

Learning a Configurable Deployment Descriptors Model in NFV

Wassim Sellil Atoui^{1,3}, Nour Assy², Walid Gaaloul³ and Imen Grida Ben Yahia¹

¹Orange Labs, Châtillon, France

²International Lebanese University, Beirut, Lebanon

³Telecom SudParis, Samovar, Evry, France

{wassimsellil.atoui, imen.gridabenyahia}@orange.com, assy.n.nour@gmail.com, walid.gaaloul@mines-telecom.fr

Abstract—The deployment descriptors in Network Function Virtualization (NFV) are usually designed and configured through static automation and manual edition by service providers without any formal strategy except best practices. Thus, leading to an error prone and time consuming approach. We propose in this paper 1) a configurable deployment descriptor model and 2) a learning approach based on machine learning to construct the configurable model automatically. Firstly, the configurable deployment descriptor model captures the relation and also the variability between the VNF elements of different deployment descriptors. It enables service providers to configure and generate customized deployment descriptors instead of designing them each time from scratch. Secondly, we define a learning approach to learn configurable deployment descriptor models by finding and federating similar VNF elements of different deployment descriptors. With our machine learning approach we construct automatically a configurable model from a set of deployment descriptors. The results of our experiments highlight the effectiveness of our approach into learning configurable deployment descriptor models.

I. INTRODUCTION

Network Function Virtualization (NFV) is a paradigm that brings more flexibility and efficiency into the management of physical and virtual resources in the network. It offers the ability to abstract and automate the control of physical and virtual resources delivering the service in the network. Network functions are separated from their hardware appliances and are virtualized into building blocks that may chain together to create communication network services (NSs). Network functions in NFV are referred to as VNFs (Virtual Network Functions).

To onboard new VNFs or NSs in the NFV platform, the service provider package them with descriptor files containing all the information required for the deployment. The descriptors specify for a network service [1], the virtual network function (VNFs) that compose it, the topology of interconnecting the VNFs and the data flow direction between them. Also the deployment descriptors for the VNFs indicate the deployment and operational behavior in terms of connectivity, interface and virtualized resource requirements [2].

The descriptors are configuration files written often manually by domain experts in semi structured languages like XML, JSON and YAML. They are complex and tedious to design manually or with static automation as they contain numerous

components that are dependent on each other. Also, there is no formal strategy to assist the creation of the deployment descriptors except experience and best practices of domain experts. This leads to a plethora of strategies of deployments. Different domain experts may choose different configurations for deploying the same network service/function, which makes it even harder to learn from past deployment strategies and identify deployment characteristics related to a network service/function. Moreover, despite some efforts to find a consensus on a common information model to use for modeling the deployment descriptors like the ETSI NFV initiative [1], there is still an absence of a common data model for the deployment descriptors, which resulted on a diversity of solutions, such as: TOSCA, YANG, Hot, etc. This diversity obliges service designers to constantly adapt the descriptors to the heterogeneous NFV platforms in order to enable the integration of the network services/functions. Therefore, designing the deployment descriptors for the network services/functions in such a highly dynamic environment, becomes a highly complex, time-consuming and tedious task.

In this work, we propose a configurable deployment descriptor model that capitalizes on a catalogue of deployment descriptors to help the service provider designing and generating NFV deployment descriptors. It merges similar component elements into a single model along with configurable operators that captures the configuration variability. We focus our attention more particularly on the Virtual Network Function Descriptors (VNFD) as they are the main component of NFV that compose each network service. The proposed model learns from past experience, gained through NFV deployment descriptors to extract useful and implicit knowledge that helps network service/function providers with decision making on designing the descriptors. The model is basically a tree structured graph that represents the component elements that appear in the local deployment descriptors along with configurable connectors that capture different structures depending on the suitable configuration. The model captures best practices in creating deployment descriptors in a specific domain and can be used as a “plug and play” model that is easily configured by service providers.

Learning configurable VNFD models is a challenging task. That is to find and group together similar elements in the set of VNFD files. It is challenging because the VNFDs could

be designed by different service providers or domain experts. Therefore, similar VNFD elements with similar functionalities could be labeled or structured differently. The configuration could also differ between the elements. To overcome this problem, we propose an algorithm based on machine learning to search and cluster similar VNFD elements based on the distance between the features of the elements. We define what are the features that characterize the elements and also the metrics that measure the similarity between the elements.

The remainder of the paper is organized as follows: Section 2, discusses the related work. Section 3, defines a representation for the VNFDs. Section 4, presents our proposed configurable descriptor model. Section 5, describe our machine learning based approach to learn automatically the configurable model. Section 6, presents the results of our experiments on our proposed solution and section 8 conclude the paper.

II. RELATED WORKS

Recent surveys on the current challenges in NFV orchestration have indicated that the generation of deployment descriptors is a challenging problem that needs to be tackled in order to enable a full automation of network configuration and allow more agility in the orchestration [3] [4]. Indeed, the orchestrator needs to adapt to different conditions of the network by adjusting the deployment resources while preserving the requirements of the network service/function providers.

Unfortunately, the problem of automating the generation of deployment descriptors in NFV has received only few attentions in the literature. In [5], the authors proposed a tool and a methodology to aid the NFV designer to rapidly design and on-board new services and applications. In [6], a characterization approach based on machine learning is proposed to identify the most appropriate types of resources to be allocated to a VNF at deployment time. Also in [7], an approach based on machine learning is considered to estimate VNFs needs in term of resource requirements from descriptors. [8], proposed a VNFD generator web application to automatically create a VNFD template that the operators and VNF vendors can quickly validate and deploy. The authors in [9] provided a detailed view on the stages that a network slice provider must follow to deploy the ordered network slice instance, accommodating it into a multi-domain infrastructure, and putting it operative for tenant's consumption. In our previous works [10], [11], we proposed a deep learning approach that learns a model from a catalogue of deployment descriptors that recommends and completes deployment descriptor. Despite those efforts, there is still no formal methodology that allows the service providers to capitalize on past NFV deployments. To the best of our knowledge, our approach to learn from deployment descriptors using a configurable deployment descriptor model has not been yet investigated in the context of NFV.

