from the intersection of Web Services with Semantic Web technology. Semantic Services bases the service creation on semantic representations, explicit descriptions and ontological representations.

In the networking area, the use of a Semantic Web approach has already been initiated. Ontologies like Network Modeling Language (NML) (HAM *et al.*, 2013) and Infrastructure and Network Description Language (INDL) (GHIJSEN *et al.*, 2013), are two projects with the aim of standardizing the terminology of infrastructure and networking resources. However, both ontologies are used just to create models and store the information on (graph) databases (e.g., (SOUZA *et al.*, 2015)), with the aim of having a common view of all resources. Therefore, a semantic service approach has not been fully explored yet.

In the area of semantic services, there are many projects attempting to create explicit descriptions, such Web Service Modeling Ontology (WSMO) or Semantic markup for Web services (OWL-S). However, this related work does not fulfill the expectations in terms of service description or automatic service discover and interoperability (VERBORGH *et al.*, 2013). RESTdesc (RESTdesc, 2011) appears as a good semantic service technology to avoid those flaws due a mechanism to describe service functionality, allowing software agents to discover in a autonomous manner what is offered by a service and how to use it.

In this work, the need of manual intervention to do NFV service integration is seen as an inefficient manner to achieve interoperability. As NFV is on early years and there are few commercial implementations until today, in order to overcome the issues related to the expected

[8] "The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation." — Tim Berners-Lee et al.

integration and testing processes early on, this work proposes and implements two proposals. The first proposal is the design and implementation of a NFV Ontology based on the Semantic Web approach and using the ETSI specifications as a guideline. The second proposal is the implementation of semantic services using a RESTdesc approach and the NOn model to create explicit service descriptions.

As a first step and in order to do bear the NOn design, this work provides a brief analysis on the currently available data structures and variable definitions present in VNFD files. The result of the analysis serves as a proof of our assumption on the lack of a common understanding of the descriptors contained in the NFV specifications. Another result are the problems in terms of interoperability by not being able to reuse VNFDs files across NFV implementations (i.e., an inter-domain scenario). As a next step, NOn is used to evolve VNFDs from a syntax level to a semantic level thus creating semantic VNFD files. As a third step, NOn and other ontologies on the field of computing and networking are used to create Semantic nFV Services. In this process, Representational State Transfer (REST) services from current NFV implementations are used and enhanced by adding a RESTdesc description. Finally, as a proof of concept, a *Generic Client* was developed to read, interpret, and consume semantic services. The client –through the use of an inference engine– is capable of creating a chain of semantic services (workflow) and self adapt to consume those services.

The research objectives of this work can be summarized as follows:

- Develop a Network Function Virtualisation ontology (NOn) using as base the ETSI Virtual Function Network Descriptor (VNFD).
- Create a semantic representation of the VNFD.
- Implement NFV interfaces following a semantic service approach.
- Automate NFV service integration by using NOn and a semantic service implementation.
- Validate the concept of NFV semantic services by proof of concept implementations showcasing automatic service integration.

The structure of this work is as follows: Chapter 2 contains the literature review on background technologies and related work. Chapter 3 introduces the research problem and includes the NFV descriptor analysis. Chapter 4 describes the design and implementation of NFV Ontology. Chapter 5 presents the implementation of Semantic nFV Services and the use cases proposed as a proof of concept. Finally, Chapter 6 provides concluding remarks and a description of future work.

# 2 Background and Related Work

This chapter presents the theoretical basis of this work and is divided into three sections, first section includes technological background used to develop the work. In the second section are the related work, including data and interoperability models. Final section concludes the chapter.

## 2.1 Background

This section describes the technologies necessaries to understand and develop the proposal for this work and to implement the defined use cases.

### 2.1.1 Representational State Transfer

REST architectural style is defined by Roy T. Fielding doctoral thesis (FIELDING, 2000). The Web can be seen as a network-base of architectural styles and software design, furthermore, each element can be seen as a reference to a resource, additionally each resource has a resource *identifier*, such the Universal Resource Identifier (URI). Components of REST architecture can perform actions over the representation of resources (RICHARDS, 2006), this means to make an action over any useful information about the state of the resource. In REST, there are two types of state: one representing the information about the resource (server side) the other representing the information about resource on the consuming application (client side) (FENG *et al.*, 2009). This representational state is transferred across the client and server, thus it receives the name: Representational State Transfer. REST is not a standard or a protocol for this reason there is no specification done by the World Wide Web Consortium (W3C) or any other standard institute (COSTELLO, 2007).

#### 2.1.1.1 REST Web Services

Although REST is not a standard, it relies on the use of several standards such Hypertext Transfer Protocol (HTTP), URI, eXtensible Markup Language (XML) or HyperText Markup Language (HTML) and does not deal with implementation details. Instead, follows some constrains in order to do implementations:

- Client-Server
- Stateless Interactions
- Self-descriptive messages

- Uniform Interface
- Named Resources
- Interconnected resource representations
- Layered components

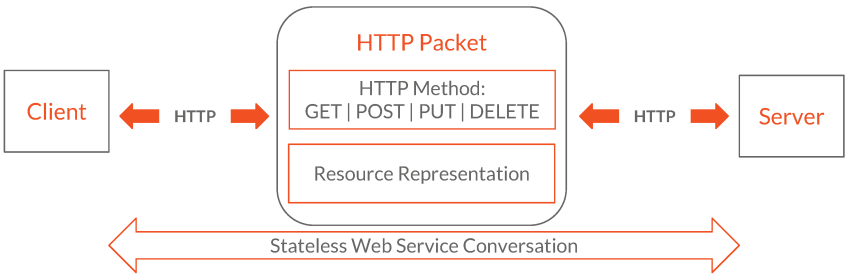


Figure 1 – RESTful Web services architecture

Figure 1 illustrates how is the REST architecture, in which a client access or modify the state of a resource representation through the use of an HTTP request and one of its methods.

### 2.1.2 Semantic Web

Current Web works in a way that can be interpreted by humans and not for machines. The Web is a linked network of Web pages referenced among them, built on the top of a tag language known as HTML. Despite HTML is a machine language, machines hardly interpret and process the info contained in Web pages. Furthermore, additionally is necessary to create other tools in order to allow the data being interpreted and useful to machines.

The Semantic Web, born with the aim of making information contained on Web pages, can be consumed and interpreted by machines in an autonomous manner, through the use of ontologies and semantic expressions in the deployment process (BERNERS-LEE *et al.*, 2001). Semantic Web relies on RDF (RDF, 2014) as the base language and ontology vocabulary as the structure (Figure 2). Adding semantics to the Web, there is a new path of data processing, data analysis and data retrieval for smart software agents (WANG; HALANG, 2013).

As an example of the impact in current Web applications, imagine a search engine using Semantic Web technology. The engine should bring the answer instead of retrieving a list of Web pages with the possible answer.

#### 2.1.2.1 RDF

RDF is a data model created to represent identified objects or data in the form of a triples-base structure (subject, predicate and object) and uses the URI to identify each element in the triples. The concept of using RDF is to map the Web data on an explicit way thus making

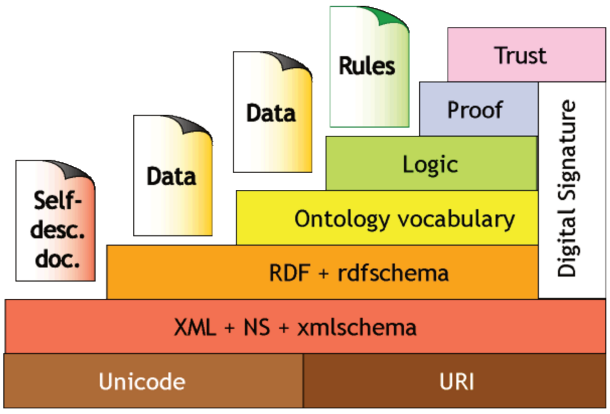


Figure 2 – The language layers of the Semantic Web

information comprehensible by software agents. RDF involves a graph network to make system representations of the real world.

RDF has properties (type, subClassOf, subPropertyOf, range, domain, label and comment), allowing the use of an inference process to infer new facts from other facts. Figure 3 shows and inference example, where it can be inferred that a Firewall is a type of Network Function. This is due a Virtual Function Network is a sub-class of Network Function and at the same time a Firewall is a type of Virtual Function Network.

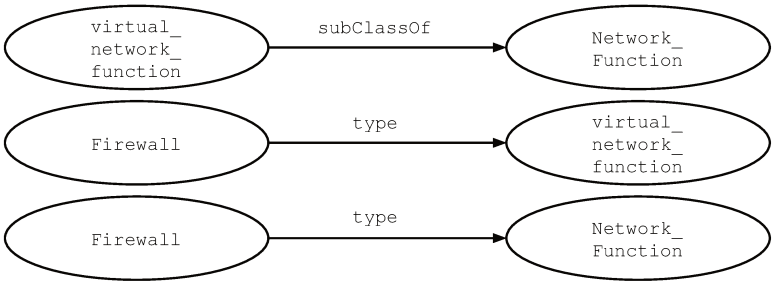


Figure 3 – RDF Example - Inference

Recommendations with the RDF schema are on (RDF Schema, 2014) and with the syntax are on(RDF Syntax, 2014).

#### 2.1.2.1.1 Notation 3

Notation 3 (N3) is a super set logic language of RDF and extends the data model through the implementation of formulae. Formulae, are literals representing graphs themselves, using variables, logical implication and functional predicates, additionally providing a textual syntax to represent RDF/XML components (NOTATION3, 2011).

Listing 2.1 illustrates the implementation of a rule in N3 language.

Listing 2.1 – Simple Rule Using N3

```
1 @prefix ppl: <http://example.org/people#>.
2 @prefix foaf: <http://xmlns.com/foaf/0.1/>.
3 {
4     ppl:Cindy foaf:knows ppl:John.
5 }
6 =>
7 {
8     ppl:John foaf:knows ppl:Cindy.
9 }.
```

Above Listing is a rule created using N3. The rule means: **IF** the object **Cindy** from the ontology **ppl** (`ppl:Cindy`) has the predicate (property) **knows** from the ontology **foaf** (`foaf:knows`) linked to the subject **John** from the ontology **ppl** (`ppl:John`) implies that (**=>**) the object **John** from the ontology **ppl** (`ppl:John`) has the predicate **knows** from ontology **foaf** (`foaf:knows`) with subject **Cindy** from the ontology **ppl** (`ppl:Cindy`). In other words, if Cindy knows John it implies John knows Cindy. The listing is a simple implication process:

$$P(x) => Q(x) \tag{2.1}$$

Some features of the language are:

- URI implementations using namespaces and `@prefix` N3 parameter (lines 1 and 2).
- Allows RDF to be expressive.
- Allows repetition of multiple objects for a same subject and predicate using a comma **","**.
- Allows repetition of multiple predicates for a same subject a semicolon **";"**.
- Allows formulae through the quote of N3 graphs using brackets **"{}"**.
- Allows rules quantification through the quantification of variables.
- Readable and natural through its consistent and simple grammar.

#### 2.1.2.2 Ontologies

An ontology can be seen as a knowledge representation of a specific domain (e.g. Gene Ontology). For the Semantic Web, an ontology is a set of properties, rules and a defined taxonomy in a software domain (ALESSO; SMITH, 2004). Taxonomy defines classes, subclasses and relationships among the objects of the domain. One of the purposes for the creation of an ontology is to have a shared vision and a common understanding of the specific domain (NOY *et al.*, 2001).

The main components of an ontology are:

- Class: a group of objects sharing common characteristics.

- Individuals: an instance of a Class. An object in a domain.
- Properties:
  - Object Property: Relationships among classes.
  - Data Property: Relationships among classes and primitive objects (e.g *Integer*)

The Web Ontology Language (OWL) has been created as a mechanism “To develop ontologies that are compatible with the WWW" (SEMANTIC-WEB-AFFINITY-GROUP, 2007). Proposed by the W3C, OWL is an effort to give a structure, to enhanced RDF in order to make the Web easier to interpret by machines.

### 2.1.3 Semantic Web Service

Semantic Web Service born where the Semantic Web crosses with the Web Service (Figure 4). Generally, WS descriptions are written in an implicit manner, this implies to have a context or background in order to use and consume the service capabilities. Instead, the semantic service technology has the aim of doing service descriptions in an explicit manner, thus allowing to use the service capabilities without previous knowledge. Furthermore, semantic service clients must be able to consume the services by reading just the descriptions and using ontologies to interpret them.

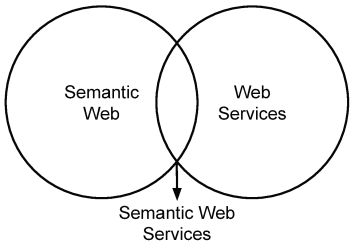


Figure 4 – Web Service & Semantic Web Integration

In order to do the implementation of Web Service in a semantic and dynamic manner, the services must be coded in an explicit manner, thus, a software agent consuming the services should know how, what and when to do, by reading the service description. Consequently, implementations with SWS attempt to be autonomous systems. Thus, creating a new path of intelligent services and environments (GUDIVADA; KALAVALA, 2005).

#### 2.1.3.1 RESTdesc

RESTdesc (RESTdesc, 2011) is a project attempting to remove the manual intervention on the WS consumption, through the implementation of semantic descriptions. RESTdesc is a description method for RESTful services, implemented on N3 language. Service descriptions in RESTDesc are centered on the service capabilities and relies on the use of an inference

engine and smart agents to perform the interpreting and the consumption process. RESTdesc uses ontologies to make service descriptions, for this reason does not need any variable declaration in order to describe functionalities. Indeed, the Uniform Resource Locators (URLs) used to consume services or to access resources are created in a dynamic and automated way through the inference engine. The engine creates a context in runtime according with the service descriptions and ontologies received as inputs (VERBORGH *et al.*, 2011b), then it builds the REST requests based on those inputs. The implementation of RESTdesc, additionally allows the creation of workflows, this means, a plan with a sorted list of WSs to be consumed in order to achieve one objective, a specific goal (VERBORGH *et al.*, 2011a).

#### 2.1.3.2 Inference Engine (Reasoner)

Inference Engines are software components implemented in the Semantic Web to deduce new knowledge from an already defined knowledge, generally using If/Then implications (e.g. Formula 2.1). This kind of engines infers new facts from a set of predefined rules by searching inside the knowledge base with the aim of achieve the rules. If a rule can be satisfied, then is placed in a plan (ALESSO; SMITH, 2004).

There exist two types of inference engine:

- Backward Chaining: for this type is given to the engine an hypothesis (goal or objective) and the engine backtracks the knowledge base to prove if the hypothesis is valid.
- Forward Chaining: for this type is given to the system some data and the engine attempts to reach a conclusion by using and inference process.

### 2.1.4 Network Function Virtualisation - NFV

Nowadays, NO and SP use a model of physical appliances to create NF, for example Load Balancer, Firewalls or WAN Accelerators. Each appliance represents one function and a service chain of the functions represents an E2E connection, known as NS. Additionally, in order to make the E2E connection (NS) feasible, it is necessary to use as many appliances as functions are needed. Thus, making the CAPEX and OPEX cost to increase, in addition, most of these appliances have some interoperability constrains, such vendor locks. For this reason, some appliances and NFs do not work properly with appliances and functions from different brands. Furthermore, updating, upgrading or adding new functions must be done in place, causing an increase on the OPEX costs for NO and SP. As an example, it can be considered the cost of updating a routing table from a switch.

Born in 2012, NFV has the aim of changing the current model of NSs deployment, thus, attempting to remove typical constraints of the appliance model as the vendor locks or the high costs of CAPEX and OPEX. This is planned to be achieved with the implementa-

tion of a software defined approach. Figure 5 shows the high level architecture of NFV. In this architecture, NFs are decoupled from the physical layer (the appliances) and deployed as virtualised applications over a software layer. These applications are created under the name of VNF and hosted at the top of the NFV Infrastructure. Different from the current model, the NFVI has three main components: physical resources (Compute, Networking, Storage), virtualisation layer (abstracting resources in a software plane) and the virtualised resources (Virtual Compute, Virtual Networking and Virtual Storage).

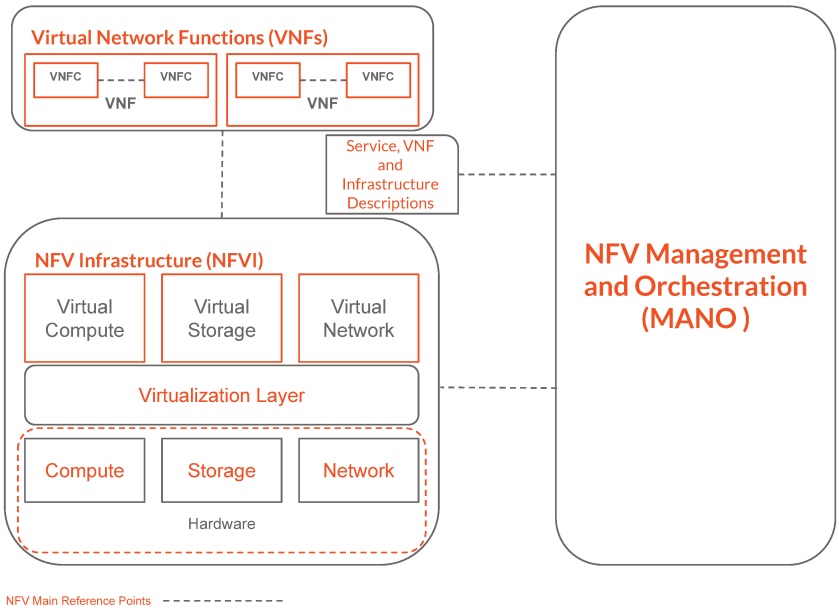


Figure 5 – NFV High Level Architecture

As mentioned above, a VNF is a software defined NF and is composite with one or more VNF Component (VNFC). These components are software deployments linked between them and can represent a Network Function itself. For this reason one VNF can be composite with more than one function. NFVI provides the resources necessaries (e.g.memory, bandwidth) to deploy a VNF in one or more Virtual Machines (VMs). VMs are configured using the parameters included in a description file created for each VNF. This files is named as VNFD and has the operational behavior (Management Operations) and deployment configuration to be setup over the NFVI (including resources, components and relationships). Figure 6 shows some information elements defined for the VNFD. In order to deploy a NS, is necessary to make a service chain of VNFs using a VNF Forwarding Graph (VNF-FG) file to structure and link the functions. The file includes the E2E service description, named as NSD. These descriptions files are predefined guidelines in the ETSI specifications and does not represent a defined implementation.

In the right side of the above figure is shown the NFV Management and Orchestration component, aiming to orchestrate and manage all the aspects related to the NS and VNF deployment. These aspects include a Service Level Agreement (SLA), life cycle (instantiate,

scale, update, upgrade and terminate) or resource allocation (physical or/and virtual) or communication interfaces. In order to do the deployment MANO component reads, interpret and execute the descriptor files over the NFVI.

| Identifier | Type | Cardinality | Description |
|---|---|---|---|
| Id | Leaf | 1 | ID (e.g., name) of this VNFD. |
| Vendor | Leaf | 1 | The vendor generating this VNFD. |
| descriptor _version | Leaf | 1 | Version of the VNF Descriptor |
| version | Leaf | 1 | Version of VNF software, described by the descriptor under consideration |
| vdu | Element | 1...N | This describes a set of elements related to a particular VDU |
| virtual Link | Element | 0...N | Represents the type of network connectivity mandated by the VNF vendor between two or more Connection Points |
| connection _point | Element | 1...N | This element describes an external interface exposed by this VNF enabling connection with a VL.<br>NOTE: The connection between the VNF and the VL is expressed by the VLD referencing this Connection Point. The Connection Point may also be attached to internal Virtual Links (vnfd: virtual_link:id). |

Figure 6 – VNFD Base Information Elements Template

To create communication among components, ETSI defined the Reference Points (communication interfaces). The dotted lines in the NFV architecture represent these Reference Points. In the case of VNF composition, internal interfaces link VNFC among them, additionally, these interfaces can change their configuration without affecting external interfaces. Such the ETSI does not define how, to implement *reference points*, this work assumes their implementation as a WS interface (ETSI GS NFV-MAN, 2014) (ETSI GS NFV-SWA, 2014) (ETSI GS NFV, 2014) (ETSI, 2012b) (ETSI, 2014b).

#### 2.1.4.1 NFV Projects

##### 2.1.4.1.1 OpenBaton

Openbaton is a compliant implementing the NFV Network Function Virtualization Orchestrator component and is based on the ETSI specifications. Openbaton is a project coded in JAVA and uses the RMI architecture aiming make to it easily extensible. Openbaton uses OpenStack as a Virtualized Infrastructure Manager (VIM) component for the orchestration of NFVI resources (OpenBaton, 2014).

The main components of OpenBaton are:

- A fully designed and implemented NFVO following ETSI specifications.
- A generic VNF Manager (VNFM) to manage VNF life-cycle using its VNFD.

- A Software Development Kit (SDK) to build proprietary components.

Figure 7 shows the architecture of OpenBaton. At the top of the figure are the Graphical User Interface (GUI) component and a VNF Package containing the VNF to be deployed. The NFVO and VIM components are represented by OpenBaton and OpenStack Projects respectively and the other elements on the figure are remaining components of the NFV architecture.

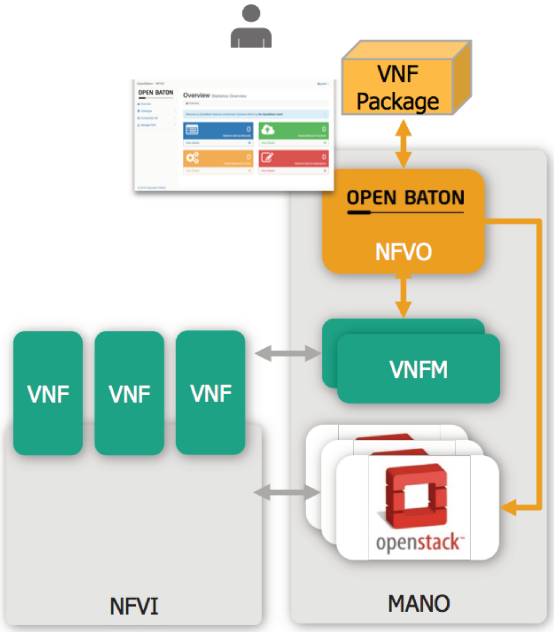


Figure 7 – OpenBaton Architecture

#### 2.1.4.1.2 OpenMano

OpenMano is an open source project aiming to implement the NFV MANO functional block (OpenMano, 2014). The project has three main components:

- openvim: is an implementation of the NFV VIM component. It offers and REST based interface to communicate (openvim Application Programming Interface (API)) with NFVI components.
- openmano: is an implementation of NFVO component. it communicates with openvim through a REST based interface (openmano API).
- openmano-gui: a GUI interacts with the openmano component through the REST interface.

#### 2.1.4.1.3 T-NOVA

The T-NOVA Project (XILOURIS *et al.*, 2014) has the aim of providing NFV as a Service (NFVaaS) in a business environment. This implementation has a new concept of a Network Function Store, which is to offer VNFs as apps are offered in a typical app store (such Google Play or Apple Apps Store). Furthermore, giving the possibility of third party developers to publish their own VNFs and to Service Provider to buy as they need. At the top of the T-NOVA architecture exists a set of northbound RESTful interfaces (with the issues mentioned above), each VNF uploaded into the NF Store has a metadata file in order to describe functionalities and how to manage them. However, such a VNFD and the metadata file has a lack of a semantic approach, leaves a gap in how to represent common components from one VNF to another (this gap additionally affects in an inter-domain plane). An ideal world for the future VNF developers will be to code a VNF once and reply n-times over different NF Stores or NFV domains. However, so far NFV orchestrators are programmed to read descriptor from syntactic and static file (such the metadata descriptor) and NFV interfaces are created base on specific needs and generally using Web Services. This Web Services are used in order create interoperability between components.

#### 2.1.4.1.4 Unify

Unifying Cloud and Carrier Networks (UNIFY) is a project focused on the research, development and evaluation of means for the orchestration, verification and observation E2E service delivery networks by through the use of core networks to data centers (UNIFY, 2014). Under UNIFY project is created a component called *Virtualizer*, an element responsible for resource allocation of networking, computing and storage components and other execution environments (SZABó *et al.*, 2014). *Virtualizer* uses a Yet Another Next Generation (YANG) data model to make the resource allocation. Listing B.2 of Annex B represents the *Virtualizer* data model in an XML format.

## 2.2 Related Work

This section is a brief summary of related work covering objectives similar to this work. Literature review of this section is divided in two different areas: (i) ontology data models for networking and computing infrastructure, (ii) interoperability models across domains. Finally, there are the general conclusions for the section.

### 2.2.1 Ontology Data Models

In *Semantic Distributed Resource Discovery for Multiple Resource Providers* (PITTARAS *et al.*, 2012) is proposed a mechanism to discover and share information about the

physical resources for resource providers (aka peers) in an inter-domain scenario. Currently, peers have proprietary models and databases to create and store their resource descriptions. However, there is a gap in terms of interoperability and variable definition to share those models across domains. For this reason, aiming to supply the absence of a standardized data model to describe resources, INDL ontology was created. In addition to the data model, also an external component with name Resource Information Service (RIS) was created. This component is capable of synchronize, translate and abstract resource information into a triple store, named as RIS database. RIS, acts as an independent middleware for the resource providers, taking the information from the providers databases and parsing it into the INDL data model, thus, making information available in a same language with other peers implementing the RIS module. Aiming to achieve interoperability between providers, RIS has an additional module (discovery module). Nevertheless, the discovery module finds only the providers with the RIS component implemented, thus, adding a constrain to the interoperability process.

Although, *Semantic Distributed Resource Discovery for Multiple Resource Providers* has some similarities with our work, such the implementation of an ontology, this work denotes two major weak points: (i) the sync between databases, them cannot be fully synchronized in real time, (ii) the implementation of an external component to create interoperability across domains.

In *A Semantic-Web Approach for Modeling Computing Infrastructures* (GHIJSEN *et al.*, 2013) is described an evolution of the INDL ontology. The information model was improved from a standalone to a Semantic Web approach. This change was done aiming the reuse of other Semantic Web models, e.g. NML (HAM *et al.*, 2013). Additionally to the new approach, is aimed re-usability of the model and an independence of the implementing technology. INDL was built as an extension of NML ontology and is used to create or enhance other ontological representations. As a proof of concept, INDL was implemented to make resource and networking descriptions of three different projects, Cinegrid, NOVI and GEYSERS (GHIJSEN *et al.*, 2012). NML (HAM *et al.*, 2013) is mentioned for INDL as the base project. NML is known as a common effort to create a standardize manner to make networking description, this effort is leaded by specialists in the networking field conforming the Open Grid Forum (OGF, ). NML data model contains all the components necessaries to create high level and detailed networking topologies, thus, giving flexibility to NOs for building models according specific needs. In addition, creating a common understanding across domains.

NML was developed using two types of syntax, one, using pure XML and XML Schema and the other using the Semantic Web approach with OWL RDF/XML and OWL schema. In some implementations using NML and INDL can be appreciated the reuse of other knowledge representations, thus, crating new representations and enhance current ones.

### 2.2.2 Interoperability Models

As an example, it is taken the mechanism done by OpenStack[1] (OpenStack, 2011) and Open DayLight[2] (OpenDayLight, 2013) projects to gain interoperability between them. It was implemented an external plugin, named Modular Layer 2 (ML2) (ML2, 2013), to parse request across the projects. ML2 additionally provides communication between OpenStack and other third party projects (such Brocade Mechanism Driver or Cisco Nexus Mechanism Driver, soft-switches), however, for each project exist the need of having a specific module in the plugin to create the communication. Consequently, a need of creating and coding as many modules as projects exists, is generated.

On the other hand, OpenStack is not a one project implementation, instead, is a stack of multiple projects (e.g. Neutron, Nova, Horizon) aiming to work together in order to create the platform. In consequence, a simple installation of OpenStack relies on reading of a hundred pages manual. Installation process includes: installing and configuring each component separately, then each component must be setup to communicate with the other components. Thus, attempting to make implementations easier, arises the need of developing external tools, such Ubuntu JUJU (Ubuntu, 2014) or make installation scripts, like RDO Project (RDO, 2016).

*REST API Design Patterns for SDN Northbound API* (ZHOU *et al.*, 2014), is an effort to create a more flexible northbound interface to SDN controllers, in addtion, there are shown some gaps that current northbound APIs have. For example, the use of static URIs to identify resources creates issues in terms of adaptability, thus, if the URI changes the response type (e.g. from JavaScript Object Notation (JSON) to XML), the interface client will be useless to face the change. With the aim of reducing this gap, it was developed a "truly" REST northbound interface. This new interface is done using all REST style pattern designs, thus giving a loosely-coupled architecture. However, service consumption process still relying in the developer entity and the use of the interface must be done through an external plugin to connect OpenStack Neutron project (and other cloud projects) with the Python-based Software-Defined Networking (POX) (POX, 2016) controller.

### 2.2.3 Gap Analysis

The use of a Semantic Web approach appears as an appealing manner to represent and use common data models in real scenarios. These models can be used as a common language to represent same concept across different kind of domains. Additionally, the knowledge can be reused to enhance other models. However, common knowledge is an initial step to achieve towards the creation of an automated interoperability across NFV domains.

