Charms and Virtual Network Functions Primitives Experiments using Open Source MANO Framework

Andra Ciobanu ,Cosmin Conțu, Eugen Borcoci
Telecommunications Department
University POLITEHNICA of Bucharest – UPB
Bucharest, Romania
Emails: andraciobanu90@yahoo.com, cosmin.contu@elcom.pub.ro, eugen.borcoci@elcom.pub.ro

***Abstract*—Network Function Virtualization (NFV) is a strong technology to support the development of flexible and customizable virtual networks in multi-tenant and multi-domain environment. Open issues still exist for architectural, interoperability, design and implementation aspects. The contributions of this paper consist in introducing Open Source MANO OSM framework, present deeper knowledge about Virtual Network Function (VNF) charms and primitives. The objective of this deeper knowledge is accomplished with a proxy charm experiment. Also, a bug fix for proxy charm is presented and other capabilities and possible limitations in different contexts are going to be examined and sorted out for the future works. Towards this aim, this paper continues the authors work and develops new functionalities along with the previous ones and integrates them in a complex network topology using OSM. This work can help the developers implementing NFV systems based on OSM**

Keywords—VNF;OSM;Charm;Primitives;Bug

I. INTRODUCTION

Telecommunication infrastructures consist of a myriad of technologies from specialized domains such as radio, access, transport, and core and (virtualized) data center networks. Designing, deploying and operating end-to-end services are commonly manual and long processes performed via traditional Operation Support Systems (OSS) resulting in long lead times (weeks or months) until effective service delivery [2]. Moreover, the involved workflows are commonly hampered by built-in hazards of infrastructures strongly coupled to physical topologies and hardware-specific constraints.

Technological advances under the ages of *Software Defined Networking (SDN*) [3] and *Network Function Virtualization (NFV)* [3] bring new ways in which network operators can create, deploy, and manage their services. SDN, NFV, and cloud computing technologies are powerful tools enabling services, and systems to meet certain objectives (e.g., a customer requesting a specific network service). Altogether, the process shall be timely, consistent, secure, and lead to cost reduction due to automation and virtualization. We refer to *Network Service Orchestration (NSO)* as the automated management and control processes involved in services deployment and operations performed mainly by telecommunication operators and service providers [4].

However, to realize this paradigm, there is a need to model the end-to-end (E2E) service and have the ability to abstract and automate the control of physical and virtual resources delivering the service. The coordinated set of activities behind such process is commonly referred to as orchestration.

In this paper, another orchestration framework has been studied and used, i.e., Open *Source MANO (OSM).* The reason for this is that SONATA framework and project itself it is a part of entire OSM and is not going to be treated alone anymore in the future, but as an integrated part of OSM.

OSM is an ETSI-hosted open source community delivering a production-quality Management and Orchestration (MANO) stack for NFV, capable of consuming openly published information models, available to everyone, suitable for all Virtual Network Functions (VNFs), operationally significant and Virtual Infrastructure Management (VIM)-independent. OSM is aligned to NFV ISG information models while providing first-hand feedback based on its implementation experience [5].

The main purpose of this paper is to continue with development of previous experiments [1][9], but which are now based on OSM framework in order to understand the capabilities of the framework, and to develop and test some custom VNFs and service chains. Previous experiments were presented in coauthor's papers [1], and they consist in VNF and network services development with SONATA and OSM frameworks.

The paper is organized as follows. Section II is an overview of related work and a short high-level overview of the OSM. Section III presents a selective view in explanation of "day 0, day 1, day 2 VNF configurations and concepts, VNF primitives and charms, all of them integrated with OSM framework. Section IV contains the results of the charm experiments done with OSM framework and all the steps taken. Section V presents conclusions and future work.

 ISBN: 978-1-61208-802-0

II. Related Work-OSM Architecture and Functions

This section shortly presents a selective view on an open-source solution and some related work dedicated to service development and orchestration in virtualized networks and its relation to OSM architecture, when applicable.

The goal of ETSI European Telecommunications Standards Institute (ETSI) OSM is the development of a community-driven production-quality E2E Network Service Orchestrator (E2E NSO) for telco services, capable of modelling and automating real telco-grade services, with all the intrinsic complexity of production environments. OSM provides a way to accelerate maturation of NFV technologies and standards, enable a broad ecosystem of VNF vendors, and test and validate the joint interaction of the orchestrator with the other components it has to interact with: commercial NFV infrastructures (NFVI+VIM) and Network Functions (either VNFs, Physical Network Functions- PNFs or Hybrid ones).

