NOn: Network Function Virtualization Ontology towards Semantic Service Implementation

Luis Cuellar Hoyos and Christian Esteve Rothenberg
School of Electrical and Computer Engineering (FEEC)
University of Campinas (UNICAMP), Campinas SP, Brazil
Email: {lcuellar, chesteve}@dca.fee.unicamp.br

***Abstract*—A hazard of ongoing Network Function Virtualization (NFV) realizations is the lack of a common understanding in support of development, deployment and operation tasks related to Virtual Function Networks (VNFs), NFV components and interfaces. In the current state of affairs, NFV stakeholders commonly create their own terminology to define and describe NFV components, following going the specifications led by European Telecommunications Standard Institute but also adopting telecommunication- and software-centric definitions. As a consequence, portability and interoperability goals of NFV get compromised since NFV technology providers have hard times in understanding and using definitions and descriptions across different domains. Furthermore, VNF data models of operational systems and deployment configuration software need to be re-defined, re-coded, and re-compiled to make them work over different NFV platforms. In this work, we present the design and implementation of our proposed NFV Ontology (NOn) enabling Semantic nFV Services (SnS) to reduce manual intervention during the integration process of heterogeneous NFV domains and effectively overcome the costly re-work hazards of current NFV implementation approaches. We present the proof of concept implementation of a Generic Client leveraging SnS/NOn to create and consume dynamic workflows in an open source testbed based on OpenStack and OpenBaton.**

Keywords: Network Function Virtualization, NFV, Semantic Services, Ontology.

I. Introduction

NFV [1] arises as a networking technology trend aiming at changing the current physical appliance model to a software-based approach using standard hardware technologies to deliver networking services. As today, a number set of specifications and guidelines [2] are available defining NFV architectural views and the functional description of the main components and their interaction (e.g., reference points, interfaces). As usual in standard developments, the specifications are meant to be read, interpreted, and implemented by human developers, and hence allow a high degree of freedom on the semantics used to develop NFV elements. As consequence, we encounter heterogeneous manners to express the same components and lack of common understanding across NFV domains. Moreover, interoperability among NFV components is still an open challenge generally approached by using Web Service (WS) [3] relying on implicit service descriptions with diverse semantics. Furthermore, service integration requires costly and error-prone manual intervention throughout 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 paper proposes the use of a common domain language to describe NFV components and to avoid manual intervention process through an automatic service integration by means of two cornerstones: 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 application of the Semantic Services [4] 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 software entity capable of reasoning by means of an inference engine that allows to create and consume dynamic WS workflows. 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. Finally a proof of concept was implemented in order to validate the potential of the proposed NOn and SnS approaches to realize NFV.

II. Problem Review

An important requirement of DevOps is the possibility of interactions between autonomous machine and human users to verify and validate services integration and troubleshooting via common Application Programming Interfaces (APIs) [5]. Generally, APIs have implicit service description in a syntax manner. For example, Web Semantic relies on the use of Resource Description Framework (RDF) and ontological representations of real world to change the manner of how the web works today. Some semantics (e.g., WSMO or OWL-S) create services based on representations and explicit descriptions but fall short of native service description, automatic discovery and interoperability [6]. Recently, RESTdesc [7] has been proposed to describe service functionality allowing the creation of automated services based on ontologies replacing variable declaration to describe functionality. As a result,

978-1-5090-4758-1/16//$31.00

TABLE I
VNF DESCRIPTOR (VNFD) BASE INFORMATION ELEMENTS

Identifer	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.
connection _point	Element	1...N	This element describes an external interface exposed by this VNF enabling connection with a Virtual Link.

RESTdesc delivers native functional and explicit description of services and automated discovery.

The Semantic Web principles using ontologies to describe infrastructure and networking resources has been a recent trend with remarkable examples such as Network MarkUp Language [8] (NML) and Infrastructure and Network Descriptor Language [9] (INDL). These efforts attempt to standardize terminologies and concepts around networking, computing and storage entities. While these languages have been used to model and store information of the Service Provider (SP) other key features of Web Semantic approaches such the use of semantic services have not been exploited yet. Another current weakness is the need of resource discovery methods and external components to synchronize, translate and abstract SP current data information model into the semantic approach.