On the other hand, communication interfaces and WS are developed to solve prob-

[1] A cloud orchestrator. It can be seen as a VIM for the NFVI component
[2] A Software Defined Networking (SDN) Controller

lems in specific scenarios and using specialized coding languages, thus, limiting interoperability to one domain. In addition, there is not a standardized manner for software developers to code, and interfaces usually cannot be consumed from one software to another software implementation without using an external components. Hence, aiming to remove above limitations, this work proposes the definition of a common data model to represent NFV components and the implementation of semantic services.

# 3 Research Problem

At least three main problems can be identified when attempting to integrate NFV technologies: (*i*) lack of well defined semantics (i.e. domain specific language), (*ii*) absence of a common understanding (i.e. shared vision) of NFV, and (*iii*) need of manual intervention to interpret, use and integrate components. Currently, software components, interfaces and services require manual intervention (e.g., to adapt interfaces, translate the semantics of variable names, parameters, tool chains, etc.) when attempting to inter-work and integrate different pieces of the NFV puzzle. While the NFV methodology to describe interfaces and abstractions (ETSI GS NFV-INF, 2014) is a guideline for developers to be followed, this document is subject to interpretation and by any means interpretable by software services and components. As a consequence, problems inherent to interface integration negatively affect NFV implementations contributing to the time and development costs along the path towards NFV services discovery and interoperability.

## 3.1 Semantics on NFV Descriptors

In order to understand how semantics are used to describe NFV elements and how the lack of a shared representation affects implementations, a brief analysis was done over two equivalent pieces of VNFD files. The syntax used to declare was compared a management interface from a Virtual Firewall defined by the ETSI (ETSI GS NFV-INF, 2014) and the OpenMano Project (OpenMano, 2014), Listings 3.1 and 3.2 respectively. In order to do, the comparison following questions were considered:

- How NFV components are described?
- How NFV definitions are done?
- How NFV terminology is defined?

Towards solve questions above, was decided to divide the analysis in two parts. One comparing descriptions between VNFD files. The other comparing descriptions within same VNFD file.

### 3.1.1 Comparison of VNFD Files

Table 1 shows the comparison items from the files. First column, is the name of the component declared by the ETSI. Second and third columns represent the declaration of the component of the files (listings 3.1 and 3.2). Although the components on table above represents

same concepts (defined by the ETSI), they can be seen as syntactically different from each other. However, on the description field can be noticed the same definition for both descriptor files. Therefore, using human reasoning, it can be deduced that the objects described are equal or equivalents.

Listing 3.1 – VNFD ETSI File

```
<connection-points>
  <management-port>
    <name>mgmt-interface</name>
    <description>Management interface</description>
  </management-port>
</connection-points>
  <pkt-in>
    <name>pkt-in</name>
    <description>Interface for packet in</description>
  </pkt-in>
  <pkt-out>
    <name>packet-out</name>
    <description>Packet out interface</description>
  </pkt-out>
```

For humans, doing a reasoning process based on contexts is an easy task. Furthermore, when exist a lack of information to do process, people start to make questions in order to understand or enhance the context. For example, on Table 1 a person may ask to himself: *is a connection point is the same concept of an external connection?*, *what is a connection? what does a management interface mean?*. The answer to those questions can be solved by reading NFV specifications, thus filling information gaps by the improving in the context and making implicit deductions. Due NFV specs are not meant to be interpreted by machines, the specifications do not represent meaningful information in a software-centric reasoning process. Hence, manual intervention is needed to make a reasoning process and a syntax interpretation.

Listing 3.2 – VNFD OpenMano File

```
vnf:
    name: TEMPLATE
    description: This is a template to help in the creation
       of your own VNFs
    -   name:                mgmt0
        type:                mgmt
        VNFC:                TEMPLATE-VM
        local_iface_name:    mgmt0
        description:         Management interface
```

Due VNFD files belong to different implementations domains (OpenMano and ETSI) and have different syntax them are useless on an inter-domain scenario, from one domain to the other. Aiming to remove the syntax boundary is necessary to do manual intervention and create a parsing mechanism to translate syntax across domains.

Table 1 – Syntax Used to declare a Manager Interface

| Component | Listing 3.1 | Listing 3.2 |
|---|---|---|
| Connection Point | connection-points | external-connections |
| Management Interface | management-port | {type:} mgmt |
| Name | mgmt-interface | mgmt0 |
| Description | Management Interface | Management Interface |

Making a review of each component and doing simple human reasoning process, following conclusions were done:

- **connection point** and an **external connection** may or may not be equivalent or equal components.
- as **mgtm** is not a real word does not have meaningful information. It may or may not be and an abbreviation for management and may or may not be equivalent to a management port. There is not additional information to clarify.
- names are chosen in and "arbitrary" way without a general pattern, the fact of being different ways to express equal components evidence this.
- description field gives information that can clarify the described component, however machines are not able to interpret it.
- to understand the components it is necessary to have some previous background/context. For example, read the ETSI specifications.

Going into the VNFD files composition, is denoted that each file is done with proprietary manner to describe components in the modeling language (e.g. XML, YAML) and a different structure. In both cases, the machines are able to read descriptors. However, due implicit descriptions are necessary to have a predefined context and/or background (such a descriptor schema or software code) in order to interpret and use the files.

Due the implicit descriptions and different NFV syntax, an inter-domain scenario is less feasible without using middleware or parser mechanisms. Thus, it can be concluded that the lack of a common understanding makes necessary manual intervention to integrate components and even though, ETSI specifications acts as a guideline to make deployments, by following them there is not guarantee a higher interoperability across domains.

### 3.1.2 Comparison Within Same VNFD Files

In this part of the analysis, it was taken equivalent components from a same descriptor file and compared their definitions. In Listing 3.1, *description* and *name* variables were taken from the components `pkt-in` and `pkt-out`. Assuming them as similar components

(clarifying that one acts as an input and the other as an output) it was easy to denote that: although them are equivalents there is not a structure to define the variables. To support the previous affirmation, additionally it was compared the variable `name`, defined as "pkt-in" for component `pkt-in` but defined as "packet-out" for component `pkt-out`. As them are formal descriptions (from ETSI specifications) and equivalent variables should have some structure or conventions on definitions. For example "pkt-in" and "pkt-out" or "packet-in" and "packet-out". This is also more noticeable in the `description` variable for *pkt-in* component is defined as an "Interface for packet in" and for the *pkt-out* component is defined as "Packet out interface". Finally, looking at the names given to some variables and components, them were defined by this work as word abbreviations without a meaningful expression for people with the correct background. In Listing 3.1, is defined the contraction of the word *packet* as *pkt* and in Listing 3.2, the word *management* as *mgmt*. This last file also has terms that without a description are useless (e.g. `local-iface-name`).

From the previous analysis is concluded that both terminology and definitions were made in an arbitrary manner and predefined for each context.

## 3.2 Semantics on NFV Deployments

Current software implementations are done using variables and methods defined by developers. These implementations have a semantic domain limited by the scope of a specific implementation. In other words, syntax and semantics used to develop a software are only useful to the software itself. Software syntax is defined without the use of common conventions or structure and cannot be extended to other complementary domains or in some cases can not be reused by components of same domain. For this reason and in order to have a common understanding of the communication syntax, is necessary the use of manual intervention to interpret syntax or create external elements to translate vocabulary from one domain to another. Furthermore, generating integration costs in order to achieve interoperability. With the aim of understanding how the issues above affect NFV implementations, it was done two scenarios of study: (i) A NFV local domain implementation. (ii) A NFV inter-domain implementation.

### 3.2.1 NFV Local Domain Scenario

Figure 8 In Figure 8 a more detailed view of NFV MANO components is presented. NFVO: incorporates new NS and VNF Packages, manage NS life-cycle [1] and manage global resources. VNFM: manage the life-cycle of the VNF instances. VIM: controls and manage the NFVI resources (computing, storage and networking).

[1] The life-cycle includes instantiation, scale-out/in, performance measurements, event correlation, termination of the instances

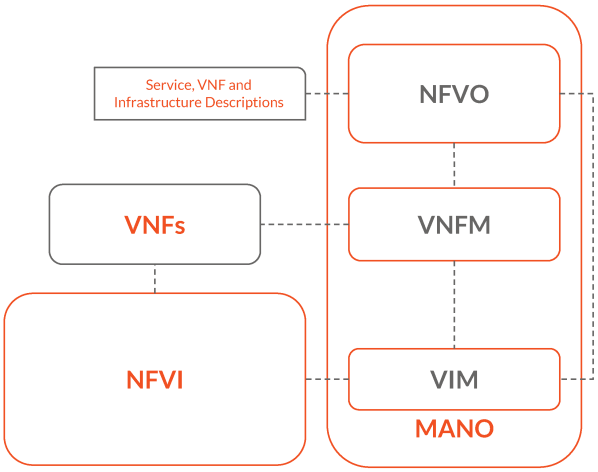


Figure 8 – NFV Architecture - MANO

From the figure above, it can be seen how NFV elements are connected among them by some defined interfaces (reference points). These interfaces allow to have interoperability between NFV elements and are defined in the ETSI specifications. However, NFV components as well as the reference points can be implemented by different developers, using different technologies, communication protocols and syntax. Thus, opening the possibility for the creation of different types of semantics and data structures to represent same concepts, such VNFs deployment process or the services developed at the top NFV.

The differences mentioned above generate the need of manual intervention in order to create interoperability. This manual intervention refers to the necessity of having people reading, interpreting and using (according to own purposes) descriptions and capabilities of components, interfaces and services. In other words, people must understand the capabilities and descriptions to integrate one component to another component within the same domain.

### 3.2.2 NFV Inter-Domain Scenario

Figure 9 shows two different NFV implementations with the same components and interfaces. Assuming deployments from different providers it may be found different semantics, coding technologies and communication protocols representing same ETSI concepts. Thereby, generating a need of manual intervention[2] in order to create interoperability across domains. For implementation A, components and interfaces are represented by doted lines and the font **Droid Sans** represents the semantics. In implementation B, straight lines are used to represent same concepts and **`Courier New`** font is used to represent the semantics.

In the figure above implementations are connected to each other through interfaces. However, interfaces from domain A only can interpret its semantics (**Droid Sans**) and can use dotted line capabilities (same thing occurs with domain B but opposite). This is due the software

[2] This work refers as manual intervention to the necessity of having humans reading, interpreting and using descriptions and capabilities of components, interfaces and services.

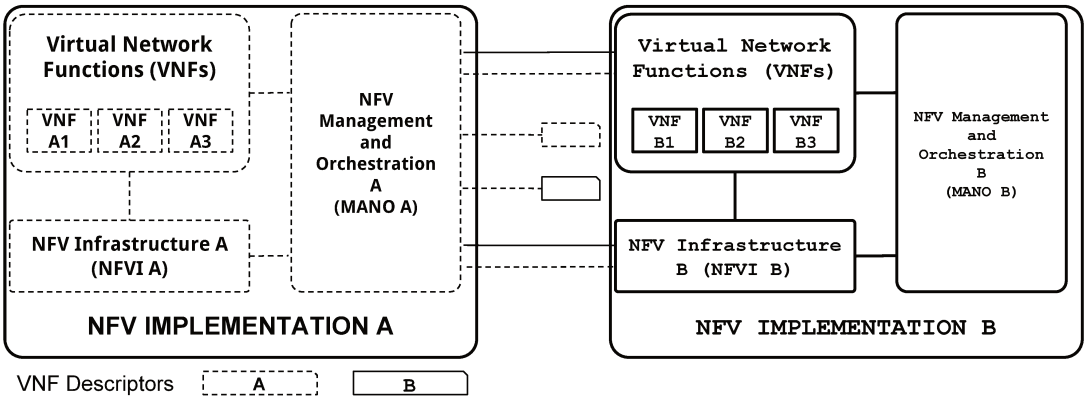


Figure 9 – Different NFV Implementations

developments are done in a syntactically manner and using implicit descriptions, without a common understanding. Hence, for NFV implementations, same components and interfaces are implemented with different syntax and descriptions. Furthermore, them is necessary to have the correct background/context.

To create interoperability across domains is necessary to do manual intervention to re-write code or to do an implementation of a middleware (Figure 10) to parse requests across domains in order to add the capability to interpret and use the semantics and components. Furthermore, this problem is more noticeable when a NFV MANO attempts to use VNFD from other implementation. For example, MANO **A** reading VNFD **B**. As shown on Figure 10, manual intervention can give multiple kinds of integration. (i) has an external component (Parser) to translate descriptors from domain **B** to domain **A**. In this case interface A remains equal. (ii) in this scenario the parser transforms requests from domain A into domain B format and vice-versa. For this case, interfaces are connected to the parser and the parsing process is not noticed by either of both domains. (iii) a new layer is added on components of domain A and B to interpret requests from both domains. For this integration, interfaces from both domains can be used.

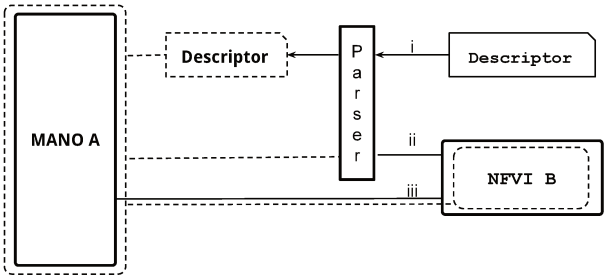


Figure 10 – NFV Domain Integration

Integration mentioned above gives interoperability to NFV. However, it involves inherent costs associated with the new developments. Furthermore, if the number of domains increases, the number of integration processes grows as well as the costs.

## 3.3 Concluding Remarks

Because of the semantics problems and implicit descriptions discussed above, manual intervention is found as an essential part of integration process. This intervention can be seen from two different perspectives: (i) to read and interpret NFV implementations (components, services and interfaces), and (ii) to use capabilities and functionalities. Both perspectives are correlated to each other (**ii** can not be done without **i**.) but seen separately due to the different costs involved. There are time and resource costs involved. For **i**, in workshops, training or self-learning (not mentioning the time spend writing manuals) and in **ii** for the integration process. Integration can include the modification of current developments or the development of an external middleware. Furthermore, if capabilities or semantics change in the service provider (server side) it may produce/bring modifications about the integration already done, thus increasing the costs. Concluding this section we found that currently manual intervention is needed to achieve interoperability for NFV in local and inter domain scenario.

# 4 NFV Ontology (NOn)

With the aim of solve the analyzed issues on the last section, this work proposes NOn, a common data model representation of NFV for SP, NO and developers. The main goal of NOn is to reduce software integration costs generate by a lack of a common manner to express semantics in NFV descriptors and WS.

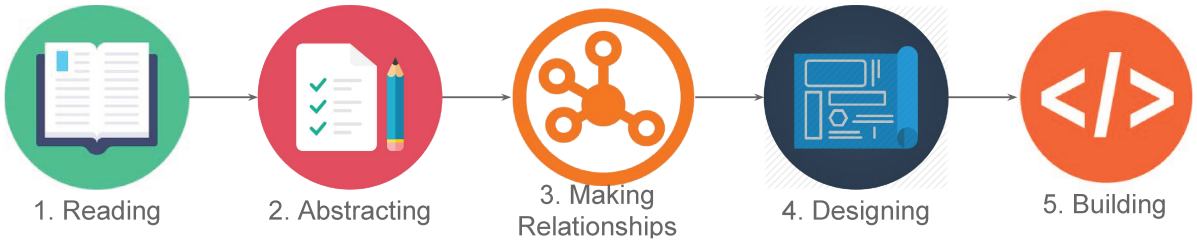


Figure 11 – NOn Implementation Road Map

This chapter presents the design and implementation process of NOn. Figure 11 shows the path followed to implement the ontology. In phase one, trough the reading of the ETSI specifications, were identified the elements of the ontology. Then, in phase two, the elements identified were abstracted and divided on two main groups: NFV descriptor elements and NFV framework elements. In phase three, data and object properties were modeled, in order to define the relationships among the ontology elements. In the last phase, using the abstracted elements, properties and relationships an Entity/Relation diagram was designed. Finally, using the resulting model the ontology was implemented.

## 4.1 Design

The implementation of NOn is based on the ETSI specifications (ETSI GS NFV-MAN, 2014) (ETSI GS NFV, 2014) (ETSI GS NFV-SWA, 2014) (ETSI, 2014b), and the model was created using as a guideline *Ontology Development 101: A Guide to Creating Your First Ontology* (NOY *et al.*, 2001). To design and create NOn data model, it was followed each step recommended for the development of an ontology. However, some steps were followed in a more rigorous manner than others. This work sees the ontology design as an iterative process, consequently the design went back and forward through the steps aiming to improve the model.

Aiming to give an overall view of the design process, the following list enumerates the steps for an ontology design. Each item contains a brief summary of the actions executed:

1. *Determine the domain and scope of the ontology:* in order to define the scope and the domain, the necessary elements to deploy a VNF were considered. Starting from the

VNFD base information elements (Table 2) and going up to the NFV framework elements (Figure 8)

2. *Consider reusing existing ontologies:* in section 2.2.1 were considered the use of NML and INDL ontologies as the base of NOn. However, as the elements contained in the data models are considered not necessary to create the VNFD information elements, in consequence, both ontologies are not reused in NFV Ontology. Nevertheless, some key intersection points were highlighted further in the chapter, aiming to make reuse in the future.

3. *Enumerate important terms in the ontology:* in this item were included the important components from the VNFD base information elements[1]. For example, deployment flavor, connection point or virtual image. These components are bounded by the defined scope.

4. *Define the classes and the class hierarchy:* the ontology classes were created from the components of the NFV architecture and those elements composing the VNFD. This work used a Down-Up strategy to create the hierarchy, starting from descriptor elements and going up to functional blocks.

5. *Define the properties of classes (slots):* the properties were created in accordance with the relationships defined for the VNFD and NFV framework elements.

6. *Define the facets of the slots:* the principal facets defined for the slots were cardinality and type values. For example, type value `string` for `description_version` element.

7. *Create instances:* the instances created were based on existing VNFD models for current NFV research projects. These models were mapped using the components defined in above items.

Table 2 – VNFD Base Information Elements

| Identifier | Type | Cardinality | Description |
|---|---|---|---|
| `vendor` | Leaf | 1 | The vendor generating this VNFD. |
| `vdu` | Element | 1...N | This describes a set of elements related to a particular VDU, see clause 6.3.1.2. |
| `connection_point` | Element | 1...N | This element describes an external interface exposed by this VNF enabling connection with a Virtual Link, see clause 6.3.1.4 (see note). |

As is presented in Table 2, base information elements (ETSI GS NFV-MAN, 2014) are used to define VNF descriptor elements. Above table shows three base information elements of a VNF descriptor. For the design process, first column was used to abstract and give the

[1] An important term is considered to those elements necessaries to deploy a VNF

name for the components of the ontology. **Type** column, was used to define components as a resource or as a data objects (Element and Leaf respectively). **Cardinality** column, gives the slot facet cardinality for components. Finally, **Description** column was used to define properties and to create the relationships among elements. As an example, elements above were defined as follows: **vendor** field is as a *data property* with *String* value and *cardinality*: 1. **vdu** and **connection_point** were defined as resources with *resource property* has_*component* (has_vdu, has_connection_point) and *cardinality*: minimum 1.

Figure 12 shows the resulting graph of modeling elements and the relationships from Table 2. In the following subsections is explained in more detail how was the process to obtain the elements and relations of the graph.

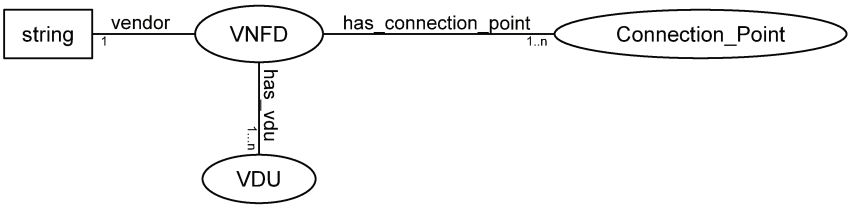


Figure 12 – Modeling NOn

### 4.1.1 Elements

To abstract the elements for NOn, the process was divided in two parts: (i) abstracting the elements contained on the VNF descriptor, named as NOn low level elements. (ii) abstracting main components of the NFV architecture, named as NOn high level elements. Thus, attempting to fulfill basic deployment variables and creating an initial data model for NFV.

#### 4.1.1.1 NOn Low Level Elements

Table 3 shows the elements of NFV taken from the VNFD information base (Table 2) and used to create low level elements of the ontology. The first column is the name given to an element in the ontology. Second column, classifies the elements: as an **Object**, if is composted by other elements or as a **Slot**, if does not have any composition. Forth column, is the cardinality defined for each element and fifth column, is the value for the elements defined as slots (e.g String). In last column, is defined to which descriptor belongs each element.

Table 3 – VNFD Elements

| Element | Type | Cardinality | Slot Type | Descriptor |
|---|---|---|---|---|
| vnfd | Object | 1...* | N/A | VNFD |
| vdu | Object | 1...* | N/A | VNFD |
| vendor | Slot | 1 | String | VNFD |

Afterwards, all components from the VNFD information base were abstracted, the process continued with the abstraction of other elements defined as **Objects**. For example, it

were taken the VDU information base elements (Table 4) to continue with the abstraction. The elements from below Table 4 were used to extend the data model. Additionally, two variations were noticed in the *vm_image* field:

- Cardinality value, maximum 1.
- Description field makes clarity on "provides a reference", thus, it was decided to have a link value (anyURI value) for the **Slot Type** field.

Table 4 – VDU Base Information Elements

| Identifier | Type | Cardinality | Description |
|---|---|---|---|
| `id` | Leaf | 1 | A unique identifier of this VDU within the scope of the VNFD, including version functional description and other identification information. This will be used to refer to VDU when defining relationships between them. |
| `vm_image` | Leaf | 0...1 | This provides a reference to a VM image |
| `vnfc` | Element | 1...N | Defines minimum and maximum number of instances which can be created to support scale //out/in. |

Table 5 is the result of mapping VDU and add them to the VNFD elements (Table 3). In this table can be observed how new object elements are defined (e.g. VNFC). The modeling process continues with the abstraction of the new objects. Thus, the process goes until all the object elements necessaries (considered as important) to deploy the VNF were abstracted.

Table 5 – NOn VNFD and VDU Low Level Elements

| Element | Type | Cardinality | Slot Type | Descriptor |
|---|---|---|---|---|
| vnfd | Object | 1...* | N/A | VNFD |
| vdu (Virtual Device Unit) | Object | 1...* | N/A | VNFD |
| vendor | Slot | 1 | String | VNFD |
| virtual_memory_resource_element | Slot | 1 | Integer | VDU |
| computation_requirement | Slot | 1 | Integer | VDU |
| id | Slot | 1 | String | VDU |
| vm_image | Slot | 0...1 | anyURI | VDU |
| vnfc | Object | 1...* | N/A | VDU |

As a final result, a table with the elements mapped from the NFV components was created. These elements, are meant to be included in the ontology as a **Class** or as a **Slot**. Table 6 is a summary of the final abstraction.

#### 4.1.1.2 NOn High Level Elements

To model NOn top level elements, the NFV framework architecture (Figure 8) was abstracted. Figure 13 shows the resulting hierarchic tree model obtained from the abstraction

Table 6 – NOn Low Level Element Summary

| Element | Type | Cardinality | Slot Type | Descriptor |
|---|---|---|---|---|
| vnfd | Object | 1...* | N/A | VNFD |
| id | Slot | 1 | String | VNFD |
| descriptor_version | Slot | 1 | String | VNFD |
| vnf_version | Slot | 1 | String | VNFD |
| vdu (Virtual Device Unit) | Object | 1...* | N/A | VNFD |
| vendor | Slot | 1 | String | VNFD |
| deployment_flavor | Object | 0...* | N/A | VNFD |
| connection_point | Object | 1...* | N/A | VNFD |
| virtual_memory_resource_element | Slot | 1 | Integer | VDU |
| scale_in_out | Slot | 0...1 | Integer | VDU |
| computation_requirement | Slot | 1 | Integer | VDU |
| id | Slot | 1 | String | VDU |
| vm_image | Slot | 0...1 | anyURI | VDU |
| vnfc | Object | 1...* | N/A | VDU |
| id | Slot | 1 | String | VNFC |
| connection_point | Object | 1...* | N/A | VNFC |
| type | Slot | 1 | String | Connection Point |
| id | Slot | 1 | String | Connection Point |

process. The elements included on the tree are classified by the ETSI as functional blocks of the NFV Architectural Framework (ETSI GS NFV-MAN, 2014), for this reason the root of the tree is the `functional_blocks` object. The second level of the tree represents the main components of NFV architecture, which includes `mano`, `nfvi`, `vnf` and `descriptor` objects. In third level, are the objects composing the main elements. For example, for the `mano` element are the `nfvo`, `vnfm` and `vim` objects and for `nfvi` element are defined `hardware`, `virtualisation` and `hypervisor` objects. Finally, for the `descriptors` element, there are the descriptors necessaries to deploy a VNF instance. Consequently, in this element was done a merge with the low level elements defined in section 4.1.1.1. The subsequent levels on the tree are the elements defined by the ETSI to composite NFV superior level components. Hence, the first definition of classes for NOn was made.

Having in count, the resulting hierarchic tree of NFV and the NML (HAM *et al.*, 2013) and INDL (GHIJSEN *et al.*, 2013) data models, this work highlights on the NFVI block a match between the elements of the ontologies. Thus, generating a possibility of reusing elements from NML and INDL in NOn. The matching elements can be observed embraced between brackets ({}) on Figure 13. The reuse of elements is feasible due, NFVI can be seen as a resource provider (networking elements included). However, these elements are not necessary for the VNFD to make the deployment of a VNF. For this reason, reusing NML and INDL ontologies is considered as out of the scope in this work.

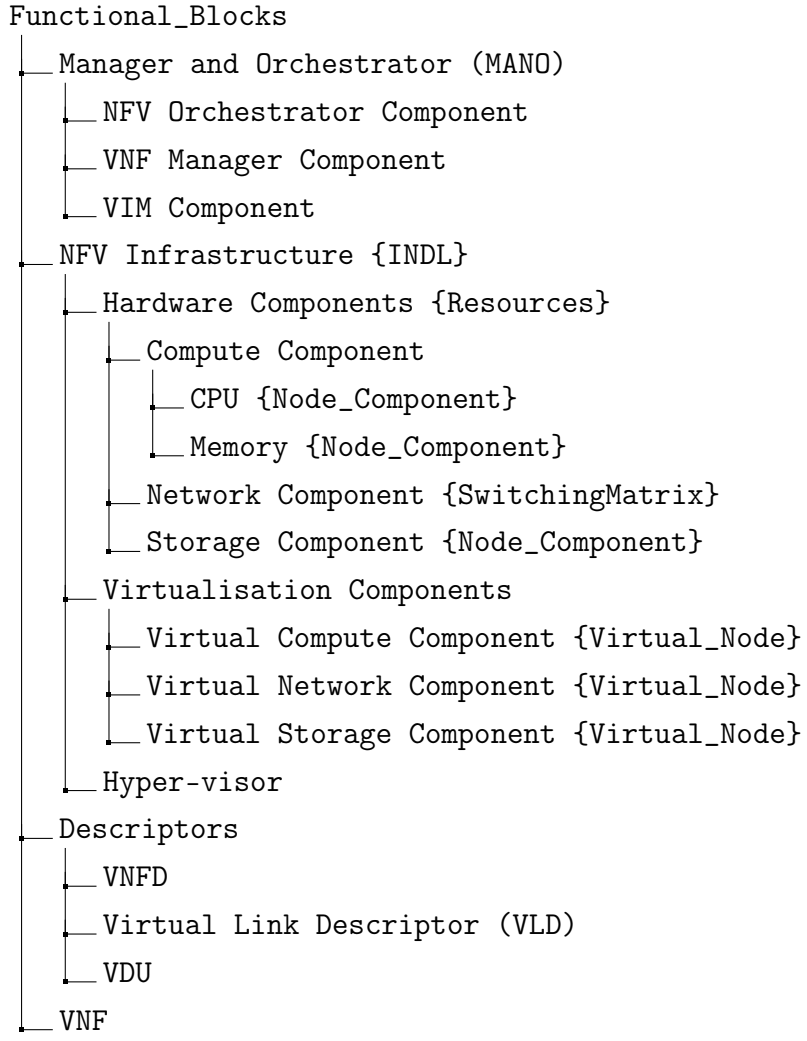


Figure 13 – NFV Framework Elements

### 4.1.2 Relationships

Afterwards, the elements in the ontology were defined, a process to create a relation among them took place. For high level elements the only relationships created were Class and Subclass. This is due, our focus is the definition of the elements necessaries in the VNFD to deploy a VNF. As is shown in Table 6, there are two types of elements associated with a descriptor file, **Objects** and **Slots**. In the descriptor, the **Slot** acts as a property with same **Cardinality** and **Slot Type** defined in the table. Instead, **Objects** are composite by other objects and slots, furthermore, with the aim of creating relationships, it was necessary to use the description field from base information elements (Table 2). In addition, the property `has_object` was defined to be use with those elements composted by other elements. For example, VNFD instance `has_vdu` VDU instance.

