Service Provider DevOps for Large Scale Modern Network Services

Juhoon Kim[1], Catalin Meirosu[2], Ioanna Papafili[3], Rebecca Steinert[4], Sachin Sharma[5], Fritz-Joachim Westphal[1], Mario Kind[1], Apoorv Shukla[6], Felicián Németh[7], Antonio Manzalini[8]

[1]Deutsche Telekom AG, *Germany*; [2]Ericsson AB, *Sweden*; [3]OTE, *Greece*; [4]SICS, *Sweden*; [5]iMinds, *Belgium*; [6]TU-Berlin, *Germany*, [7]BME, *Hungary*; [8]TI, *Italy*

***Abstract*—Network service providers are facing challenges for deploying new services mainly due to the growing complexity of software architecture and development process. Moreover, the recent architectural innovation of network systems such as Network Function Virtualization (NFV), Software-defined Networking (SDN), and Cloud computing increases the development and operation complexity yet again. One of the emerging solutions to this problem is a novel software development concept, namely DevOps, that is widely employed by major Internet software companies. Although the goals of DevOps in data centers are well-suited for the demands of agile service creation, additional requirements specific to the virtualized and software-defined network environment are important to be addressed from the perspective of modern network carriers.**

In this paper, we thoroughly debate DevOps requirements for developing a modern service creation platform by taking EU FP7 project UNIFY as a reference architecture and suggest the corresponding extensions of UNIFY interfaces that meet the discovered requirements.

I. Introduction

Network service providers are victims of their own success in the telecommunication business. System architectures and development processes become more and more complex due to the evolution of network technologies and innovation based on the rich Internet infrastructure. Moreover, recent research efforts made in academia and industry show a strong trend towards utilizing virtualized and software-defined network environments that adds complexity to the development and operation of large scale network services. This complexity keeps network service providers tardy in creation and deployment of new services and eventually hinders the further evolution of network technologies.

Major Internet software companies have been experiencing similar issues and designed tools and methods that aim at improving the efficiency of the software development and at narrowing the distance between the development and operations. The concept of unifying such modern development methods and tools is commonly referred to as *DevOps*. The communication and collaboration between developers as well as the integration of an individual piece of work are especially emphasized tasks of DevOps in the software development.

Several projects in data centers put this novel development concept into practice by tackling multiple challenges mainly brought up by the geographical distribution of network nodes. Examples of such challenges are:

- A high cost of operation in terms of time and human/financial resources due to the physical management of distributed nodes
- Limited visibility of network and service states that makes it difficult to assure the Quality of Experience (QoE)
- Difficulties in pinpointing the cause and location of problems (troubleshooting) and debugging
- Difficulties in deploying services quickly and frequently, e.g., due to the validation of service integration or regression test

This movement in data centers inspires service providers to adopt automated development and operation processes within the scope of virtualized and software-defined network infrastructure. The virtualized network infrastructure differs from the traditional definition of network resources in the data center in the perspective of its independence from the fixed location and/or hardware; hence provides great opportunities to network carriers to lower operation costs. However, existing management techniques and tools are developed to solve particular and well-defined problems of static network environments and therefore would need to be adapted to the requirements of virtualized network environments.

In this paper, we discuss DevOps requirements within dynamic network environments by taking EU FP7 project UNIFY [1] as a reference architecture built on the highly virtualized network infrastructure. Based on this discussion, we extend the UNIFY APIs in order to support developers in creating DevOps tools that fulfil the requirements of modern network service providers. We call this extended DevOps concept *Service-provider DevOps* or *SP-DevOps* in this paper. Further, we show how SP-DevOps can be used for the actual service creation in the UNIFY framework by applying this concept to one of use cases developed in [2].

The rest of this paper is structured as follows. In Section II, we discuss DevOps requirements in dynamic network environments such as virtualized and software-defined networks. Section III gives an overview of UNIFY architecture that is used as a reference architecture throughout this paper. In Section IV, four main processes of SP-DevOps are described. After that, the extension of UNIFY APIs and the use case of SP-DevOps are illustrated in Section V and Section VI, respectively. Finally, Section VII presents the related studies and Section VIII summarizes this paper.

II. DevOps Requirements for Service Providers

The UNIFY project [3], as well as the IETF [4] and the TMForum [5] have recently initiated activities that investigate the challenges related to adopting DevOps practices in carrier networks. In this section, we summarize the current status of the discussion that we are driving in IETF on this topic, and make some additional considerations.

