Received May 19, 2018, accepted June 22, 2018, date of publication July 9, 2018, date of current version July 30, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2852649

On the Feasibility of Using Hierarchical Task Networks and Network Functions Virtualization for Managing Software-Defined Networks

WILLIAM VILLOTA[1], MARIO GIRONZA[2], ARMANDO ORDOÑEZ[3], AND OSCAR MAURICIO CAICEDO RENDON[2]

[1]Instituto de Computação, Universidade Estadual de Campinas, Campinas 13083-970, Brazil
[2]Telematics Engineering Group, Telematics Department, Universidad del Cauca, Popayán 19003, Colombia
[3]Intelligent Management Systems Group, Foundation University of Popayán, Popayán 19003, Colombia

Corresponding author: Oscar Mauricio Caicedo Rendon (omcaicedo@unicauca.edu.co)

The work of O. M. Caicedo Rendon was supported by the Universidad del Cauca (Unicauca), Colombia.

ABSTRACT Management is an essential process to ensure the proper operation of computer networks. There are a lot of proposals to manage software-defined networks (SDN) from their application plane. However, such proposals share some shortcomings related to low automation of network management tasks, long time needed to handle network situations, and the lack of flexibility and workability. In this paper, we introduce JANO to overcome these shortcomings and investigate the feasibility of using automated planning and network functions' virtualization to manage SDN from a vertical management plane. JANO uses hierarchical task networks' planning to generate automatically plans that automate management tasks and reduce the time required by administrators to face network situations. Also, JANO performs an instantiation of the management orchestrator of network functions virtualization to provide flexibility and workability in the generation and execution of plans targeted to addressing network situations. We evaluate JANO in a proof of concept. The evaluation results corroborated that JANO is a feasible solution to manage SDN, since the planning time slightly and linearly increased with the number of primitive tasks that form a plan, the time-consuming needed by administrators to addressing a network situation was short, and the additional traffic was low.

INDEX TERMS Automated planning, hierarchical task network, network functions virtualization, software-defined networks.

I. INTRODUCTION

Management is an essential part of computer networks because it ensures their proper operation, maintenance, administration, and provisioning [1]. In a traditional network environment, management is complex because both forwarding and control planes are in the same network device [2]–[4]. This means that, typically, Network Administrators (hereinafter called Administrators) have to manage each network device separately to handle the network. The Software-Defined Networking (SDN) is a new network paradigm that makes management tasks easier than in conventional networks by centralizing the network control [2]. Nevertheless, SDN also needs to be appropriately managed. In this sense, SDN presents a typical problem of new paradigms, the importance of its management is underestimated. Indeed, the first SDN definitions considered the planes of Data, Control, and Application and they did not take into account the Management plane.

The SDN management is a big challenge that needs to be tackled to prevent technological patching [2]. This patching is generated by the arising of many management solutions, located at the Application Plane, each one performing specific management tasks. There are many proposals for managing SDN from its application plane, such as Set Cover Problem [5], middleboxes [6], interactive visualization [7], Software Defined Infrastructure [8], and RESTful APIs [9]. These proposals share some shortcomings, such as: (*i*) low-automation of network management tasks, (*ii*) long-time required to perform network management tasks; and (*iii*) lack of flexibility and workability (*i.e.,* the ability to work in existing networks) in management. In our previous work [10]–[13], we have also proposed solutions for managing SDN from the application plane by using mashups and the Situational Management (SM). Nevertheless, in such solutions, we have only achieved a medium-automation level granted by mashups themselves.

Ulterior investigations have included a management plane in the SDN architecture [2], [3], [14]–[16] to address the patching aforementioned. In this sense, recently, proposals have emerged to achieve self-driving SDNs [17], [18] by using Machine Learning (ML).

To overcome the shortcomings related to low-automation, long-time, and lack of flexibility and workability in performing network management tasks of current SDN solutions in the Application Plane and unlike the ML-based solutions, in this paper, we propose JANO. JANO extends our previous work [19] and aims to investigate the feasibility of using HTN and NFV to automate the SDN vertical Management Plane (introduced in [2] and modeled by us in [16]) to achieving SDN management. HTN is a planning technique that decomposes tasks into subtasks and leads their execution orderly to meet a goal [20]. We argue that HTN is a crucial tool to manage SDN since it facilitates the work of Administrators by automating many management tasks. Such automation could reduce the management complexity and would eliminate the need for Administrators to address network situations (*e.g.,* faults, wrong configuration, anomalies in SDN controllers or OpenFlow-compliant switches, and change of controllers during runtime) manually one at a time. As a consequence of using HTN, the Administrators can face and overcome network situations in a shorter time.