Table 7 shows the relationships obtained from the analysis of ETSI specifications and NOn low level elements (Table 6). First column, represents an element of the ontology. Second column, represents the properties attached to the element. Third and fourth column

represent, slot facets of the properties.

Table 7 – NOn Relationships

| Element | Property | Carinality | Value |
|---|---|---|---|
| **VNFD** | id | 1 | String |
| | descriptor_version | 1 | String |
| | vnf_version | 1 | String |
| | has_vdu | 1...* | vdu |
| | vendor | 1 | String |
| | has_deployment_flavor | 0...* | deployment_flavor |
| | has_connection_point | 1...* | connection_point |
| **VDU** | virtual_memory_resource_element | 1 | Integer |
| | scale_in_out | 0...1 | Integer |
| | computation_requirement | 1 | Integer |
| | id | 1 | String |
| | vm_image | 0...1 | anyURI |
| | has_vnfc | 1...* | vnfc |
| **VNFC** | id | 1 | String |
| | has_connection_point | 1...* | connection_point |
| **Connection Point** | type | 1 | String |
| | id | 1 | String |

### 4.1.3 Model Realization

The abstraction of NOn elements, was an iterative and incremental process. Thus, the project started by modeling the elements and relationships from the VNFD and was extended with other modeled elements. Figure 14 illustrates how, NOn is being extended from Figure 12, using new relationships and elements. The modeling process continued by adding all the abstracted elements.

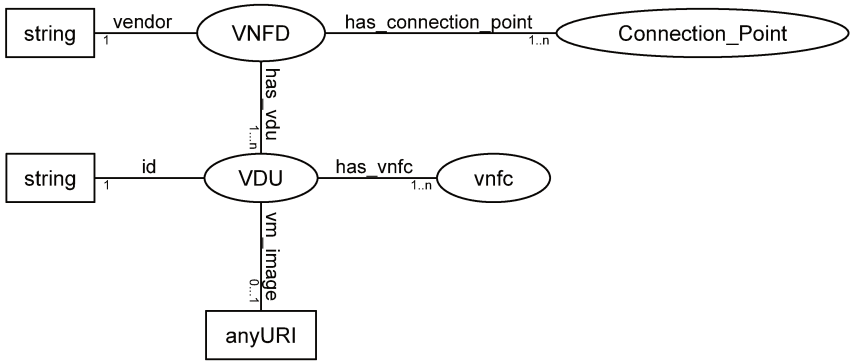


Figure 14 – Extending NOn

In ontology modeling, there exists non restriction on the size of an ontology in terms of elements, relationships or slots. Furthermore, an ontology model is not a view of one person or a group, instead, it must be a shared vision of a domain. For this reason, the model is based on

a set of ETSI specifications (developed with a common vision of experts on NFV) and limited by the elements necessaries to create a VNFD file for a VNF deployment.

The final design for the first version of NOn is presented in Figure 15, using the entity relationship diagram annotation. The figure contains all the abstracted elements, slots and slot facets (cardinality and value) from the previous sections. Due above figure is and overall of the abstracting process, the implementation is mainly based on this design.

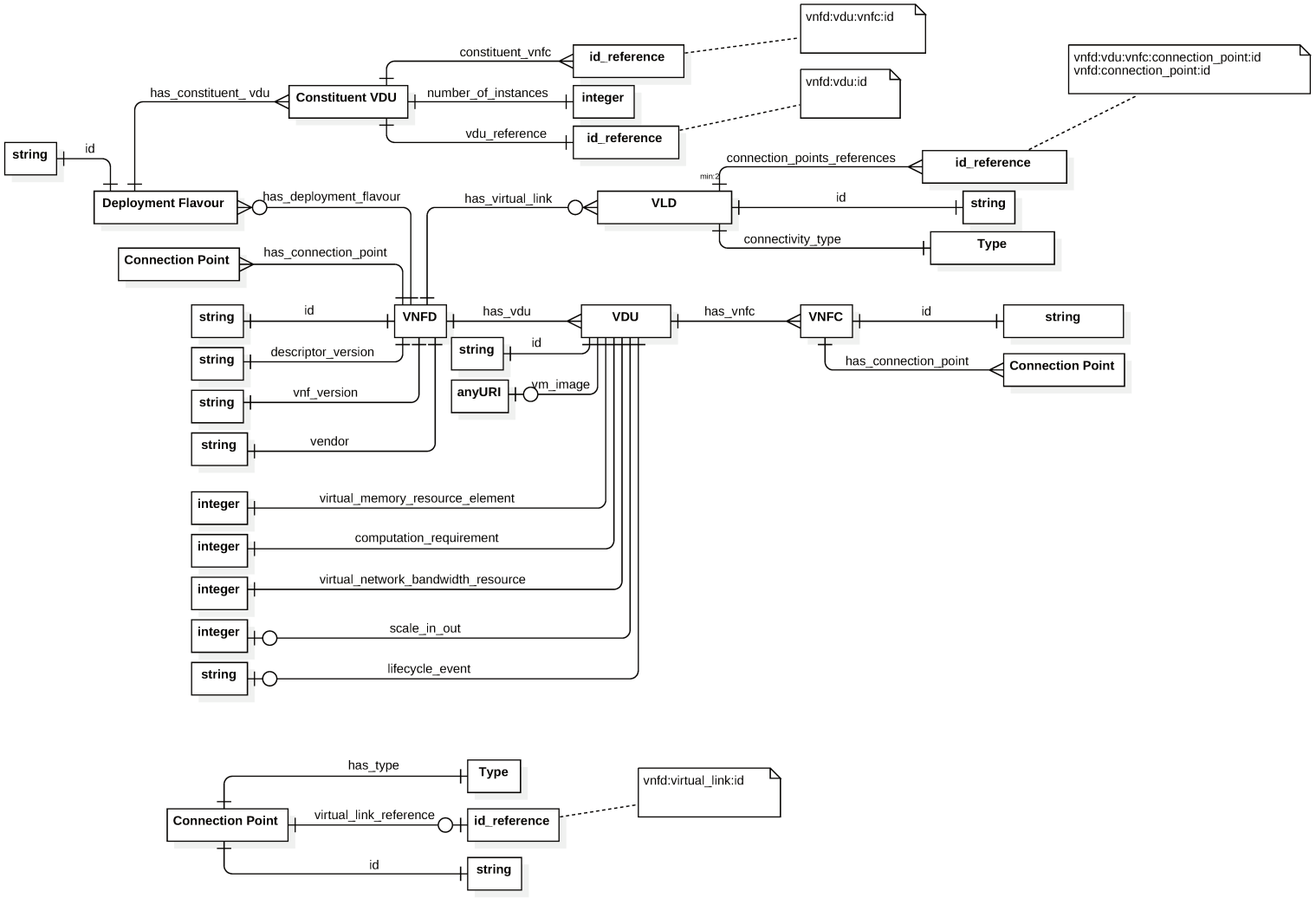


Figure 15 – NOn Model

## 4.2 Implementation

Using as a reference point the model in the Figure 15, the data and resource properties and elements were implemented using Protégé modeling tool (Protégé, 2016). As a first step, the Classes and Subclasses of the ontology were implemented. Then, data and object properties (slots) were created. Finally, VNFD instances were created using as guideline different descriptor models, taken from a NFV implementation. thus semantic VNF Descriptor files were created.

### 4.2.1 Classes and Sub-classes

The implementation process started with the creation of classes and Subclasses (high and low level elements). Figure 16 shows the resulting hierarchic tree from modeling elements in Protégé. Elements from level one[2] and two of the image are considered as top level classes. Level three, is considered as middle level classes and below level forth are bottom level classes.

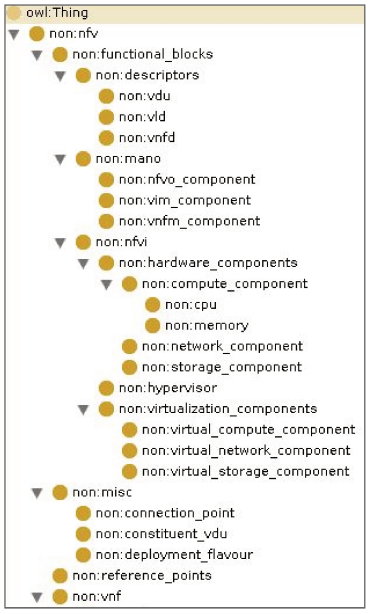


Figure 16 – NOn Classes and Sub-classes - Protégé

From the above figure, can be noticed a top level class (not modeled) named as *Misc*. This class was created in order of classifying elements that does not belong to any NFV functional blocks or descriptor files.

Each element of the ontology has the following structure:

```
Name_Space:Class
```

[2] NFV is considered as root level of the ontology. Subsequent levels start at level one (functional_blocks level)

*Name Space* variable, represents an abbreviation for the URI containing the ontology (e.g. http://www.intrig.com/ontology/non.owl) and *Class* variable, represents the class name (e.g. mano). For example, instead representing ontology classes with the complete URI:

```
http://www.intrig.com/ontology/non.owl#mano
http://www.intrig.com/ontology/non.owl#vdu
```

RDF allows to present a short version using of *name spaces*, thus, it is defined a `@prefix` for the ontology, and subsequent objects from the ontology are defined using the prefix.

```
@prefix non: <http://www.intrig.com/ontology/non.owl>
non:mano
non:vdu
```

After the class implementation was done on Protégé we generate the first data model was. Listing 4.1 presents the class data model of NOn generated by the modeling tool. The data model is generated in RDF/XML format.

Listing 4.1 – NOn Data Model: Classes and Sub-classes

```
<owl:Class rdf:about="#functional_blocks">
 <rdfs:subClassOf rdf:resource="#nfv"/>
</owl:Class>
<owl:Class rdf:about="#mano">
        <rdfs:subClassOf>
        <owl:Class rdf:ID="functional_blocks"/>
        </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#descriptors">
        <rdfs:subClassOf>
        <owl:Class rdf:about="#functional_blocks"/>
        </rdfs:subClassOf></owl:Class>
<owl:Class rdf:ID="vnfd">
        <rdfs:subClassOf>
        <owl:Class rdf:about="#descriptors"/>
        </rdfs:subClassOf>
</owl:Class>
```

Even though, at this point none of the properties of NOn are implemented, users of the ontology can use some degree of inference through the inherited rules from RDF and OWL. Following example shows how, an inference process can be achieved:

(i) `<non:descriptors><rdfs:subClassOf><non:functional_blocks>`
(ii) `<non:vnfd><rdfs:subClassOf><non:descriptors>`

From (i) and (ii) rules, (iii) can be inferred:

(iii) `<non:vnfd><rdfs:subClassOf><non:functional_blocks>`

The pattern adopted to name NOn elements is the same defined in the ETSI information elements. Using lowercase and underscore symbols (_).

### 4.2.2 Data and Object Properties

Afterwards ontology elements were implemented, modeling properties was the following step. Each descriptor and misc classes were used to add their corresponding data and object properties.

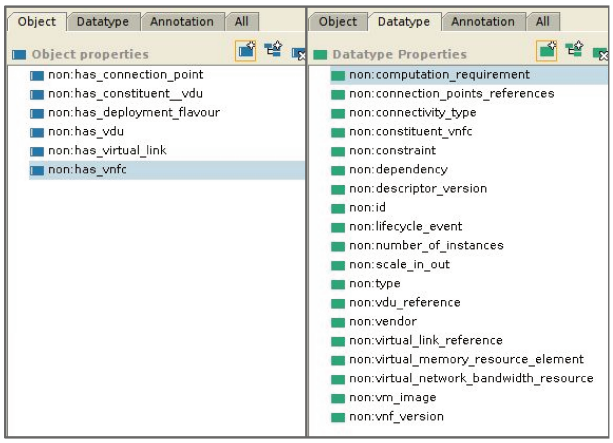


Figure 17 – NOn Data and Object Properties - Protégé

Figure 17 shows the implemented data and object properties. The structure of properties is similar to class structure, `name_space:property`. Left side of the figure represents object properties and right side represents data properties. In addition to the implementing process, there exist the need of adding a relationship between the ontology elements. Thus, it is necessary to create a **domain** and a **range** for the properties.

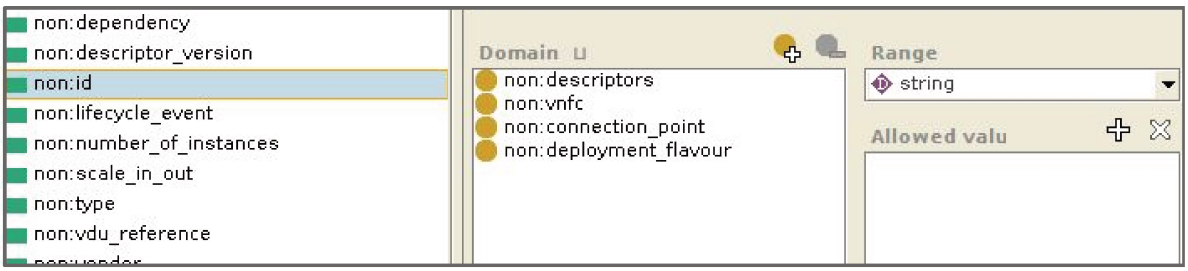


Figure 18 – NOn Properties Range and Domain - Protégé

**Domain** defines the classes allowed to use the property, and **Range** defines the slots type. Figure 18 shows the process for the **Domain** and **Range** creation. As an example, in the figure is shown the data property `non:id` with the **Range** `string` value (slot type) and the

**Domain** limited by four classes, descriptors, vnfc, connection point and deployment flavour. Finally, when all the properties were created and relationships were done, cardinality facet was added to the slots[3].

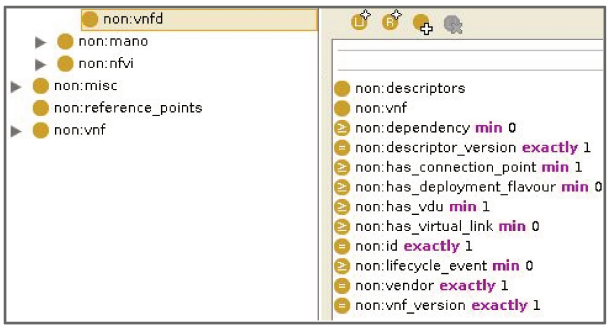


Figure 19 – NOn Property Cardinality - Protégé

Figure 19 shows the cardinality added to the elements (objects and slots) of the VNFD class. Cardinality property works as follows: **`min`** and **`max`** restrictions are used respectively for greater or equal than (**>=**) and less or equal than (**<=**) facets. **`exactly`** restriction is used for equality (**=**) facet. For example, a VNFD has one to many (1...*) VDU and is represented as: `non:has_vdu` **`min`** 1.

The result of implementing the properties can be seen in Listing 4.2. At the top of the listing (lines 1 to 15) are the object properties and in bottom (lines 15 to 29) are the data properties. In addition, the listing shows the `domain` and `range` of the properties. However, due cardinality restriction is added in the classes through the slots, is not presented in the listing.

Listing 4.2 – NOn Data Model: Data and Object Properties

```
1  ### Object Properties
2
3  <owl:ObjectProperty rdf:about="#has_vdu">
4          <rdfs:range rdf:resource="#vdu"/>
5          <rdfs:domain rdf:resource="#vnfd"/>
6  </owl:ObjectProperty>
7  <owl:ObjectProperty rdf:about="#has_deployment_flavour">
8          <rdfs:range rdf:resource="#deployment_flavour"/>
9          <rdfs:domain rdf:resource="#vnfd"/>
10 </owl:ObjectProperty>
11 <owl:ObjectProperty rdf:about="#has_vnfc">
12         <rdfs:range rdf:resource="#vnfc"/>
13         <rdfs:domain rdf:resource="#vdu"/>
14 </owl:ObjectProperty>
15
16 ### Data Properties
17
18 <owl:DatatypeProperty rdf:about="#vnf_version">
19         <rdfs:domain rdf:resource="#vnfd"/>
20         <rdfs:range rdf:resource="http://www.w3.org/2001/
              XMLSchema#string"/>
21 </owl:DatatypeProperty>
```

[3] Cardinality facet was added one by one to each element of the ontology.

```
22 <owl:DatatypeProperty rdf:about="#dependency">
23         <rdfs:domain rdf:resource="#vnfd"/>
24         <rdfs:range rdf:resource="http://www.w3.org/2001/
             XMLSchema#string"/>
25 </owl:DatatypeProperty>
26 <owl:DatatypeProperty rdf:about="#virtual_link_reference">
27         <rdfs:domain rdf:resource="#connection_point"/>
28         <rdfs:range rdf:resource="http://www.w3.org/2001/
             XMLSchema#string"/>
29 </owl:DatatypeProperty>
```

In ontology modeling cardinality facet is presented as a subclass of RDF and an property of OWL and added to the classes as a slot. Listing 4.3 shows cardinality as an RDF *subclass of* (line 5) OWL *restriction class* (line 6), additionally has the OWL minimum cardinality property with the RDF data type property *integer* and value *one* (line 8).

Listing 4.3 – NOn Data Model: Cardinality Restriction

```
1  <owl:Class rdf:ID="vnfd">
2    <rdfs:subClassOf>
3     <owl:Class rdf:about="#descriptors"/>
4    </rdfs:subClassOf>
5    <rdfs:subClassOf>
6     <owl:Restriction>
7      <owl:onProperty><owl:ObjectProperty rdf:ID="has_vdu"/></
         owl:onProperty>
8      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/
       XMLSchema#int">1</owl:minCardinality>
9     </owl:Restriction>
10   </rdfs:subClassOf>
11 </owl:Class>
```

When all the cardinality facets were implemented, the first version of NOn is considered as finished. As a consequence of the implementation process, additionally, a first semantic VNFD template was created. This template follows the ETSI specification and is ready to be used for instance creation.

## 4.3 NOn Use Cases: Semantic VNFD

Aiming to test NOn in real NFV implementations, two VNFD descriptors from different NFV implementations were parsed into NOn VNFD instances. In following use cases, descriptors from current implementation were taken and attempted to be matched with the elements and components included in the ontology. Protégé tool was used to create the instances.

Table 8 – Test Parameters

| **Parameter** | **Description** |
|---|---|
| Goal | The main objective to be achieved by executing the test. |
| Preconditions and Assumptions | Which are the conditions or assumptions necessaries to perform the test. |
| Test Data | Which are the data inputs to perform the test. |
| Description | Description of the Test process. |
| Testing tools | Tools needed to perform the test. |
| Post-Conditions | After performing the test which is the expected state. |
| Expected Results | Which are the expected results after executing the test. |
| Expected vs Obtained Results | A comparison between the expectations and the reality after perform the test. |

Table 8 shows the parameters included in the analysis for each use case. First column, is the name of the parameter to define each use case and the second column is the description of the parameter. For each use case the following process was executed:

- Take a VNFD file from current NFV deployments (e.g. Listing 4.4).
- Use Protégé GUI to parse elements (Figure 20).
- Made a comparison among the original descriptor and the semantic one.

The above process is explained in detail in Subsection 4.3.1. Hence, subsequent use case omits the detailed explanation, instead contains the results of the test.

### 4.3.1 Use Case I: OpenBaton VNFD

Table 9 shows the test parameters and the results obtained for the OpenBaton use case.

**Use Case Process:**

Through the use of Protegé tool, was created a VNFD instance from a OpenBaton descriptor file (Listing 4.4). To create the semantic VNFD, there exist two options, a Top-Down and Down-to-Top mapping process. First option, starts by creating the VNFD instance (Figure 20) and going down to create lower level elements (e.g. `non:vdu`) and their subsequent

Table 9 – OpenBaton Use Case

| Parameter | Description |
|---|---|
| Goal | Parse a OpenBaton VNFD file, into a semantic file using NOn vocabulary. |
| Preconditions and Assumptions | VNFD file must be working in a real implementation. |
| Test Data | OpenBaton VNFD file (OpenBaton VNFD, 2014). Listing 4.4 shows a descriptor from an OpenBaton example. |
| Description | Use a OpenBaton VNFD file to create a semantic descriptor using an ontology modeling tool. In this process each element contained in the descriptor is interpreted and attempted to be parsed with the elements in NOn. |
| Testing tools | Protegé, for instance creation. |
| Post-Conditions | Semantic VNFD created. |
| Expected Results | All ETSI OpenBaton VNFD elements mapped with NOn. Proprietary OpenBaton elements not mapped |
| Expected vs Obtained Results | All the elements belonging to ETSI specifications were able to be included in the semantic descriptor. Proprietary elements such `type` or `event` (lines 4, 21) were not able to be mapped in the semantic descriptor. Properties such `name` can be mapped using `non:id` element. However, if OpenBaton is reserving this field for orchestration purposes, it can generate conflict. Other properties like `endpoint` or `vimInstancename` can be included using elements from other NOn objects, such `non:vnfm` and `non:vim` respectively. |

elements (e.g. `non:vnfc`). The other option starts by creating low level elements of the OpenBaton VNFD (e.g. `non:connection_point`) and continue creating upper level elements that contain lower components (e.g. `non:vnfc`). Both options are finished when all elements are mapped.

Listing 4.4 – OpenBaton VNFD

```
1  { "vendor":"fokus",
2    "version":"0.1",
3    "name":"iperf-client",
4    "type":"client",
5    "endpoint":"generic-vnfm",
6    "vdu":[{
7      "vm_image":["iperf_client_image"],
8       "virtual_memory_resource_element":"1024",
9       "virtual_network_bandwidth_resource":"1000000",
10      "vimInstanceName":"10.1.1.25-vim-instance",
11      "vdu_constraint":"",
12      "scale_in_out":2,
13      "vnfc":[{
14        "connection_point":[{
15                  "virtual_link_reference":"private"
16        }]}]}],
17      "virtual_link":[{"name":"private"}],
18      "lifecycle_event":[{
19        "event":"INSTANTIATE",
20        "lifecycle_events":["install.sh"]},
```

```
21          {"event":"CONFIGURE",
22           "lifecycle_events":["server_configure.sh"]
23          }
24        ],
25       "deployment_flavour":[{
26          "df_constraint":["constraint1","constraint2"],
27          "costituent_vdu":[],
28          "flavour_key":"m1.small"
29          }]
30 }
```

Since in this work is attempted to execute a sequential process to map the elements we opt for a Top-Down technique allowing Protégé to guide the creation. When a top level class is being created, Protégé underline mandatory classes and slots to be instantiated. Furthermore, giving the opportunity of creating those elements from the top level class. Figure 20 shows the VNFD mapping process. In the main software window, can be seen the mapping process of the VNFD elements. Small software window , is used to create `non:vdu` lower element.

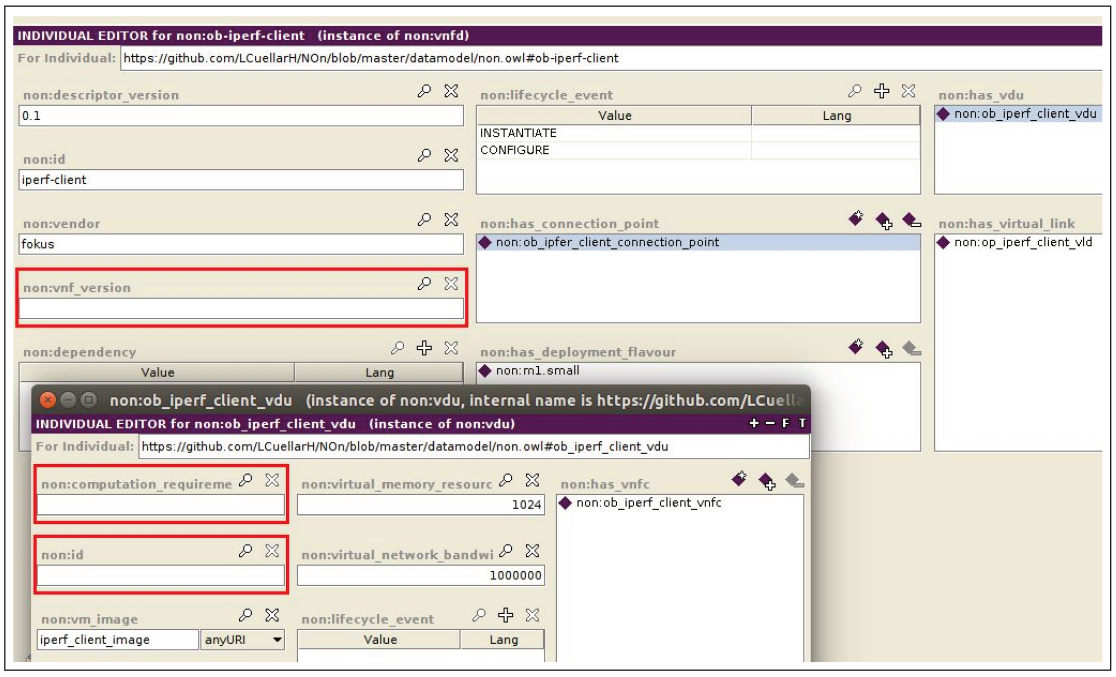


Figure 20 – Parsing OpenBaton VNFD File

The red frames show the facets defined as an **`exactly`** or **`min`** one cardinality, which means a mandatory property. However, the Protegé tool allows creating instances of the classes without instantiation of mandatory properties. In the mapping process, the elements and properties defined by the ETSI were created. However, there exists proprietary elements defined by OpenBaton in the descriptor file that were not able to be mapped. Other properties included in the OpenBaton file, defined by the ETSI and modeled in the ontology but not included in the base information elements, can be mapped using NOn top level classes. However, as Protégé tool uses inference to create the instances and currently for NOn there is not a direct relation between descriptors and some top level elements this mapping must be done directly on the semantic file (Listing 4.5).

Listing 4.5 – OpenBaton Semantic VNFD File

```
1  @prefix non:https://github.com/LCuellarH/NOn/blob/master/
       datamodel/non.owl.
2  ### non:#ob-iperf-client
3  non:ob-iperf-client rdf:type owl:NamedIndividual,
4                                 non:vnfd;
5        non:descriptor_version "0.1"^^xsd:string;
6        non:lifecycle_event "CONFIGURE"^^xsd:string,
7                            "INSTANTIATE"^^xsd:string;
8        non:vendor "fokus"^^xsd:string;
9        non:id "iperf-client"^^xsd:string;
10       non:has_deployment_flavour non:m1.small;
11       non:has_vdu non:ob_iperf_client_vdu;
12       non:has_connection_point;
13       non:ob_ipfer_client_connection_point;
14       non:has_virtual_link non:op_iperf_client_vld.
15
16 ###  non:#ob_iperf_client_vdu
17 non:ob_iperf_client_vdu rdf:type owl:NamedIndividual,
18                                   non:vdu;
19       non:virtual_network_bandwidth_resource "1000000"^^xsd
           :int;
20       non:virtual_memory_resource_element "1024"^^xsd:int;
21       non:scale_in_out "2"^^xsd:int;
22       non:vm_image "iperf_client_image"^^xsd:anyURI;
23       non:has_vnfc non:ob_iperf_client_vnfc.
```

Listing 4.5 shows the resulting file for the OpenBaton VNFD instance[4] and its subsequent VDU instance (implemented in Figure 20). Above file, is written in RDF/N3 (NOTATION3, 2011) language[5], a compact and readable alternative to RDF/XML. Line 11 and 12 shows the object property (`non:has_element`) between `non:vnfd` instance and the `non:vdu` and `non:connection_point` instances. Hence, using this relationship an inference process can be executed.

Due OpenBaton has developed its components (such descriptors) following ETSI specifications, this work assumes OpenBaton descriptor model as an excellent option to test NOn model. In addition, some interesting elements included in OpenBaton descriptor were found as an opportunity to enhance and increase the model, for example, `lifecycle_events` (lines 22, 23). However, possible new elements must be in consideration with NFV community (including ETSI) before being added the into the model.

### 4.3.2 Use Case II: OpenMano VNFD

Table 10 shows the test parameters and the results obtained for the OpenMano use case.

This work finds OpenMano as a project that does not follow ETSI specifications to define its components and uses proprietary syntax. Thus, increasing difficulty of the mapping process. Furthermore, this work foresees interoperability issues of OpenMano with other NFV

[4] To see complete file refer to Annex C Listing C.1

[5] From now on listings with pieces of semantic code will be represented on N3 language

Table 10 – OpenMano Use Case

| Parameter | Description |
|---|---|
| Goal | Parse OpenMano VNFD file into a semantic file using NOn vocabulary. |
| Preconditions and Assumptions | VNFD file must be working in a real implementation. External Connection and Bridge-ifaces are connection points |
| Test Data | OpenMano VNFD. |
| Description | Use an OpenMano VNFD file to create a semantic descriptor using an ontology modeling tool. Through the use of a Down-to-Top process is interpreted each element contained in the descriptor and tried to make a match with the elements in NOn. |
| Testing tools | Protegé, to create VNFD instance. |
| Post-Conditions | Semantic VNFD created. |
| Expected Results | ETSI OpenMano VNFD elements mapped with NOn. OpenMano elements not mapped. |
| Expected vs Obtained Results | `non:connection_point` and `non:vnfc` were able to be mapped. However, as in OpenMano file there is not a direct relation between VNFC and VNFD elements in the OpenMano file, it was not possible to create a relationship between them. It is possible to make this relationship and the mapping of `non:vm_image` property, creating a `non:vdu` element. It was found that some elements not mapped with NOn are elements defined in the ETSI specifications. |

implementations. However, the elements not included in the ontology, but defined by the ETSI and included in OpenMano descriptor are seen as an opportunity to enhance and increase the model.