Our proposed solution is inspired from the configurable process models in Business Process Management (BPM)[12]. Designing high-quality process models is often time-consuming,

error-prone, and costly. The tasks in BPM need to be ordered as well as their execution plan, i.e. whether the tasks are executed in parallel or are alternatives to each. Configurable process models significantly reduce process modeling costs by allowing the reuse of existing process models and enabling their adaptation for different needs. [13]. Our solution is more closely related to the work introduced by Rosemann et al. [14], [15]. The authors propose a Configurable EPC (C-EPC) notation which extends EPC with configurable elements in order to improve the configurability of Enterprise systems and reference models such as SAP R/3 reference model. Basically, the EPC notation consists of three main control-flow elements: *event*, *function* and *gateway*. An event can be seen as a pre- and/or post-condition that triggers a function. A function describes the kind of work which must be done. Three types of connectors, OR, exclusive OR (XOR) and AND are used to model the splits and joins. C-EPC adds two constructs to the EPC language: *configurable functions* and *configurable gateways*. A configurable function has two configuration alternatives: *ON* and *OFF*. A configuration *ON* means that the function is included in the process variant while a configuration *OFF* means that it is excluded. A configurable connector has configuration alternatives that are of equal or less restrictive behavior. For example, a configurable *OR* has the configuration alternatives: *OR* (no restriction is applied), *XOR* and *AND* with possibly restricted outgoing/incoming branches.

We propose a graphical representation of a VNFD model that consists of component instances, relations and gateways. Our proposed composition, allocation and connection gateways are different than those in the BPM domain (i.e. OR, XOR and AND gateways). The configurable VNFD model contains configurable gateways that have OR and XOR operators. The behavior of these operators is similar to that of BPM gateways. However, their configuration is different.

III. MODEL OF THE VIRTUAL NETWORK FUNCTION DESCRIPTOR (VNFD)

VNFD is a configuration file that describes the NFV infrastructure resource requirements for a VNF in a service provider environment and the operational behavior of the VNF including lifecycle events (e.g. scaling, upgrading).

The ETSI NFV [1] has released a specification that defines the requirements for the structure and format of a VNFD. The VNFD is composed of one or many virtual deployment units (VDUs) that describe the deployment resources and operation behavior of a VNF component (VNFC). VDUs are virtual machines that host the VNF or parts of it. Each part of the VNF is a VNFC and can be deployed on one or more VDUs. Each VDU is characterized by, among others, the software image loaded on it and the resources needed to deploy it. A VDU describes mainly the virtual compute (VC), virtual storage (VS) and virtual memory (VM) resources that are necessary for deploying a VNFC and it could be linked via connection points (CPD) to other VDUs or to external VDUs that belong

to other VNFs via external CPD. Virtual links in the VNFD indicate how the VDUs are connected and via which CPD.

The scope of this paper is to propose a configurable model for describing the deployment requirements of a VNF and an approach to learn it. Therefore, in the following, we propose a formal definition of a VNFD model representation in terms of structure and deployment requirements. However, our definition is generic and can be extended to include the operational behavior perspective.

A. VNFD representation

As the structure of a VNFD can be mapped to a graph, we choose graph theory to represent a VNFD model as a tree-like structure. For sake of simplicity, we restrict our definition to the general elements that compose a VNFD regardless of any data model. We consider for that matter the ETSI NFV specification [1].

Fig. 2 and Fig. 3 illustrate two basic examples of a VNFD representation of a firewall and CPE respectively. Fig. 1(a) shows the basic elements that compose the VNFD model representation. The nodes in the VNFD model represent the different component instances that compose the VNFD. Each component instance has one of the following types: *VNFC*, *VDU*, *VM*, *VS*, *VC* or *CPD* and may have one or more attributes. The root node represents an instance of the type VNFD. Fig. 2 presents an example where the root node v_1 is an instance of type VNFD and has the attribute “name” which has the value “vFirewall”. Graphically, we use “:” in the node label to indicate the component from which the instance is derived. For sake of simplicity, in our example, we define one attribute for each node and we indicate it between “()” in the node.

The relations between the nodes (component instances) can be one of the following three types: *composition*, *allocation* or *connection*. A composition relation between two nodes v_1 and v_2 indicates that the instance v_1 is composed by the instance v_2 . That means that v_2 is part of v_1 and cannot exist without it. The composition relation exists typically between (i) VNFDs and VNFs, (ii) VNFCs and VDUs and (iii) VDUs and CPDs. Graphically, a 1:1 composition relation is represented by a solid line. For example, in Fig. 2, the instance of VDU v_4 is composed of the instance of CPD v_7 . In case a 1:n composition relation exists between multiple nodes, we use a *composition gateway* to connect the nodes. Graphically, the composition gateway is represented by a grey diamond with the label “C”. In our example, the instance of VNFC v_2 is composed of two instances: VDU v_3 and VDU v_4 .