Focusing now on NFV, at least three problems can be identified as technology integration hazards: (*i*) absence of a common understanding (i.e. shared vision) around NFV), (*ii*) lack of well defined semantics (i.e. domain specific language), and (*iii*) need of manual intervention to interpret, use, and integrate components. As today, 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 interface and abstractions [10] provides a guideline for developers, the document is subject to interpretation and by any means interpretable by software services and components. As a consequence, problems inherent to interface integration negatively affects NFV implementations by increasing development time and costs when attempting NFV service discovery and interoperability.

III. NFV ONTOLOGY (NON)

We propose NOn to provide a common knowledge and language across NFV domains useful for all stakeholders such as SP, Network Operator (NO), and developers. The main goal of NOn is to reduce integration costs caused by the current semantic diversity of NFV descriptors and WS implementations.

A. Designing NOn

The design of NOn is based on European Telecommunications Standard Institute (ETSI) specifications [2] [11] [12] [13] and follows the principles of *Ontology Development 101: A Guide to Creating Your First Ontology* [14]. Some steps were followed in a more rigorously way than others. Considering ontology design an iterative process, our design work went back and forward through the steps to improve the model.

Table I shows three base information elements of a VNF descriptor according to [2]. The first column is used to abstract and name components on the ontology. The Type column defines components as a resource or a data type variable (Element and Leaf respectively). The cardinality column stores the number for components (slot facet), and finally, the description field describes properties and relationships among elements.

After modeling the VNFD, we moved our attention to components defined as resources. Table II shows the element definitions given for Virtual Device Unit (VDU) components. The process was repeated until all components and relationships were defined resulting in the NOn ontology.

Figure 1 presents the resulting graph as an example of modeling elements and relationships corresponding to the abstractions of Table II (top) and I (bottom).

TABLE II
VNFD:VDU BASE ELEMENTS

Identifer	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.

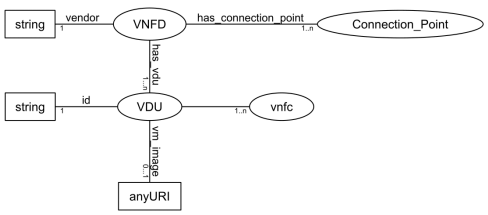

Fig. 1. Extending NOn

B. NOn Example: Semantic VNFD

Aiming to exemplify NOn with real NFV implementations, two VNFD descriptors from different NFV implementations were parsed into NOn VNFD instances. We used the Proteg modeling tool[1] to develop the classes and sub-classes of the ontology ind addition to Data and Object properties (slots). Instances of the VNFD element were created using different descriptor models from a NFV implementation, resulting in

[1]http://protege.stanford.edu/

a semantic VNFD template following ETSI specification and ready for a descriptor instance creation.

We used descriptors from two NFV open source implementations (OpenMano[15] and OpenBaton[16]) and tried to match their components into the ontology. Interestingly, during the parsing process, we observed common elements included in both descriptors (e.g., lifecycle events or name) but not defined in the ETSI specifications. Likewise, we discovered some elements not present in the first version of the ontology but defined in ETSI documents, an opportunity we used to enhance the NOn model in spirit of our iterative approach. Listing 1 shows some features modeled in the OpenBaton VNFD descriptor file matching elements defined in Fig. 1.

Listing 1. OpenBaton Semantic VNFD File

```
### non:#ob-iperf-client
non:ob-iperf-client rdf:type owl:
    NamedIndividual,
non:vnfd;
non:has_vdu non:ob_iperf_client_vdu;
###  non:#ob_iperf_client_vdu
non:ob_iperf_client_vdu rdf:type owl:
    NamedIndividual,
non:vdu;
non:vm_image "iperf_client_image"^^xsd:
    anyURI;
non:has_vnfc non:ob_iperf_client_vnfc.
```

As a result of parsing both descriptors we found the current OpenBaton descriptor model a better option to validate our NOn model, which can be explained by OpenBaton defining its components (such descriptors) closely following ETSI specifications whereas OpenMano provides a solution that makes more use of proprietary syntax.

IV. Semantic nFV Services

SnS is our proposed concept to add semantic service descriptions to NFV Web Service interfaces and APIs in order to reduce the need of manual intervention when integrating services and to improve overall interoperability.

A. Creating Semantic Services

In spirit of software and knowledge reuse, SnS leverages recent software technologies (Representational State Transfer (REST) and Notation 3 (N3) language. More specifically, We opted for RESTdesc [17] as the key enabling technology to create the semantic services. Fundamentally, RESTdesc blends together existent ontologies (e.g. NOn) and already deployed REST services. Furthermore, its implementation does not require modifications in the service capabilities and methods, instead RESTdesc provides a mechanism to describe them.