For more information about the VNFD file and NOn VNFD file, refer to Annex B (Listing B.1) and Annex C (Listing C.2) respectively.

## 4.4 Conclusions

NOn represents an initial step to reduce gaps of interoperability in terms of vocabulary among different NFV implementations. Due mapping reasons NOn is a more appropriated solution for those implementations following ETSI specifications in the element definitions. However, as NOn is a language and not a semantic instance implementation, in addition, by adding more effort in the mapping process is possible to create/increase more elements mapped from VNFD files to the semantic approach. To do this its necessary to create files without the use of a modeling tool. Furthermore, it is possible to add proprietary syntax (with Semantic Web approach) not included on ETSI specifications, for proprietary components.

# 5 Semantic NFV Services (SnS)

The definition of Web Service methods, capabilities and semantics are particular for each implementation, additionally, relying in the developing entity. In software development each software can be considered a different world. Thus, the process to create definitions is something that cannot be standardized in a multi-domain scenario. Even though, REST protocol[1] attempts to be self descriptive, due its implicit service descriptions, is necessary to have a context or background to consume and use the service capabilities, consequently, manual intervention is needed. With the aim of exploring the above-mentioned issues in an NFV context, imagine a scenario in which the MANO component from one provider attempts to consume NFVI services from a different provider. In this case, an integration process is impossible to achieve without manual intervention.

This chapter presents the concept and implementation of Semantic nFV Services (SnS) validated with a few open source Network Function Virtualization projects. SnS is a proposal aiming to add semantic service descriptions to NFV Web Service (interfaces and APIs), in order to reduce manual intervention to integrate services.SnS attempts to reuse current software technologies (such REST and N3 language) and NFV projects, instead, creating new technologies or develop new components with the same capabilities, for example a semantic NFVO.

Firstly, we show the process of creating semantic services for NFV and developing a *Generic Client* to consume those services. Secondly, we present the implementation of an inference process to create a goal based semantic services workflows. Additionally, an improvement for the *Generic Client* is presented by adding the capability of creating and consuming inferred workflows. Finally, we describe a Proof of Concept implementation to prove the usability for *Generic Client* in existing NFV projects.

## 5.1 Creating Semantic Services

One of the goals of SnS is to reuse current technologies, consequently, this work have chosen RESTdesc (RESTdesc, 2011) technology to create the semantic service descriptions. RESTdesc allows the reuse of existing ontologies (e.g. NOn) and already developed REST services. Furthermore, its implementation does not require modifications in the service capabilities and methods, instead provides a mechanism to describe and enhanced them.

With the implementation of semantic service technology, this work aims to reduce the need for manual intervention. However, to introduce semantic technologies, is necessary to use or implement smart agents and inference engines in order to consume the semantic de-

[1] This work assumes the use of WSs for NFV implementations. Particularly REST protocol

scriptions. Thus, adding extra components in the service deployment process. In addition, the implementation of new components and the creation of descriptions, increases the developing cost, in terms of time and resources, nevertheless, this cost can be compensated by reducing the creation of manuals, training or services integration.

The implementation of SnS was done in two stages: (i) adding semantic descriptions to (current and new) REST WS; (ii) creating a generic REST client to consume the services.

### 5.1.1 Adding Descriptions to Services

The Figure 21 shows an abstraction of how the service descriptions are added in current Web Service. In the left side of the figure is shown the semantic descriptions (based on NOn data model). In the right side of the figure is represented the WSs themselves. Additionally, by adding semantic descriptions to the services, the normal behavior of the WS and its capabilities are not affected, instead, the usability is enhanced.

This work names as Semantic nFV Services, to the combination of semantic descriptions and the NFV Web Services.

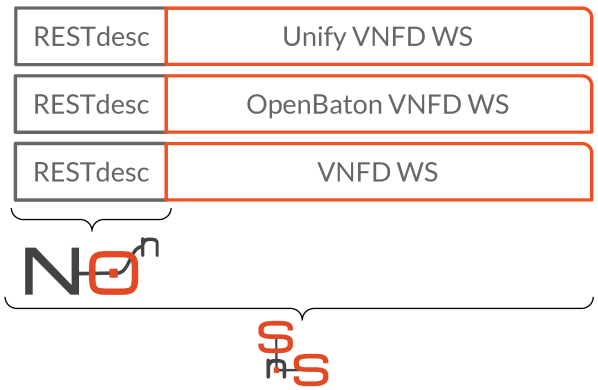


Figure 21 – SnS Adding Semantic Descriptions

With the aim of adding semantic descriptions to NFV services, this work developed Web Services with the capability of generating files used by OpenMano and OpenBaton projects to deploy VNFs, such VNF Descriptor. These services were implemented using JAVA Spring Framework (Spring Framework, 2002).

Initially, and using OpenBaton VNF deployment process as reference, two services were created[2]:

- A service with the HTTP GET Method retrieving a JSON file with the corresponding VNFD format, Listing 5.1.

[2] This process will be discussed in section 5.3.2

- A service with HTTP GET Method retrieving a YAML Ain't Another Markup Language (YAML) file containing associated Metadata (Listing 5.4) for VNF Descriptor, Listing 5.5.

Both services, receive deployment parameters as inputs (e.g. `vm_image`) and retrieve a file filled with those parameters (e.g Listing 4.4).

Listing 5.1 – OpenBaton VNFD WS

```
1  GET /nfv/parser/openbaton/vnf/vnfd?
2  vendor="value"&
3  version="value"&
4  name="value"&
5  type="value"&
6  endpoint="value"&
7  vim_instance="value"
8  configurations="value"&
9  vm_image="value"&
10 virtuallink="value"&
11 minCPU="value"&
12 minBW="value"&
13 minRAM="value"&
14 lifecycle="value"&
15 dev_flavour="value"&
16 provides="value"&
17 scaleinout="value"&
18 HTTP/1.1
19 Host: localhost:8080
20 Content-Type: application/json
```

Listing 5.1 is the representation of the VNFD service. Line 1, represents the service method and the service URI. Lines 2 to 17, represent the parameters accepted by the service. Line 18, represents the communication protocol. Lines 19 and 20 respectively are: the Host IP and the type of the retrieving file. From the listing, using human reasoning process and having an adequate background, developers can interpret some parameters in the listing, for example, `vendor` or `minCPU`. In addition, the background can be used to consume the service. However, to people without a manual, background or service description, it is difficult to understand and use each parameter to consume the service.

Table 11, shows a brief summary with the parameters used to develop VNFD Web Service. In the first column, the elements are taken from OpenBaton[3] file, in second column is the name given to the parameters in the service. With the description of above table, parameters in Listing 5.1 are easier to understand and use. However, the interpretation of service capability can not be achieved without manual intervention.

To remove or reduce the manual intervention constrain the WS consumption process, a semantic service description was created and added to the service using RESTdesc and NOn and HTTP data models. Thus, service method, content type and URI were described using an ontology defined for HTTP and the query parameters were described using NOn.

[3] To see specification of each parameter refer to openbaton.github.io/documentation/vnf-descriptor/

Table 11 – OpenBaton VNFD WS Parameters

| **OpenBatonVNFD Parameters** | **WS Parameter** |
|---|---|
| vendor | vendor |
| version | version |
| name | name |
| type | type |
| endpoint | endpoint |
| vim_instance | vim_instance |
| configurations | configurations |
| vm_image | vm_image |
| virtual_link | virtuallink |
| computation_requirement | minCPU |
| virtual_network_bandwidth_resource | minBW |
| virtual_memory_resource_element | minRAM |
| life_cycle | lifecycle |
| deployment_flavour | dev_flavour |
| provides | provides |
| scale_in_out | scaleinout |

Listing 5.2 shows the service description[4] done for OpenBaton VNFD Web Service. From the listing, can be observed how, RESTdesc description is implemented using N3 language and divided in three parts:

Listing 5.2 – OpenBaton VNFD Semantic Service Description

```
1  {#Pre-conditions
2  ?vnfd a non:vnfd;
3    non:id ?vnfd_id;
4    non:descriptor_version ?ver_des;
5    non:has_vdu ?vdu.
6  ...
7  ?vdu a non:vdu;
8    non:vm_image ?vm_image
9  ...
10 ?vl a non:vld;
11   non:connectivity_type ?vl_type.
12 }
13 =>
14 {#Process
15 _:request http:methodName "GET";
16   http:MessageHeader "Content-Type: application/json";
17   http:requestURI ("http://localhost:8080/nfv/parser/
       openbaton/vnf/vnfd?vendor="?vendor"&version="?ver_des"&
       vm_image="?vm_image"&virtuallink="?vl_type"");
18   http:resp [ http:body json:openbaton_vnfd].
19 #Post-conditions
20 ?vnf non:has_vnfd ?non_vnfd.
21 }.
```

- **Pre-Condition:** Lines 1 to 10, represent the set of rules that must be executed in order

[4] Due practical manners, there are some of the parameters implemented in the service description but removed from the lsiting, such, ontology parameter `@prefix`

to consume the service capability. For this WS, NOn ontology is used to perform the variable declaration. Furthermore, in the description exists a match between each element of Table 11 and the elements defined in NOn. For example, **vendor** variable match with `non:vendor` element. Additionally, in N3 language, is necessary to pass the elements taken from an input file to a variable in the descriptor file, thus, variables can be used on execution time. For example, `non:vnfd` pass all the elements belonging to the VNFD semantic file (Listing 4.5) to `?vnfd` variable, then, all the elements taken from `?vnfd` belongs to the descriptor file and can be used without referencing the file, question mark (**?**) represents the variable declaration in N3. Line 7, illustrates last process. In the listing, is not necessary to make reference to a specific VDU element, instead, when it refers to a `non:vdu`, is inferred that the reference is to all VDU elements contained on the `?vnfd` variable.

- **Process:** These are the actions to be executed if the rules from preconditions item can be achieved. For example, an HTTP request.
- **Post-Condition:** These are the actions to perform or the is state to be achieved, if the process is executed. For example, VNF Descriptor file created.

The describing process for semantic services, is not an easy task, developers must have good knowledge of the services to be described, capabilities, the describing language (N3) and the ontologies to used (e.g. NOn). For example, due there are several objects in NOn using the data property `id`, such VNFD, VDU or VNFC, in line 4 of Listing 5.2 may be an ambiguity in the variables, however, this ambiguity is removed by the previous line. In line 3 on same listing, is defined `non:vnfd` as a variable and the line is finalized using a semicolon (;).

The use of a semicolon implies that, subsequent lines are making a reference to the variable previously declared. To end this reference (in order to declare other objects in a different context) is necessary to use a dot (line 4). Thus, the interpretation for VNFD declaration (lines 2 to 5) is:

- *if* there exist an element with the type `non:vnfd`, take the element and pass it to `?vnfd` variable.
- *then*, *if* there exist an element with the type `non:id` **and** belongs to the `?vnfd` variable, take it and pass it to the `?vnfd_id` variable.
- *then*, *if* there exist an element with the type `non:descriptor_version` **and** belongs to the `?vnfd` variable, take it and pass it to the `?ver_des` variable.
- *then*, *if* there exist an element with the type `non:has_vdu` **and** belongs to the `?vnfd` variable, take it and pass it to the `?vdu` variable.

Same process, is implemented with other elements included in the description, such `?vdu`. Precondition process continues until whole rules are satisfied. Finally, above description can be read as: *if* there exist an element with the type `non:vnfd` **and** has the parameters below (`non:vdu`, `non:vld`, etc.), execute the HTTP call. *then*, the `non:vnf` variable must be created and associate it with `non:vnfd` variable.

The OpenBaton Metadata Web Service is explained and used in Section 5.2.

### 5.1.2 Consuming Semantic Services

To consume Semantic nFV Services, is necessary to use a client capable of adapting itself, in order to consume different types of REST descriptions. Hence, in this work was developed a JAVA based client, named as *Generic Client*. This client, does not have predefined parameters to consume services, such headers or methods and its coding was done using basic libraries. Thus, instead using JAVA REST frameworks, this work opted to use the default JAVA library for HTTP connections, *java.net*. This was done just to proof the simplicity of creating a client.

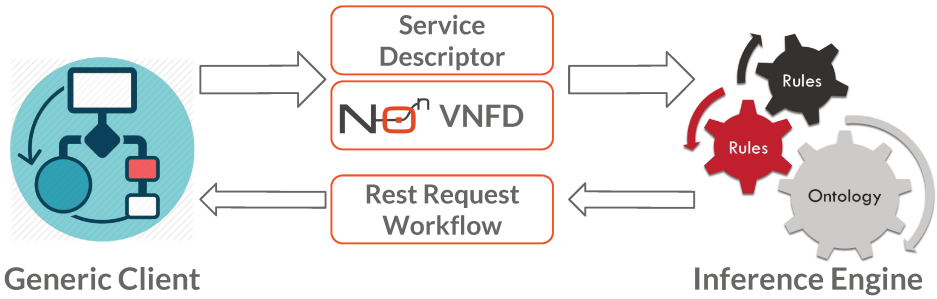


Figure 22 – Generating Dynamic WS request

On the other hand, *Generic Client* implements an external inference engine to make reasoning processes, Euler Yet another proof Engine (EYE) (Jos De Roo, 2009). EYE inference engine, receives some facts and make inference process to retrieve some other deduced facts. Figure 22 illustrates how the inference engine interacts with the *Generic Client*. First, the engine receives a set of rules (service description) and a context (Semantic VNFD) from the *Generic Client*, then, it retrieves a workflow with the inferred REST request. It is important to denote that NFV service descriptions containing a query requests, are not possible to do without NOn.

Listing 5.3 shows an example of one response given by the inference engine. This workflow can be read as:

- An HTTP GET request, called `_:sk0` is done to the `/nfv/parser/openbaton/vnf/vnfd` URI exists.
- The request, has the header type `application/json`.

- The request response represents of a `json:openbaton_vnfd` file and is given by the element `_:sk1`.

Listing 5.3 – Inference Engine Response

```
1 _:sk0 http:methodName "GET".
2 _:sk0 http:MessageHeader "Content-Type: application/json".
3 _:sk0 http:requestURI ("http://localhost:8080/nfv/parser/
    openbaton/vnf/vnfd?vendor=" "fokus" "&version=" "0.1" "&
    name=" "iperf-client" "&vm_image=" "iperf_client_image"^^
    xsd:anyURI "&virtuallink=" "private" "&lifecycle=" "
    CONFIGURE" "&dev_flavour=" "m1.small" "&scaleinout=" "2"^^
    xsd:int "").
4 _:sk0 http:resp _:sk1.
5 _:sk1 http:body json:openbaton_vnfd.
```

In execution time, and using the resulting workflow (Listing 5.3), *Generic Client* builds up the request and consumes the service. The process to create a request is as follows:

- Client, uses the `http:methodName` and `http:requestURI` (line 1 and 3 respectively) as fixed parameters. This means, for all requests is mandatory to the use both parameters.
- The other parameters, are implemented in an *if-then* implication process. For example, *if* there exists a `http:MessageHeader`) *then* put the header in the request.

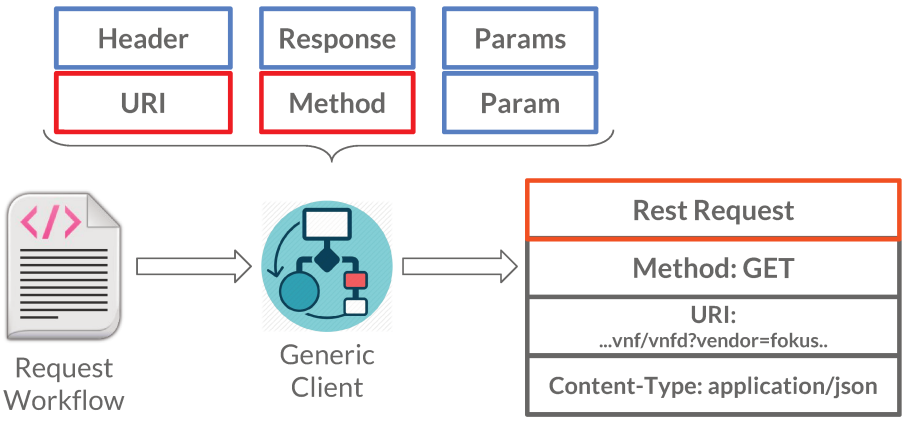


Figure 23 – Creating a Dynamic WS request

Figure 23 shows how, is the process when *Generic Client* reads the workflow file. In first step, client puts the URI and Method parameters into the JAVA request, then, the client performs an algorithm to make a match between the parameters in the file and the possible parameters composing the HTTP request. This matching process is done according to the defined method, for example, a POST method can have a body file, but a GET method cannot. In other words, *if* a parameter is in the file *then*, is added to the JAVA request. In consequence, the client adapts itself according parameters included in descriptions and using the subsequent inference workflow, thus, none of the parameters are predefined.

Thereby, the *Generic Client* born, a REST client that was created once, and is capable of consuming different REST Web Service without manual intervention to interpret capabilities.

## 5.2 SnS Workflow Inference

In the last section was used the term "workflow", to make a reference to the response given from the inference engine. This response included, the process and the parameters necessaries to create an HTTP request in order to consume a REST Web Service. In this section, same term is used to make a reference to the response given by the inference engine, with a plan including a sorted list of HTTP requests. In addition, the list is built with the aim of achieving a predefined goal.

Listing 5.4 – OpenBaton Metadata File

```
name: vnfPackage_name
scripts-link: scripts_link
image:
    upload: option
    ids: list_of_ids
    names: list_of_names
    link: image_link
image-config:
    name: image_name
    diskFormat: disk_format
    containerFormat: container_format
    minCPU: min_cpu
    minDisk: min_disk
    minRam: min_ram
    isPublic: is_public
```

In first part of the section both SnSs[5] (Listings 5.1 and 5.5), were taken and used to create a workflow in order to deploy a VNF using the OpenBaton NFVO. In the second part, *Generic Client* was improved with the capability of consuming the new workflow.

### 5.2.1 Creating Dynamic Workflow

Listing 5.4 shows the file defined by OpenBaton to create the Metadata associated to a VNFD file, and Listing 5.5 shows the definition of the REST Web Service with the capability of retrieving this Metadata file.

Listing 5.5 – OpenBaton Metadata WS

```
GET /nfv/parser/openbaton/vnf/metadata?
name="value"
 scriptslink="value&
 imaname="value"&
 upload="check"&
 link="value"&
 diskFormat="value"&
```

[5] From now on is referred as SnS to the WS created for NFV and with semantic descriptions

```
 containerFormat="value"&
 minCPU="value"&
 minDisk="value"&
 minRam="value"&
 isPublic="value"&
 scaleinout="value"&
HTTP/1.1
Host: localhost:8080
Content-Type: application/json
```

Inference engines, have the feature of infer a context and deduce facts from a given knowledge. In addition, and using the new facts to get a context, these engines are able to create a plan containing a sequence of HTTP requests. The inference process done by the engine is as follows:

- Knowledge Definition: In this step are defined the inputs for the inference engine. For example, NFV service descriptions, VNF semantic descriptor or NOn.
- Reasoning Process: Using given inputs, the inference engine makes a process to deduce new facts. For example, are used the properties defined for the ontologies and the rules defined in the descriptions.
- Create a Workflow: *if*, the rules and the deduced facts can be achieved with the given input, *then*, inference engine uses sing service descriptions to create a plan with a sorted sequence of services to be consumed.

Figure 24 illustrates above process. Instead, this work uses the capability of "enforcing" the engine to create a plan in order to achieve a specific goal, this plan is known as: Goal-Based Workflow.

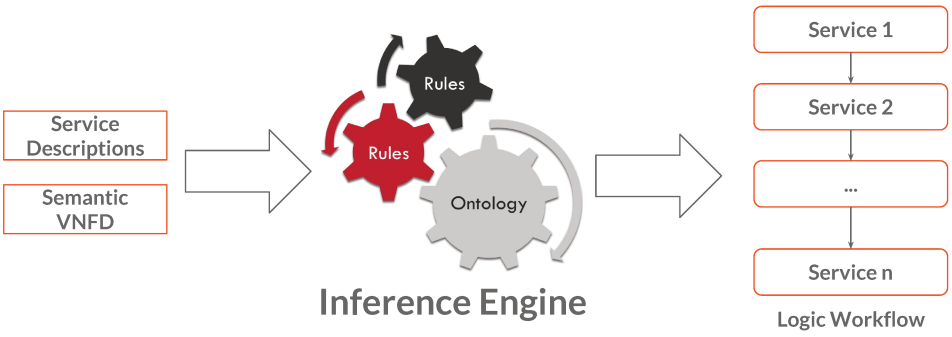


Figure 24 – Creating Context Based Workflow

For goal-based workflows, the context is created in order to achieve a specific task (in contrast, reasoning process above, creates a task based on the deduced context). With the aim of doing the goal-based reasoning process, the engine creates a plan to achieve a proposed goal[6].

[6] A goal is the desired state to be reached, when a service wokrflow is consumed.

In addition, another input file must be provided to the engine, this file includes the description of the goal [7]. The inference process done by the engine to achieve a goal is as follows:

- Assuming Facts: Inference engine starts assuming one fact as true. A fact, is a desired state in a service description (postcondition) or a predefined goal. For example, if the fact is a goal, then, engine assumes that the goal was achieved.
- Collecting Proofs: To support facts, engine must recollect proofs using the inputs and a reasoning process. Proofs are "proven", by analyzing the postcondition section of services descriptions or using other input data.
  - *if*, a service with the desired state (postcondition) is found, then, the inference engine goes to precondition section and attempts to prove the rules of the service. To prove the rules, engine uses postconditions from other service descriptions and other data inputs. or example a VNFD file.
  - *then*, *if* preconditions can be proven, the service is added in a workflow.
- Creating a Workflow: As some preconditions from service descriptions can be a postcondition from other service description, previous steps are repeated until the whole preconditions and postconditions are proven and there is not more proofs to be done. Consequently, proven services are added into the workflow, then, a plan is created.

Above process is known as backwards reasoning, this is due, inference engine goes backwards through the service descriptions attempting to create a workflow in order to achieve a goal. Summarizing the process, once a goal defined, engine finds a service to achieve this goal. Then, the service became the new goal and the engine attempts to achieve this new goal. The process continues until all the goals are achieved.

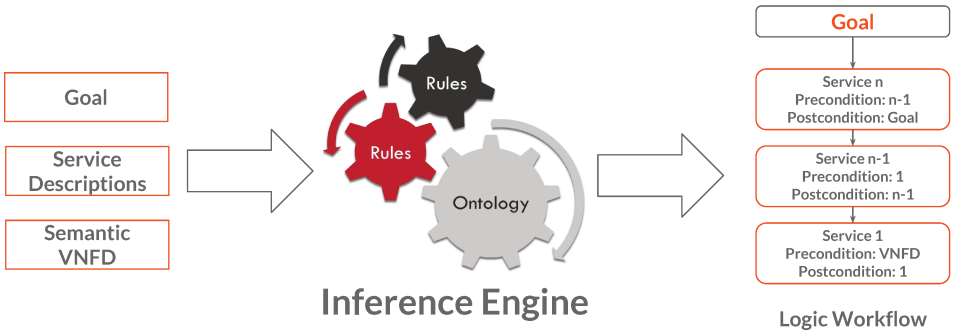


Figure 25 – Creating Goal Based Workflow

Figure 25 illustrates backwards reasoning process. In the figure, can be seen how the inference engine provides a goal-based workflow. Workflow, starts with the last service to

[7] With the aim of doing goal-based workflows, this project uses a backwards inference engine

be consumed, and goes down to the first service to be consumed. Additionally, in the figure is shown the postconditions of some services as preconditions from other services aiming to achieve the goal.

Listing 5.6 – OpenBaton Metadata Semantic Service Description

```
1  {
2  ?vnfd a non:vnfd;
3    non:vendor ?vendor;
4    non:descriptor_version ?ver_des;
5    non:id ?des_name;
6    non:lifecycle_event ?lifecycle.
7  ?vdu a non:vdu;
8    non:vm_image ?vm_image;
9    non:computation_requirement ?minCPU;
10   non:computation_requirement ?minRam.
11 ?vnf non:has_vnfd ?non_vnfd.
12 }
13 =>
14 {
15 _:request http:methodName "GET";
16   http:MessageHeader "Content-Type: application/yaml";
17   http:requestURI ("http://localhost:8080/nfv/parser/
       openbaton/vnf/metadata?name=" ?des_name "&link=" ?
       vm_image "&minCPU=" ?minCPU "&minRam=" ?minRam);
18   http:resp [ http:body yaml:vnf_metadata].
19 ?vnf  ob:has_metadata yaml:vnf_metadata.
20 }.
```

In Listing 5.6 is the semantic description done for OpenBaton Metadata service. In this description, some rules can be achieved with the semantic VNFD file, however, the precondition `has_vnfd` (added in line 11), is described as a postcondition in the VNFD semantic service (Listing 5.2, line 20). Consequently, before consuming Metadata Web Service, is necessary to consume VNFD Web Service. In contrast, in Listing 5.7 is described a goal defined as a postcondition in the Metadata service description. Thus, in order to achieve the goal, is necessary to consume both services

Listing 5.7 – OpenBaton Metadata Semantic Service Goal

```
1  {?vnf  ob:has_metadata yaml:vnf_metadata}=>
2  {?vnf  ob:has_metadata yaml:vnf_metadata}.
```

Using the inference engine, OpenBaton Semantic nFV Services and above goal, a goal-based workflow was created. Listing 5.8 shows a piece of the workflow retrieved by the inference engine[8]. In the listing, is presented the backwards reasoning process performed by the. The process is executed as follows:

- In the lines 1 to 5, is defined the goal, the desired state to be achieved, Listing 5.7.
- Inference engine, assumes the achievement of the goal as a fact (lemma1, line 6) and starts to recollect evidence to prove it (lemma2, line 7).

[8] To see complete file refer to annex D Listing D.1

- In the lines 10 to 14, is realized by the engine that, the defined goal is a postcondition in the Metadata service description WS (Listing 5.6), thus, the achievement of the goal is proved and the service is defined as the new goal. Then, inference engine starts to recollect evidence to prove the new fact (lemma4 to 12, line 15).

- In line 19 (lemma4), is proved that, `non:vnfd` rule can be achieved by taking the parameter from the VNFD semantic descriptor file (Listing 4.5). In addition, other parameters included in the precondition section (such `non:vdu`, `non:vm_image`) are proven in the same way. Thus, the process continues until reaching lemma12 (line 21).

- In Lemma12, engine realizes that, `non:has_vnfd` precondition is defined as a postcondition in the VNFD service description (Listing 5.2), thus, all the proofs for the Metadata service are recollected. In consequence, the service is added in the workflow and VNFD WS is defined as the new goal.

- As done with Metadata service, inference engine starts recollecting evidence to prove the new fact, lemma4 to 10 and 14 to 18 (line 29). In Lemma14 (line 31), is proved that, `non:scale_in_out` rule can be achieved by taking the parameter from the VNFD semantic descriptor. In addition, other parameters are proven in the same way.

- Finally, in lemma19, are recollected all the proofs for VNFD service. Then, the service is added in the workflow and the process is finished. This is due, at this point there is not more evidence to recollect, consequently .