An allocation relation between two nodes v_1 and v_2 indicates that v_2 is a resource that needs to be allocated to v_1 . The allocation relation can only exist between VDUs on the one hand and VS, VC and VM on the other hand, i.e. only the virtual storage, virtual memory and virtual component resources can be allocated to the VDUs. Graphically, the allocation relation is represented by a dashed line. As an example, in Fig. 2, the VS v_5 is allocated to the VDU v_3 . In case multiple resources are allocated to a VDU (i.e. 1:n

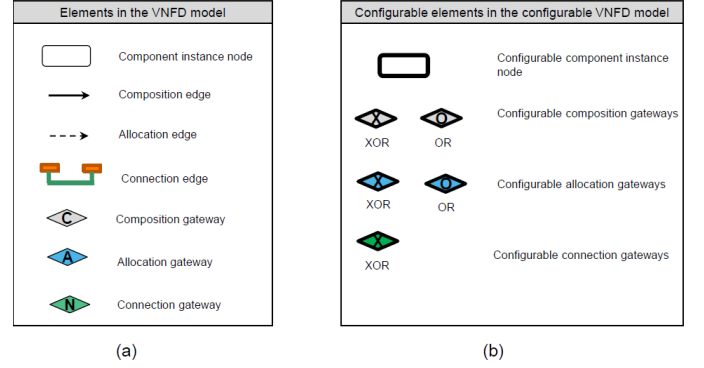


Fig. 1. (a) graphical representation of the elements used in a VNFD model and (b) graphical representation of the configurable elements in a configurable VNFD model

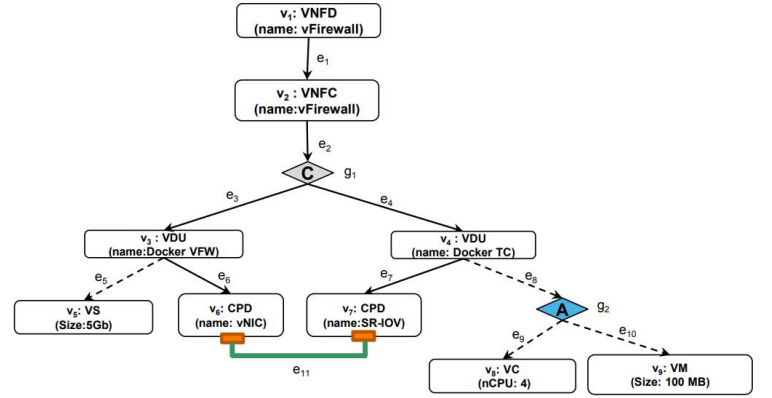
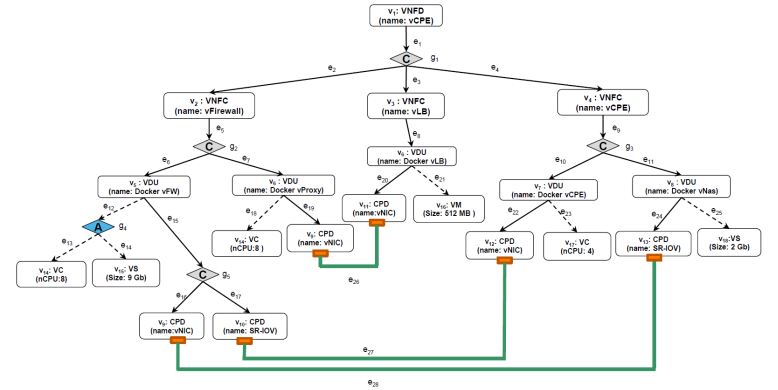


Fig. 2. An example of a vFireWall VNFD represented graphically as a tree-like structure



instances of type CPD and is a bidirectional relation. It represents the virtual links that connect the CPDs of the internal VDUs together or to external VDUs of external VNFs. Graphically, a connection relation is represented by a solid green line. As an example, in Fig. 2, the relation between the CPD v_6 and the CPD v_7 indicates that a virtual link connects the two connection points. We also introduce the *connection gateway* (as shown in Fig. 1) that allows to connect multiple connection points together. The connection gateway is graphically represented by a green diamond with the label “N”.

B. Formal Definition of a VNFD

Let $\mathcal{U}_A$ be the universe of attributes’ names and $\mathcal{U}_V$ be the universe of attributes’ values. The definition of a VNFD model is given in Definition 1.

Definition 1 (VNFD model):

A VNFD model is a tuple $D = (V, \hat{V}, G, \hat{G}, E, \hat{E}, \lambda)$ where:

- $V = \{v_1, v_2, \dots, v_n\}$ is the set of nodes;
- $\hat{V} : V \rightarrow \{VNFD, VNFC, VDU, VS, VM, VC, CPD\}$ is a function that assigns a component type to a node $v \in V$;
- G is the set of gateways;
- $\hat{G} : G \rightarrow \{C, A, N\}$ is a function that assigns a type to a gateway $g \in G$ such that:
 - $\hat{G}(g) = C$ indicates a composition gateway,
 - $\hat{G}(g) = A$ indicates an allocation gateway and
 - $\hat{G}(g) = N$ indicates a connection gateway.
- $E \subset (V \cup G) \times (V \cup G)$ is the set of edges;
- $\hat{E} : E \rightarrow \{composition, allocation, connection\}$ is a function that assigns a type to an edge $e \in E$;
- $\lambda : V \cup E \rightarrow \mathcal{P}(\mathcal{U}_A \times \mathcal{U}_V)$ is a function that assigns for either a node $v \in V$ or an edge $e \in E$ a set of attributes and their corresponding values, i.e. for an element $m \in V \cup E$, $\lambda(m) = \{(attr_i, value_i) \mid i \geq 1\} \in \mathcal{P}(\mathcal{U}_V \times \mathcal{U}_A)$. We use the shorthand $\lambda_{attr}(m) = value$ to refer to the pair $(attr, value) \in \lambda(m)$.

In our example in Fig. 2, the definition of the VNFD model is as following: $V = \{v_1, \dots, v_9\}$; $\hat{V}(v_1) = VNFD$, $\hat{V}(v_2) = VNFC$ and so on; $G = \{g_1, g_2\}$; $\hat{G}(g_1) = C$, $\hat{G}(g_2) = A$; $E = \{e_1, \dots, e_{11}\}$; $\hat{E}(e_1) = composition$, $\hat{E}(e_5) = allocation$ and so on; $\lambda_{name}(v_1) = vFirewall$, $\lambda_{size}(v_5) = 5GB$ and so on.