A. Observability

Telecom operator infrastructure is characterized by a wide distribution over geographical areas, large numbers of nodes and functions involved in providing services and stringent contractual demands on high availability.

In software-defined infrastructures, we define observability as the property that provides visibility on the status of both physical and virtual components of the infrastructure at the time scale relevant for a particular task. One important requirement for observability is scalability of the components involved in the various processes, including message busses and databases. The scalability is also affected by trade-offs between the communication overhead and the reliability of a particular estimate and the strategic placement of the monitoring functions.

The SNMP agent paradigm commonly employed in management is clearly limiting the capability to provide scalable observability, and the OpenFlow extensions for monitoring unfortunately follow the same unscalable communication pattern. We believe that distributed information exchange between all types of virtual functions (with monitoring functions as a major use case) is key to providing scalable observability in a carrier environment. From a DevOps perspective, such distributed information exchange is an important enabler for pushing automation beyond simple sets of scripts that replicate tested and trialed situations.

Programmability is another key requirement for monitoring functions. It is clear that given the amount of information that can be collected from high-speed communication links, it is not feasible for a for-profit enterprise to monitor everything all the time. Exposing monitoring functions through programmability interfaces that enable to select what and how to monitor through expressive, declarative languages is important for applying DevOps principles from the data center to carrier-grade software-defined infrastructures.

B. Stability

We define stability as a property of software-defined infrastructures that deliver highly-available telecommunication services. For developers following that agile paradigm, constant changes are a way of life, while for the operators instability is the result of external, usually unforeseen, events that need to be handled. The difference in mentalities between these two traditionally separate organizations is thus significant.

The infrastructure itself may be in a state of continuous changes and thus inherently unstable, but the services that execute on top of the infrastructure are expected to be always available. The instability in the infrastructure may be due to manual processes (such as routine management and maintenance tasks) or, in particular in software-defined infrastructure, to the actions of a myriad of controllers and orchestrators that optimize resource usage and functional configurations. However, in order to maximize the stability at the service level, any instability (whether observed, predicted or scheduled) in the infrastructure needs to be communicated to all the interested parties and programmable interfaces that allow actuation have to be made available.

However, cascading actions originated by different processes that have different objectives and views of the overall operational situation need to be mitigated. Failing that, they may lead to the emergence of non-linear behaviors where local dynamics may lead to radically different global effects when combined.

C. Troubleshooting

Identifying the cause of a failure in an infrastructure that is continuously undergoing changes is a major issue that needs to be addressed for achieving reliable operations in software-defined infrastructures. From a DevOps perspective, troubleshooting usually involves a series of tasks that need to be performed by either a developer or an operator according to a pre-defined incident handling description. However, such descriptions are usually built for physical infrastructure and network functions that are implemented in boxes that have fixed locations. Everything changes in a software-defined infrastructure, where a multitude of actors may trigger changes at any time and not all changes might be available to be logged in a central point for investigation. Methods that allow automatic definitions of such workflows and allow automatically adapting the workflows to the changes in the infrastructure and virtual network functions are a significant requirement for next-generation infrastructure management. An increased degree of automation is in line with DevOps principles and enables more efficient use of resources.

In particular in SDN, a large amount of information regarding the actual state of the infrastructure is stored by controllers and orchestrators, while virtual function managers and orchestrators hold that state information at the functional levels. Exposing information that helps building automated workflows becomes key to addressing this requirement. Based on this information, tools could be built that assist both developers and operators by generating tailor-made troubleshooting workflows for particular services composed of virtual network functions.

D. Machine-to-Machine Interaction

The creation of new services is expected to be more strategic in the future as communication services become primary commodity in daily life. One of the scenarios being investigated in the telecom business is based on the assumption that, in the future, telecom infrastructure provides new types of services (e.g., Cognition-as-a-Services) to a growing number of non-human users such as smart devices loaded with the sophisticated cognitive software. This assumption is relevant to the subject of *Softwarization* or Network Function Virtualization (NFV). In this perspective, one important requirement for DevOps is to enable both human users and autonomous machine to verify/validate the integration of the service and to troubleshoot/debug causes of a failure via common APIs. When the autonomous machine accesses DevOps APIs, all