Listing 5.8 – OpenBaton Metadata Goal-Based Workflow - Compact

```
1  [ a r:Proof, r:Conjunction;
2    r:component <#lemma1>;
3    r:gives {
4      _:sk3 ob:has_metadata yaml:vnf_metadata.
5    }].
6  <#lemma1> a r:Inference; r:gives {_:sk3 ob:has_metadata yaml:
       vnf_metadata}; r:evidence (
7    <#lemma2>);
8    r:rule <#lemma3>.
9  <#lemma2> a r:Inference; r:gives {_:sk4 http:methodName "GET
       ".
10   _:sk4 http:MessageHeader "Content-Type: application/json".
11   _:sk4 http:requestURI ("http://localhost:8080/nfv/parser/
       openbaton/vnf/metadata?name=" "iperf-client" "&link=" "
       iperf_client_image"^^xsd:anyURI "&minCPU=" _:sk5 "&minRam
       =" "1024"^^xsd:int).
12   _:sk4 http:resp _:sk6.
13   _:sk6 http:body yaml:vnf_metadata.
14   _:sk3 ob:has_metadata yaml:vnf_metadata}; r:evidence (
15   <#lemma4>  <#lemma5> ... <#lemma11>  <#lemma12>);
16   r:rule <#lemma13>.
17 <#lemma3> a r:Extraction; r:gives {{?x0 ob:has_metadata yaml:
       vnf_metadata} => {?x0 ob:has_metadata yaml:vnf_metadata}};
18   r:because [ a r:Parsing; r:source <file:///ServiceDescriptor
       /OpenBaton/goals/Metadata-iPerf-Client-goal.n3>].
```

```
19 <#lemma4> a r:Extraction; r:gives {non:ob-iperf-client a non:
     vnfd}; r:because [ a r:Parsing; r:source <file:///
     ServiceDescriptor/OpenBaton/resources/iperf_client.n3>].
20 ...
21 <#lemma12> a r:Inference; r:gives {_:sk0 http:methodName "GET
     ".
22   _:sk0 http:MessageHeader "Content-Type: application/json".
23   _:sk0 http:requestURI ("http://localhost:8080/nfv/parser/
       openbaton/vnf/vnfd?vendor=" "fokus" "&version=" "0.1" "&
       name=" "iperf-client" "&vm_image=" "iperf_client_image"
       ^^xsd:anyURI "&virtuallink=" "private" "&lifecycle=" "
       CONFIGURE" "&dev_flavour=" "m1.small" "&scaleinout=" "2"
       ^^xsd:int "").
24   _:sk0 http:resp _:sk1.
25   _:sk1 http:body json:openbaton_vnfd.
26   _:sk2 a json:file.
27   _:sk2 a ob:vnfd.
28   _:sk3 non:has_vnfd _:sk2}; r:evidence (
29   <#lemma4>...<#lemma10><#lemma14>...<#lemma18>);
30   r:rule <#lemma19>.
31 <#lemma14> a r:Extraction; r:gives {non:ob_iperf_client_vdu
     non:scale_in_out "2"^^xsd:int};
32   r:because [ a r:Parsing; r:source <file:///ServiceDescriptor
       /OpenBaton/resources/iperf_client.n3>].
33 <#lemma19> a r:Extraction; r:gives {{?x0 a non:vnfd.
34   ?x0 non:vendor ?x1.
35 ...
36   ?x10 non:id ?x11} => {_:x12 http:methodName "GET".
37   _:x12 http:MessageHeader "Content-Type: application/json".
38   _:x12 http:requestURI ("http://localhost:8080/nfv/parser/
       openbaton/vnf/vnfd?vendor=" ?x1 "&version=" ?x2 "&name="
        ?x3 "..." "&dev_flavour=" ?x11 "&scaleinout=" ?x7 "").
39 ...
40   _:x15 non:has_vnfd _:x14}};
41   r:because [ a r:Parsing; r:source <file:///ServiceDescriptor
       /OpenBaton/services/parser/OpenBaton-Parser.n3>].
```

From the listing, Lemmas 3 (line 17), 13 (refer to annex D Listing D.1) and 19 (line 33) are used as proven facts added in the workflow. However, this project uses inferred facts to consume the services, thus, the workflow used in the *Generic Client* is composted by Lemmas 2 and 12 (lines 9 and 21 respectively). This is due, inferred facts include parameter values, necessaries to execute queries in the HTTP requests. For example, `non:scale_in_out "2"xsd:int`, the value is **2**.

### 5.2.2 Consuming Dynamic Workflows

Currently, *Generic Client* can adapt its code to create HTTP requests dynamically in order consume a REST Web Service. However, this adaptability only works to consume one service at the time (Figure 22). In consequence, inference engine is not capable of using a goal-based workflow. Thus, main challenge in this research was to improve the client by adding the capability of reading and interpreting these kind of workflows.

The Workflows retrieved by the inference engine (Listing 5.8) include, in addition to the inferred facts, lemmas and proven facts. These last two, are considered unnecessary data in order to consume the workflow. Consequently, *Generic Client* has a process to remove this

data, without affecting the inferred facts and its sequence (Figure 25).

Figure 26 illustrates the interaction between the *Generic Client* and the inference engine, aiming to create and interpret a workflow. In first part of the process, the client pass as inputs, the service descriptions, the semantic VNFD file and the goal, to the engine. Then, inference engine uses the inputs to infer and retrieve a goal-based workflow (e.g. Listing 5.8). Finally, *Generic Client* interpret the generated workflow to execute the plan.

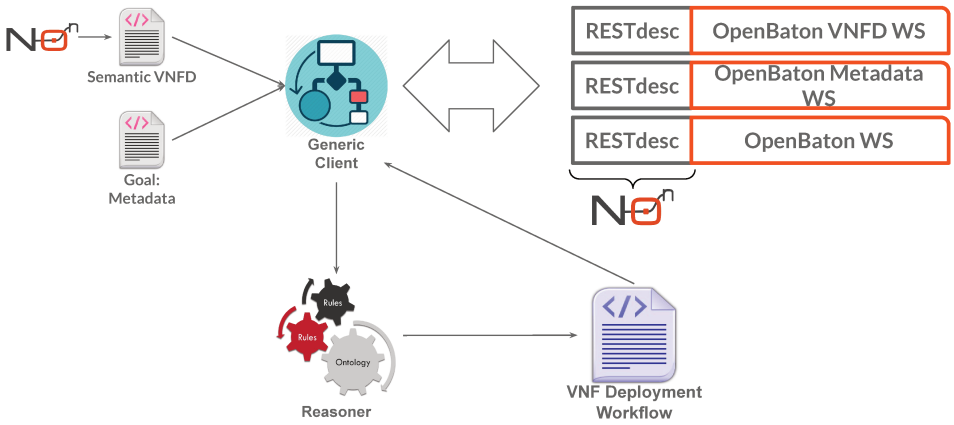


Figure 26 – Generic Client Consuming a Goal-Based Workflow

To interpret the workflow, *Generic Client* initiates a process to remove lemmas and proven facts included in the workflow. This removal process, excludes inferred facts. Then, is executed cleaning process over to the inferred facts, thus, unnecessary characters are removed from the HTTP requests. Finally, a new version of the workflow is generated. This version only includes, the HTTP requests and the ontology `@prefixes`. Listing 5.9 shows the new version of the Metadata based-goal workflow (Listing 5.8).

Listing 5.9 – OpenBaton Metadata Generic Client Workflow

```
1  PREFIX non: <https://github.com/LCuellarH/NOn/blob/master/
       datamodel/non.owl#>
2  ...
3  PREFIX http: <http://www.w3.org/2011/http#>
4  #request
5  _:sk4 http:methodName "GET".
6  _:sk4 http:MessageHeader "Content-Type: application/json".
7  _:sk4 http:requestURI ("http://localhost:8080/nfv/parser/
       openbaton/vnf/metadata?name=" "iperf-client" "&link=" "
       iperf_client_image"^^xsd:anyURI "&minCPU=" _:sk5 "&minRam="
        "1024"^^xsd:int).
8  _:sk4 http:resp _:sk6.
9  _:sk6 http:body yaml:vnf_metadata.
10 _:sk3 ob:has_metadata yaml:vnf_metadata}
11 #request
12 _:sk0 http:methodName "GET".
13 _:sk0 http:MessageHeader "Content-Type: application/json".
14 _:sk0 http:requestURI ("http://localhost:8080/nfv/parser/
       openbaton/vnf/vnfd?vendor=" "fokus" "&version=" "0.1" "&
       name=" "iperf-client" "&vm_image=" "iperf_client_image"^^
       xsd:anyURI "&virtuallink=" "private" "&lifecycle=" "
       CONFIGURE" "&dev_flavour=" "m1.small" "&scaleinout=" "2"^^
       xsd:int "").
```

```
15 _:sk0 http:resp _:sk1.
16 _:sk1 http:body json:openbaton_vnfd.
```

The new workflow is parsed in a JAVA List of REST requests objects. Then, the order of the list is inverted, transforming a backwards plan into a forward plan. Afterwards, to do the service consumption, *Generic Client* uses the process defined in Section 5.1.2 (Figure 23). Assuming that, all services parameters are correct, both services are consumed properly. Hence, a enhanced version of the *Generic Client* is created, a client capable of consuming REST WS workflows without having a predefined context.

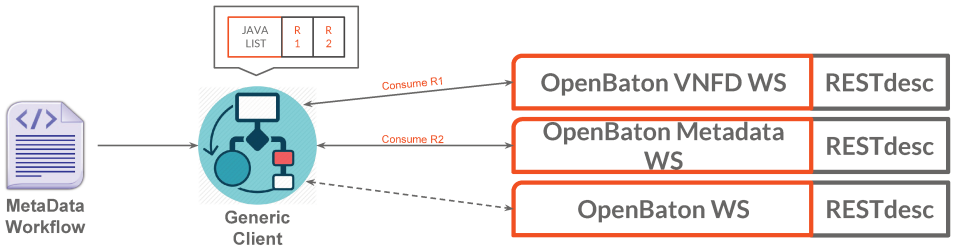


Figure 27 – Generic Client Consuming Workflow

Figure 27 illustrates the service consumption process. In the figure is shown how, to consume services is not necessary to use the semantic description. Instead, *Generic Client* consumes service capabilities using HTTP requests. In the JAVA List, are the services and the order (**R1**,**R2**) to be consumed. Then, *Generic Client* proceeds to consume each service.

To finalize the section, Figure 28 summarizes *Generic Client* execution cycle. In stage one and two, the client reads and uses semantic inputs to make an inference process and create a context. Then, using the context and the service descriptions, client decides are the possible services to be consume and add them in a plan (workflow). Finally, client executes the inferred workflow.

## 5.3 SnS Use Cases: Semantic Services on NFV projects

Previously in the chapter, it was created a *Generic Client* capable of adapting itself to consume workflows without using a predefined context or background. Instead, the client creates the context in execution time. Additionally, some Semantic nFV Services were created for OpenMano and OpenBaton Projects.

In this section the *Generic Client* and the SnS are used to create an automation process to deploy VNFs using a minimum of manual intervention. Thus, three use cases were created. First use case, includes a "benchmarking" process for the creation of VNFD files. In second use case, a VNF is deployed using a fully automated process. Final use case, is an attempt to use *Generic Client* to automate the integration of two NFV components (NFVO and VIM) from different implementations.

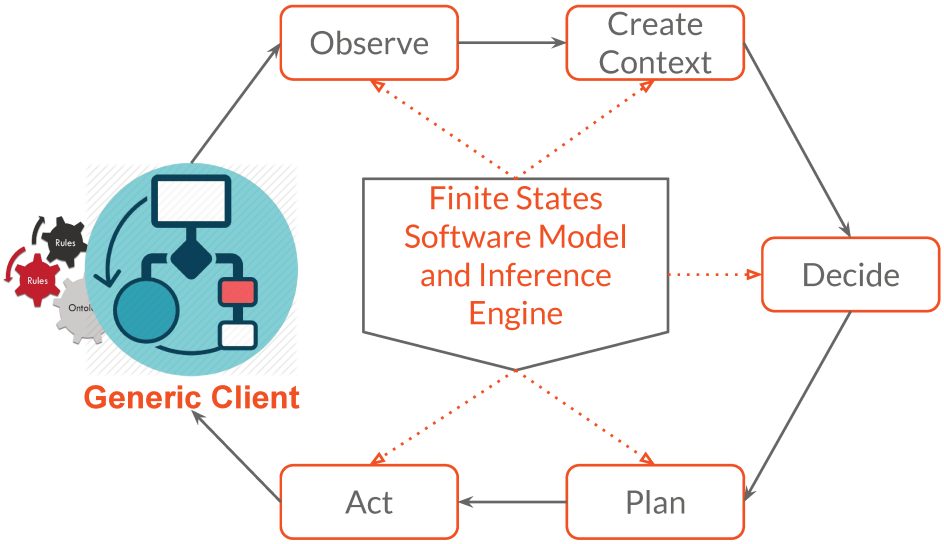


Figure 28 – Generic Client Process Cycle

### 5.3.1 Use Case I: Semantic VNFD Generator Service

The aim of implementing NOn, is to avoid: (i) the creation of multiple descriptors and different syntax, (ii) the use of parsers to translate syntax across domains. Unfortunately, to achieve the purposed goal, it is necessary the implementation of semantic components in current NFV implementations. However, a more feasible scenario, is the use of NOn to create semantic NFV descriptors and implement parsers those to translate descriptors into proprietary data models. Figure 29 illustrates the purposed scenario.

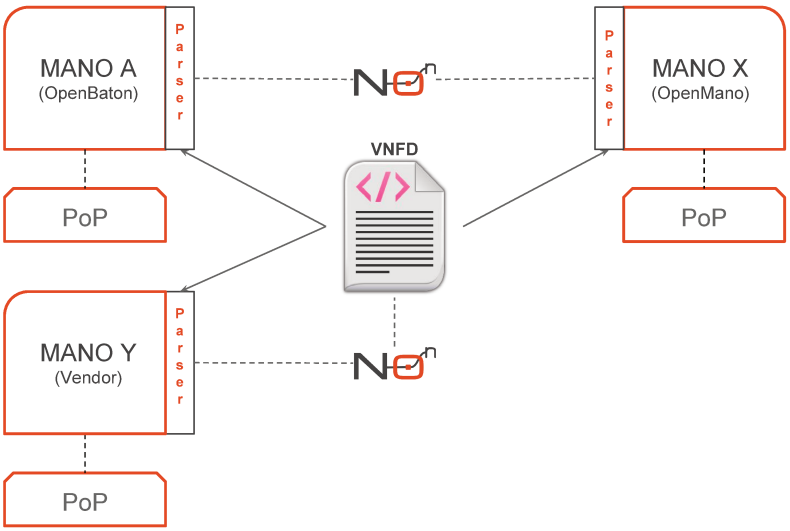


Figure 29 – SnS Not Implemented Use Case

In above figure, each NFV implementation has one semantic parser in order to translate NOn into proprietary data models. Thus, allowing to have an universal language to describe current descriptors (e.g. VNFD). However, this type of parsing methods is a main issue that, this work tries to avoid. Consequently, this scenario was deprecated.

Figure 30 shows the real scenario purposed for use case I. In this scenario, is changed the parsing approach for a Semantic nFV Services approach. Thus, instead having a parser for each implementation, is implemented a SnS capable of retrieving proprietary data models. Hence, the *Generic Client* and the inference engine are used to automatically consume each service.

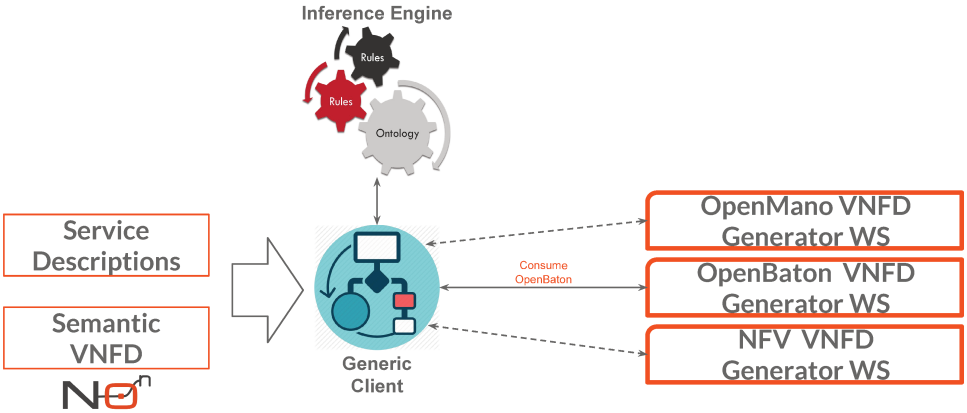


Figure 30 – SnS Use Case I: Semantic VNFD Generator Service

To implement the use case, were developed two semantic services to create OpenBaton and OpenMano VNFD file. Left side of the figure, shows the inputs used by the client, in right side represents the Web Service[9]. Thus, using the inputs, *Generic Client* and inference engine deduce which WS can be consumed, then, the service is consumed and the file retrieved. For example, if the parameter `non:vim_component` exists in semantic VNFD file, then, OpenBaton service is consumed. This is due, in the descriptor for OpenBaton service the parameter is used in precondition section and OpenMano does not.

For Use Case I, were created three different scenarios to observe the behavior of *Generic Client*. In each scenario, *Generic Client* acts as a "Benchmarking" process, finding a service to be consumed according with the inputs.

In all scenarios same inputs were given to the client:

- Semantic VNFD
- OpenBaton SnS
- OpenMano SnS

#### 5.3.1.1 Scenario I: Using OpenBaton Semantic Descriptor

Following table represents the parameters used in scenario I:

[9] Service description in left side of the figure, corresponds to the Web Service on the right side.

Table 12 – SnS Use Case I: Scenario I

| **Parameter** | **Description** |
|---|---|
| Goal | Consume OpenBaton SnS. |
| Preconditions and Assumptions | Semantic VNFD file.<br>Semantic file created using NOn and based on the OpenBaton data model.<br>Goal not predefined. |
| Test Data | Semantic VNFD.<br>OpenBaton SnS.<br>OpenMano SnS. |
| Description | *Generic Client* receives test data inputs and through the inference engine starts a benchmarking process. This process tries to find which service is adequate to be consumed according with the inputs. If, the inputs fully-fill the preconditions of any SnS the service is consumed and the VNFD file is created. |
| Testing tools | *Generic Client* instance. |
| Post-Conditions | SnS Consumed |
| Expected Results | OpenBaton SnS Consumed. |
| Expected vs Obtained Results | None of the services was consumed:<br>Due the absence of VIM component on the semantic VNFD, OpenBaton SnS was not consumed. |

Listing 5.10 – Vim Component

```
### https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#vim_openstack_25
non:vim_openstack_25 rdf:type owl:NamedIndividual,
        non:vim_component;
        non:id "10.1.1.25-vim-instance" ^^xsd:string.
```

With the aim of making the service consumed, VIM component was manually added in the semantic VNFD (Listing 5.10). Thus, the service was consumed and the OpenBaton VNFD file was successfully created. Listing 5.11 shows the resulting request given by the inference engine and used by the *Generic Client*.

Listing 5.11 – OpenBaton SnS Generic Client Request

```
_:sk0 http:methodName "GET".
_:sk0 http:MessageHeader "Content-Type: application/text".
_:sk0 http:requestURI ("http://localhost:8080/nfv/parser/
    openbaton/vnf/vnfd?vendor=" "fokus" "&version=" "0.1" "&
    name=" "iperf-client" "&vm_image=" "iperf_client_image"^^
    xsd:anyURI "&virtuallink=" "private" "&lifecycle=" "
    CONFIGURE" "&dev_flavour=" "m1.small" "&scaleinout=" "2"^^
    xsd:int "&vim=" "10.1.1.25-vim-instance""").
_:sk0 http:resp _:sk1.
_:sk1 http:body json:openbaton_vnfd.
_:sk3 ob:has_vnfd _:sk2.
```

#### 5.3.1.2 Scenario II: Using OpenMano Semantic Descriptor

Following table represents the parameters used in scenario II:

Table 13 – SnS Use Case I: Scenario II

| **Parameter** | **Description** |
|---|---|
| Goal | Consume OpenMano SnS. |
| Preconditions and Assumptions | Semantic VNFD file.<br>Semantic file created using NOn and based on the OpenMano data model.<br>Goal not predefined. |
| Test Data | Semantic VNFD.<br>OpenBaton SnS.<br>OpenMano SnS. |
| Description | *Generic Client* receives test data inputs and through the inference engine starts a benchmarking process. This process tries to find which service is adequate to be consumed according with the inputs. If, the inputs fully-fill the preconditions of any SnS the service is consumed and the VNFD file is created. |
| Testing tools | *Generic Client* instance. |
| Post-Conditions | SnS Consumed |
| Expected Results | OpenMano SnS Consumed. |
| Expected vs Obtained Results | None of the services were consumed:<br>Due the absence of many components of OpenMano semantic VNFD , OpenMano SnS was not consumed. |

With the aim of making the service consumed, OpenMano service description was manually modified: to include parameters that can be provided by other components of NOn and to delete parameters not contained on the ontology. Listing 5.12 shows the final version of the service. Thus, service was consumed and OpenMano VNFD was created.

Listing 5.12 – OpenMano VNFD Semantic Service Description

```
{
    ?vnfd a non:vnfd.
#       non:id ?vnf_name;
#       non:descriptor_version ?vnf_description.
#   ?vdu a non:vdu;
#       non:vm_image ?vm_image.
    ?vnfc a non:vnfc;
        non:id ?vnfc_name;
        non:has_connection_point ?vnfc_conn.
    ?vnfc_conn a non:connection_point;
        non:type ?ext_conn_type;
        non:id ?ext_conn_name.
}
=>
{
        _:request http:methodName "GET";
        http:MessageHeader "Content-Type: application/json";
        http:requestURI ("nfv/parser/openmano/vnf/vnfd?
           vnf_description="?vnf_description"&vnf_name="?
           vnf_name"&vnfc_name="?vnfc_name"&vnfc_description=
           ""&vm_image="?vm_image"&ext_conn_name="?
           ext_conn_name"&ext_conn_iface_name="?ext_conn_name
           "&ext_conn_description="?ext_conn_type"&
           ext_conn_type"?ext_conn_type"");
        http:resp [ http:body yaml:openbaton_vnfd].
```

```
20          ?non_vnfd a yaml:file;
21                    a om:vnfd.
22          ?vnf non:has_vnfd ?non_vnfd.
23 }.
```

Above listing, shows the service description of OpenMano VNFD WS. Lines with numeral sign (#), represents the objects deleted from the descriptor in order to consume the service, lines 3 to 6.

Listing 5.13 – OpenMano SnS Generic Client Request

```
1 _:sk0 http:methodName "GET".
2 _:sk0 http:MessageHeader "Content-Type: application/json".
3 _:sk0 http:requestURI ("nfv/parser/openmano/vnf/vnfd?
    vnf_description=" _:sk1 "&vnf_name=" _:sk2 "&vnfc_name=" "
    TEMPLATE-VM" "&vnfc_description=" "&vm_image=" _:sk3 "&
    ext_conn_name=" "mgmt0" "&ext_conn_iface_name=" "mgmt0" "&
    ext_conn_description=" "mgmt0" "&ext_conn_type" "mgmt0" ""
    ).
4 _:sk0 http:resp _:sk4.
5 _:sk4 http:body json:openbaton_vnfd.
6 _:sk6 ob:has_vnfd _:sk5.
```

#### 5.3.1.3 Scenario III: Using Generic Semantic Descriptor

Following table represents the parameters used in Scenario III:

Table 14 – SnS Use Case I: Scenario III

| **Parameter** | **Description** |
|---|---|
| Goal | Consume any SnS. |
| Preconditions and Assumptions | Semantic VNFD generic file (Annex C, Listing C.3).<br>Semantic file created using NOn.<br>Predefined Goal (create VNFD). |
| Test Data | Semantic VNFD.<br>OpenBaton SnS.<br>OpenMano SnS. |
| Description | *Generic Client* receives test data inputs and through the inference engine starts a benchmarking process. This process tries to find which service is adequate to be consumed according with the inputs. If, the inputs fully-fill the preconditions of any SnS the service is consumed and the VNFD file is created. |
| Testing tools | *Generic Client* instance. |
| Post-Conditions | SnS Not Consumed |
| Expected Results | SnS Not Consumed. |
| Expected vs Obtained Results | OpenMano service was consumed:<br>This occurs due, the minimum components needed to consume OpenMano service (Listing 5.12), in Listing 5.14 is shown the resulting request. In addition, OpenBaton service was not consumed due the absence of VIM component. |

In below listing, due is used the complete version of OpenMano SnS (without deleting rules[10]), more values are filled in the request parameter (line 3) in comparison with scenario II (Listing 5.13, line 3).

Listing 5.14 – Generic VNFD Resulting Request

```
1 _:sk0 http:methodName "GET".
2 _:sk0 http:MessageHeader "Content-Type: application/json".
3 _:sk0 http:requestURI ("nfv/parser/openmano/vnf/vnfd?
    vnf_description=" "0.2" "&vnf_name=" "iperf-server" "&
    vnfc_name=" "ob_vnfc1" "&vnfc_description=" "&vm_image=" "
    ubuntu-14.04-server-cloudimg-amd64-disk1"^^xsd:anyURI "&
    ext_conn_name=" "ob1" "&ext_conn_iface_name=" "ob1" "&
    ext_conn_description=" "bridge" "&ext_conn_type" "bridge"
    "").
4 _:sk0 http:resp _:sk1.
5 _:sk1 http:body json:openbaton_vnfd.
6 _:sk2 a json:file.
7 _:sk2 a ob:vnfd.
8 _:sk3 ob:has_vnfd _:sk2
```

In order to analyze how, *Generic Client* and inference engine react when two SnS can achieve same goal, it was decided to add VIM component in semantic VNFD generic file (Listing 5.10).

By doing the modification, OpenMano SnS was consumed (Listing 5.14) over OpenBaton. This is due, the server version of the inference engine allows to use *Quick Answer* capability. This capability is used to find one answer (the fastest one) to achieve the goal. However, desktop versions of the inference engine allows to infer more than one answer to accomplish the goal, thus *Simple Answer* capability of the desktop engine was used.

Listing 5.15 – Generic VNFD Resulting Request - Simple Answer Capability

```
1 #request
2 _:sk0 http:methodName "GET".
3 _:sk0 http:MessageHeader "Content-Type: application/json".
4 _:sk0 http:requestURI ("http://localhost:8080/nfv/parser/
    openmano/vnf/vnfd?vnf_description=" _:sk1 "&vnf_name=" _:
    sk2 "&vnfc_name=" "generic_vnfc1" "&vnfc_description=" "&
    vm_image=" _:sk3 "&ext_conn_name=" "ob1" "&
    ext_conn_iface_name=" "ob1" "&ext_conn_description=" "
    bridge" "&ext_conn_type" "bridge" "").
5 _:sk0 http:resp _:sk4.
6 _:sk4 http:body yaml:openmano_vnfd.
7 non:generic-vnfd-1 a yaml:file.
8 _:sk5 non:has_vnfd non:generic-vnfd-1
9 #request
10 _:sk6 http:methodName "GET".
11 _:sk6 http:MessageHeader "Content-Type: application/text".
12 _:sk6 http:requestURI ("http://localhost:8080/nfv/parser/
    openbaton/vnf/vnfd?vendor=" "fokus" "&version=" "0.2" "&
    name=" "iperf-server" "&vm_image=" "ubuntu-14.04-server-
    cloudimg-amd64-disk1"^^xsd:anyURI "&virtuallink=" "private
    " "&lifecycle=" "INSTANTIATE-install.sh-install-srv.sh" "&
    dev_flavour=" "m1.small" "&scaleinout=" "2"^^xsd:int "&vim
    =" "10.1.1.25-vim-instance" "").
13 _:sk6 http:resp _:sk7.
```

[10] Numeral sign (#) are removed from service description on Listing 5.12.

```
14 _:sk7 http:body json:openbaton_vnfd.
15 non:generic-vnfd-1 a json:file.
16 _:sk8 non:has_vnfd non:generic-vnfd-1
```

Listing 5.15 shows the resulting workflow using *Simple Answer* capability [11]. Thus, the *Generic Client* was able to consume an create both (OpenMano and OpenBaton) descriptors using just one semantic description.

This use case is finalized by concluding that, the implementation of NOn and SnS are good solution to create multidomain NFV interoperability. The use case opens a possibility of using an unique representation of NFV to create multiple types of descriptors VNFD, and the implementation of semantic services to reduce manual intervention in the Web Service consumption process.

### 5.3.2 Use Case II: Workflow Inference - Deploying a VNF Semantic Services

#### 5.3.2.1 Goal

- Create an automated process for deploying VNFs on the OpenBaton project.

#### 5.3.2.2 Preconditions and Assumptions:

- OpenStack installed.
- OpenBaton installed.
- OpenBaton SnS installed.
- A method to create Tape ARchiver (TAR) files.

#### 5.3.2.3 Test Data

- OpenBaton VNFD.
- OpenBaton VNF Deployment Service Description: This description corresponds to a service created by the OpenBaton Project[12], to deploy a VNF over a NFVI using the NFVO. Listing 5.17.
- OpenBaton SnS installed.
- A Goal for VNF deployment. Listing 5.16.

[11] To see complete answer of the inference engine refer to Annex D Listing D.2

[12] This works refers as a OpenBaton Project objects to all elements and services developed by OpenBaton, such OpenBaton Project deployment WS or NFVO

Listing 5.16 – OpenBaton Deploy VNF Semantic Service Goal

```
1 {?vnf ob:state ob:vnf_deployed.}=>
2 {?vnf ob:state ob:vnf_deployed}.
```

Listing 5.17 – OpenBaton Project Deploy VNF Service Description

```
1  {
2  ?vnf ob:has_metadata yaml:vnf_metadata.
3  }=>{
4  _:request http:methodName "POST";
5    http:MessageHeader "Content-Type: application/json";
6    http:requestURI "http://localhost:8080/nfv/parser/upload";
7    http:body       [ http:formData ("file" ?vnf_package)];
8    http:resp [ http:body http:response].
9  ?vnf ob:state ob:vnf_deployed.
10 }.
```

#### 5.3.2.4 Testing Tools

- Public Inference Engine.
- Generic Client.

#### 5.3.2.5 Test Description

Figure 31 illustrates the current process to deploy a VNF using OpenBaton Project. In step one and two are manually created the OpenBaton VNFD (Listing 4.4) and the Metadata files (Listing 5.4). Step three is a process (can be manually or using a programmed task) to create an OpenBaton VNF package. This package is a TAR file, including VNFD and Metadata files. Final step, is the process to upload the package into OpenBaton platform. If, all the steps are done correctly all component must be shown in the OpenBaton and OpenStack dashboard.