OSM's approach aims to minimize integration efforts thanks to four key aspects:

A well-known *Information Model (IM),* aligned with ETSI NFV, that is capable of modelling and automating the full lifecycle of Network Functions (NF) (virtual, physical or hybrid), Network Services (NS), and Network Slices (NSI), from their initial deployment (instantiation, Day-0, and Day-1) to their daily operation and monitoring (Day-2). OSM's IM is completely infrastructure-agnostic, so that the same model can be used to instantiate a given element (e.g., VNF) in a large variety of VIM types and transport technologies, enabling an ecosystem of VNF models ready for their deployment everywhere.

OSM provides a *unified northbound interface (NBI),* based on NFV SOL005, which enables the full operation of system and the Network Services and Network Slices under its control. In fact, OSM's NBI offers the service of managing the lifecycle of Network Services (NS) and Network Slices Instances (NSI), providing as a service all the necessary abstractions to allow the complete control, operation and supervision of the NS/NSI lifecycle by client systems, avoiding the exposure of unnecessary details of its constituent elements.

The OSM extended the concept of "Network Service" , so that an NS can span across the different domains identified like virtual, physical ones or technological like Radio access network (RAN) core, and transport networks. Therefore, it is possible to control the full lifecycle of an NS interacting with VNFs, PNFs and Hybrid Network Functions (HNFs) in an undistinguishable manner along with on demand transport connections among different sites. In addition, OSM can also manage the lifecycle of Network Slices, assuming when required the role of Slice Manager, extending it also to support an integrated operation [6].

III. VNF Configurations and Concepts, VNF Primitives and Charms

The OSM's approach aims to minimize integration efforts, so a well-known Information Model (IM), aligned with ETSI NFV is capable of modelling and automating the full lifecycle of Network Functions.

Virtual Network Function Descriptor [VNFD] defines the resources required for realizing the VNF. The descriptor includes various components that are part of the VNF. In addition, it also defines the VNF level configuration information.

The VNFD connects *Virtual Deployment Units (VDUs)* using the internal *Virtual Links (VLs)*. Each VDU represents a Virtual Machine (VM)/Container. The following diagram from Figure 1 illustrates the internal structure of VNFD.

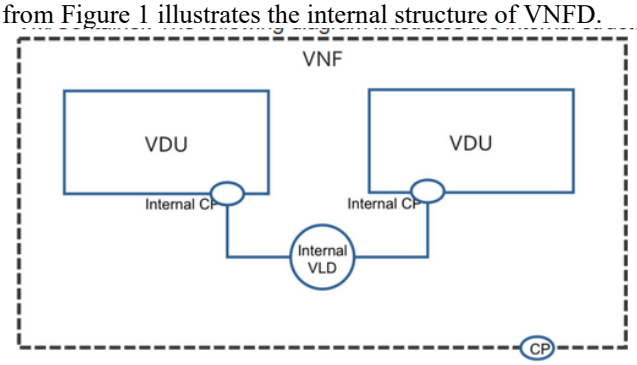

Figure 1 Virtual Network Function Descriptor [7]

The VDUs attach to the internal VLs using the internal *Connection Points (CPs).* So, the VNFD captures the list of VDUs and the internal VLs that connect the VDUs [7]. NFV promises to go from traditional network management to native NFV management, with highly efficient automation and operation.

In order to fulfill the complete onboarding process, a VNF Package will be produced and it will be part of the OSM catalogue for its inclusion in a Network Service. The onboarded VNF should aim to fulfil the lifecycle stages. It requires to function properly, for the NFV MANO layer to be able to automate. To accomplish the lifecycle stages, the resulting package includes all the requirements, instructions and elements which are: basic instantiation (a.k.a. "Day0"), service initialization (a.k.a. "Day1") and runtime operations (a.k.a. "Day2") – see Figure 2.