IV. VNFD CONFIGURABLE MODEL

This section introduces the notion of a configurable VNFD model. The configurable representation of a VNFD aims to capture different predefined VNFD models into one customizable model. The component instances and gateways can be configurable in this model. A configurable element is graphically represented with a thick line. In addition to the elements shown in Fig. 1(a), Fig. 1(b) shows the configurable elements that can exist in a configurable VNFD. Fig. 4 illustrates an example of a configurable VNFD that represents a consolidation of the VNFDs in Fig 2 and 3. In the following, we explain in more details the configurable component instances (Section IV-A) and the configurable gateways (Section IV-B).

Afterwards, we formally define a configurable VNFD model (Section IV-C).

A. Configurable component instances

A configurable component instance can be configured in two ways. First, it can be configured to *ON* (i.e. the component instance is included in the VNFD variant that is derived from the configurable model) or *OFF* (i.e. the component instance is excluded from the VNFD variant). A component instance configured to OFF results in the deactivation of all the component instances that are linked to it. For example, in Fig. 4, the component instances v_3 and v_4 are configurable. If v_3 is configured to OFF, all the component instances v_3 , v_8 , v_{11} and v_{18} will be deactivated and later removed from the resulting VNFD variant.

Second, a configurable component instance can have configurable attributes. A configurable attribute is an attribute whose value can be adapted. Configurable attributes are graphically represented in red color. In our example in Fig. 4, the VS v_{16} has a configurable attribute “size” whose value is in the range $[5 - 9]GB$. This means that, during configuration, the size of the VS can be set to any number between 5GB and 9GB. However, not all attributes can be configurable. In fact, some attributes such as the identifier of a component instance are considered as a characteristic that describe the component and therefore cannot be configurable. In our proposed model, we consider the attribute “name” as a characteristic of a component instance.

A configurable attribute value can be either expressed as a *range* (for numerical values) or as a *discrete set* (for numerical or textual values). A range describes the possible values that the configurable attribute could have. It is an interval of the minimal and maximal values that are recommended to be set. It can be configured to a single numerical value that exists in it. A discrete set is used mainly with textual values to describe the possible finite set of values an attribute can have. It can be configured to a single value that is picked from the set. For example, in Fig. 4, the range is used for the configurable attribute “size” of the VS v_{12} . The discrete set can be used for instance if multiple images could be used to deploy a VDU. In this case, the attribute “image” of the VDU is configurable and its value can be defined as a discrete set that includes the different images.

B. Configurable gateways

The composition, allocation and connection gateways in the VNFD model allow to model 1:n relations between the component instances. In a configurable model, a configurable gateway allows to model a variable number of relations. That is, some of the gateway’s relations can be deactivated and therefore excluded from the derived VNFD variant. In order to guide the configuration of a gateway, we introduce two operators, *OR* and *XOR*, that define the behavior of a configurable gateway and constrain its possible configurations.

A configurable OR composition gateway that connects a component instance v_1 to a set of component instances

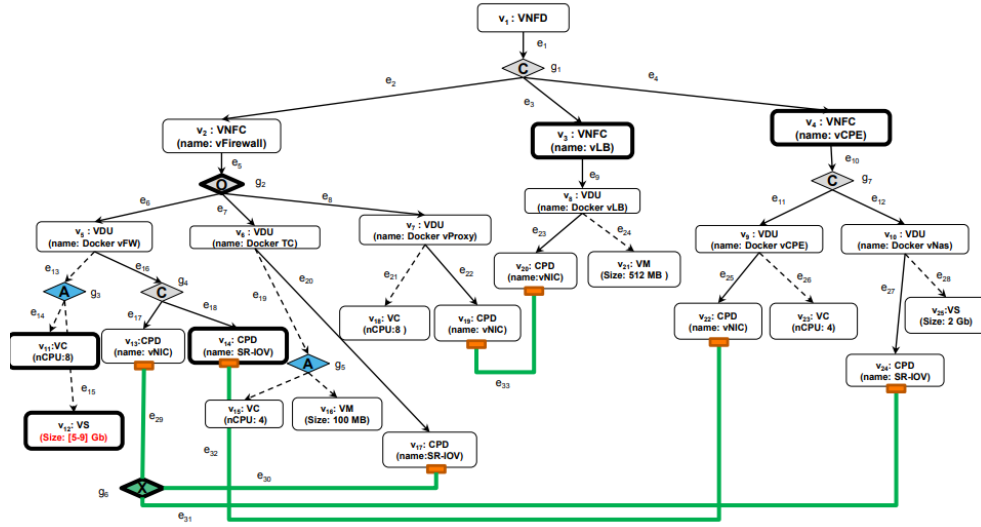


Fig. 4. An example of a configurable deployment descriptor model that combines the VNFDs in Figs. 2 and 3

$v_2, \dots, v_n$ indicates that v_1 can be composed of any combination of the component instances in the set $\{v_2, \dots, v_n\}$. In case a 1:1 composition relation is selected, we say that the OR composition gateway is configured to a composition edge. In case a 1:n composition relation is selected, we say that the OR composition gateway is configured to a composition gateway. Graphically, a configurable OR composition gateway is represented with a thick grey diamond and a circle inside. For example, in Fig. 4, g_1 is a configurable OR composition gateway. Therefore, the VNFC vFirewall can be composed of any combination of the three VDUs: Docker vFW, Docker TC and Docker vProxy. In case Docker vFW and Docker TC are selected, Docker vProxy with its allocated resources and CPDs are excluded and the configuration of the gateway becomes a composition gateway that connects vFirewall to Docker vFW and Docker TC.

A configurable XOR composition gateway indicates that at configuration time, a component instance can be composed of one and only one of the connected component instances. Therefore, the configuration of an XOR composition gateway is always a composition edge. Graphically, a configurable XOR composition gateway is represented with a thick grey diamond and a cross inside.