NFV is a promising technology that targets to decouple Network Functions from the hardware on which they run [21]. Although some research proposals have explored the relationship between SDN and NFV [21]–[23], to the best of our knowledge, the use of NFV to perform SDN management has been scarcely studied. In this sense, we argue JANO might become a valuable tool to manage SDN because it exploits some fundamental NFV characteristics such as flexibility, elasticity, scalability, and workability. In particular, JANO uses the NFV Management and Orchestration (MANO) concept to achieve SDN management via a vertical plane.

To sum up, our key contributions are:

- An architecture based on NFV and HTN, called JANO, intended to manage SDN from a vertical plane; note that we are pioneers on using HTN to offer network management tasks.
- A prototype of the proposed architecture. This prototype was deployed in a traditional environment and in a cloud environment.
- An evaluation by a proof-of-concept that reveals JANO is a feasible solution to manage SDN regarding time-planning, time-consuming and network traffic. Time-planning is the time spent by JANO to automatically generating a plan to cope with a situation in SDN. Time-consuming is the time occupied by an administrator to face a situation in SDN.

The remainder of this paper is organized as follows. In Section II, we present SDN, NFV, HTN, and related work. In Section III, we introduce JANO. In Section IV, we expose and analyze a proof-of-concept of our proposal. In Section V, we provide some conclusions and implications for future work.

II. BACKGROUND AND RELATED WORK

This section presents main concepts of SDN, NFV, and HTN. Also, this section introduces the related work to JANO.

A. BACKGROUND

1) MANAGEMENT OF SOFTWARE-DEFINED NETWORKING

SDN is a network paradigm that aims to make more customizable, flexible and dynamic computers networks [24]. The SDN architecture has four planes [2], [3], [14], [15], [24]: Data, Control, Application and Management. The Data Plane represents forwarding devices formed by two elements [25]: a functional one (software) responsible for holding data and performing forwarding tasks, and a physical element consisting of components such as ports, memory, processor, and storage. The Control Plane aims to make high-level decisions and enforce settings in forwarding devices. This Plane interacts with the other planes by different interfaces called SouthBound with the Data Plane, NorthBound with the Application Plane, EastBound with other controllers, and Management with the Management Plane. The Application Plane represents network applications, such as load balancers, proxies, and firewalls.

The Management Plane aims to offer Management Functions, such as Fault, Configuration, Accounting, Performance, and Security (FCAPS), to the other SDN planes. In this sense, recently, proposals based on Artificial Intelligent techniques have emerged to provide specific network management tasks, such as handover control, energy consumption, reconfiguration, and traffic classification [26]–[29]. In particular, in SDN, there are new proposals that use Machine Learning (ML) to achieve autonomous [17] and cognitive [18] SDN management. Unlike proposals that use ML, it is to highlight that, in this paper, we propose an approach based on HTN and NFV to automate the SDN vertical management plane introduced by us in [16].

2) NETWORK FUNCTIONS VIRTUALIZATION

NFV is a technology that offers flexibility and cost saving in the services provisioning [30]. This technology is intended to decouple Network Functions (NF) (*e.g.,* firewalls and load balancers) from the hardware on which they run [21]. The latter means that a Virtualized Network Function (VNF) can run on any server in the network [31]. The Network Functions Virtualization Infrastructure (NFVI), VNFs, and MANO form NFV. NFVI is the combination of physical (hardware) and virtual resources (software) that form the environment in which VNFs run. A VNF may implement one or more NFs and run on virtual resources (*e.g.,* VMs running services and Dockers running micro-services) [32].

MANO is formed by [33] and [34]: (*i*) an NFV Orchestrator in charge of performing the composition of

NFVI resources, (*ii*) a Virtualized Network Functions Manager (VNFM) responsible for managing the lifecycle of VNFs; and (*iii*) a Virtualized Infrastructure Manager (VIM) in charge of virtualizing, monitoring, configuring and controlling network resources. We argue that NFV MANO is a valuable support for achieving SDN management because JANO inherits from it essential features like flexibility, elasticity, scalability, workability, and mainly the ability to automatically compose management applications.