To introduce semantic technology smart agents and inference engines (*reasoner*) are necessary to consume service descriptions. We argue that the extra components needed during the service deployment process and development efforts of the semantic descriptions are compensated by the cost reduction due to manual intervention, training and hazards of services integration.

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

1) Adding Descriptions to Services: in order to add semantic descriptions for NFV services, a REST Web Service was developed with the ability to generate VNF deployment files (e.g VNFD) for OpenMano and OpenBaton. Initially, two different services were created:

- A service using the Hypertext Transfer Protocol (HTTP) GET Method (Listing 2) to retrieve a JavaScript Object Notation (JSON) file with the corresponding VNFD OpenBaton format.
- A service with HTTP GET Method to retrieve a YAML Ain't Another Markup Language (YAML) file with the associated metadata for the OpenBaton VNFD.

Both services receive VNF deployment parameters as inputs (e.g. `vm_image`) contained in the retrieved file.

Listing 2. OpenBaton VNFD WS

```
GET /nfv/parser/openbaton/vnf/vnfd?vendor    1
    ="value"& name="value"& type="value"&
    endpoint="value"&vim_instance="value"
    vm_image="value"&minCPU="value"&minBW=
    "value"&minRAM="value"&dev_flavour="
    value"
Host: localhost:8080                         2
Content-Type: application/json               3
```

Listing 2 shows the resulting call to the first service using HTTP. Line 1 represents the service method, the service Universal Resource Identifier (URI) and the query parameters to construct the file. Lines 2 and 3 are Host IP and the type of the retrieving file, respectively. While a human with adequate software development and NFV background could reasoning and interpret some parameters (e.g., `vendor` or `minCPU`), other developers could easily misunderstand the parameters and not be able to consume the service as expected.

To circumvent manual intervention, the semantic service implementation describes the service method, content type and URI based on the HTTP ontology while services parameters using NOn ontology.

Listing 3. OpenBaton VNFD Semantic Service Description

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

Listing 3 presents the resulting service description[2] of OpenBaton VNFD WS. The RESTdesc description uses N3 and can be divided in three parts:

Precondition: Lines 1 to 10 represent a precondition that must be accomplish to consume the service. The NOn ontology is used to do variable declaration. In N3 language, it is necessary to pass ontology elements to defined variables in the descriptor in order to be used at execution time. For example `non:vnfd` passes all elements contained in the VNFD semantic file 1 to the `?vnfd` variable. Question mark (**?**) represents the variable declaration in N3.

Process: the actions that must be executed if preconditions are accomplished. For example a HTTP request.

Postcondition: final state reached once the process is executed, e.g. VNFD created.

The process of generating service descriptions is not an easy task and attention needs to be paid to write properties explicit to an object. The description above can be read as: *if* there exist an object that is an `non:vnfd` an has all the parameters below (`non:vdu`, `non:vld`, etc.), *then* execute the HTTP call and set the variable VNF with the associated VNFD.

2) Consuming Semantic Services: to consume SnS a service client is needed capable of adapting as required to consume different types of REST requests. We developed a JAVA based client named as *Generic Client* without predefined parameters to consume services (e.g headers, methods).

Generic Client uses an external inference engine to do reasoning (Euler Yet another proof Engine (EYE) [18]) as follows: (*i*) receives rules (service descriptor) and a context (Semantic VNFD) from the *Generic Client*, and (*ii*) derives a workflow with the inferred REST request. It is important to denote that NFV service descriptions would not be possible without NOn.