The behavior of the configurable allocation and connection gateways is similar to that of the composition gateway. Exceptionally, a configurable connection gateway does not have an OR operator since the connection relation is always a 1:1 relation. Graphically, the configurable allocation and connection gateways are represented similarly to the configurable composition gateway but with blue and green colors respectively.

In terms of behavior, the behavior of the XOR operator is included in that of the OR operator since an OR operator can be also configured to a 1:1 relation. However, we included the XOR operator to add more expressiveness to our configurable

model.

C. Formal definition of a configurable VNFD

In this section, we provide a formal definition of a configurable VNFD. let $\mathcal{U}_A$ and $\mathcal{U}_V$ be the universes of attributes' names and values; $\mathcal{U}_V^c$ is the universe of configurable attributes' values which are defined in terms of range and discrete set.

Definition 2 (Configurable VNFD):

A configurable VNFD is a tuple $D^c = (V, \hat{V}, G, \hat{G}, E, \hat{E}, \lambda, V_a^c, V_a^c, G^c, \hat{G}^c, \lambda^c)$ where:

- $V, \hat{V}, G, E, \hat{E}, \lambda$ are as specified in Definition 1;
- $\hat{G} : G \setminus G^c \rightarrow \{C, A, N\}$ is a function that assigns for a non-configurable gateway $g \in G \setminus G^c$ a type as specified in Definition 1;
- $V_a^c \subseteq V^c \subseteq V$ is the set of configurable nodes; V_a^c is the set of nodes whose attributes are configurable;
- $G^c \subset G$ is the set of configurable gateways;
- $\hat{G}^c : G^c \rightarrow (\{C, A\} \times \{OR, XOR\}) \cup \{(N, XOR)\}$ is a function that assigns a type for a configurable gateway $g^c \in G^c$ such that "C" indicates composition, "A" indicates allocation and "N" indicates connection; e.g. $\hat{G}^c(g^c) = (C, OR)$ indicates a configurable OR composition gateway.
- $\lambda^c : V_a^c \rightarrow \mathcal{P}(\mathcal{U}_A \times \mathcal{U}_V^c)$ is a function that assigns for a configurable node $v^c \in V_a^c$ a set of attributes whose corresponding values are configurable, i.e. $\lambda^c(v^c) = \{(attr_i, value_i^c) \mid i \geq 1, attr \in \mathcal{U}_A \wedge value^c \in \mathcal{U}_V^c\}$. We use the shorthand $\lambda_{attr}^c(v^c) = value^c$ to refer to the pair $(attr, value^c) \in \lambda^c(v^c)$.

In our example in Fig. 4, the definition of the configurable VNFD is as following: $V = \{v_1, \dots, v_{24}\}$; $\hat{V}(v_1) = VNFD$, $\hat{V}(v_3) = VNFC$ and so on; $G = \{g_1, \dots, g_7\}$; $\hat{G}(g_1) = C$ and so on; $E = \{e_1, \dots, e_{31}\}$; $\hat{E}(e_1) = composition$, $\hat{E}(e_{14}) = allocation$ and so on; $\lambda_{name}(v_2) = vFirewall$,

$\lambda_{nCPU}(v_{11}) = 8$ and so on; $V^c = \{v_3, v_4, v_{11}, v_{12}, v_{14}\}$; $V_a^c = \{v_{12}\}$; $G^c = \{g_2, g_6\}$; $\hat{G}^c(g_2) = (C, OR)$, $\hat{G}^c(g_6) = (C, XOR)$; $\lambda_{size}^c(v_{12}) = [5 - 9]Gb$.

V. LEARNING THE DEPLOYMENT CONFIGURABLE MODEL

In this section, we propose an approach based on machine learning that constructs automatically the configurable VNFD model. Initially, each VNFD file in the data set is represented by a tree-like structure mode (as described in section III). The configurable VNFD model is afterward constructed by federating similar nodes from different VNFD models into a single model.

We propose in this section an algorithm that searches automatically for similar nodes in the set of VNFDs models that could be federated together. It is not obvious to recognize similar nodes in different VNFD models, where each VNFD could be designed by different domain experts or service providers. Therefore, nodes with similar functionalities could have different labels, different attributes, different configurations and different relations to other nodes.

Our algorithm clusters together similar nodes in the set of VNFD models based on four characteristics that define each node: label, type, attributes and relations. Clearly, only nodes with similar type could be compared together. Thus, the node label, attributes and relations are the node main features that will be used to measure the similarity between nodes.

Before discussing the algorithm, we define first the metrics that will be used by the algorithm.

1) *Similarity metrics between the nodes*: The similarity between the nodes is measured in terms of their features. Let $\ddot{V} = \{\ddot{v}_1, \ddot{v}_2, \dots, \ddot{v}_{\ddot{N}}\}$, be the set that contains all the nodes that are extracted from the VNFD data models, $\ddot{N}$ is the number of all the nodes. The features of each node $x \in \ddot{V}$ are formally defined as the following:

A node x has a type denoted $x.type$, a label denoted $x.label$ and a set of attributes denoted $x.attributes$. The node type is a predefined notation that is specified in the data model of the VNFD. The label of the node is a string that is set arbitrary by the service provider. The attributes are a set of key/value pairs that characterize the configuration of the node (component instance). To represent the node's attributes, we define $att_x = \{a_1, a_2, \dots, a_A\}$, $x.attributes = att_x$ as the attributes vector of a node $x \in \ddot{V}$. att_x contains all the possible attributes that occur in the set of nodes $\ddot{V}$. The index i of $a_i \in Att_x$ represents a particular attribute key and its content is the value of that key. If an attribute $a_i \in Att_x$ does not exist in the node x , its value is then equal to 0. The qualitative attributes of the nodes are considered as categories. Each possible category is added as an individual attribute in att_x . It is equal to 1 if its value occurs in the node and equals to 0 otherwise. The quantitative attributes are in the other hand normalized so that their values are restricted between $[0, 1]$. We use for that the basic normalization formula:

$$a_i^x = \frac{a_i^x - \min_{\ddot{V}}(a_i)}{\max_{\ddot{V}}(a_i) - \min_{\ddot{V}}(a_i)} \quad (1)$$

where $a_i^x = a_i \in Att_x$, $\max_{\ddot{V}}(a_i)$ and $\min_{\ddot{V}}(a_i)$ are respectively the maximum and minimum values of a_i across all nodes attributes in $\ddot{V}$.