3) HIERARCHICAL TASK NETWORKS

HTN is an Artificial Intelligence technique that aims to automatically create plans (formed by a set of primitive tasks) that modify initial states to achieve goals [35]. To create plans, HTN uses Planning Problems and Planning Domains [36], [37]. A Planning Problem contains an initial state (i.e., the state to modify) and a goal (i.e., the state to reach). Goals are, for instance, guarantee a particular level of service in a bank network supported by a campus network handled by an SDN controller and migrate such controller without losing services continuity. A Planning Domain contains a set of operators/actions and methods to break compound tasks. A compound task (*e.g.,* replace a link by its backup) is a high-level action that must be decomposed in subtasks up to get primitive ones. A primitive task (*e.g.,* execute the re-start command for a controller) is an indivisible low-level action [38]. In brief, the automatic HTN planning process starts by using the methods to decompose non-primitive tasks into subtasks, aiming to achieve a goal. This process is performed over and over until only obtaining primitive tasks that can be directly executed using the planning operators [20].

HTN planning is helpful in domains that meet certain conditions. First, the domain knowledge must be well structured so that domain goals and activities are expressable as tasks and plans. Second, detailed information is needed to describe how to find a solution (achieve a goal) using a set of tasks (create a plan). These conditions enable HTN planners to be fast and scalable [37] as well as ideal to overcome real-world situations in dynamic systems like computer networks (traditional and based on SDN and/or NFV). The network management domain meets these conditions because, first, FCAPS are well known and definable by atomic tasks forming management plans. Second, different languages, such as the Common Information Model (CIM) [16], JavaScript Object Notation (JSON) [13], and XML [39], can be used to describe these plans intended to achieve the goals defined in the Service Level Agreements (SLAs). It is noteworthy that, in our approach, HTN is used to create an instance of NFV MANO that automatically generates plans targeted to handle *nmsits*. These *nmsits* are unexpected, dynamic, and heterogeneous network management situations, such as a sudden packet loss in a core router of a network backbone and an unforeseen slowness in data transmission over a link between virtual routers [13].

B. RELATED WORK

1) SDN MANAGEMENT

There are a lot of proposals, located at the Application Plane, that does not consider NFV concepts to manage SDN. For instance, FlowCover is an approach that introduces a low-cost monitoring scheme for timely recovering of flow statistics in SDN [5]. This approach optimizes the flow statistics collection scheme and models the polling switches selection as a weighted set cover problem. Furthermore, FlowCover focuses on reducing the network consumption during the traffic measurement and decreases the communication cost of monitoring the network performance by polling requests and replies.

Reference [7] presents an analysis of signaling traffic in SDN to explain the general effects of the communication between the Control Plane and the Data Plane regarding resource consumption and network performance. This analysis includes Administrators in the management loop. Thus, they can configure and reconfigure SDN parameters depending on traffic needs. The Software-defined Unified Virtual Monitoring Function to Large-scale Networks (SuVMF) is a component of a Virtual Network Management System (vNMS) [6] that offers unified control and management for virtualized infrastructures in SDN. SuVMF allows integration of management services and provides control and management abstraction and filtering layer among vNMS, SDN controllers, legacy NMS, and OpenFlow switches. Chowdhury *et al.* [9] present PayLess, a network monitoring framework to collect flow statistics in SDN. This framework uses an adaptive statistics collection algorithm that delivers highly accurate information in real-time generating low network overhead. Also, PayLess offers a flexible application that proposes an abstract view of networks and a uniform way to request statistics from the underlying network devices.

Lin *et al.* [8] present MonArch, an architecture targeted to monitoring SDN. This architecture provides integrated measurement and monitoring functionalities available to users by open APIs. MonArch is based on the Software-Defined Infrastructure concept that allows to design Smart Applications on Virtual Infrastructure Testbed. This concept aims to enable the experimentation with future applications and services. Reference [25] proposes a framework based on SDN for handling fixed backbone networks. This framework supports static and dynamic resource management by including a distributed management layer and a control layer formed by a set of local managers and local controllers, respectively. Yang *et al.* [40] propose Flo-v, a framework intended to provide accurate network monitoring in a virtual SDN. Also, Flo-v monitors traffic in a network hypervisor with lower traffic overhead than existing schemes by using selective and adaptive mechanisms. It is to emphasize that Flo-v uses a virtualized environment that does not consider NFV.