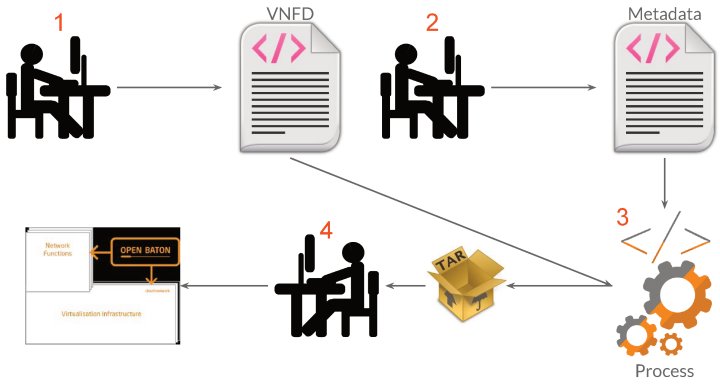


Figure 31 – OpenBaton VNF Deployment Process

**Scenario:**

Figure 32 illustrates the scenario implemented in this use case. NFV components were installed and configured in different servers using the Information & Networking Tech-

nologies Research & Innovation Group (INTRIG)[13] facilities. For the NFVI component, were used the resources on the INTRIG Cloud. Additionally, the cloud is orchestrated by OpenStack, thus, OpenStack is used as the VIM component in the scenario. On the other hand, the VNFM and NFVO components were installed on an internal server on INTRIG's lab. In consequence, main components of the NFV Framework were installed and implemented.

The next step, was to connect and register VIM component in the NFVO component. Thereby, the access to all resources provided by the NFVI[14] was guaranteed.

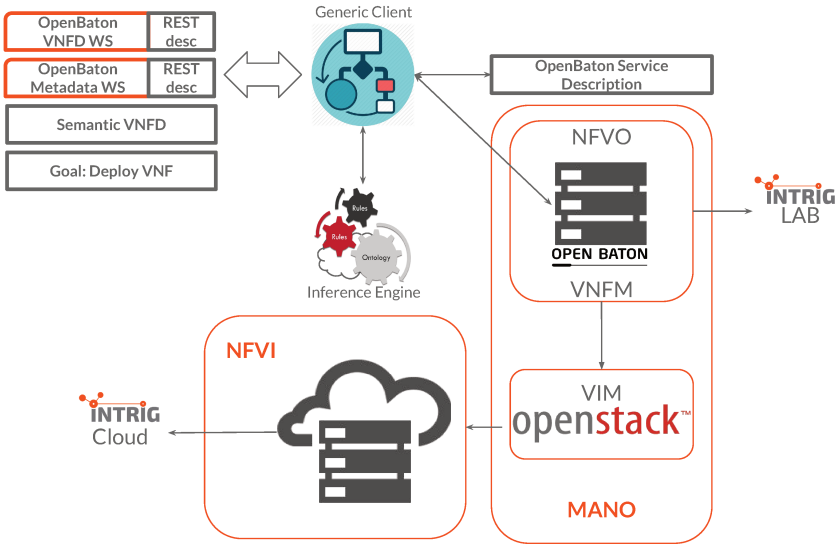


Figure 32 – SnS Use Case II: Test Scenario

*Generic Client*, was used in a personal sessions of different VMs over the servers in INTRIG's lab. Additionally, OpenBaton VNFD and MetaData Web Service were installed on the same machine with *Generic Client*. All components were connected among them using a local network. Finally, *Generic Client* was connected via WS to an online and public EYE inference engine (VERBORGH; ROO., 2012) and the RESTdesc descriptions were located on the GitHub repository of the project (PROJECT, 2016).

**SnS Deployment Process:**

Afterwards, the installation of previous scenario was done, some tests were done to prove correct behavior. A VNF was deploy using a manual process (Figure 31) and following OpenBaton use case[15] (OpenBaton Use Case, 2014). The test was successful and the VNF from the tutorial was able to be deployed. IPerf is a tool to do measurements of bandwidth to calculate network performance (GATES; WARSHAVSKY, 2012).

Instead, an automated deployment process for same scenario was done as follows:

13 The research group that the author is part of.

14 Some issues were found during the installation and configuration of both OpenStack and OpenBaton projects (specially in the communication part among them), however this issues were solved through the use of the mailing list.

15 The tutorial was done until the VNF IPerf Client and Server deployment

1. A user creates the Semantic VNFD and the Goal (5.16).

2. *Generic Client* takes service descriptions, semantic file and the goal and pass them to the inference engine.

3. Inference engine creates the workflow and retrieved it to the *Generic Client*.

4. Client interpret the workflow and creates the deployment version of the file. Listing 5.18 shows the new version of workflow:

Listing 5.18 – *Generic Client* Deploy VNF Workflow

```
#request
_:sk_6 http:methodName "POST".
_:sk_6 http:MessageHeader "Content-Type: application/json
    ".
_:sk_6 http:requestURI "http://localhost:8080/nfv/parser/
    upload".
_:sk_6 http:body _:sk_7.
_:sk_7 http:formData ("file" _:sk_8).
_:sk_6 http:resp _:sk_9.
_:sk_9 http:body http:response.
#request
_:sk_4 http:methodName "GET".
_:sk_4 http:MessageHeader "Content-Type: application/json
    ".
_:sk_4 http:requestURI ("http://localhost:8080/nfv/parser
    /openbaton/vnf/metadata?name=" "iperf-server" "&link="
     "ubuntu-14.04-server-cloudimg-amd64-disk1"^^xsd:
    anyURI "&minCPU=" "2" "&minRam=" "2").
_:sk_4 http:resp _:sk_5.
_:sk_5 http:body yaml:vnf_metadata.
#request
_:sk_0 http:methodName "GET".
_:sk_0 http:MessageHeader "Content-Type: application/json
    ".
_:sk_0 http:requestURI ("http://localhost:8080/nfv/parser
    /openbaton/vnf/vnfd?vendor=" "fokus" "&version=" "0.2"
     "&name=" "iperf-server" "&vm_image=" "ubuntu-14.04-
    server-cloudimg-amd64-disk1"^^xsd:anyURI "&virtuallink
    =" "private" "&lifecycle=" "INSTANTIATE-install.sh-
    install-srv.sh" "&dev_flavour=" "m1.small" "&
    scaleinout=" "2"^^xsd:int "").
_:sk_0 http:resp _:sk_1.
_:sk_1 http:body json:openbaton_vnfd.
```

5. Client consumes OpenBaton VNFD WS and creates locally the retrieving JSON file.

6. Client consumes OpenBaton Metadata WS and creates locally the retrieving YAML file.

7. Client consumes OpenBaton Project Deploy VNF WS:

   - Creates VNF package using VNFD and Metadata files.
   - Upload VNF package via OpenBaton Project WS.

8. OpenBaton Project NFVO deploys the VNF over INTRIG's cloud using OpenStack. Finally OpenBaton Project returns the id given to the deployed package.

Figure 33 illustrates previous process on a sequence diagram.

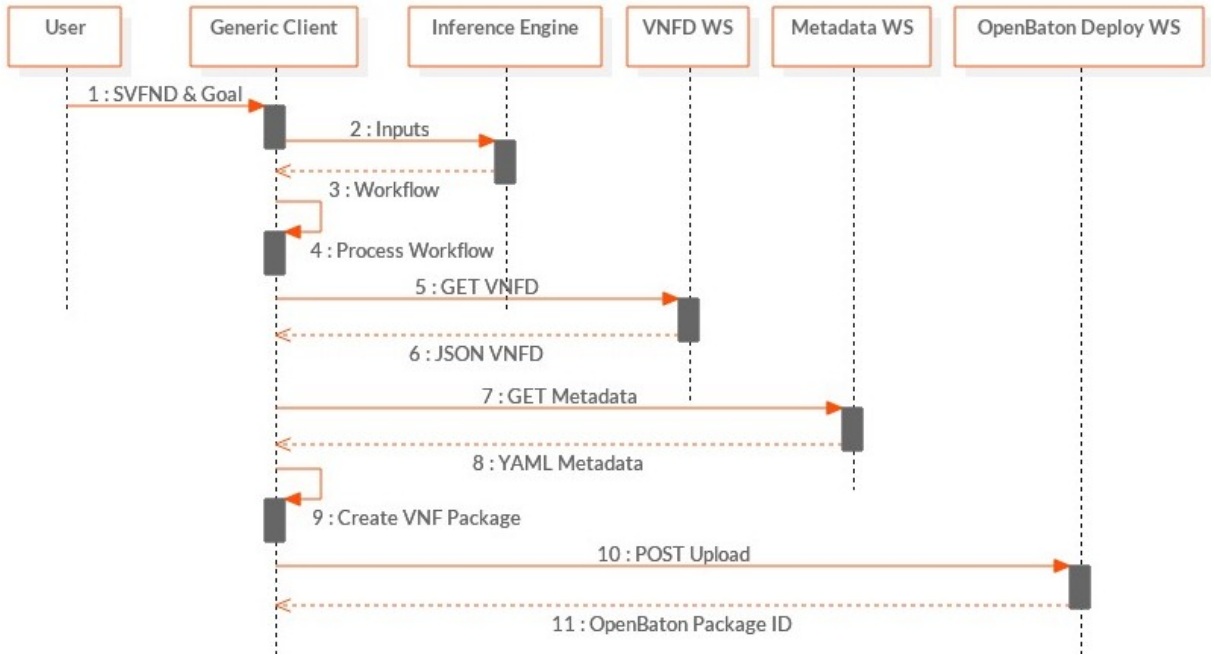


Figure 33 – SnS Use Case II: Sequence Diagram

**Test:**

In order to test the automated process, OpenBaton use case (OpenBaton Use Case, 2014) was used. As a difference with the OpenBaton use case, semantic VNFD files for IPerf client and server and a Goal were defined.

#### 5.3.2.6 Postcondition

- iPerf Server VNF deployed
- iPerf Client VNF deployed

#### 5.3.2.7 Expected Results

Both, server and client VNFs deployed without using manual intervention other than the creation of semantic descriptors and the goal.

#### 5.3.2.8 Expected vs Obtained Results

Both VNFs were successfully deployed (separately).

#### 5.3.2.9 Conclusions

This work foresees the implementation of SnS, NOn and the *Generic Client* as a good option to reduce manual intervention service integration process. Instead, creating differ-

ent to consume REST WS capabilities, one client is used to consume and interpret service descriptions in an autonomous manner. It was proven a reduction of manual intervention through the implementation of the Semantic nFV Services.

Aiming to increase the automate process, all capabilities NFV implementations must represented as resources or services using REST style. Thus, interoperability across domain can be improved and more complex workflows can be generated. For example, in order to use OpenStack, OpenBaton creates a plugin to consume one by one the REST Web Service, this is achieved creating a predefined context in the OpenBaton plugin. Instead, if each service of OpenStack would have a semantic description, the predefined context can be replaced with a generic client and the inference engine.

### 5.3.3 Use Case III: OpenBaton - Unify Integration Proposal

#### 5.3.3.1 Goal

Create a proposal and possible integration between OpenBaton Project and UNIFY *Virtualizer* element.

#### 5.3.3.2 Preconditions and Assumptions

- OpenBaton installed
- *Virtualizer* component installed
- A *Virtualizer* Semantic nFV Services: To create the UNIFY data model.

#### 5.3.3.3 Test Data

- *Virtualizer* data model
- Semantic VNFD

#### 5.3.3.4 Testing Tools

- A public Inference Engine
- A Generic Client
- An UNIFY *Virtualizer*

#### 5.3.3.5 Test Description

**Scenario:**

UNIFY *Virtualizer* acts as a VIM component, in which case for the NFV architecture *Virtualizer* will be at the bottom of OpenBaton NFVO and VNFM. Figure 34 illustrates the scenario.

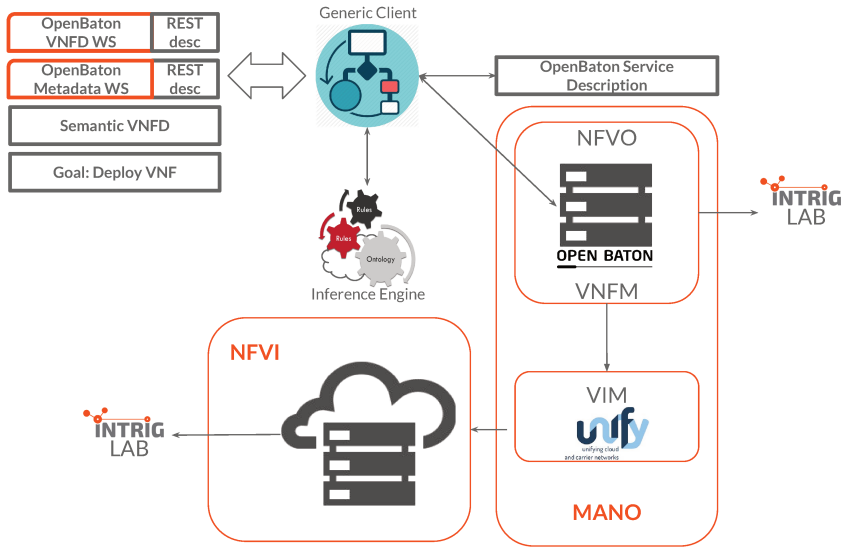


Figure 34 – SnS Use Case II: Sequence Diagram

Above figure, illustrates how OpenStack (VIM) component is replaced with UNIFY component. However, as OpenBaton is a project based on a JAVA RMI architecture, it has a VIM Driver plugin module to handle new modules created to access VIM components (NFVO Architecture, 2016).

Currently, in OpenBaton exists a module for the integration with OpenStack. In addition, this module extends all methods of *VIM Drivers* plugin module to executes all requests to OpenStack. Consequently, to integrate OpenBaton with UNIFY's *Virtualizer*, a new plugin must be created. Furthermore, the inhered methods must be implemented to consume *Virtualizer* API (VIM Plugin, 2016).

With the aim of using *Generic Client* in the integration process, the UNIFY's plugin should be included some features from client, thus, achieving, a decoupling of the plugin and the *Virtualizer*.

**Limitations:**

- As *Virtualizer* data model does not follow the ETSI specifications, there exist a gap in order to map VNFD elements into UNIFY data model.
- In order to create the plugin, inherited methods must be implemented from *VIM Drivers* component (VIM Plugin, 2016). Furthermore, some of these methods include: `List <DeploymentFlavour>listFlavors`, that returns the list of the Deployment Flavour or `NFVImage addImage`, that adds a new NFVImage. However, these methods or similar ones are not available in the *Virtualizer* API. Instead, *Virtualizer* deployment process relies in the interpretation of the data model. Thus, making integration process a difficult task.

- As *VIM Drivers* methods are based on the WS from OpenStack API, this might limit integration with other tools.
- As OpenBaton has predefined methods for new plugins, there exist a boundary in order to consume the *Virtualizer* SnS. This is due, one of the inputs for the services is the semantic VNFD and the use of the *Generic Client*.
- The registration process of a VIM component on OpenBaton Project is done by two means (OpenStack and Test). Making difficult the process of registering other components.

**Test:**

Mainly, it was attempted to deploy a NF using OpenBaton NFVO and *Virtualizer* (VIM component), through a UNIFY SnS.

#### 5.3.3.6 Postcondition

- VNF deployed.

#### 5.3.3.7 Expected Results

- Integration achieved.
- VM added.
- VNF deployed.

#### 5.3.3.8 Expected vs Obtained Results

Due to incompatibilities with the methods and the variables used by the *Virtualizer* and the variables provided by OpenBaton. It was not possible to do the integration. Furthermore, SnS was not able to be consumed, due the variables needed to consume the service are not provided by the plugin methods (such `NFVImage`)On the other hand, the VIM registration was not able to be achieved. This is due, in OpenBaton there not exist an option to register a VIM component different from OpenStack or Test type.

## 5.4 Final Remarks

While NOn is as a promising option to create service descriptions in the context of NFV, implementations that follow in a minor degree the ETSI specifications appear less susceptible to be described with the ontology compared to implementations that closely follow the specifications. From Use Case I, we note that the creation of a mechanism capable of self creating VNFD files can be a first step to use a unique descriptor in order to deploy VNFs in multiple

domains. From Use Case II, it can be seen how VNF deployment workflows can be generated if a proper service exists for each stage of the deployment. In that case, the deployment process can be completely defined by REST WS and the integration and consumption processes can be done in a fully automatic manner (i.e. without manual intervention). Finally, reviewing Use Case III and taking in count other VIM components APIs (such OpenStack or OpenMano VIM), we discovered further gaps on interface definitions.

As OpenMano and OpenStack interfaces are quite similar to implement, in which case for an inference engine equivalent components can be consumed using similar parameters contained in the descriptions (e.g., same incoming and out-coming), which results in more flexibility to create the workflow. In contrast, the UNIFY *Virtualizer* model has a different way to deploy VNF with inputs, especially made for the project prototype implementation and not necessary fully following the ETSI specifications. For this reason, a multi-vendor scenario using automatic service integration becomes more troublesome in the case of UNIFY compared to other open source components following a common pattern (ETSI specifications) resulting in higher integration costs.

This section helped to improve the NOn model in order to create service descriptions. It was realized that two more data properties (*Slots*) can be added to some Object components on the ontology, `non:name` and `non:description` *Slots*. This is due to most of the NFV implementations having in their description files both properties which can be used to differentiate between one implementation to another. In addition, the Object property `non:has_vnfd` was added to the `non:vnf` high level element in order to create the condition on the descriptors.

# 6 Conclusions and Future Work

This work aims at advancing the state of affairs of interoperability in the context of Network Function Virtualization technology through the application of semantic principles and technologies. We explored the integration of NFV with Web Semantic approaches in order to create a common representation of virtualisation technologies that can be shared across administrative domains and used to create descriptions for semantic service implementations of NFV embodiments. This work combines theoretical and practical considerations for the implementation and merging of the two key technologies –the Semantic Web and NFV– in order to deliver automatic Web Service integration.

In this work, some of the interoperability gaps were identified and can be explained by different semantic implementations on NFV and how the lack of unifying semantics affects the communications and VNF deployment across domains. This gap was attempted to be removed by the implementation of a common NFV data model known as NOn. However, the actual prototype implementation was limited by how close open source NFV projects are currently following ETSI specifications. Projects based on the the information models defined by the ETSI exhibit better chance inter-working in multi-domain scenarios without requiring manual intervention.

The implementation of NFV Ontology opened the door to create Semantic nFV Services. Through the implementation of REST interfaces, explicit service descriptions and the use of inference engines we implemented proof of concepts of automatic integrated services without human intervention. This was showcased through the implementations of the proposed use cases. A *Generic Client* was capable of self adapting to consume dynamic REST Web Service workflows without the need of humans in the loop. Furthermore, it was noticed that if the deployment process is defined step by step using WS, the need for manual intervention is reduced. Nevertheless, as current NFV implementations do not use semantic technologies in their developments, implementation of semantic technologies were scoped just to component interfaces, otherwise it would be necessary to re-write code to build semantic components, for example semantic MANO or VNFM.

Altogether, the implementation of NOn and SnS are an initial step towards automatic service integration. However, there are still multiple roadblocks and gaps ahead that we regard as future work. The current state of NOn does not fulfill all the needs of a complete data model capable of describing all possible VNFD files. Furthermore, there are other types of descriptors and components in the NFV evolving architecture that can and should be abstracted and added to the ontology in order to reach higher levels of interoperability and automation.

This work defends the use of semantic technologies to describe WS implementa-

tions of NFV interfaces. However, the full potential was not explored in the sense of leveraqging of semantic technologies to build full components. These implementations can be done using NOn to define variables and components to avoid ambiguity across implementations by using a common terminology to refer to the same variables and components. This would allow higher levels of interoperability and flexibility compare to SnS.

The *Generic Client* based on an inference engine appears as a promising approach to take care of NFV orchestration processes. To achieve this goal, it is necessary to fully decouple NFV framework components (Figure 5) and comprehensive REST services definitions to implement SnSs. By decoupling components and defining SnS, is would be possible for the *Generic Client* to realize dynamic orchestration of VNFs without manual intervention. Thus, it would no be necessary to define in the the client which components do what (e.g. OpenStack to do NFVI orchestration). In contrast, the inference engine takes care of creating the complete workflow and choosing the best option according to the inputs and the inferred context, as illustrate in Figure 35.

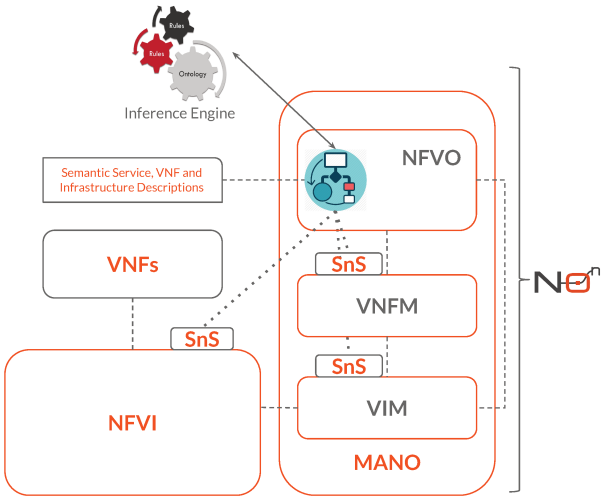


Figure 35 – *Generic Client* - NFVO Proposal

The figure shows the proposed method to include the *Generic Client* into the NFVO component. In addition, it shows the proposed implementation of SnSs in components and interfaces, and the use of semantic descriptors to deploy Network Function. The aim of this approach is to use the *Generic Client* and the inference engine for the reasoning process in order to create dynamic workflow consumption in the orchestration task of NFV. By decoupling the elements of NFV architecture and implementing SnS, it is possible to create non restriction in terms of interoperability, this can be achieved by allowing the orchestrator to integrate itself with other components without having a predefined context or previous knowledge of other domains. However, to make this proposal feasible, more semantic models are needed in order to describe other capabilities besides the orchestration process, such as Policies or Life-cycle management.

Finally, this work also explored the implementation of semantic technologies on the

developing process of the *Generic Client* using the Jena Framework API (JENA Framework, 2014) and a first prototype was developed. However, runtime consumption of the framework randomized the variable matching and changed the order of the parameters from the workflow given by the inference engine. For example, `non:vendor` parameter was mismatched with other parameter value such `non: description_version` value. For this reason, improvements to the *Generic Client* using other semantic framework rather than Jena or waiting for a newer release of the framework is required.

# Bibliography

ALESSO, H.; SMITH, C. F. *Developing Semantic Web Services*. [S.l.]: A K Peters/CRC Press, 2004. 178 p. Citado 2 vezes nas páginas 24 and 26.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. *et al.* The semantic web. *Scientific american*, New York, NY, USA:, v. 284, n. 5, p. 28–37, 2001. Citado 2 vezes nas páginas 19 and 22.

COSTELLO, R. L. Building web services the rest way. *URL: http://www. xfront. com/REST-Web-Services. html. Ultima Consulta*, v. 11, p. 2007, 2007. Citado na página 21.

ETSI. *ETSI - European Telecommunications Standards Institute*. 1988. Disponível em: <http://www.etsi.org>. Citado na página 18.

ETSI. *Network Functions Virtualisation*. 2012. Disponível em: <http://www.etsi.org/technologies-clusters/technologies/nfv>. Citado na página 17.

ETSI. Network Functions Virtualisation - White Paper #1. 2012. Disponível em: <https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper1.pdf>. Citado 2 vezes nas páginas 18 and 28.

ETSI. *ETSI Network Function Virtualisation enters Phase 2*. 2014. Disponível em: <http://www.etsi.org/index.php/news-events/news/850-2014-12-news-etsi-network-function-virtualization-enters-phase-2>. Citado na página 18.

ETSI. Network Functions Virtualisation - White Paper #3. 2014. Disponível em: <https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf>. Citado 4 vezes nas páginas 17, 18, 28, and 41.

ETSI GS NFV. Architectural Framework. 2014. Disponível em: <http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01-_60/gs_NFV002v010201p.pdf>. Citado 2 vezes nas páginas 28 and 41.

ETSI GS NFV-INF. Methodology to describe Interfaces and Abstractions. 2014. Disponível em: <http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/007/01.01.01_60/gs_NFV-INF007v010101p.pdf>. Citado na página 34.

ETSI GS NFV-MAN. NFV Management and Orchestration. 2014. Disponível em: <http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf>. Citado 4 vezes nas páginas 28, 41, 42, and 45.

ETSI GS NFV-SWA. Virtual Network Functions Architecture. 2014. Disponível em: <http://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_NFV-SWA001v010101p.pdf>. Citado 2 vezes nas páginas 28 and 41.

FENG, X.; SHEN, J.; FAN, Y. Rest: An alternative to rpc for web services architecture. In: IEEE. *Future Information Networks, 2009. ICFIN 2009. First International Conference on*. [S.l.], 2009. p. 7–10. Citado na página 21.

FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. Tese (Doutorado) — University of California, Irvine, 2000. Citado na página 21.

GATES, M.; WARSHAVSKY, A. *Iperf version 2.0. 3*. 2012. Citado na página 83.

GHIJSEN, M.; HAM, J. V. D.; GROSSO, P.; LAAT, C. D. Towards an infrastructure description language for modeling computing infrastructures. In: IEEE. *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*. [S.l.], 2012. p. 207–214. Citado na página 31.

GHIJSEN, M.; HAM, J. V. D.; GROSSO, P.; DUMITRU, C.; ZHU, H.; ZHAO, Z.; LAAT, C. D. A semantic-web approach for modeling computing infrastructures. *Computers & Electrical Engineering*, Elsevier, v. 39, n. 8, p. 2553–2565, 2013. Citado 3 vezes nas páginas 19, 31, and 45.

GUDIVADA, N.; KALAVALA, M. Semantic web services. In: . [S.l.]: the Consortium for Computing Sciences in Colleges., 2005. Citado na página 25.

HAM, J. van der; DIJKSTRA, F.; ŁAPACZ, R.; BROWN, A. The network markup language (nml) a standardized network topology abstraction for inter-domain and cross-layer network applications. In: *Proceedings of the 13th Terena Networking Conference*. [S.l.: s.n.], 2013. Citado 3 vezes nas páginas 19, 31, and 45.

JENA Framework. *Apache JENA*. 2014. Disponível em: <https://jena.apache.org/>. Citado na página 92.

Jos De Roo. *Euler Yet another proof Engine - EYE*. 2009. Disponível em: <http://eulersharp.sourceforge.net/>. Citado na página 65.

ML2. *Neutron/ML2*. 2013. Disponível em: <https://wiki.openstack.org/wiki/Neutron/ML2>. Citado na página 32.

NFVO Architecture. *NFVO Architecture*. 2016. Disponível em: <http://openbaton.github.io/documentation/nfvo-architecture/>. Citado na página 87.

NOTATION3. *Notation3 (N3): A readable RDF syntax*. 2011. Disponível em: <https://www.w3.org/TeamSubmission/n3/>. Citado 2 vezes nas páginas 23 and 58.

NOY, N. F.; MCGUINNESS, D. L. *et al. Ontology development 101: A guide to creating your first ontology*. [S.l.]: Stanford knowledge systems laboratory technical report KSL-01-05 and Stanford medical informatics technical report SMI-2001-0880, Stanford, CA, 2001. Citado 2 vezes nas páginas 24 and 41.

OGF. *Open Grid Forum*. Disponível em: <https://www.ogf.org/>. Citado na página 31.

OpenBaton. *OpenBaton*. 2014. Disponível em: <http://openbaton.github.io/>. Citado na página 28.

OpenBaton Use Case. *Use case example: Iperf client - server*. 2014. Disponível em: <http://openbaton.github.io/documentation/use-case-example/>. Citado 2 vezes nas páginas 83 and 85.

OpenBaton VNFD. *Virtual Network Function Descriptor*. 2014. Disponível em: <http://openbaton.github.io/documentation/vnf-descriptor/>. Citado na página 56.

OpenDayLight. *OpenDaylight Platform*. 2013. Disponível em: <http://www.opendaylight.org/>. Citado na página 32.

OpenMano. *OpenMano Project*. 2014. Disponível em: <https://github.com/nfvlabs/openmano>. Citado 2 vezes nas páginas 29 and 34.

OpenStack. *OpenStack*. 2011. Disponível em: <https://www.openstack.org/>. Citado na página 32.

PITTARAS, C.; GHIJSEN, M.; WIBISONO, A.; GROSSO, P.; HAM, J. V. D.; LAAT, C. D. Semantic distributed resource discovery for multiple resource providers. In: IEEE. *Semantics, Knowledge and Grids (SKG), 2012 Eighth International Conference on*. [S.l.], 2012. p. 225–228. Citado na página 30.

POX. *About POX*. 2016. Disponível em: <ttp://www.noxrepo.org/pox/about-pox/>. Citado na página 32.

PROJECT, S. *SnS Project*. 2016. Disponível em: <https://github.com/LCuellarH/SnS>. Citado na página 83.

Protégé. *Protégé Project*. 2016. Disponível em: <http://protege.stanford.edu/>. Citado na página 50.

RDF. *Resource Description Framework*. 2014. Disponível em: <http://www.w3.org/RDF/>. Citado 2 vezes nas páginas 19 and 22.

RDF Schema. *RDF Schema 1.1*. 2014. Disponível em: <https://www.w3.org/TR/rdf-schema/>. Citado na página 23.

RDF Syntax. *RDF 1.1 XML Syntax*. 2014. Disponível em: <https://www.w3.org/TR/rdf-schema/>. Citado na página 23.

RDO. *RDO Project*. 2016. Disponível em: <https://www.rdoproject.org/Quickstart>. Citado na página 32.

RESTdesc. *RESTdesc – Semantic descriptions for hypermedia APIs*. 2011. Disponível em: <http://restdesc.org/>. Citado 3 vezes nas páginas 19, 25, and 60.