Listing 4. 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.
```

The code in listing 4 shows the inference engine response, which can be read as: HTTP GET request (called `_:sk0`) to `/nfv/parser/openbaton/vnf/vnfd` URI exists; has header type `application/json`, and response is `_:sk1` with a representation of a `json:openbaton_vnfd` file.

Using the resulting workflow at run-time, the *Generic Client* builds the request using the `http:methodName` and `http:requestURI` (lines 1 and 3 respectively) as fixed parameters. Other parameters are filled as *if-then* variables.

[2]Due to space constraints, some of the implemented are not presented in the listing, e.g., ontology parameter `@prefix`.

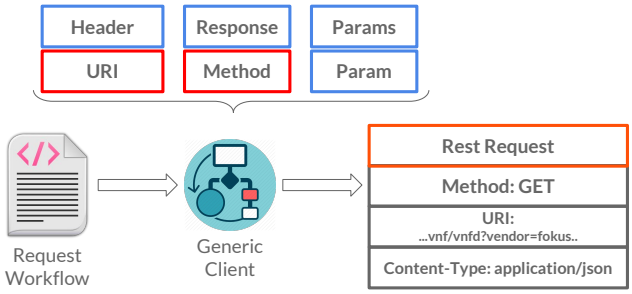

Fig. 2. Creating a Dynamic WS request.

For example, *if* there exists a `http:MessageHeader` *then* put the header in the request.

Figure 2 illustrates the request creation process. After the *Generic Client* reads the workflow file, the client puts URI and method parameters in the JAVA request. Then, the client makes a match between parameters in the file and the parameters belonging to the HTTP request according to the method at hand (e.g., POST method may have a body file and GET methods not). `IF` a parameter is in the file, `THEN` it is added to the JAVA request. This way, the client adapts itself according to the parameters included in the descriptions and the subsequent inference workflow gets predefined.

To wrap up, we present the *Generic Client* as a flexible REST client that is created once but capable of consuming multiple, dynamically defined REST services.

B. SnS Workflow Inference

We now focus on the wider inference workflow as the overarching process of making a plan with an ordered sequence of HTTP requests in order to accomplish a specific goal.

1) Creating Dynamic Workflow: inference engines have the ability of inferring context and deducing facts from a given knowledge base. Thus, these kind of engines can be used to create a plan (workflow) with a sequence of HTTP requests from the inferred context to get a conclusion. In our case, we are interested in the engine's ability to create a plan to reach a specific goal[3].

A goal-based workflow is used to create the necessary context for a specific task, i.e., the inference engine makes a plan to achieve the proposed goal. Besides knowledge inputs (e.g. service descriptions), the inference engine needs to know the objective to be accomplished. In this project, we opt for a *backwards inference engine* in order to create the plan, the engine starts assuming the fact that the "goal proposed was accomplished". It then starts to take proofs to support this fact based on the possibles services to be consumed to reach the goal. This is done through the analysis of the postcondition section of the service descriptions. When a service with the proper post condition is found, the inference engine starts to find if the preconditions can be achieved using other services

[3]An objective/goal is the desired state of a resource or a service to be accomplish

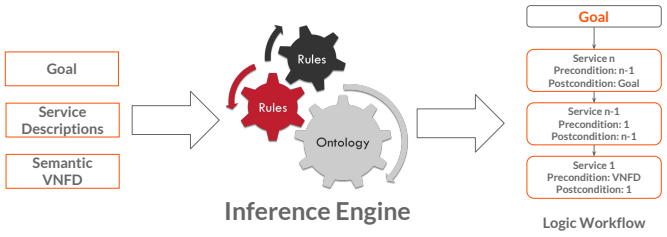

Fig. 3. Creating Goal Based Workflow

descriptions or data inputs (e.g. VNFD file). If preconditions can be achieved, the service is put into the plan. Some of the service preconditions may be a postcondition of another service and the inference engine must go backwards in order to analyze if preconditions can be achieved. If so, the service is added to the plan and the same process is done to find the proof of the goal. Hence, the inference engine goes backwards creating the plan and chaining services needed to accomplish the desired goal. The backwards process goes until there are no more proofs to be done.

Figure 3 illustrates the process in order to accomplish a goal, the inference engine provides a backwards service plan. The plans starts with the last service to be consumed and goes down until the first service is consumed. We can see how postconditions of some services are others preconditions hence opening possibility of service planning, it is worth to note that one of the inputs of the engine must be the predefined goal.

2) Consuming Dynamic Workflows: while the *Generic Client* has the ability of self-adapting to dynamically create REST requests, service consumption is done one by one, i.e., only one SnS is received and consumed at a time. The challenge becomes now allowing the client to read and directly interpret a workflow process given by the inference engine. Since a workflow includes REST requests along lemmas to proof facts, it is necessary for the client to find a manner to process the workflow and remove unnecessary data without affecting the inferred REST requests and the sequence to be consumed (see Fig. 3).