The similarity distance $dist(x, y)$, $x, y \in \ddot{V}$, measures how similar two nodes x and y are to each other based on the features. It is defined as follows:

$$dist(x, y) = (\omega_1 * sim_l) + (\omega_2 * sim_a) + (\omega_3 * sim_r) \quad (2)$$

where $sim_l = sim_l(x.label, y.label)$ is the similarity between the labels of x and y . $sim_a = sim_a(x.attributes, y.attributes)$ is the similarity between the attributes of x and y . sim_r is a similarity that measures how much relations x and y have in common. $\omega_1, \omega_2, \omega_3$ are weights between the similarities, such that $\sum_{i=1}^3 \omega_i = 1$

The label similarity, sim_l , is a string metric that measures the syntactical similarity. We propose to use the Levenshtein distance to compute the matching between two nodes labels. sim_l is defined as follows :

$$sim_l(x.label, y.label) = \frac{LevenshteinDistance(x.label, y.label)}{\max(|x.label|, |y.label|)} \quad (3)$$

The similarity between the nodes attributes is the distance between their corresponding attribute vector. we propose to use for that the euclidean distance, sim_a is defined as follows :

$$sim_a(x.attributes, y.attributes) = \sqrt{\sum_{k=1}^A (x.a_k - y.a_k)^2} \quad (4)$$

The relational similarity, sim_r , between two nodes, measures the common elements that both nodes are in relation with. There are different levels of relations that could occur between the nodes in the same VNFD model. We divide this relation levels based on the reachability of the nodes. For example, the first level relation of node v_1 in Fig. 2 are the nodes that are directly related to it i.e. v_2 . The second level relation are the nodes v_5 and v_6 . It is important in the relational similarity nodes to account for the level of the nodes when comparing the relations. Nodes that are in lower relation levels are more relevant to account their similarity than the farthest relation levels. Therefore, we define the relational similarity as following:

$$Sim_r(x, y) = \sum_{z=1}^Z \frac{\sum_{\gamma \in \Gamma(x, y)^z} \frac{Z+1-z}{Z} \times dist_r(\gamma)}{|\Gamma(x, y)^z|} \quad (5)$$

where Z is the number of relational levels to be considered in the similarity. $\Gamma(x, y)^z$ is the set of all combination of nodes (x', y') in the relational level z , such that x' is a node related to x and y' is a node related to y . $dist_r(\gamma)$ is the distance between the nodes in the relational set that measures the similarity in terms of the label and the attributes. It is defined in eq. 6

$$dist_r(x, y) = (\omega'_1 * sim_l) + (\omega'_2 * sim_a) \quad (6)$$

where ω'_1, ω'_2 are weights. $\omega'_i = \omega_i + \frac{\omega_3}{2}, i \in \{1, 2\}$

2) *Nodes clustering algorithm*: The clustering algorithm groups the nodes with similar types together in clusters. Each cluster contains the nodes that are likely to be similar. The resulted set of clusters are afterward added in the configurable VNFD model, each cluster is a single node in the configurable VNFD model.

Given a set of nodes with similar types, we propose to use the k-medoids method [16] to find k clusters that groups similar nodes together. Initially, the k-medoids method selects arbitrary k nodes from the nodes set as the center of k independent clusters. At each iteration, the method minimizes the distance between all nodes and the center of clusters by assigning each node to the closest cluster. The center of clusters is afterward reselected at as the closest node in the cluster to all the other nodes in the same cluster. Algorithm 1 summarizes the clustering algorithm.

Algorithm 1: Clustering algorithm

Input: $\vec{V}^T$ (set of nodes of type T), K (number of clusters), Max_I (maximum number of iterations)

Output: $Cl_1, Cl_2, \dots, Cl_K$ clusters of nodes

// Initialization

- 1 $I=0$ (Initial iteration)
- 2 $Center_k = Select(x \in \vec{V}^T)$ (choosing K unique nodes as cluster centroids)
- 3 **while** (*Nochangesinall* $Center_k$) **do**
 - // Constructing the K clusters
 - 4 **For each** $x \in \vec{V}^T$, **Add** (x, Cl_j) **such that** $dist(x, center_j) = Min(dist(x, center_k)), \forall k \in [1, K]$
 - // Updating the clusters centroids
 - 5 **For each** $Cl_k, k \in [1, K]$, $Center_k := x, x \in Cl_k$ **such that** $\sum_{y' \in Cl_k} dist(x, y') <= min(\sum_{y' \in Cl_k} dist(x', y')), \forall x' \in Cl_k$
 - 6 $I++$
- 7 **end**
- 8

A. Constructing the configurable VNFD model

The configurable model is constructed starting from the root node. The root is by default named ($VNFD^c$), which is the grouping of all the root nodes in the VNFD models set. Each node is linked to other nodes for which it has relations with. If the node is created in the second phase (federation of other similar nodes) then different relations could exists depending on which configuration is selected. Configurable operators are added to account for this variability.