As mentioned in the Background, the first SDN definitions [4], [24] only consider three SDN planes (Data, Control, and Application). Ulterior efforts came up to

TABLE 1. Proposals for SDN management.

Reference	Technology			Metric		
	SDN	NFV	HTN	Time-planning	Time-consuming	Bandwidth
[5]	App Plane					✓
[6]	App Plane					✓
[7]	App Plane					✓
[8]	App Plane					✓
[9]	App Plane					✓
[17]	Mngt Plane					
[18]	Mngt Plane					
[11-13]	App Plane			✓	✓	✓
[16]	Mngt Plane			✓	✓	✓
[21]	App Plane	✓			✓	
[22]	App Plane	✓				✓
[23]	App Plane	✓			✓	
[25]	App Plane					
[40]	App Plane					✓
Our work	Mngt Plane	✓	✓	✓	✓	✓

consider a management plane in SDN [2], [3], [14], [15], but up to now this plane has been barely studied. In fact, recent advances like the Knowledge-Defined Networking (KDN) [17] and the Cognitive Network Management (CNM) [18] are just in an early stage to realize the SDN management plane. KDN and CNM propose to use ML in SDN, aiming at learning the behavior of the network and, in some cases, automatically operate the network according to the learned. The final goal of these advances is to achieve networks where the network management system is fully autonomous.

Unlike the related work, in our previous works, we have proposed approaches for managing SDN by using mashups and SM [10]–[13]. In these approaches, we facilitate the daily work of Administrators when they need to deal with *nmsits* by providing mechanisms to recognize them automatically and dynamically compose mashments (*i.e.,* mashups for facing *nmsits*), as well as the architecture that supports these models and mechanisms. It is noteworthy that, up to now, we always consider the Administrator in the operational management loop. Thus, our approaches are not fully autonomous. In fact, we have only achieved a medium-automation level during the carrying out of network management tasks because of intrinsic mashups limitations.

In the literature, some approaches use SDN and NFV jointly, but they usually do not focus on network management. For instance, the work [21] introduces an architecture that allows to compose and deploy VNFs in SDN rapidly. This work highlights the synergies between NFV and SDN, they are highly complementary. Matias *et al.* [22] present the evolution from a pure NFV solution to an entirely SDN-enabled. In such an evolved solution, SDN supports the NFV deployment and, also, the design of VNFs. The work [23] describes how NFV supports the implementation and deployment of a particular network function (to know a Session Border Controller). This work also provides an overview of the SDN/NFV ecosystem, highlighting its scalability and flexibility.

Table 1 presents diverse investigation works about SDN management, revealing several facts. First, some investigations [21]–[23] consider the NFV/SDN ecosystem but they do not aim at managing SDN, wasting the flexibility and elasticity provided by NFV. Second, to the best of our knowledge, none of the previous works combines NFV and HTN for managing SDN. Third, most of aforementioned works were evaluated using metrics like bandwidth and time-response; by contrast, we consider two additional metrics: time-planning and time-consuming. Fourth, most of these works are located at the Application Plane. Following, directions from ITU and IRTF, our approach is located at the SDN Management Plane.

2) HTN PROPOSALS

HTN has been used in different application domains to achieve a goal by composing plans automatically. For instance, the work [41] proposes a system that creates mission plans for environmental disaster situations. In this work, the authors introduce a planning system that selects the primitive tasks necessary to addres particular disasters. Ullrich and Melis [20] propose PAIGOS, a framework that uses pedagogical knowledge to create structured courses. PAIGOS offers operators and methods to achieve multiple learning goals by learners. Chen *et al.* [35] combine Markov decision processes and HTN plans to compose Web Services automatically. In particular, web service composition is to create multiple executable plans in business processes. Thus, they offer a more flexible and optimal solution to business customers.

Recently, the HAUTO framework offers a solution centered in the user to compose telecommunication services [38] by using three main modules: (*i*) a requests processing module that transforms natural language and context information to computer language, (*ii*) an automatic composition that creates plans using HTN; and (*iii*) an execution environment for convergent services that executes the created plans. Milliez *et al.* [42] propose a solution to carry out human-robot collaborative activities. This solution uses HTN planning to describe the robot tasks and a dynamic model to track its movements.