The proposed interaction between the *Generic Client* and the inference engine is as follows. Firstly, the client takes the service descriptions and the semantic VNFD. Secondly, it parses the goal to be achieved among the inputs which are passed to the inference engine. Thirdly, the inference engine responds with a deduced workflow, and, finally, the *Generic Client* interprets and executes the plan.

V. Proof Of Concept Experiment

Figure 4 illustrates the experimental scenario to validate our Proof of Concept implementation. The different NFV components are installed and configured on different servers at the Information & Networking Technologies Research & Innovation Group (INTRIG) facilities. The NFV Infrastructure (NFVI)/Virtualized Infrastructure Manager (VIM) is based on OpenStack while OpenBaton VNFD and MetaData WS were

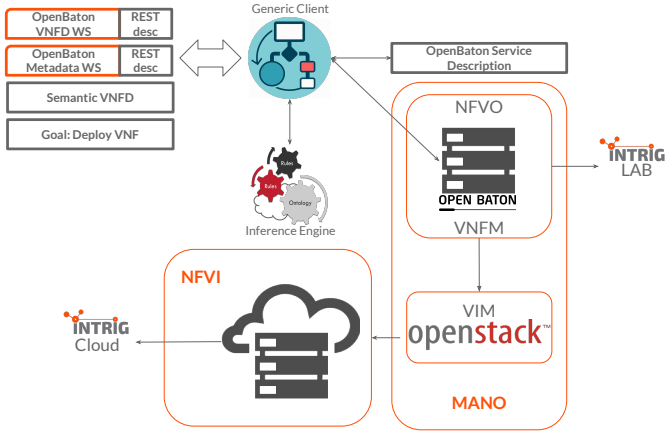

Fig. 4. Experimental Scenario of SnS.

installed along the server running the *Generic Client* which communicates with the online inference engine [19] using RESTdesc public service descriptions available in our GitHub repository. [4]

Listing 5. OpenBaton Deploy VNF Semantic Service Goal

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

Functional tests were carried to verify the VNF deployment process first using a manual process following OpenBaton tutorial guidelines, and then by means of SnS for the same deployment process as follows:

1) User creates Semantic VNFD and Goal (Listing 5).
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 pass it to *Generic Client*.
4) Client process the answer and creates a request file.
5) Client consumes the OpenBaton VNFD WS and creates the JSON file to retrieved.
6) Client consumes the OpenBaton Metadata WS and creates the YAML file to be retrieved.
7) Client consumes the OpenBaton Deploy VNF WS:
   - Creates VNF package using VNFD and Metadata files.
   - Upload VNF package via the OpenBaton WS.
8) OpenBaton Network Function Virtualization Orchestrator (NFVO) deploys the VNF over INTRIG's cloud using OpenStack APIs.
9) Finally, OpenBaton returns the `id` given for the deployed package. VIM.

Each file was successfully deployed in separately manner. Both VNF were deploy without creating clients for each service, instead *Generic Client* and inference engine do all the process taking as a start point service descriptions, semantic VNFD and goal defined.

[4]https://github.com/intrig-unicamp/sns_sermantic_nfv_services

VI. Conclusion and Future Work

This papers presents an approach to leverage semantic technologies towards interoperability and model-based software automation in the context of Network Function Virtualization technology. We explore the applicability of the the Web Semantic approach to interfaces of NFV by leveraging a common representation of the concepts and data models provided by the proposed NFV Ontology (NOn), which is reusable across domains and allows to create meaningful descriptors for semantic service implementation (Semantic nFV Services). Our work does not remain just at a theoretical level –in which almost anything is plausible– but presents the proof of concept implementations combining state of the art technologies of the Semantic Web (e.g., RESTdesc, N3), cloud infrastructure (e.g. OpenStack), and NFV management and orchestration (e.g., OpenMano, OpenBaton) to assess the practicality and promised benefits of automatic Web Service integration.

Along our journey, we encountered multiple issues related to interoperability due to diverse usage and interpretation of NFV concepts and their software implementation. Our attempt to address the resulting implementation and integration challenges is based on adopting a common and semantically meaningful NFV data model (NOn). However, when brought into open source NFV projects (e.g., OpenMano, OpenBaton), we realized limitations caused by their actual adherence to ETSI specifications. Projects closely following the glsetsi information models defined, which are not (yet) fully consistent across different documents, results in better possibilities of realizing multi-domain, distributed NFV scenarios [20] based on heterogeneous implementations without using falling into costly human intervention.