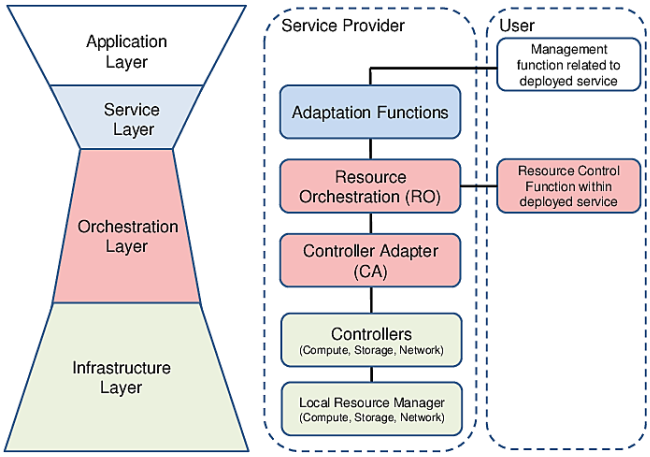

Figure 1: Three-layered UNIFY architecture

the necessary steps need to be triggered without a human intermediation. This eventually leads telecom providers to the reduction of Opex caused by human operations.

III. UNIFY

Although the afore-discussed properties are common DevOps requirements underlying the most of modern telecom service creations, viable action towards fulfilling them differs from one architecture to another. Therefore, we take EU FP7 project UNIFY as a reference architecture to show how such requirements can be effectively dealt in a large scale service creation.

A. Architecture Overview

To remedy today's limits in deploying and managing telecommunication services, the UNIFY project exploits the benefit of SDN-enabled infrastructure that pursues full network and service virtualization so as to enable rich and flexible services and operational efficiency.

UNIFY introduces a three-layer architecture (as shown in Figure 1) which comprises the service layer, orchestration layer, and infrastructure layer. Practically, each layer is employed to group functional components with similar abstractions. First, the service layer comprises traditional and modern (e.g., virtualization, SDN, and/or cloud) management/business functions and is responsible for translating user's abstract service request (Service Graph) into the detailed service description (Network Function Forwarding Graph). Second, the orchestration layer maintains the global view of resources and capabilities, what is more, it performs policy enforcement and resources orchestration between the upper layer functions and the underlying resources. Finally, the infrastructure layer includes resources and local resource agents (i.e., compute, storage and network) and their corresponding local agents (e.g., OpenFlow switch agent).

The UNIFY architecture exhibits specific characteristics which are important to SP-DevOps such as [i] *interalia* model-based decomposition so as to build services out of elementary blocks, [ii] recursive orchestration due to multi-level virtualization of storage, compute and network resources, and [iii] capability for multiple administrative domains allowing the possibility to separate roles and responsibilities among multiple business actors, e.g., network service providers, OTT providers, or end-users.

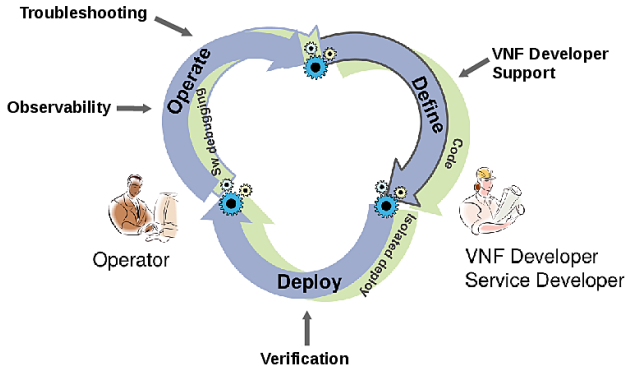

Figure 2: SP-DevOps cycle in UNIFY service creation framework

B. Dev & Ops in UNIFY

In UNIFY, two groups of developers are defined depending on their roles. The first group is *service developers* that design and create a service by determining its resource requirements and interconnecting desired network functions. This type of development is carried out based on a simple service description language. The second group is actual software developers that implement Virtualized Network Functions (VNFs). A VNF is a logical network component that runs within an independent container (e.g., virtual machine) and that is responsible for specific network tasks (e.g., firewall, NAT, or payload inspection). In a NFV architecture, a service is created by combining (or chaining) multiple VNFs. In UNIFY, those who design and implement VNFs are referred to as *VNF developers* and they correspond with developers in the software development domain. Since the architecture restricts the service development to pre-verified sets of physical/virtual infrastructure resources, the greater part of the development complexity lies in implementing VNFs. Thus, unless differently stated, the term *developers* addresses *VNF developers* in the remaining of this paper.