RICHARDS, R. Representational state transfer (rest). In: *Pro PHP XML and Web Services*. [S.l.]: Springer, 2006. p. 633–672. Citado na página 21.

SEMANTIC-WEB-AFFINITY-GROUP. Rdf and owl - a simple overview of the building blocks of the semantic web. In: . [S.l.: s.n.], 2007. Citado na página 25.

SOUZA, T. P. C.; SANTOS, M. A. S.; PAULA, L. B.; ROTHENBERG, C. E. Towards semantic networks models via graph databases for sdn applications. In: *Europe Workshop on Software Defined Networks (EWDSN 2015)*. [S.l.: s.n.], 2015. p. 49–54. Citado na página 19.

Spring Framework. *Spring Framework*. 2002. Disponível em: <https://spring.io/guides/gs/rest-service/>. Citado na página 61.

SZABó, R.; SONKOLY, B.; KIND, M. *Deliverable 2.2: Final Architecture*. 2014. Disponível em: <http://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/UNIFY%20Deliverable%202.2%20Final%20Architecture.pdf>. Citado na página 30.

Ubuntu. *Ubuntu JUJU*. 2014. Disponível em: <https://jujucharms.com/>. Citado na página 32.

UNIFY. *Nnifying Cloud and Carrier Networks*. 2014. Disponível em: <https://www.fp7-unify.eu/>. Citado na página 30.

VERBORGH, R.; ROO., J. D. *Eye Public Reasoner*. 2012. Disponível em: <http://eye.restdesc.org/>. Citado na página 83.

VERBORGH, R.; STEINER, T.; DEURSEN, D.; WALLE, R. Van de; VALLÉS, J. G. Efficient runtime service discovery and consumption with hyperlinked restdesc. In: IEEE. *Next Generation Web Services Practices (NWeSP), 2011 7th International Conference on*. [S.l.], 2011. p. 373–379. Citado na página 26.

VERBORGH, R.; STEINER, T.; DEURSEN, D. V.; COPPENS, S.; MANNENS, E.; WALLE, R. Van de; VALLÉS, J. G. Integrating data and services through functional semantic service descriptions. In: *Proceedings of the W3C Workshop on Data and Services Integration*. [S.l.: s.n.], 2011. Citado na página 26.

VERBORGH, R.; STEINER, T.; DEURSEN, D. V.; ROO, J. D.; WALLE, R. Van de; VALLÉS, J. G. Capturing the functionality of web services with functional descriptions. *Multimedia tools and applications*, Springer, v. 64, n. 2, p. 365–387, 2013. Citado na página 19.

VIM Plugin. *Create Vim Plugin*. 2016. Disponível em: <http://openbaton.github.io/documentation/vim-plugin/>. Citado na página 87.

WANG, X.; HALANG, W. *Discovery and Selection of Semantic Web Services*. [S.l.]: Springer-Verlag Berlin Heidelberg, 2013. 9– 22 p. Citado na página 22.

XILOURIS, G.; TROUVA, E.; LOBILLO, F.; SOARES, J. M.; CARAPINHA, J.; MCGRATH, M.; GARDIKIS, G.; PAGLIERANI, P.; PALLIS, E.; ZUCCARO, L. *et al.* T-nova: a marketplace for virtualized network functions. In: IEEE. *Networks and Communications (EuCNC), 2014 European Conference on*. [S.l.], 2014. p. 1–5. Citado na página 30.

ZHOU, W.; LI, L.; LUO, M.; CHOU, W. Rest api design patterns for sdn northbound api. In: IEEE. *Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on*. [S.l.], 2014. p. 358–365. Citado na página 32.

# Annex

# ANNEX A – NFV Ontology Notation 3 File

Listing A.1 shows the complete definition of NOn in N3 language.

Listing A.1 – NFV Ontology File

```
@prefix : <https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#>.
@prefix non: <https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix xml: <http://www.w3.org/XML/1998/namespace>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix xsp: <http://www.owl-ontologies.com/2005/08/07/xsp.owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix swrl: <http://www.w3.org/2003/11/swrl#>.
@prefix swrlb: <http://www.w3.org/2003/11/swrlb#>.
@prefix protege: <http://protege.stanford.edu/plugins/owl/protege#>.
@base <https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl>.

<https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl>

rdf:type owl:Ontology.

#################################################################
#
#    Object Properties
#
#################################################################

https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#has_connection_point

non:has_connection_point rdf:type owl:ObjectProperty;

        rdfs:range non:connection_point;
        rdfs:domain [ rdf:type owl:Class;
        owl:unionOf ( non:vnfc
        non:vnfd
        )
        ].
###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#has_constituent__vdu

non:has_constituent__vdu rdf:type owl:ObjectProperty;
        rdfs:range non:constituent_vdu;
        rdfs:domain non:deployment_flavour.

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#has_deployment_flavour

non:has_deployment_flavour rdf:type owl:ObjectProperty;
        rdfs:range non:deployment_flavour;
        rdfs:domain non:vnfd.

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#has_vdu
```

```
47 |
48 |non:has_vdu rdf:type owl:ObjectProperty;
49 |         rdfs:range non:vdu;
50 |         rdfs:domain non:vnfd.
51 |
52 |###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
   |   non.owl#has_virtual_link
53 |
54 |non:has_virtual_link rdf:type owl:ObjectProperty;
55 |         rdfs:range non:vld;
56 |         rdfs:domain non:vnfd.
57 |
58 |###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
   |   non.owl#has_vnfc
59 |
60 |non:has_vnfc rdf:type owl:ObjectProperty;
61 |         rdfs:domain non:vdu;
62 |         rdfs:range non:vnfc.
63 |
64 |#########################################################
65 |#
66 |#    Data properties
67 |#
68 |#########################################################
69 |
70 |https://github.com/LCuellarH/NOn/blob/master/datamodel/non.
   |   owl#computation_requirement
71 |
72 |non:computation_requirement rdf:type owl:DatatypeProperty;
73 |         rdfs:range xsd:string;
74 |         rdfs:domain non:vdu.
75 |
76 |###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
   |   non.owl#connection_points_references
77 |
78 |non:connection_points_references rdf:type owl
   |   :DatatypeProperty;
79 |         rdfs:range xsd:string;
80 |         rdfs:domain non:vld.
81 |
82 |###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
   |   non.owl#connectivity_type
83 |
84 |non:connectivity_type rdf:type owl:DatatypeProperty;
85 |         rdfs:range xsd:string;
86 |         rdfs:domain non:vld.
87 |
88 |###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
   |   non.owl#constituent_vnfc
89 |
90 |non:constituent_vnfc rdf:type owl:DatatypeProperty;
91 |         rdfs:range xsd:string;
92 |         rdfs:domain non:constituent_vdu.
93 |
94 |###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
   |   non.owl#constraint
95 |
96 |non:constraint rdf:type owl:DatatypeProperty;
97 |         rdfs:domain non:constituent_vdu.
98 |
99 |###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
   |   non.owl#dependency
100|
101|non:dependency rdf:type owl:DatatypeProperty;
102|         rdfs:range xsd:string;
103|         rdfs:domain non:vnfd.
104|
```

```
105 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#descriptor_version
106 
107 non:descriptor_version rdf:type owl:DatatypeProperty;
108         rdfs:range xsd:string;
109         rdfs:domain non:vnfd.
110 
111 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#id
112 
113 non:id rdf:type owl:DatatypeProperty;
114 
115         rdfs:range xsd:string;
116         rdfs:domain [ rdf:type owl:Class;
117         owl:unionOf ( non:connection_point
118         non:deployment_flavour
119         non:descriptors
120         non:vnfc
121         )
122         ].
123 
124 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#lifecycle_event
125 
126 non:lifecycle_event rdf:type owl:DatatypeProperty;
127 
128         rdfs:range xsd:string;
129         rdfs:domain [ rdf:type owl:Class;
130         owl:unionOf ( non:vdu
131         non:vnfd
132          )
133          ].
134 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#number_of_instances
135 
136 non:number_of_instances rdf:type owl:DatatypeProperty;
137 
138         rdfs:range xsd:int;
139         rdfs:domain non:constituent_vdu.
140 
141 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#scale_in_out
142 
143 non:scale_in_out rdf:type owl:DatatypeProperty;
144 
145          rdfs:range xsd:int;
146          rdfs:domain non:vdu.
147 
148 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#type
149 
150 non:type rdf:type owl:DatatypeProperty;
151           rdfs:range xsd:string;
152           rdfs:domain non:connection_point.
153 
154 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#vdu_reference
155 
156 non:vdu_reference rdf:type owl:DatatypeProperty;
157         rdfs:range xsd:string;
158         rdfs:domain non:constituent_vdu.
159 
160 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#vendor
161 
162 non:vendor rdf:type owl:DatatypeProperty;
163         rdfs:range xsd:string;
```

```
164         rdfs:domain non:vnfd.
165 
166 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#virtual_link_reference
167 
168 non:virtual_link_reference rdf:type owl:DatatypeProperty;
169 
170         rdfs:range xsd:string;
171         rdfs:domain non:connection_point.
172 
173 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#virtual_memory_resource_element
174 
175 non:virtual_memory_resource_element rdf:type owl
    :DatatypeProperty;
176 
177         rdfs:range xsd:int;
178         rdfs:domain non:vdu.
179 
180 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#virtual_network_bandwidth_resource
181 
182 non:virtual_network_bandwidth_resource rdf:type owl
    :DatatypeProperty;
183 
184         rdfs:range xsd:int;
185         rdfs:domain non:vdu.
186 
187 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#vm_image
188 
189 non:vm_image rdf:type owl:DatatypeProperty;
190 
191         rdfs:range xsd:anyURI;
192         rdfs:domain non:vdu.
193 
194 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#vnf_version
195 
196 non:vnf_version rdf:type owl:DatatypeProperty;
197         rdfs:range xsd:string;
198         rdfs:domain non:vnfd.
199 
200 ######################################################
201 #
202 #    Classes
203 #
204 ######################################################
205 
206 https://github.com/LCuellarH/NOn/blob/master/datamodel/non.
    owl#compute_component
207 
208 non:compute_component rdf:type owl:Class;
209         rdfs:subClassOf non:hardware_components.
210 
211 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#connection_point
212 
213 non:connection_point rdf:type owl:Class;
214         rdfs:subClassOf non:misc ,
215         [ rdf:type owl:Restriction;
216         owl:onProperty non:virtual_link_reference;
217         owl:maxCardinality "1"^^xsd:nonNegativeInteger
218         ] ,
219         [ rdf:type owl:Restriction;
220         owl:onProperty non:id;
221         owl:cardinality "1"^^xsd:nonNegativeInteger
```

```
222         ] ,
223         [ rdf:type owl:Restriction;
224         owl:onProperty non:type;
225         owl:cardinality "1"^^xsd:nonNegativeInteger
226         ].
227
228 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
        non.owl#constituent_vdu
229
230 non:constituent_vdu rdf:type owl:Class;
231         rdfs:subClassOf non:misc ,
232         [ rdf:type owl:Restriction;
233         owl:onProperty non:constraint;
234         owl:minCardinality "0"^^xsd:nonNegativeInteger
235         ] ,
236         [ rdf:type owl:Restriction;
237         owl:onProperty non:vdu_reference;
238         owl:cardinality "1"^^xsd:nonNegativeInteger
239         ] ,
240         [ rdf:type owl:Restriction;
241         owl:onProperty non:number_of_instances;
242         owl:cardinality "1"^^xsd:nonNegativeInteger
243         ] ,
244         [ rdf:type owl:Restriction;
245         owl:onProperty non:constituent_vnfc;
246         owl:minCardinality "1"^^xsd:nonNegativeInteger
247         ].
248
249 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
        non.owl#cpu
250
251 non:cpu rdf:type owl:Class;
252          rdfs:subClassOf non:compute_component.
253
254 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
        non.owl#deployment_flavour
255
256 non:deployment_flavour rdf:type owl:Class;
257
258         rdfs:subClassOf non:misc ,
259         [ rdf:type owl:Restriction;
260         owl:onProperty non:id;
261         owl:cardinality "1"^^xsd:nonNegativeInteger
262         ].
263
264 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
        non.owl#descriptors
265
266 non:descriptors rdf:type owl:Class;
267         rdfs:subClassOf non:functional_blocks.
268
269 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
        non.owl#functional_blocks
270
271 non:functional_blocks rdf:type owl:Class;
272         rdfs:subClassOf non:nfv.
273
274 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
        non.owl#hardware_components
275
276 non:hardware_components rdf:type owl:Class;
277         rdfs:subClassOf non:nfvi.
278
279 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
        non.owl#hypervisor
280
```

```
non:hypervisor rdf:type owl:Class;
        rdfs:subClassOf non:nfvi.

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#mano

non:mano rdf:type owl:Class;
         rdfs:subClassOf non:functional_blocks.

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#memory

non:memory rdf:type owl:Class;
        rdfs:subClassOf non:compute_component.

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#misc

non:misc rdf:type owl:Class;
         rdfs:subClassOf non:nfv.

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#network_component

non:network_component rdf:type owl:Class;
        rdfs:subClassOf non:hardware_components.

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#nfv

non:nfv rdf:type owl:Class.

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#nfvi

non:nfvi rdf:type owl:Class;
         rdfs:subClassOf non:functional_blocks.

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#nfvo_component

non:nfvo_component rdf:type owl:Class;
        rdfs:subClassOf non:mano.

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#reference_points

non:reference_points rdf:type owl:Class;
        rdfs:subClassOf non:nfv.

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#storage_component

non:storage_component rdf:type owl:Class;
        rdfs:subClassOf non:hardware_components.

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#vdu

non:vdu rdf:type owl:Class;

         rdfs:subClassOf non:descriptors ,
        [ rdf:type owl:Restriction;
        owl:onProperty non:virtual_network_bandwidth_resource
             ;
        owl:cardinality "1"^^xsd:nonNegativeInteger
        ] ,
```

```
337         [ rdf:type owl:Restriction;
338         owl:onProperty non:id;
339         owl:cardinality "1"^^xsd:nonNegativeInteger
340         ] ,
341         [ rdf:type owl:Restriction;
342         owl:onProperty non:computation_requirement;
343         owl:cardinality "1"^^xsd:nonNegativeInteger
344         ] ,
345         [ rdf:type owl:Restriction;
346         owl:onProperty non:scale_in_out;
347         owl:maxCardinality "1"^^xsd:nonNegativeInteger
348         ] ,
349         [ rdf:type owl:Restriction;
350         owl:onProperty non:vm_image;
351         owl:maxCardinality "1"^^xsd:nonNegativeInteger
352         ] ,
353         [ rdf:type owl:Restriction;
354         owl:onProperty non:virtual_memory_resource_element;
355         owl:cardinality "1"^^xsd:nonNegativeInteger
356         ] ,
357         [ rdf:type owl:Restriction;
358         owl:onProperty non:lifecycle_event;
359         owl:minCardinality "0"^^xsd:nonNegativeInteger
360         ] ,
361         [ rdf:type owl:Restriction;
362         owl:onProperty non:has_vnfc;
363         owl:minCardinality "1"^^xsd:nonNegativeInteger
364         ].
365
366 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#vim_component
367
368 non:vim_component rdf:type owl:Class;
369         rdfs:subClassOf non:mano.
370
371 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#virtual_compute_component
372
373 non:virtual_compute_component rdf:type owl:Class;
374         rdfs:subClassOf non:virtualization_components.
375
376 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#virtual_network_component
377
378 non:virtual_network_component rdf:type owl:Class;
379         rdfs:subClassOf non:virtualization_components.
380
381 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#virtual_storage_component
382
383 non:virtual_storage_component rdf:type owl:Class;
384         rdfs:subClassOf non:virtualization_components.
385
386 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#virtualization_components
387
388 non:virtualization_components rdf:type owl:Class;
389         rdfs:subClassOf non:nfvi.
390
391 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#vld
392
393 non:vld rdf:type owl:Class;
394
395          rdfs:subClassOf non:descriptors ,
396         [ rdf:type owl:Restriction;
```

```
397         owl:onProperty non:connection_points_references;
398         owl:minCardinality "2"^^xsd:nonNegativeInteger
399         ] ,
400         [ rdf:type owl:Restriction;
401         owl:onProperty non:id;
402         owl:cardinality "1"^^xsd:nonNegativeInteger
403         ] ,
404         [ rdf:type owl:Restriction;
405         owl:onProperty non:connectivity_type;
406         owl:cardinality "1"^^xsd:nonNegativeInteger
407         ].
408
409 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#vnf
410
411 non:vnf rdf:type owl:Class;
412          rdfs:subClassOf non:nfv.
413
414 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#vnfc
415
416 non:vnfc rdf:type owl:Class;
417           rdfs:subClassOf non:vnf ,
418         [ rdf:type owl:Restriction;
419         owl:onProperty non:has_connection_point;
420         owl:minCardinality "1"^^xsd:nonNegativeInteger
421         ] ,
422         [ rdf:type owl:Restriction;
423         owl:onProperty non:id;
424         owl:cardinality "1"^^xsd:nonNegativeInteger
425         ].
426
427 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
    non.owl#vnfd
428
429 non:vnfd rdf:type owl:Class;
430           rdfs:subClassOf non:descriptors ,
431         non:vnf ,
432         [ rdf:type owl:Restriction;
433         owl:onProperty non:dependency;
434         owl:minCardinality "0"^^xsd:nonNegativeInteger
435         ] ,
436         [ rdf:type owl:Restriction;
437         owl:onProperty non:lifecycle_event;
438         owl:minCardinality "0"^^xsd:nonNegativeInteger
439         ] ,
440         [ rdf:type owl:Restriction;
441         owl:onProperty non:has_deployment_flavour;
442         owl:minCardinality "0"^^xsd:nonNegativeInteger
443         ] ,
444         [ rdf:type owl:Restriction;
445         owl:onProperty non:vendor;
446         owl:cardinality "1"^^xsd:nonNegativeInteger
447         ] ,
448         [ rdf:type owl:Restriction;
449         owl:onProperty non:has_connection_point;
450         owl:minCardinality "1"^^xsd:nonNegativeInteger
451         ] ,
452         [ rdf:type owl:Restriction;
453         owl:onProperty non:vnf_version;
454         owl:cardinality "1"^^xsd:nonNegativeInteger
455         ] ,
456         [ rdf:type owl:Restriction;
457         owl:onProperty non:has_virtual_link;
458         owl:minCardinality "0"^^xsd:nonNegativeInteger
459         ] ,
```

```
460         [ rdf:type owl:Restriction;
461         owl:onProperty non:descriptor_version;
462         owl:cardinality "1"^^xsd:nonNegativeInteger
463         ] ,
464         [ rdf:type owl:Restriction;
465         owl:onProperty non:has_vdu;
466         owl:minCardinality "1"^^xsd:nonNegativeInteger
467         ] ,
468         [ rdf:type owl:Restriction;
469         owl:onProperty non:id;
470         owl:cardinality "1"^^xsd:nonNegativeInteger
471         ].
472
473 ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
        non.owl#vnfm_component
474
475 non:vnfm_component rdf:type owl:Class;
476         rdfs:subClassOf non:mano.
```

# ANNEX B – NFV/VNFD Deployment Files

Listing B.1 shows the VNFD file for OpenMano implementations.

Listing B.1 – OpenMano VNFD File

```
---
vnf:
    name: TEMPLATE
    description: This is a template to help in the creation of your own VNFs
    # class: parent      # Optional. Used to organize VNFs
    external-connections:
    -   name:               mgmt0
        type:               mgmt        # "mgmt" (autoconnect to management net), "bridge", "data"
        VNFC:               TEMPLATE-VM # Virtual Machine this interface belongs to
        local_iface_name:   mgmt0       # interface name inside this Virtual Machine (must be defined in the VNFC section)
        description:        Management interface
    -   name:               xe0
        type:               data
        VNFC:               TEMPLATE-VM
        local_iface_name:   xe0
        description:        Data interface 1
    -   name:               xe1
        type:               data
        VNFC:               TEMPLATE-VM
        local_iface_name:   xe1
        description:        Data interface 2
    -   name:               ge0
        type:               bridge
        VNFC:               TEMPLATE-VM
        local_iface_name:   ge0
        description:        Bridge interface
    VNFC:                               # Virtual machine array
    -   name:       TEMPLATE-VM         # name of Virtual Machine
        description: TEMPLATE description
        VNFC image: /path/to/imagefolder/TEMPLATE-VM.qcow2
        # image metadata: {"bus":"ide", "os_type":"windows", "use_incremental": "no" } #Optional
        # processor:                    #Optional
        #     model: Intel(R) Xeon(R) CPU E5-4620 0 @ 2.20GHz
        #     features: ["64b", "iommu", "lps", "tlbps", "hwsv", "dioc", "ht"]
        # hypervisor:                   #Optional
        #     type: QEMU-kvm
        #     version: "10002|12001|2.6.32-358.el6.x86_64"
        # vcpus: 1          # Only for traditional cloud VMs. Number of virtual CPUs (oversubscription is allowed).
        # ram: 1024         # Only for traditional cloud VMs. Memory in MBytes (not from hugepages, oversubscription is allowed)
        # disk: 10          # disk size in GiB, by default 1
        numas:
        -   paired-threads: 5           # "cores", "paired-threads", "threads"
```

```
43              paired-threads-id: [ [0,1], [2,3], [4,5], [6,7],
                   [8,9] ] # By default follows incremental order
44              memory: 14                  # GBytes
45              interfaces:
46              -   name:       xe0
47                  vpci:       "0000:00:11.0"
48                  dedicated: "yes"          # "yes"(passthrough)
                       , "no"(sriov with vlan tags), "yes:sriov"(
                       sriovi, but exclusive and without vlan tag
                       )
49                  bandwidth: 10 Gbps
50                  # mac_address: '20:33:45:56:77:44' #avoid
                       this option if possible
51              -   name:       xe1
52                  vpci:       "0000:00:12.0"
53                  dedicated: "yes"
54                  bandwidth: 10 Gbps
55                  # mac_address: '20:33:45:56:77:45' #avoid
                       this option if possible
56          bridge-ifaces:
57          -   name:       mgmt0
58              vpci:       "0000:00:09.0"    # Optional. Virtual
                   PCI address
59              bandwidth: 1 Mbps             # Optional.
                   Informative only
60              # mac_address: '20:33:45:56:77:46' #avoid this
                   option if possible
61              # model:        'virtio'      # ("virtio","e1000
                   ","ne2k_pci","pcnet","rtl8139") By default, it
                    is automatically filled by libvirt
62          -   name:       ge0
63              vpci:       "0000:00:10.0"
64              bandwidth: 1 Mbps
65              # mac_address: '20:33:45:56:77:47' #avoid this
                   option if possible
66              # model:        'virtio'      # ("virtio","e1000
                   ","ne2k_pci","pcnet","rtl8139") By default, it
                    is automatically filled by libvirt
67          devices:                          # Optional, order
               determines device letter asignation (hda, hdb,
               ...)
68          -   type:       disk              # "disk","cdrom","xml"
69              image:      /path/to/imagefolder/SECOND-DISK.qcow2
70              # image metadata: {"bus":"ide", "os_type":"
                   windows", "use_incremental": "no" }
71              # vpci:       "0000:00:03.0"   # Optional, not for
                    disk or cdrom
72          -   type:       cdrom
73              image:      /path/to/imagefolder/CDROM-IMAGE.qcow2
74              # image metadata: {"bus":"ide", "os_type":"
                   windows", "use_incremental": "no" }
75          -   type:       xml
76              image:      /path/to/imagefolder/ADDITIONAL-DISK.
                   qcow2    # Optional, depending on the device
                   type
77              image metadata: {"bus":"ide", "os_type":"windows"
                   , "use_incremental": "no" }  # Optional,
                   depending on the device type
78              vpci:       "0000:00:03.0"
                                    # Optional, depending
                    on the device type (not needed for disk or
                   cdrom)
79              xml:    '   xml text for XML described devices. Do
                    not use single quotes inside
80                          The following words, if found, will be
                                replaced:
81                          __file__    by image path, (image must
```

```
                                be provided)
82                          __format__  by qcow2 or raw (image
                               must be provided)
83                          __dev__     by device letter (b, c, d
                               ...)
84                          __vpci__    by vpci (vpci must be
                               provided)
85                  ,
86      # Additional Virtual Machines would be included here
```

Listing B.2 shows the Virtualizer file for Unify implementations.

Listing B.2 – Unify VNFD File

```
1  <?xml version="1.0" ?>
2  <virtualizer>
3          <id>UID01</id>
4          <name>Local Orchestrator - Docker</name>
5          <nodes>
6                  <node>
7                          <id>x86_64-elxa2chld12</id>
8                          <name>elxa2chld12-Linux</name>
9                          <type>Linux-3.13.0-79-generic</type>
10                         <ports>
11                                 <port>
12                                         <id>wlan0</id>
13                                         <name>wlan0</name>
14                                         <port_type>c4:d9:87:
                                              b1:c9:fc</
                                              port_type>
15                                 </port>
16                         </ports>
17                         <resources>
18                                 <cpu>4</cpu>
19                                 <mem>35520499712</mem>
20                                 <storage>16697704448</storage
                                       >
21                         </resources>
22                         <NF_instances>
23                                 <node>
24                             <id>NF1</id>
25                             <name>Firewall</name>
26                             <type>Container</type>
27                             <!-- example may contain <
                                   resources> here -->
28                             <ports>
29                                 <port>
30                                     <id>1</id>
31                                     <name>Firewall-1</name>
32                                     <port_type>unify</
                                           port_type>
33                                     <metadata>
34                                       <key>network</key>
35                                       <value>1</value>
36                                     </metadata>
37                                 </port>
38                                 <port>
39                                     <id>2</id>
40                                     <name>Firewall-2</name>
41                                     <port_type>unify</
                                           port_type>
42                                     <metadata>
43                                         <key>network</key>
44                                         <value>1</value>
45                                     </metadata>
46                                 </port>
```

```
                                </ports>
                                <metadata>
                                  <key>command</key>
                                  <value>tail -f /dev/null</value
                                    >
                                </metadata>
                                <metadata>
                                  <key>image</key>
                                  <value>unify/transit:0.1</value
                                    >
                                </metadata>
                                <metadata>
                                  <key>ports</key>
                                  <value>8080</value>
                                </metadata>
                                <metadata>
                                  <key>privileged</key>
                                  <value>true</value>
                                </metadata>
                            </node>
                            </NF_instances>
                    </node>
        </nodes>
</virtualizer>
```

# ANNEX C – NOn Semantic Descriptor Files

Listing C.1 shows the resulting OpenBaton VNFD file usig NOn.

Listing C.1 – NOn OpenBaton VNFD File

```
@prefix : <https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#> .
@prefix non: <https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix xsp: <http://www.owl-ontologies.com/2005/08/07/xsp.owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix swrl: <http://www.w3.org/2003/11/swrl#> .
@prefix swrlb: <http://www.w3.org/2003/11/swrlb#> .
@prefix protege: <http://protege.stanford.edu/plugins/owl/protege#> .
@base <https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl> .

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#m1.small
non:m1.small rdf:type owl:NamedIndividual ,
     non:deployment_flavour ;
     non:id "m1.small"^^xsd:string .

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#ob-iperf-client

non:ob-iperf-client rdf:type owl:NamedIndividual ,
     non:vnfd ;
     non:descriptor_version "0.1"^^xsd:string ;
     non:lifecycle_event "CONFIGURE"^^xsd:string ,
                         "INSTANTIATE"^^xsd:string ;
     non:vendor "fokus"^^xsd:string ;
     non:id "iperf-client"^^xsd:string ;
     non:has_deployment_flavour non:m1.small ;
     non:has_vdu non:ob_iperf_client_vdu ;
     non:has_connection_point non:ob_ipfer_client_connection_point ;
     non:has_virtual_link non:op_iperf_client_vld .

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#ob_iperf_client_vdu
non:ob_iperf_client_vdu rdf:type owl:NamedIndividual ,
     non:vdu ;
     non:virtual_network_bandwidth_resource "1000000"^^xsd:int ;
     non:virtual_memory_resource_element "1024"^^xsd:int ;
     non:scale_in_out "2"^^xsd:int ;
     non:vm_image "iperf_client_image"^^xsd:anyURI ;
     non:has_vnfc non:ob_iperf_client_vnfc .

###  https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#ob_iperf_client_vnfc
non:ob_iperf_client_vnfc rdf:type owl:NamedIndividual ,
     non:vnfc ;
```

```
45      non:has_connection_point non
            :ob_ipfer_client_connection_point .
46
47  ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
        non.owl#ob_ipfer_client_connection_point
48  non:ob_ipfer_client_connection_point rdf:type owl
        :NamedIndividual ,
49      non:connection_point ;
50      non:virtual_link_reference "private"^^xsd:string .
51
52  ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
        non.owl#op_iperf_client_vld
53
54  non:op_iperf_client_vld rdf:type owl:NamedIndividual ,
55      non:vld ;
56      non:id "private"^^xsd:string .
```

Listing C.2 shows the resulting OpenMano VNFD file usig NOn.