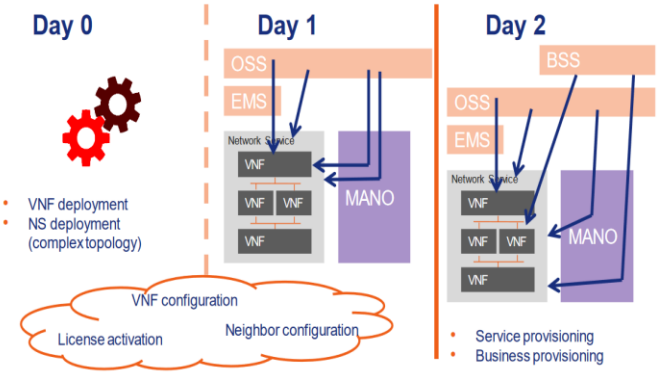

Figure 2 High Level Overview Day 0, 1 , 2 VNF [7]

 ISBN: 978-1-61208-802-0

In **Day0** stage, the VNF is instantiated and the management access is established so that the VNF can be configured at a later stage. Furthermore, *cloud-init* files can be used to inject this minimal configuration to the VNF.

The main objective of the **Day1** stage is to configure the VNF so it starts providing the expected service. The service is defined in the initial configuration which will run automatically after the VNF in instantiated.

The main objective of **Day2** is to be able to re-configure the VNF so its behavior can be modified during runtime. In Day2, main Key Performance Indicators KPIs and their run scaling actions can be monitored.

Juju is a generic Virtual Network Function Manager (VNFM) in the ETSI NFV architecture [7]. Juju is a universal service modelling system; it models services, their relationships and scale, independent of substrate (cloud, virtualized or physical). **Juju** is an open source modeling tool, composed of a controller, models, and charms, for operating software in the cloud. It can handle configuration, relationships between services, lifecycle and scaling, which ensures that common elements such as databases, messaging systems, key value stores, logging infrastructure and other 'glue' functions are available as charms for automatic integration, reducing the burden on vendors and integrators.

A **charm** is a collection of actions and hooks that encapsulate the operational logic of an application. A charm is a piece of software that runs scripts over some targets. Traditionally the charms wrote in Juju are used inside an application or in the same machine as an application.. Charms make it easy to reliably and repeatedly deploy applications, and then scale them as required with minimal effort.

Charmed OSM is an Open Source MANO distribution, developed and maintained by Canonical, which uses Juju charms to simplify its deployments and operations. Charmed OSM enables Telecommunication Service Providers (TSPs) to easily deploy pure upstream Open Source MANO in highly available, production-grade and scalable clusters. Hooks manage the lifecycle events of an application, from installation, configuration (day-0), and scaling, in a repeatable and reliable way. Actions are on-demand functions that can handle day 1 and day 2 configuration. Type of charm depends on type of Network Function, as it can be seen in Figure 3.

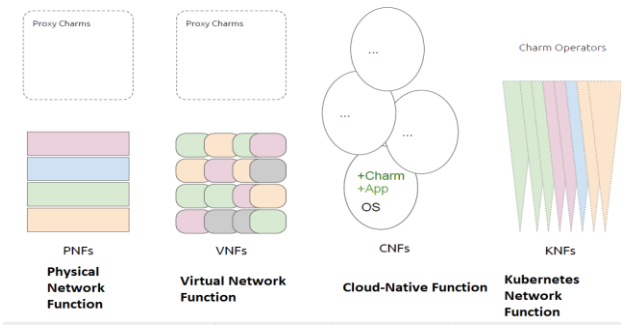

Figure 3. Generation of Network Function with OSM [7]

Types of Charms: Proxy – the focus on this paper

- Used for Physical and Virtualized Network Functions
- Runs in a Linux LXD container, separate from the VNF
- Only handles day 1 and day 2 configuration
- Usually communicates with VNF via SSH

Types of Charm: Machine

- Used for Cloud-native Network Functions (CNF) CNFs are like VNFs, but they run on lighter-weight containers, providing greater agility and ease of deployment compared with VMs
- Runs on the same machine as the VNF

Types of Charm: KNF

- Used for *Kubernetes Network Functions*

Charm runs as Operators in Kubernetes and manages lifecycle and day 0, day 1 and day 2 configuration.