The major role of operators in UNIFY is the performance management of a service. More specifically, operators ensure that performance indicators of the service meet the requirements specified in the service graph. For this, operators need a fair degree of monitoring capabilities on virtual/physical elements of the infrastructure.

IV. SP-DevOps Processes of UNIFY

In this section, we show how the DevOps requirements for service providers discovered in Section II can be addressed within the scope of the service creation in the UNIFY framework. Figure 2 briefly illustrates mapping between the four large processes (*VNF developer support, verification, observability, troubleshooting*) of Service Provider DevOps (SP-DevOps) and the lifecycle of the service creation in UNIFY with regard to roles of operators and developers.

A. VNF development support

A Virtualized Network Function (VNF) is a basic functional unit in a Network Function Virtualization (NFV) architecture. In order to facilitate VNF development to be performed directly in the production system, there is need for a set of supporting functions provided by the architecture towards the VNF developer on top of the service layer. Once a VNF under development is deployed within the production system, VNF developers are supported with the observability, verification and troubleshooting capabilities described afterwards. For VNF development support functionality, three sub-processes are considered:

- **Adding a new VNF to the production environment:** This sub-process allows developers to add a new or updated VNF in the production environment for testing and debugging purposes. The service layer stores a description of the VNF capabilities and resource requirements given by the developer, and informs the orchestration layer about the existence of the VNF.
- **Modifying an already deployed service graph with a new or updated VNF:** A developer-request to deploy a VNF is received at the service layer and forwarded to the orchestration layer, which assesses the resource allocation for current and new VNFs by querying the service catalogue. The controller layer allocates the resources needed for the new instance and configures the policies associated with it.
- **Attach VNF to software development tools:** The developer queries the service layer by providing a service graph identifier and the type of VNF for the purpose of debugging. The request is forwarded to the orchestration layer, which provides an identifier for the VNF instance and associated resources. Given the identifier, the developer can connect to the VNF instance and run tools for distributed software debugging.

B. Verification process

In a large scale service architecture such as UNIFY, the verification process is not limited to the code validation. The emphasized goal of the verification in UNIFY is rather the assurance that the service configuration is in concord with the service definition, e.g., quality indicators and performance indicators.

Verification is considered as a set of features providing gatekeeper functions to validate both the abstract service models and the proposed resource configuration before actual instantiation on the infrastructure layer takes place.

In UNIFY, the verification process is performed across multiple layers in the architecture:

- **The service layer** verifies topological inconsistencies and loops of the service graph (SG) and network function forwarding graph (NF-FG).
- **The orchestration layer** validates mapping between the requirement of NFs and the allocation of the resource and capability. This layer also verifies whether or not the placement of a new NF violates the policy (and the performance) of the already deployed NF-FG.
- **The infrastructure layer** (controllers) checks consistency of specific configuration instances such as inconsistent network configuration in the form of OpenFlow rules.

C. Observability process

The observability process provides visibility onto the operational performance of service graphs deployed in the unified production environment. This process is carried out by selecting points and targets of measurements and choosing corresponding tools based on the key performance indicators (KPIs) and key quality indicators (KQIs) specified in the service graph. As a result of such measurements, the observability process generates and notifies diverse statistics of traffic, e.g., latency, throughput, and packet/byte counts, and status reports of infrastructure components, e.g., resource usage and availability. The definition of the measurement is delivered from developers down to the infrastructure layer in a chain and the notification traverses in reverse order.

D. Troubleshooting process

A troubleshooting process is requested either by a developer manually or by some components on service/orchestration layers automatically. The requested troubleshooting process aims to follow up on reported bugs, faults, and anomalous states. Such a process may include automated deployment of relevant verification and observability tools when the faulty or anomalous condition cannot be immediately localized from existing observations and reports. Detected and localized faults and performance degradations in the infrastructure layer are asynchronously reported to the virtualized infrastructure management layer (and forwarded if necessary to higher layers) where further investigation may take place.

V. Extension of UNIFY APIs

In order to provide SP-DevOps capabilities to the creation of services, it is crucial to ensure an appropriate level of programmability in the monitoring infrastructure. Taking UNIFY as an example, a set of APIs specific to DevOps are required to be implemented in order to support developers in implementing SP-DevOps applications (see Section VI) that are an essential part of the service creation and maintenance. Due to the abstracted and layered nature of the UNIFY architecture and to the wide scope of the DevOps operation in the architecture, the APIs need to be employed between almost all the layers in the architecture and are sequentially invoked across the layers.