Given the resulted clusters $Cl = cl_1, cl_2, \dots, cl_K$ from the second phase. We define for each cluster $cl_i \in Cl$ three sets for each relation type that the cluster could have with other nodes or clusters of nodes. $N_{i,r}^-, N_{i,r}^+, N_{i,r}^*$, where i is the cluster index and r is the relation type (composition, allocation or connection). The three sets contain the nodes or clusters of nodes that the cluster cl_i is in relation with (relation type = r). The cluster cl_i contains nodes that are likely to be similar.

tab. I. The complexity metrics for the two considered scenarios

Complexity metric	Multiple VNFDs	Configurable model
CFC	18	26
ACD	2.9	4.2
CNC	0.8	0.6
Density	0.2	0.05

The particularity of each set is on how common the nodes are included in the cluster cl_i share the relations with the other nodes. The set $N_{i,r}^+$, contains nodes that are in relation with more than one node in cl_i . $N_{i,r}^-$, contains nodes that are in relation with only one node in cl_i and $N_{i,r}^*$, contains nodes that are in relation with all the nodes in cl_i .

The configurable gateways are added to the cluster cl_i relations depending on the sets $N_{i,r}^-, N_{i,r}^+, N_{i,r}^*$. A configurable *XOR* gateway is added between the cluster relations with the nodes in $N_{i,r}^-$, with respect to the relation type r . A configurable *OR* gateway is added between the cluster relations with the nodes in $N_{i,r}^+$. The relation with the nodes in $N_{i,r}^*$ are linked using a simple gateway or with a direct edge if this are only one node in $N_{i,r}^*$.

VI. EXPERIMENTS

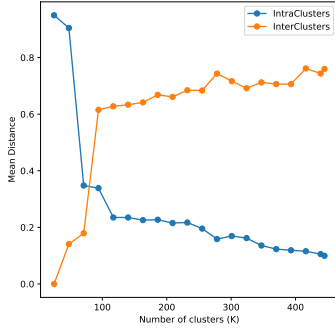
A. Environment settings

In our experiment, we consider approximately 100 VNFDs collected from various organizations and private telco vendors. The data set include several variations of VNFDs for different VNFs such as Mobility Management Entity (MME), virtual Customer Premises Equipment (vCPE), virtualized Subscriber Data Management (vSDM), etc. The VNFDs are YAML files that follows the TOSCA data model. We implemented a python program to construct the VNFD models from the data set. The number of nodes extracted, with respect to their type, are : 65 VNFC, 453 VDUs, 453 VCs, 421 VMs, 357 VSSs, 520 CPs.

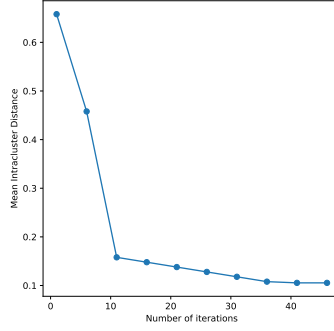
B. Complexity of the configurable VNFD model

To evaluate the complexity of our proposed solution, we consider a scenario in which a service provider wants to design a new VNF. The only information that he can rely on to have insights about the deployment configuration of the VNF is a repository of the 30 VNFDs. In order to capitalize on the knowledge provided by the 30 VNFDs, we evaluate the complexity of learning from a representation given by the configurable VNFD model (scenario 1) against a complexity of learning from the representation of 30 VNFDs graphs (scenario 2).

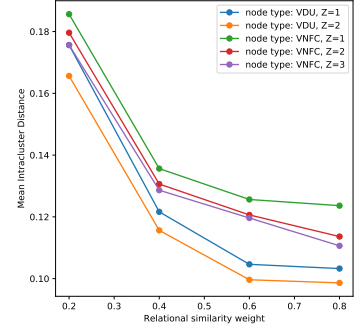
We assess the structural complexity of the two representations by computing the well-known complexity metrics: CFC (Control Flow Complexity), ACD (Average Connector Degree), CNC (Coefficient of Network Connectivity) and density. CFC metric [16] evaluates the complexity of the representation with respect to the configurable elements (OR, XOR). ACD metric [17] calculate the number of nodes that an operator has as average. CNC metric [18] gives the ratio of edges to nodes and the density metric [19] relates the number of edges to the number of maximum edges that can exist among nodes. The



(a) Inter-cluster and Intra-cluster mean similarity distance in terms of different cluster numbers, VDU nodes



(b) Intra-cluster mean similarity distance per iteration, VDU nodes, k=200



(c) Intra-cluster mean similarity distance in terms of different relation weights

Fig. 5. Experimental results of the clustering algorithm

results are summarized in tab. I. The metrics of the multiple VNFD representations are obtained by summing all the metrics resulted from each VNFD representation individually. By comparing the metrics' values of the two, we notice that the CFC of our configurable model is very high compared to the multiple VNFD representations. This can be explained by the fact that when similar VNFD elements are merged in the configurable model, more possibilities of relations with other VNFD elements are generated, especially in the case when the configurable operator "OR" is used. The lower density value of the configurable model show that it has a reasonable complexity. This can be explained by the fact that the configurable descriptor model contains less duplication of the components instances and is more expressive.

C. Nodes Clustering algorithm

We evaluate the performance of the clustering algorithm by measuring the efficiency of the clusters into grouping similar nodes. We compute for that the mean similarity distance between the nodes inside the clusters (intra-clusters similarity) and the distance between the clusters (inter-clusters similarity). The intra-cluster similarity measures how good the clusters group similar nodes and the inter-cluster similarity measures how isolated the clusters are from each other.