VNF primitives in OSM are declared in the VNF Descriptor and their initial-config-primitive (Day-1) is invoked by the LCM at instantiation time. This is where the special 'config' primitive is invoked, setting ssh credentials. The config-primitive (Day-2) is invoked by the LCM at operator demand (or demanded through the NBI e.g., from an OSS). These primitives are a 1:1 map to a charm action or the 'config' hook.

IV. PROXY CHARM BUILD EXAMPLE IN OSM AND BUG FIX EXAMPLE

As opposed to classical "Native charms", **Proxy charms** run from outside the application. In particular, it run within a model instantiated in a LXC container that configures the VNFs through their management interface, as it can be observed in Figure 4. Proxy charms cover day-1 and day-2 configuration [8].

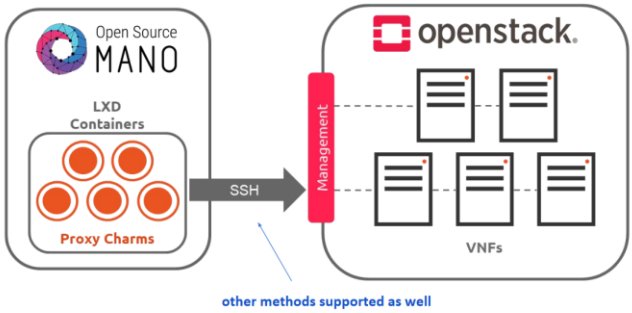

Figure 4. Proxy Charms [8]

The steps needed to build a proxy charm are the following:

1. *1) Setting up a charming environment (sudo snap install charm --classic # already installed in using shared OSM)*
   a) Create needed directories for building the charm
   b) Juju and charms environment variables

2) Creating a Proxy charm layer

a) Metadata.yaml includes all the high level information of our charm

b) Layer.yaml states all the layers on which our layer is based

c) Actions.yaml contains the high level description of the actions that will be implemented in the charm

d) Reactive/simple.py contains the actual code of the Proxy charm

3) Implementing the action

a) Append the implementation of the action to reactive/simple.py

Further on, the objective is to provide the guidelines for including all the necessary elements in the VNF Package and to provide the guidelines for including all the necessary elements in the VNF Package for its successful instantiation and management setup, so it can be further configured at later stages. The way to achieve this in OSM is to prepare the descriptor so that it accurately details the VNF requirements, prepare cloud-init scripts (if needed), and identify parameters that may have to be provided at later stages to further adapt to different infrastructures.

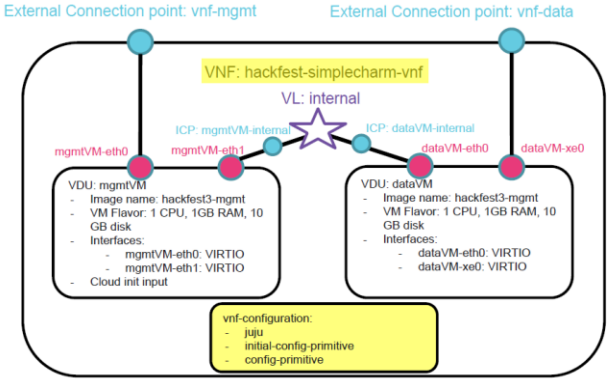

Figure 5. Practical example of adding charm [8]

The initial-config-primitive section takes care of **Day-1**

- The seq section states the order in which the initial config primitives will be called.
- The Proxy charm has ssh access to the VNFs thanks to the config primitive.
- The touch primitive is our Day-1 action created in the simple charm

The config-primitive section contains the available on-demand actions for **Day-2**.

Day-2 primitives are actions invoked on demand, so the config-primitive block is used instead of the initial-config-primitive block at the VNF or VDU level.

Proxy charms for implementing Day 2 primitives are built exactly in the same way as when implementing Day 1 primitives. In this stage, at day2 different monitoring parameters can be added:

- Collecting NFVI metrics
- Collecting VNF indicators
- Adding scaling operations

During charm implementation, some problems or bugs may occur. A common one can appear when the proxy charm presented above has been added at day-1 stage at the ssh access. The contribution of the paper has been to develop a python script in order to ensure that the workload status is set to active only when SSH proxy is properly configured.