First, APIs that create and remove a monitoring endpoint (*RegisterListener* and *UnregisterListener*) within the virtual infrastructure are necessary. The established monitoring endpoint observes the performance of service entities and validates their integrity. Second, it is important to provide developers with the capability to isolate the debugging environment from the production environment (i.e., debugging mode and release mode), thus APIs that change the execution status (*SetExecutionState and GetExecutionState*) of the service are required to be implemented. Third, multiple APIs are needed to query

and obtain the performance value in various metrics (e.g., *GetPerformanceValue*).

Furthermore, there are several capabilities that are crucial to enable automated troubleshooting and debugging features in DevOps applications. The inter-layer communication capability for notifying (*SetupNotification* and *Notify*) problems, e.g., detected and predicted failure, resource shortage, and/or malicious activities, is one of them and the verification capability (*Verify*) is another. These capabilities are required to be configurable via dedicated APIs.

VI. Use Case: SSL VPN Network Service

In this section, we consider the application of SP-DevOps by exploring the SSL VPN network service, i.e., one of the use cases developed in UNIFY project [2]. SSL VPN network service is an example of a value-added service that a service provider might offer on the virtualized network infrastructure.

A. Initiation of observability and troubleshooting

The configuration of a VPN service comprises several steps such as the configuration of the core network, the definition of a virtual template interface which enables the dynamic configuration of virtual access interface per user upon request, the formation and association of each VPN to a Virtual Routing and Forwarding (VRF) configuration, and the configuration of user profiles and Authentication, Authorization and Accounting (AAA) services at the customer premises.

Regarding maintenance of a VPN service, the service provider is obliged to frequently monitor a multitude of interfaces and protocols such as: i) concerning the core: validation of successful running of the routing protocol, and ii) concerning the VPN itself: validation of VRF configurations and routing tables, verification of associations of provider/customer edge (PE/CE) routers. The procedures for provisioning and maintenance of VPN include the participation of multiple dedicated appliances, e.g., AAA server, customer premises equipments (CPEs), DHCP server, edge and core routers. Operational procedures related to VPN are inefficient due to the complex manual configuration and monitoring of the different nodes associated with it. Adding another service on top of VPN, e.g., firewall, considerably increases further this complexity. Such manual activities are highly time-consuming and prone to errors, which in turn imply significant operating costs for the service provider. Such costs can be further increased by penalties due to SLA violations with regard to delivery or troubleshooting times.

Employing the UNIFY architecture, an SSL VPN service can be offered over software-defined virtual infrastructure. The devices deployed at customer premises are extremely simple and limited to packet forwarding, whereas all other relevant CPE functionality would be included in the vCPE[1] VNF deployed on a PoP at the edge of the IP network in co-location with a vPE[1] VNF that takes over the physical PE. A vSSL[1] VNF would implement acceleration of SSL encryption and decryption that are too intensive to be performed at high speed on the computational resources deployed in the PoPs.

[1] 'v' is the abbreviation for 'virtualized'

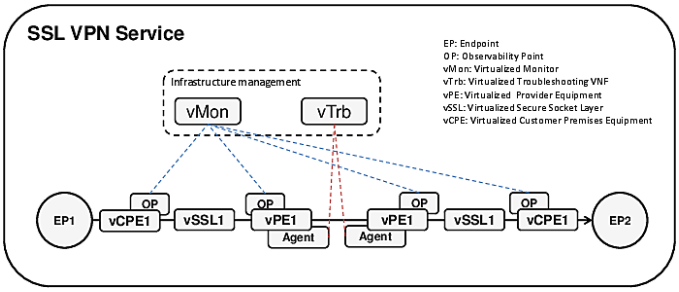

Figure 3: Simplified graphical representation of SSL VPN.