The clustering algorithm is applied separately for each node type in the VNFD model. The number of clusters to use is determined empirically. In Fig 5a, we measure the performance of the algorithm in terms of different number of clusters. We considered in that experiment the clustering of the VDU nodes. We notice that the distance intra-clusters decreases when more clusters are considered and the distance inter-cluster increases with the number of clusters. This is because the algorithm isolates better the similar nodes in a large data set (450 nodes) when more clusters are used. However, the similarity distance is relatively stable after a while (when 130 clusters are used). This happens when an efficient number of clusters is reached. Additional number of clusters will not improve significantly the performance in this

case. The algorithm converges relatively faster to a minimal mean intra-cluster similarity. Fig 5b, shows the performance of the algorithm after a number of iterations. The experiment is also conducted on the VDU nodes. The number of clusters is fixed to 200. Initially, the performance increases significantly until it stabilizes after 10 iterations. This is due to the random initialization of the clusters centroids. Fig. 5c shows the performance of the algorithm in terms of different relational weight values (ω_3). The other weights ω_1 and (ω_2) are fixed to $(1 - \omega_3)/2$. We considered in this experiment the clustering of VDU nodes and VNF C nodes. We notice that the performance proportionally to the relational weight. This is because the VDU and the VNFDC nodes have few attributes and many relations to the other nodes. We evaluated also the impact of the zone Z parameter. It indicates the level of the relations to be considered. We notice that this parameter can also enhance considerably the performance of the algorithm

VII. CONCLUSION

In this work we defined a configurable deployment descriptor model that merges similar VNF component elements into a single model. The configurable model captures the relation and also the variability between the VNF elements. It is a solution that can enable service providers to generate and variate VNF descriptors easily. Also, we proposed a machine learning approach to learn the configurable model from different VNF descriptors. For that we defined an algorithm based on KNN that automatically cluster similar descriptor elements/nodes based on a defined similarity distance metric. The experiments showed that the configurable model has a good complexity compared to a repository of VNF descriptors. Our learning algorithm showed also a good performance for clustering similar descriptor elements. As future work, we intend to extend our model in order to incorporate the behavioral aspect in the deployment descriptors like the scalability, the policy constraints, the affinity/anti-affinity relations, etc.

REFERENCES

- [1] ETSI, “Gs nf-v-ifa 011.” <https://standards.globalspec.com/std/13271186/gs-nfv-ifa-011>, 2019.
- [2] ETSI, “Gs nf-v-ifa 014.” <https://standards.globalspec.com/std/13271188/gs-nfv-ifa-014>, 2019.
- [3] K. Katsalis, N. Nikaein, and A. Edmonds, “Multi-domain orchestration for nf-v: Challenges and research directions,” in *15th Inter. Conf. on Ubiquitous Comp. and Commun.*, pp. 189–195, Dec 2016.
- [4] N. F. S. de Sousa, D. A. L. Perez, R. V. Rosa, M. A. S. Santos, and C. E. Rothenberg, “Network service orchestration: A survey,” *arXiv:1803.06596*, 2018.
- [5] C. Makaya and D. Freimuth, “Automated virtual network functions onboarding,” in *IEEE Conf. on Network Function Virtualization and Software Defined Networks*, pp. 206–211, Nov 2016.
- [6] V. Riccobene, M. J. McGrath, M. Kourtis, G. Xilouris, and H. Koumaras, “Automated generation of vnf deployment rules using infrastructure affinity characterization,” in *IEEE NetSoft Conf. and Workshops*, pp. 226–233, June 2016.
- [7] H. Jmila, M. I. Khedher, and M. A. El Yacoubi, “Estimating vnf resource requirements using machine learning techniques,” in *Neural Information Processing*, (Cham), pp. 883–892, Springer, 2017.
- [8] T. Nguyen and M. Yoo, “A vnf descriptor generator for tacker-based nf-v management and orchestration,” in *In Proc. Int. Conf. on Information and Communication Technology Convergence (ICTC)*, pp. 260–262, Oct 2018.
- [9] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.
- [10] W. S. Atoui, I. G. Ben Yahia, and W. Gaaloul, “Using deep learning for recommending and completing deployment descriptors in nf-v,” in *IEEE Conf. on Net. Softwarization (NetSoft)*, pp. 233–235, June 2019.
- [11] W. S. Atoui, I. Grida Ben Yahia, and W. Gaaloul, “Virtual network function descriptors mining using word embeddings and deep neural networks,” in *IFIP/IEEE Symp. on Integrated Net. and Service Management (IM)*, pp. 515–520, April 2019.
- [12] F. Gottschalk, T. A. C. Wagemakers, M. H. Jansen-Vullers, W. M. P. van der Aalst, and M. La Rosa, “Configurable process models: Experiences from a municipality case study,” in *Advanced Information Systems Engineering* (P. van Eck, J. Gordijn, and R. Wieringa, eds.), pp. 486–500, Springer, 2009.
- [13] N. F. S. de Sousa, D. A. L. Perez, R. V. Rosa, M. A. S. Santos, and C. E. Rothenberg, “Network service orchestration: A survey,” *CoRR*, vol. abs/1803.06596, 2018.
- [14] M. Rosemann and W. M. P. van der Aalst, “A configurable reference modelling language,” *Inf. Syst.*, vol. 32, no. 1, pp. 1–23, 2007.
- [15] J. Recker, M. Rosemann, W. M. P. van der Aalst, and J. Mendling, “On the syntax of reference model configuration - transforming the C-EPC into lawful EPC models,” in *Business Process Management Workshops, BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS, Nancy, France, September 5, 2005, Revised Selected Papers*, pp. 497–511, 2005.
- [16] L. Kaufman and P. J. Rousseeuw, “Clustering by means of medoids,” 1987.
- [17] J. Cardoso, J. Mendling, G. Neumann, and H. A. Reijers, “A discourse on complexity of process models,” in *Proc. Int. Conf. on Business Process Management Workshops, BPM’06*, pp. 117–128, Springer, 2006.
- [18] B. List and B. Korherr, “An evaluation of conceptual business process modelling languages,” in *Proc. ACM Symp. on Applied Computing, SAC ’06*, pp. 1532–1539, ACM, 2006.
- [19] H. A. Reijers and I. T. P. Vanderfeesten, “Cohesion and coupling metrics for workflow process design,” in *Business Process Management*, pp. 290–305, Springer, 2004.