The main principal flow is that charms would set their own workload status. The flow for status is the following:

- the charm start out and sets “maintenance”
- during its scope, it stuff up in “active” when the workload is ready
- or if there is any probem which needs intervention, it sets “blocked” status.

Charms, in general provide layers so that someone can build a proxy charm and evaluate a risk. A piece of software is valid for a charm and it can be reused-that is why layers are used. One of the layers is the “sshproxy” layer that includes ways to connect automatically to any VNF through SSH.

The implemented script scope is to set the proxy charm’s state to active so the LCM knows it is ready to work only when SSH proxy is ok configured.

As it can be seen in Figure 6, the first part invokes the “reactive/simply.py” code, then it sets active status when ssh is configured, then, the last step is to map the action to the commands to be run.

 ISBN: 978-1-61208-802-0

```
from charmhelpers.core.hookenv import (
    action_get,
    action_fail,
    action_set,
    status_set,
)
from charms.reactive import (
    clear_flag,
    set_flag,
    when,
    when_not,
)
import charms.sshproxy


@when('sshproxy.configured')
@when_not('simple.installed')
def install_simple_proxy_charm():
    """Post-install actions.

    This function will run when two conditions are met:
    1. The 'sshproxy.configured' state is set
    2. The 'simple.installed' state is not set

    This ensures that the workload status is set to active only when the SSH
    proxy is properly configured.
    """
    set_flag('simple.installed')
    status_set('active', 'Ready!')


@when('actions.touch')
def touch():
    err = ''
    try:
        filename = action_get('filename')
        cmd = ['touch {}'.format(filename)]
        result, err = charms.sshproxy._run(cmd)
    except:
        action_fail('command failed:' + err)
    else:
        action_set({'output': result})
    finally:
        clear_flag('actions.touch')
```

Figure 6. Fixed typo in simple proxy charm

CharmHelpers provides an opinionated set of tools for building Juju charms. This script imports actions from charmHelpers, reactive and ssh.proxy,then creates a function that mentains the workload status of the proxy charm when SSH is configured.

V. CONCLUSIONS AND FUTURE WORK

This paper continued the work from our previous one [9] and got deeper into the structure of a VNF and network service from descriptors creation until management setup and instantiation. The chosen framework is OSM, which is an ETSI-hosted project to develop an Open Source NFV Management and Orchestration software stack aligned with ETSI NFV. The reason why SONATA wasn't used in this case is that it belongs to OSM community and is not so modular as OSM framework is.

As future work, new experiments will be done in OSM and contact with OSM community will be maintain in order to adjust and address possible other bugs or issues with charms and primitives.

REFERENCES

[1] A. Ţapu, C. Conţu and E. Borcoci, "Multiple Chained Virtual Network Functions Experiments with SONATA Emulator", The 12th International Conference on Communications 2018.

[2] http://www.blueplanet.com/products/multi-domain-service-orchestration.html accessed 2020-10/09

[3] D. Kreutz et.al.,"Software-Defined Networking: A Comprehensive", Survey. Proceedings of the IEEE 103, 14{76. URL: http://ieeexplore.ieee.org/document/6994333/, doi:10.1109/JPROC.2014.2371999,2015.

[4] R. Mijumbi et.al., "Network Function Virtualization: State-of-the-Art and Research Challenges", IEEE Communications Surveys & Tutorials 18, 236{262. URL: http://ieeexplore.ieee.org/document/7243304/, doi:10.1109/COMST.2015.2477041,2015.

[5] https://osm.etsi.org/wikipub/index.php/OSM_Release_FIVE.

[6] https://osm.etsi.org/docs/user-guide/02-osm-arhitecture-and-functions.html, retrived on 2020.

[7] https://osm.etsi.org/wikipub/images/0/07/Introduction_to_OSM_Zero_Touch_Carrier_Autmation_Congress_FJ.pdf, retrived on 2020.

[8] http://osm-download.etsi.org/ftp/osm-6.0-six/8th-hackfest/presentations/8th%20OSM%20Hackfest%20-%20Session%207.1%20-%20Introduction%20to%20Proxy%20Charms.pdf, retrived on 2020.

[9] A.Tapu, C.Conţu and E. Borcoci " Study on Use-Cases of Open Source Management and Orchestration Framework in 5G Projects", The Nineteenth International Conference on Networks ICN 2020.