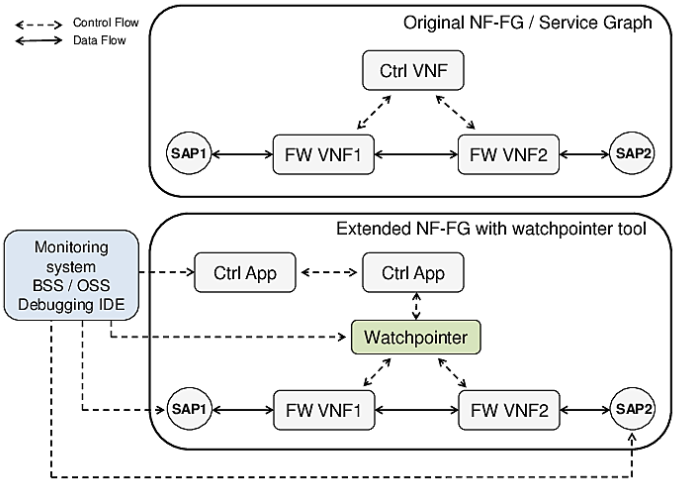

Figure 4: Use case over a watch-point inserted for observing data plane and control messages.

In the aforementioned setup, an observability VNF could be instantiated in co-location with vCPE and vPE, so as to monitor various KPIs related to the physical node (i.e., PE) as well as other VNFs associated to it. The infrastructure management enables an operator to introduce *vMon*[1], i.e., a virtualized monitoring VNF, that provides an enhanced granularity view for users. A customer representative trying to pro-actively troubleshoot connectivity problems employs *vTrb*[1], i.e., a virtualized troubleshooting VNF, that relies on trusted agents deployed in the virtual infrastructure to generate packets that test connectivity conditions at different layers on the path between the two vPE VNF instances. Finally, the vTrb VNF is integrated with the monitored infrastructure, such that in case one of the vPE VNF instances is moved automatically as a result of a scale-out operation ordered by the infrastructure management, the vTrb instance will follow the associated vPE instance and inform the customer representative of the event. Figure 3 illustrates how an SSL VPN service is extended with observability and troubleshooting capabilities.

B. An example of VNF developer support

An example of VNF development support is a stand-alone debugging tool providing network watch-points [3] that give visibility onto the OpenFlow control channel as well as actions that facilitate troubleshooting and debugging activities. The tool helps to define network watch-points for certain data path or control traffic events (policy violation, or

any OpenFlow matching rule), and is capable of triggering different troubleshooting actions. The network watch-points provide an opportunity for service developers or operators to define OpenFlow-like filters for selecting relevant packets and automatically perform pre-defined actions collectively or on a per packet basis. Due to its monitoring functionality, matching packets could be filtered out and suspicious traffic could be detected in an operational network.

For these purposes, network watch-points consist of three different parts: standard OpenFlow match fields; the switch defined by a unique data path ID (DPDI); and, troubleshooting actions (such as no action, drop, log, notify, etc). According to the OpenFlow matching fields, the DPID may contain a wildcarded value allowing a single entry to match multiple DPIDs. Moreover, since the match fields are standardized, the data plane operation mode can delegate the execution of the matching algorithm to switching hardware supporting the OpenFlow protocol.

Figure 4 illustrates an example where the network watching tool passively observes the control traffic for a service graph, for the purposes of observing and debugging the Control App. The tool performs a matching process on every OpenFlow message in order to find the longest matching watch-point entry in the watch-point repository (similar to a firewall), executing specified actions upon match. If there is no matching entry, then the tool forwards the captured OpenFlow message automatically to the controller. Additionally, watch-points can here be installed in relevant switches as flow rules with the highest priority to capture traffic that normally would not trigger a control message (such as PACKET_IN), for the purpose of troubleshooting. The tool operates transparently to flow table changes, by recording and analyzing flow modification messages between the data plane component and the controller, such that the mapping to the correct set of actions to be executed is maintained.

VII. Related Work

In current telecom industries, the TMForum enhanced Telecom Operations Map (eTOM) and the IT Information Library (ITIL) [6] are proposed to increase the ties between Dev and Ops. eTOM defines a set of business process models for the Ops part of DevOps. ITIL provides a collection of best practices and guidelines for companies and practitioners on the subject of managing IT services throughout their lifecycle. In the DevOps point of view, both the approaches (eTOM and ITIL) do not really increase the ties between Dev and Ops. Because eTOM provides a set of guidelines for only Ops part of DevOps and ITIL is difficult to implement in the telecom industries, as this involves a large number of actors and players. On the other hand, DevOps guidelines are expected to be for both activities (Dev & Ops) of telecom infrastructure and are expected to decrease the number of actors and roles, and to provide the exact procedures that are necessary to implement ITIL in a practical and realistic way.