Listing C.2 – NOn OpenMano VNFD File

```
1   @prefix : <https://github.com/LCuellarH/NOn/blob/master/
        datamodel/non.owl#> .
2   @prefix non: <https://github.com/LCuellarH/NOn/blob/master/
        datamodel/non.owl#> .
3   @prefix owl: <http://www.w3.org/2002/07/owl#> .
4   @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
5   @prefix xml: <http://www.w3.org/XML/1998/namespace> .
6   @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
7   @prefix xsp: <http://www.owl-ontologies.com/2005/08/07/xsp.
        owl#> .
8   @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
9   @prefix swrl: <http://www.w3.org/2003/11/swrl#> .
10  @prefix swrlb: <http://www.w3.org/2003/11/swrlb#> .
11  @prefix protege: <http://protege.stanford.edu/plugins/owl/
        protege#> .
12  @base <https://github.com/LCuellarH/NOn/blob/master/datamodel
        /non.owl> .
13
14  ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
        non.owl#openmano_ge0_connection_point
15  non:openmano_ge0_connection_point rdf:type owl
        :NamedIndividual ,
16      non:connection_point ;
17      non:type "bridge"^^xsd:string ;
18      non:id "ge0"^^xsd:string .
19
20  ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
        non.owl#openmano_mgmt0_connection_point
21  non:openmano_mgmt0_connection_point rdf:type owl
        :NamedIndividual ,
22      non:connection_point ;
23      non:type "mgmt0"^^xsd:string ;
24      non:id "mgmt0"^^xsd:string .
25
26  ###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
        non.owl#openmano_vnfc
27  non:openmano_vnfc rdf:type owl:NamedIndividual ,
28      non:vnfc ;
29      non:id "TEMPLATE-VM"^^xsd:string ;
30      non:has_connection_point non
            :openmano_ge0_connection_point ,
31      non:openmano_mgmt0_connection_point ,
32      non:openmano_xe0_connection_point ,
33      non:openmano_xe1_connection_point .
```

```
34 |
35 |###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
   |   non.owl#openmano_vnfd
36 |non:openmano_vnfd rdf:type owl:NamedIndividual ,
37 |     non:vnfd .
38 |
39 |###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
   |   non.owl#openmano_xe0_connection_point
40 |non:openmano_xe0_connection_point rdf:type owl
   |   :NamedIndividual ,
41 |     non:connection_point ;
42 |     non:type "data"^^xsd:string ;
43 |     non:id "xe0"^^xsd:string .
44 |
45 |###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
   |   non.owl#openmano_xe1_connection_point
46 |non:openmano_xe1_connection_point rdf:type owl
   |   :NamedIndividual ,
47 |     non:connection_point ;
48 |     non:type "data"^^xsd:string ;
49 |     non:id "xe1"^^xsd:string .
```

Listing C.3 shows a resulting Generic VNFD file usig NOn.

Listing C.3 – NOn OpenMano VNFD File

```
1  |@prefix : <https://github.com/LCuellarH/NOn/blob/master/
   |   datamodel/non.owl#>.
2  |@prefix non: <https://github.com/LCuellarH/NOn/blob/master/
   |   datamodel/non.owl#>.
3  |@prefix owl: <http://www.w3.org/2002/07/owl#>.
4  |@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
5  |@prefix xml: <http://www.w3.org/XML/1998/namespace>.
6  |@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
7  |@prefix xsp: <http://www.owl-ontologies.com/2005/08/07/xsp.
   |   owl#>.
8  |@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
9  |@prefix swrl: <http://www.w3.org/2003/11/swrl#>.
10 |@prefix swrlb: <http://www.w3.org/2003/11/swrlb#>.
11 |@prefix protege: <http://protege.stanford.edu/plugins/owl/
   |   protege#>.
12 |@base <https://github.com/LCuellarH/NOn/blob/master/datamodel
   |   /non.owl>.
13 |
14 |###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
   |   non.owl#generic-conn-1
15 |non:generic-conn-1 rdf:type owl:NamedIndividual,
16 |     non:connection_point;
17 |     non:type "bridge"^^xsd:string;
18 |     non:id "ob1"^^xsd:string;
19 |     non:virtual_link_reference "private"^^xsd:string.
20 |
21 |###  https://github.com/LCuellarH/NOn/blob/master/datamodel/
   |   non.owl#generic-vdu-1
22 |non:generic-vdu-1 rdf:type owl:NamedIndividual,
23 |     non:vdu;
24 |     non:virtual_network_bandwidth_resource "1000000"^^xsd:int
   |        ;
25 |     non:virtual_memory_resource_element "1024"^^xsd:int;
26 |     non:scale_in_out "2"^^xsd:int;
27 |     non:computation_requirement "2"^^xsd:string;
28 |     non:vm_image "ubuntu-14.04-server-cloudimg-amd64-disk1"^^
   |        xsd:anyURI;
29 |     non:id "vim-instance"^^xsd:string;
30 |     non:has_vnfc non:generic-vnfc1.
31 |
```

```
32 ### https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#generic-vld-1
33 non:generic-vld-1 rdf:type owl:NamedIndividual ,
34     non:vld;
35     non:id "generic_vld_id"^^xsd:string;
36     non:connectivity_type "private"^^xsd:string.
37
38 ### https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#generic-vnfc1
39 non:generic-vnfc1 rdf:type owl:NamedIndividual ,
40     non:vnfc;
41     non:id "generic_vnfc1"^^xsd:string;
42     non:has_connection_point non:generic-conn-1.
43
44 ### https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#generic-vnfd-1
45 non:generic-vnfd-1 rdf:type owl:NamedIndividual ,
46     non:vnfd;
47     non:descriptor_version "0.2"^^xsd:string;
48     non:vnf_version "0.2"^^xsd:string;
49     non:lifecycle_event "INSTANTIATE-install.sh-install-srv.sh"^^xsd:string;
50     non:vendor "fokus"^^xsd:string;
51     non:id "iperf-server"^^xsd:string;
52     non:has_vdu non:generic-vdu-1;
53     non:has_virtual_link non:generic-vld-1;
54     non:has_deployment_flavour non:os-m1-small.
55
56 ### https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#os-m1-small
57
58 non:os-m1-small rdf:type owl:NamedIndividual ,
59       non:deployment_flavour;
60       non:id "m1.small"^^xsd:string.
61
62 ### https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#vim_openstack_25
63 non:vim_openstack_25 rdf:type owl:NamedIndividual ,
64     non:vim_component ;
65     non:id "10.1.1.25-vim-instance"^^xsd:string.
66 ### Generated by the OWL API (version 3.5.1) http://owlapi.sourceforge.net
```

# ANNEX D – SnS Workflow Files

Listing D.1 – SnS Metadata Goal Workflow File

```
PREFIX non: <https://github.com/LCuellarH/NOn/blob/master/datamodel/non.owl#>
PREFIX yaml: <http://example.org/yaml#>
PREFIX http: <http://www.w3.org/2011/http#>
PREFIX ob: <http://example.org/openbaton#>
PREFIX json: <http://example.org/json#>
PREFIX tmpl: <http://purl.org/restdesc/http-template#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xml: <http://www.w3.org/XML/1998/namespace>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX xsp: <http://www.owl-ontologies.com/2005/08/07/xsp.owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX swrl: <http://www.w3.org/2003/11/swrl#>
PREFIX swrlb: <http://www.w3.org/2003/11/swrlb#>
PREFIX protege: <http://protege.stanford.edu/plugins/owl/protege#>
PREFIX r: <http://www.w3.org/2000/10/swap/reason#>


[ a r:Proof, r:Conjunction;
  r:component <#lemma1>;
  r:gives {
    _:sk3 ob:has_metadata yaml:vnf_metadata.
  }].

<#lemma1> a r:Inference; r:gives {_:sk3 ob:has_metadata yaml:vnf_metadata}; r:evidence (
  <#lemma2>);
  r:rule <#lemma3>.

<#lemma2> a r:Inference; r:gives {_:sk4 http:methodName "GET".
  _:sk4 http:MessageHeader "Content-Type: application/json".
  _:sk4 http:requestURI ("http://localhost:8080/nfv/parser/openbaton/vnf/metadata?name=" "iperf-client" "&link=" "iperf_client_image"^^xsd:anyURI "&minCPU=" _:sk5 "&minRam=" "1024"^^xsd:int).
  _:sk4 http:resp _:sk6.
  _:sk6 http:body yaml:vnf_metadata.
  _:sk3 ob:has_metadata yaml:vnf_metadata}; r:evidence (
  <#lemma4>
  <#lemma5>
  <#lemma6>
  <#lemma7>
  <#lemma8>
  <#lemma9>
  <#lemma10>
  <#lemma11>
  <#lemma12>);
  r:rule <#lemma13>.

<#lemma3> a r:Extraction; r:gives {{?x0 ob:has_metadata yaml:vnf_metadata} => {?x0 ob:has_metadata yaml:vnf_metadata}};
  r:because [ a r:Parsing; r:source <file:///home/ldapusers/lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-vnfd-parser/ServiceDescriptor/OpenBaton/goals/Metadata-
```

```
      iPerf-Client-goal.n3>].
48
49 <#lemma4> a r:Extraction; r:gives {non:ob-iperf-client a non:
     vnfd};
50   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
       lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
       vnfd-parser/ServiceDescriptor/OpenBaton/resources/
       iperf_client.n3>].
51
52 <#lemma5> a r:Extraction; r:gives {non:ob-iperf-client non:
     vendor "fokus"};
53   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
       lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
       vnfd-parser/ServiceDescriptor/OpenBaton/resources/
       iperf_client.n3>].
54
55 <#lemma6> a r:Extraction; r:gives {non:ob-iperf-client non:
     descriptor_version "0.1"};
56   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
       lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
       vnfd-parser/ServiceDescriptor/OpenBaton/resources/
       iperf_client.n3>].
57
58 <#lemma7> a r:Extraction; r:gives {non:ob-iperf-client non:id
      "iperf-client"};
59   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
       lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
       vnfd-parser/ServiceDescriptor/OpenBaton/resources/
       iperf_client.n3>].
60
61 <#lemma8> a r:Extraction; r:gives {non:ob-iperf-client non:
     lifecycle_event "CONFIGURE"};
62   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
       lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
       vnfd-parser/ServiceDescriptor/OpenBaton/resources/
       iperf_client.n3>].
63
64 <#lemma9> a r:Extraction; r:gives {non:ob_iperf_client_vdu a
     non:vdu};
65   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
       lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
       vnfd-parser/ServiceDescriptor/OpenBaton/resources/
       iperf_client.n3>].
66
67 <#lemma10> a r:Extraction; r:gives {non:ob_iperf_client_vdu
     non:vm_image "iperf_client_image"^^xsd:anyURI};
68   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
       lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
       vnfd-parser/ServiceDescriptor/OpenBaton/resources/
       iperf_client.n3>].
69
70 <#lemma11> a r:Extraction; r:gives {non:ob_iperf_client_vdu
     non:virtual_memory_resource_element "1024"^^xsd:int};
71   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
       lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
       vnfd-parser/ServiceDescriptor/OpenBaton/resources/
       iperf_client.n3>].
72
73 <#lemma12> a r:Inference; r:gives {_:sk0 http:methodName "GET
     ".
74   _:sk0 http:MessageHeader "Content-Type: application/json".
75   _:sk0 http:requestURI ("http://localhost:8080/nfv/parser/
       openbaton/vnf/vnfd?vendor=" "fokus" "&version=" "0.1" "&
       name=" "iperf-client" "&vm_image=" "iperf_client_image"
       ^^xsd:anyURI "&virtuallink=" "private" "&lifecycle=" "
```

```
      CONFIGURE" "&dev_flavour=" "m1.small" "&scaleinout=" "2"
      ^^xsd:int "").
  _:sk0 http:resp _:sk1.
  _:sk1 http:body json:openbaton_vnfd.
  _:sk2 a json:file.
  _:sk2 a ob:vnfd.
  _:sk3 ob:has_vnfd _:sk2}; r:evidence (
  <#lemma4>
  <#lemma5>
  <#lemma6>
  <#lemma7>
  <#lemma8>
  <#lemma9>
  <#lemma10>
  <#lemma14>
  <#lemma15>
  <#lemma16>
  <#lemma17>
  <#lemma18>);
  r:rule <#lemma19>.

<#lemma13> a r:Extraction; r:gives {{?x0 a non:vnfd.
  ?x0 non:vendor ?x1.
  ?x0 non:descriptor_version ?x2.
  ?x0 non:id ?x3.
  ?x0 non:lifecycle_event ?x4.
  ?x5 a non:vdu.
  ?x5 non:vm_image ?x6.
  ?x5 non:virtual_memory_resource_element ?x7.
  ?x8 ob:has_vnfd ?x9} => {_:x10 http:methodName "GET".
  _:x10 http:MessageHeader "Content-Type: application/json".
  _:x10 http:requestURI ("http://localhost:8080/nfv/parser/
      openbaton/vnf/metadata?name=" ?x3 "&link=" ?x6 "&minCPU=
      " _:x11 "&minRam=" ?x7).
  _:x10 http:resp _:x12.
  _:x12 http:body yaml:vnf_metadata.
  ?x8 ob:has_metadata yaml:vnf_metadata}};
  r:because [ a r:Parsing; r:source <file:///home/ldapusers/
      lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
      vnfd-parser/ServiceDescriptor/OpenBaton/services/
      metadata/OpenBaton-Parser-metadata.n3>].

<#lemma14> a r:Extraction; r:gives {non:ob_iperf_client_vdu
    non:scale_in_out "2"^^xsd:int};
  r:because [ a r:Parsing; r:source <file:///home/ldapusers/
      lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
      vnfd-parser/ServiceDescriptor/OpenBaton/resources/
      iperf_client.n3>].

<#lemma15> a r:Extraction; r:gives {non:op_iperf_client_vld a
    non:vld};
  r:because [ a r:Parsing; r:source <file:///home/ldapusers/
      lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
      vnfd-parser/ServiceDescriptor/OpenBaton/resources/
      iperf_client.n3>].

<#lemma16> a r:Extraction; r:gives {non:op_iperf_client_vld
    non:connectivity_type "private"};
  r:because [ a r:Parsing; r:source <file:///home/ldapusers/
      lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
      vnfd-parser/ServiceDescriptor/OpenBaton/resources/
      iperf_client.n3>].

<#lemma17> a r:Extraction; r:gives {<https://github.com/
    LCuellarH/NOn/blob/master/datamodel/non.owl#m1.small> a
    non:deployment_flavour};
```

```
  r:because [ a r:Parsing; r:source <file:///home/ldapusers/
     lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
     vnfd-parser/ServiceDescriptor/OpenBaton/resources/
     iperf_client.n3>].

<#lemma18> a r:Extraction; r:gives {<https://github.com/
   LCuellarH/NOn/blob/master/datamodel/non.owl#m1.small> non:
   id "m1.small"};
  r:because [ a r:Parsing; r:source <file:///home/ldapusers/
     lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
     vnfd-parser/ServiceDescriptor/OpenBaton/resources/
     iperf_client.n3>].

<#lemma19> a r:Extraction; r:gives {{?x0 a non:vnfd.
  ?x0 non:vendor ?x1.
  ?x0 non:descriptor_version ?x2.
  ?x0 non:id ?x3.
  ?x0 non:lifecycle_event ?x4.
  ?x5 a non:vdu.
  ?x5 non:vm_image ?x6.
  ?x5 non:scale_in_out ?x7.
  ?x8 a non:vld.
  ?x8 non:connectivity_type ?x9.
  ?x10 a non:deployment_flavour.
  ?x10 non:id ?x11} => {_:x12 http:methodName "GET".
  _:x12 http:MessageHeader "Content-Type: application/json".
  _:x12 http:requestURI ("http://localhost:8080/nfv/parser/
     openbaton/vnf/vnfd?vendor=" ?x1 "&version=" ?x2 "&name="
      ?x3 "&vm_image=" ?x6 "&virtuallink=" ?x9 "&lifecycle="
     ?x4 "&dev_flavour=" ?x11 "&scaleinout=" ?x7 "").
  _:x12 http:resp _:x13.
  _:x13 http:body json:openbaton_vnfd.
  _:x14 a json:file.
  _:x14 a ob:vnfd.
  _:x15 ob:has_vnfd _:x14}};
  r:because [ a r:Parsing; r:source <file:///home/ldapusers/
     lcuellar/ownCloud/Luis/NOn/Developments/UseCases/nfv-
     vnfd-parser/ServiceDescriptor/OpenBaton/services/parser/
     OpenBaton-Parser.n3>].
```

Listing D.2 – SnS Use Case I: Scenario III Workflow

```
PREFIX non: <https://github.com/LCuellarH/NOn/blob/master/
   datamodel/non.owl#>
PREFIX yaml: <http://example.org/yaml#>
PREFIX http: <http://www.w3.org/2011/http#>
PREFIX tmpl: <http://purl.org/restdesc/http-template#>
PREFIX ob: <http://example.org/openbaton#>
PREFIX json: <http://example.org/json#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xml: <http://www.w3.org/XML/1998/namespace>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX xsp: <http://www.owl-ontologies.com/2005/08/07/xsp.owl
   #>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX swrl: <http://www.w3.org/2003/11/swrl#>
PREFIX swrlb: <http://www.w3.org/2003/11/swrlb#>
PREFIX protege: <http://protege.stanford.edu/plugins/owl/
   protege#>
PREFIX r: <http://www.w3.org/2000/10/swap/reason#>

[ a r:Proof, r:Conjunction;
  r:component <#lemma1>;
  r:component <#lemma2>;
```

```
21 |   r:gives {
22 |     _:sk5 non:has_vnfd non:generic-vnfd-1.
23 |     _:sk8 non:has_vnfd non:generic-vnfd-1.
24 |   }].
25 |
26 | <#lemma1> a r:Inference; r:gives {_:sk5 non:has_vnfd non:
   |    generic-vnfd-1}; r:evidence (
27 |   <#lemma3>);
28 |   r:rule <#lemma4>.
29 |
30 | <#lemma2> a r:Inference; r:gives {_:sk8 non:has_vnfd non:
   |    generic-vnfd-1}; r:evidence (
31 |   <#lemma5>);
32 |   r:rule <#lemma4>.
33 |
34 | <#lemma3> a r:Inference; r:gives {_:sk0 http:methodName "GET
   |    ".
35 |   _:sk0 http:MessageHeader "Content-Type: application/json".
36 |   _:sk0 http:requestURI ("nfv/parser/openmano/vnf/vnfd?
   |      vnf_description=" _:sk1 "&vnf_name=" _:sk2 "&vnfc_name="
   |      "generic_vnfc1" "&vnfc_description=" "&vm_image=" _:sk3
   |      "&ext_conn_name=" "ob1" "&ext_conn_iface_name=" "ob1" "&
   |      ext_conn_description=" "bridge" "&ext_conn_type" "bridge
   |      " "").
37 |   _:sk0 http:resp _:sk4.
38 |   _:sk4 http:body yaml:openmano_vnfd.
39 |   non:generic-vnfd-1 a yaml:file.
40 |   _:sk5 non:has_vnfd non:generic-vnfd-1}; r:evidence (
41 |   <#lemma6>
42 |   <#lemma7>
43 |   <#lemma8>
44 |   <#lemma9>
45 |   <#lemma10>
46 |   <#lemma11>
47 |   <#lemma12>);
48 |   r:rule <#lemma13>.
49 |
50 | <#lemma4> a r:Extraction; r:gives {{?x0 non:has_vnfd ?x1} =>
   |    {?x0 non:has_vnfd ?x1}};
51 |   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
   |      lcuellar/SnS/sns_server/nfv-vnfd-parser/
   |      ServiceDescriptor/OpenBaton/goals/vnfd-iPerf-Client-goal
   |      .n3>].
52 |
53 | <#lemma5> a r:Inference; r:gives {_:sk6 http:methodName "GET
   |    ".
54 |   _:sk6 http:MessageHeader "Content-Type: application/text".
55 |   _:sk6 http:requestURI ("http://localhost:8080/nfv/parser/
   |      openbaton/vnf/vnfd?vendor=" "fokus" "&version=" "0.2" "&
   |      name=" "iperf-server" "&vm_image=" "ubuntu-14.04-server-
   |      cloudimg-amd64-disk1"^^xsd:anyURI "&virtuallink=" "
   |      private" "&lifecycle=" "INSTANTIATE-install.sh-install-
   |      srv.sh" "&dev_flavour=" "m1.small" "&scaleinout=" "2"^^
   |      xsd:int "&vim=" "10.1.1.25-vim-instance" "").
56 |   _:sk6 http:resp _:sk7.
57 |   _:sk7 http:body json:openbaton_vnfd.
58 |   non:generic-vnfd-1 a json:file.
59 |   _:sk8 non:has_vnfd non:generic-vnfd-1}; r:evidence (
60 |   <#lemma6>
61 |   <#lemma14>
62 |   <#lemma15>
63 |   <#lemma16>
64 |   <#lemma17>
65 |   <#lemma18>
66 |   <#lemma19>
67 |   <#lemma20>
68 |   <#lemma21>
```

```
69    <#lemma22>
70    <#lemma23>
71    <#lemma24>
72    <#lemma25>
73    <#lemma26>);
74    r:rule <#lemma27>.
75
76  <#lemma6> a r:Extraction; r:gives {non:generic-vnfd-1 a non:
        vnfd};
77    r:because [ a r:Parsing; r:source <file:///home/ldapusers/
          lcuellar/SnS/sns_server/nfv-vnfd-parser/
          ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
          ].
78
79  <#lemma7> a r:Extraction; r:gives {non:generic-vnfc1 a non:
        vnfc};
80    r:because [ a r:Parsing; r:source <file:///home/ldapusers/
          lcuellar/SnS/sns_server/nfv-vnfd-parser/
          ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
          ].
81
82  <#lemma8> a r:Extraction; r:gives {non:generic-vnfc1 non:id "
        generic_vnfc1"};
83    r:because [ a r:Parsing; r:source <file:///home/ldapusers/
          lcuellar/SnS/sns_server/nfv-vnfd-parser/
          ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
          ].
84
85  <#lemma9> a r:Extraction; r:gives {non:generic-vnfc1 non:
        has_connection_point non:generic-conn-1};
86    r:because [ a r:Parsing; r:source <file:///home/ldapusers/
          lcuellar/SnS/sns_server/nfv-vnfd-parser/
          ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
          ].
87
88  <#lemma10> a r:Extraction; r:gives {non:generic-conn-1 a non:
        connection_point};
89    r:because [ a r:Parsing; r:source <file:///home/ldapusers/
          lcuellar/SnS/sns_server/nfv-vnfd-parser/
          ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
          ].
90
91  <#lemma11> a r:Extraction; r:gives {non:generic-conn-1 non:
        type "bridge"};
92    r:because [ a r:Parsing; r:source <file:///home/ldapusers/
          lcuellar/SnS/sns_server/nfv-vnfd-parser/
          ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
          ].
93
94  <#lemma12> a r:Extraction; r:gives {non:generic-conn-1 non:id
        "ob1"};
95    r:because [ a r:Parsing; r:source <file:///home/ldapusers/
          lcuellar/SnS/sns_server/nfv-vnfd-parser/
          ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
          ].
96
97  <#lemma13> a r:Extraction; r:gives {{?x0 a non:vnfd.
98    ?x1 a non:vnfc.
99    ?x1 non:id ?x2.
100   ?x1 non:has_connection_point ?x3.
101   ?x3 a non:connection_point.
102   ?x3 non:type ?x4.
103   ?x3 non:id ?x5} => {_:x6 http:methodName "GET".
104   _:x6 http:MessageHeader "Content-Type: application/json".
105   _:x6 http:requestURI ("nfv/parser/openmano/vnf/vnfd?
```

```
      vnf_description=" _:x7 "&vnf_name=" _:x8 "&vnfc_name=" ?
      x2 "&vnfc_description=" "&vm_image=" _:x9 "&
      ext_conn_name=" ?x5 "&ext_conn_iface_name=" ?x5 "&
      ext_conn_description=" ?x4 "&ext_conn_type" ?x4 "").
106   _:x6 http:resp _:x10.
107   _:x10 http:body yaml:openmano_vnfd.
108   ?x0 a yaml:file.
109   ?x0 a non:vnfd.
110   _:x11 non:has_vnfd ?x0}};
111   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
      lcuellar/SnS/sns_server/nfv-vnfd-parser/
      ServiceDescriptor/OpenMano/services/parser/OpenMano-
      Parser.n3>].
112
113 <#lemma14> a r:Extraction; r:gives {non:generic-vnfd-1 non:
    vendor "fokus"};
114   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
      lcuellar/SnS/sns_server/nfv-vnfd-parser/
      ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
      ].
115
116 <#lemma15> a r:Extraction; r:gives {non:generic-vnfd-1 non:
    descriptor_version "0.2"};
117   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
      lcuellar/SnS/sns_server/nfv-vnfd-parser/
      ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
      ].
118
119 <#lemma16> a r:Extraction; r:gives {non:generic-vnfd-1 non:id
     "iperf-server"};
120   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
      lcuellar/SnS/sns_server/nfv-vnfd-parser/
      ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
      ].
121
122 <#lemma17> a r:Extraction; r:gives {non:generic-vnfd-1 non:
    lifecycle_event "INSTANTIATE-install.sh-install-srv.sh"};
123   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
      lcuellar/SnS/sns_server/nfv-vnfd-parser/
      ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
      ].
124
125 <#lemma18> a r:Extraction; r:gives {non:generic-vdu-1 a non:
    vdu};
126   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
      lcuellar/SnS/sns_server/nfv-vnfd-parser/
      ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
      ].
127
128 <#lemma19> a r:Extraction; r:gives {non:generic-vdu-1 non:
    vm_image "ubuntu-14.04-server-cloudimg-amd64-disk1"^^xsd:
    anyURI};
129   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
      lcuellar/SnS/sns_server/nfv-vnfd-parser/
      ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
      ].
130
131 <#lemma20> a r:Extraction; r:gives {non:generic-vdu-1 non:
    scale_in_out "2"^^xsd:int};
132   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
      lcuellar/SnS/sns_server/nfv-vnfd-parser/
      ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
      ].
133
134 <#lemma21> a r:Extraction; r:gives {non:generic-vld-1 a non:
```

```
    vld};
135   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
        lcuellar/SnS/sns_server/nfv-vnfd-parser/
        ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
        ].
136
137 <#lemma22> a r:Extraction; r:gives {non:generic-vld-1 non:
    connectivity_type "private"};
138   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
        lcuellar/SnS/sns_server/nfv-vnfd-parser/
        ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
        ].
139
140 <#lemma23> a r:Extraction; r:gives {non:os-m1-small a non:
    deployment_flavour};
141   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
        lcuellar/SnS/sns_server/nfv-vnfd-parser/
        ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
        ].
142
143 <#lemma24> a r:Extraction; r:gives {non:os-m1-small non:id "
    m1.small"};
144   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
        lcuellar/SnS/sns_server/nfv-vnfd-parser/
        ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
        ].
145
146 <#lemma25> a r:Extraction; r:gives {non:vim_openstack_25 a
    non:vim_component};
147   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
        lcuellar/SnS/sns_server/nfv-vnfd-parser/
        ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
        ].
148
149 <#lemma26> a r:Extraction; r:gives {non:vim_openstack_25 non:
    id "10.1.1.25-vim-instance"};
150   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
        lcuellar/SnS/sns_server/nfv-vnfd-parser/
        ServiceDescriptor/OpenBaton/resources/ob-vnfdv1.0.1.n3>
        ].
151
152 <#lemma27> a r:Extraction; r:gives {{?x0 a non:vnfd.
153   ?x0 non:vendor ?x1.
154   ?x0 non:descriptor_version ?x2.
155   ?x0 non:id ?x3.
156   ?x0 non:lifecycle_event ?x4.
157   ?x5 a non:vdu.
158   ?x5 non:vm_image ?x6.
159   ?x5 non:scale_in_out ?x7.
160   ?x8 a non:vld.
161   ?x8 non:connectivity_type ?x9.
162   ?x10 a non:deployment_flavour.
163   ?x10 non:id ?x11.
164   ?x12 a non:vim_component.
165   ?x12 non:id ?x13} => {_:x14 http:methodName "GET".
166   _:x14 http:MessageHeader "Content-Type: application/text".
167   _:x14 http:requestURI ("http://localhost:8080/nfv/parser/
        openbaton/vnf/vnfd?vendor=" ?x1 "&version=" ?x2 "&name="
         ?x3 "&vm_image=" ?x6 "&virtuallink=" ?x9 "&lifecycle="
        ?x4 "&dev_flavour=" ?x11 "&scaleinout=" ?x7 "&vim=" ?x13
         "").
168   _:x14 http:resp _:x15.
169   _:x15 http:body json:openbaton_vnfd.
170   ?x0 a json:file.
171   ?x0 a non:vnfd.
```

```
172   _:x16 non:has_vnfd ?x0}};
173   r:because [ a r:Parsing; r:source <file:///home/ldapusers/
          lcuellar/SnS/sns_server/nfv-vnfd-parser/
          ServiceDescriptor/OpenBaton/services/parser/OpenBaton-
          Parser.n3>].
174
175 TC=4 TP=8 BC=0 BP=0 PM=0 CM=0 FM=0 AM=0
176 reasoning 7 [msec cputime] 6 [msec walltime]
177 #ENDS 0.055 [sec] IO=41/2 TC=4 TP=8 BC=0 BP=0 PM=0 CM=0 FM=0
      AM=0
178
179 [2016-04-27T18:15:48.155Z] in=41 out=2 step=8 brake=2 inf
      =129060 sec=0.055 inf/sec=2933182
```