Concepts behind NOn and SnS can be regarded as initial steps towards the vision of automatic service integration in NFV and a real, fruitful DevOps environment. Related efforts are going on based on TOSCA[5] and NETCONF/YANG data models, neither of which are being extended so far to embrace semantic principles but are still regarded as relevant technologies for NFV management and orchestration.

We now move our attention to some gaps and limitations we intend to address in future work. In its current development state, NOn does not fulfill all requirements of being a (universal) data model capable of describing all the different VNFD. Furthermore, there are other types of descriptors and components on NFV that need to go through the abstraction and design process to become an ontology and increase the overall system interoperability. We did not explore the full potential of a native semantic approach to build components like the VIM or NFVO. Such implementations can be done using NOn to define variables and components and avoid ambiguity across implementations and allow efficient reuse of code and design principles, altogether yielding higher levels of interoperability and DevOps productivity, critical success factors in the trend of network softwarization.

[5]https://docs.oasis-open.org/tosca/tosca-nfv/

Acknowledgment

This work was supported by the Innovation Center, Ericsson Telecomunicações S.A., Brazil. We also thank Drthe Arndt, Ruben Verborgh, Jos De Roo and fellows at the Data Science Lab, Ghent University, Belgium, for their technical support on EYE and RESTdesc. Prof. Dr Luciano de Paula for his review considerations and suggestions to improve this work.

References

[1] ETSI, "Network Functions Virtualisation - White Paper #1," 10 2012. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper1.pdf

[2] ETSI GS NFV-MAN, "NFV Management and Orchestration," 12 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

[3] ETSI, "Etsi network function virtualisation enters Phase 2," 2014. [Online]. Available: http://www.etsi.org/index.php/news-events/news/850-2014-12-news-etsi-network-function-virtualization-enters-phase-2

[4] H. Alesso and C. F. Smith, *Developing Semantic Web Services*. A K Peters/CRC Press, 2004.

[5] J. Kim, C. Meirosu, I. Papafili, R. Steinert, S. Sharma, F.-J. Westphal, M. Kind, A. Shukla, F. Nemeth, and A. Manzalini, "Service provider devops for large scale modern network services," in *IFIP/IEEE IM*, May 2015, pp. 1391–1397.

[6] L. A. Kamaruddin, J. Shen, and G. Beydoun, "Evaluating usage of wsmo and owl-s in semantic web services," in *Proceedings of APCCM '12*, 2012, pp. 53–58. [Online]. Available: http://dl.acm.org/citation.cfm?id=2523782.2523790

[7] R. Verborgh, T. Steiner, D. Van Deursen, J. De Roo, R. Van de Walle, and J. G. Vallés, "Capturing the functionality of web services with functional descriptions," *Multimedia tools and applications*, vol. 64, no. 2, pp. 365–387, 2013.

[8] J. van der Ham, F. Dijkstra, R. Łapacz, and A. Brown, "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*, 2013.

[9] M. Ghijsen, J. Van Der Ham, P. Grosso, C. Dumitru, H. Zhu, Z. Zhao, and C. De Laat, "A semantic-web approach for modeling computing infrastructures," *Computers & Electrical Engineering*, vol. 39, no. 8, pp. 2553–2565, 2013.

[10] ETSI GS NFV-INF, "Methodology to describe Interfaces and Abstractions," 12 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/007/01.01.01_60/gs_NFV-INF007v010101p.pdf

[11] ETSI GS NFV, "Architectural Framework," 12 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf

[12] ETSI GS NFV-SWA, "Virtual Network Functions Architecture," 12 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_NFV-SWA001v010101p.pdf

[13] ETSI, "Network Functions Virtualisation - White Paper #3," 10 2014. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf

[14] N. F. Noy, D. L. McGuinness *et al.*, "Ontology development 101: A guide to creating your first ontology," 2001.

[15] OpenMano. (2014) Openmano project. [Online]. Available: https://github.com/nfvlabs/openmano

[16] OpenBaton. (2014) OpenBaton. [Online]. Available: http://openbaton.github.io/

[17] RESTdesc. (2011) RESTdesc Semantic descriptions for hypermedia APIs. [Online]. Available: http://restdesc.org/

[18] Jos De Roo. (2009) Euler Yet another proof Engine - EYE. [Online]. Available: http://eulersharp.sourceforge.net/

[19] R. Verborgh and J. D. Roo. (2012) Eye Public Reasoner. [Online]. Available: http://eulersharp.sourceforge.net/

[20] R. V. Rosa, M. A. S. Santos, and C. E. Rothenberg, "Md2-nfv: The case for multi-domain distributed network functions virtualization," *2015 International Conference and Workshops on Networked Systems (NetSys)*, vol. 00, no. undefined, pp. 1–5, 2015.