The SDN solutions to support the DevOps concept are for the tools: (1) FlowSense [7] and OpenSketch [8], which estimate performance metrics, (2) OF-Rewind [9] and automatic test packet generation tool [10] which localize and find root-cause analysis of detected faults, and performance degradations and (3) NICE [11], VeriFlow [12], and NetPlumber [13], which compare expected and detected system states. The advantage of FlowSense is that it follows a passive monitoring approach, which is a scalable way of estimating the behavior in the network, but the limitations are related to timing - all estimates of the link utilization requires data based on completed flow sessions, which may not be suitable in an environment where timing is essential for dynamic service-chains. OpenSketch, however, offers a high degree of automation, but the maintenance of 'sketches', i.e., data structures for storing information about packet states, takes resources.

The advantages of OFRewind is that it is capable of recording both control and data traffic traces of an OpenFlow network, and is capable of replaying it in a custom OpenFlow network to reproduce the bugs. The challenges in replaying include timing accuracy, multi-instance synchronization, and online replay of multiple network elements. The automatic test packet generation tool, however, finds issues in both Dev and Ops by sending real test packets in a network. However, the problem is that it requires gathering all data at a centralized location or massive generation of test traffic, which puts a high load on the infrastructure. The other tools such as NICE, VeriFlow, NetPlumber, enable a (formal) verification of specific SDN configurations (e.g., availability of a path to the destination, absence of routing loops, access control policies, or isolation between virtual networks). However, these tools operate on the (centralized) programmability of the control plane only. This might be a limitation if we consider a possible network deploying active network functions, i.e., an environment that also enables programmability of the data plane in a distributed fashion.

VIII. Summary

In this paper, we discussed the DevOps requirements specific to the modern network service creations that follow a trend towards employing virtualization and software-defined network technologies. Furthermore, we proposed an extended DevOps concept (SP-DevOps) that addresses the discovered requirements and demonstrated how it can be used in a large scale service creation by taking EU FP7 project UNIFY as a reference architecture.

In order to further explore the concept of SP-DevOps, we described the four main processes, i.e., VNF developer support, verification process, observability process, and troubleshooting process, that together form the SP-DevOps lifecycle in a UNIFY service creation. In this paper, it is explained how each of these processes effectively tackles the corresponding DevOps requirement for a large scale service creation on virtualized and software-defined network environments.

Acknowledgement

This work is supported by FP7 UNIFY, a research project partially funded by the European Community under the Seventh Framework Program (grant agreement no. 619609). The views expressed here are those of the authors only. The European Commission is not liable for any use that may be made of the information in this document.

REFERENCES

[1] "Unify: Unifying cloud and carrier networks," 2014. [Online]. Available: https://www.fp7-unify.eu/

[2] "Use cases and initial architecture," 2014. [Online]. Available: https://www.fp7-unify.eu/

[3] "Initial requirements for the sp-devops concept, universal node capabilities and proposed tools," 2014. [Online]. Available: https://www.fp7-unify.eu/

[4] C. Meirosu, A. Manzalini, J. Kim, R. Steinert, S. Sharma, and G. Marchetto, "Devops for software-defined telecom infrastructures," 2014.

[5] "Zero-touch orchestration, operations and management project," 2014. [Online]. Available: http://www.tmforum.org/zoom/16335/home.html

[6] "Gb921-w working together - itil and etom, v 11.3," 2011. [Online]. Available: http://www.tmforum.org/DownloadRelease135/15584/home.html

[7] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: Monitoring Network Utilization with Zero Measurement Cost," in *ACM PAM*, 2013.

[8] M. Yu, L. Jose, and R. Miao, "Software Defined Traffic Measurement with OpenSketch," in *USENIX NSDI*, 2013.

[9] A. Wundsam, D. Levin, S. Seetharaman, A. Feldmann *et al.*, "OFRewind: Enabling Record and Replay Troubleshooting for Networks." in *USENIX ATC*, 2011.

[10] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "Automatic Test Packet Generation," in *ACM CoNEXT*, 2012.

[11] M. Canini, D. Venzano, P. Peresini, D. Kostic, J. Rexford *et al.*, "A NICE Way to Test OpenFlow Applications," in *USENIX NSDI*, 2012.

[12] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey, "Veriflow: Verifying network-wide invariants in real time," *ACM SIGCOMM CCR*, 2012.

[13] P. Kazemian, M. Chan, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis." in *USENIX NSDI*, 2013.