From Table 2, we can conclude that HTN has been successfully used in a wide range of domains, such as education, web services composition, convergent telecommunication

TABLE 2. HTN-based proposals.

Reference	Domain	Metric		
		Time-consuming	Time-planning	Traffic
[20]	Education		✓	
[35]	Web Services		✓	✓
[38]	Convergent services		✓	
[41]	Environmental		✓	
[42]	Robotic		✓	
Our work	Network Management	✓	✓	✓

services, and environmental. Unlike the research mentioned above, our approach focuses on network management. In this sense, it is important to highlight that we are pioneers in using HTN and NFV to manage SDN.

III. JANO

This section presents some motivating scenarios. Also, this section introduces JANO and its architectural elements.

A. MOTIVATING SCENARIOS

First, let us consider a scenario in which a live migration of SDN controllers must be performed because of technological obsolescence or because new network services are required during the expansion of a Campus Network. In this scenario, it is evident that the task of live migration needs to be carried out without losing campus services continuity. Thus, this task should be performed automatically and in the shortest time possible. Note that, the live migration of SDN controllers may be time-consuming if Administrators use non-integrated solutions; it is needed to turn off the current controller and turn on the new one, implying associate and dissociate the involved switches. Also, since Administrators are not expert developers, writing a management script to cope with this task is hard for them.

Let us consider a second scenario in which providers offer SDNs formed by multiple slices that, in turn, provide Internet services to companies such as banks, hospitals, and governments. Sometimes, unexpected situations can break the connection and stop the normal functioning of services offered by these companies through their corresponding slices. These providers offering SDN slices often use human Administrators to overcome such situations. They are responsible for redirecting, re-configuring and, in the worst case, rebooting the controller and all the OpenFlow switches that form the network. As a result, overcoming a situation in an SDN context may consume a long time because human intervention usually tends to be slow and error-prone.

Let us consider a third scenario where an Administrator needs to perform configuration and monitoring tasks in a network formed by a lot of devices supporting various protocols, such as OpenFlow and ForCES (Forwarding and Control Element Separation). To perform these tasks, a lot of resources for processing and storage are necessary. If the Administrator decides to use a management application running on a physical machine, some problems like overload and low performance could arise. These problems are related to the network dynamicity that leads to such an application to consume more and more resources (memory, processing and storage). Nowadays, Cloud Computing represents a suitable alternative to face this scenario since resources are offered on demand. By using Cloud Computing, management applications may be supplied by Software as a Service (SaaS). In this sense, when JANO is a SaaS, the customers (*e.g.,* Administrators, SDN Providers, and Virtual Network Providers) obtain a flexible and workable tool that allows automatically performing network management tasks.

Traditional SDN management solutions present several shortcomings to address the above scenarios: (*i*) low automation since in most cases, Administrators must manage the controllers and devices singly, (*ii*) time-consuming, Administrators need a lot of time to face and overcome situations (*e.g.,* faults, wrong configuration of devices and controllers, and traffic anomalies); and (*iii*) lack of flexibility and workability, typically, Administrators have just one option (coming from their knowledge) to address and solve a situation. The above shortcomings lead to loss of money for both providers and customers. Therefore, it is evident the need for automating the decision-making process to face and overcome *nmsits*. In response to these shortcomings, we propose JANO.

B. OVERVIEW

JANO is a solution that automates the SDN management tasks by using HTN planning and NFV MANO. In JANO, HTN is a crucial enabler for providing a more straightforward and less time-consuming way of generating possible solutions (plans) to face any *nmsit*. The planned solutions are built by combining different alternatives from known compositions used for dealing with similar *nmsits*. Thus, an Administrator has a set of options instead of only one from his/her own experience. JANO also offers a user-friendly interface to display executed and on running plans. In turn, MANO provides to JANO essential Cloud Computing features like flexibility and workability.

Figure 1 depicts the overall operation of JANO: (*1*) it collects data from the Managed SDN to recognize *nmsits*, (*2*) it automatically builds plans (*i.e.,* solutions formed by a set of primitive management tasks) to manage recognized *nmsits*, (*3*) it executes plans built to deal with *nmsits*, (*4*) it stores and publishes plans; and (*5*) it depicts information about plans (executed and on running) and their primitive tasks to facilitate the decision-making process when *nmsits* arise.

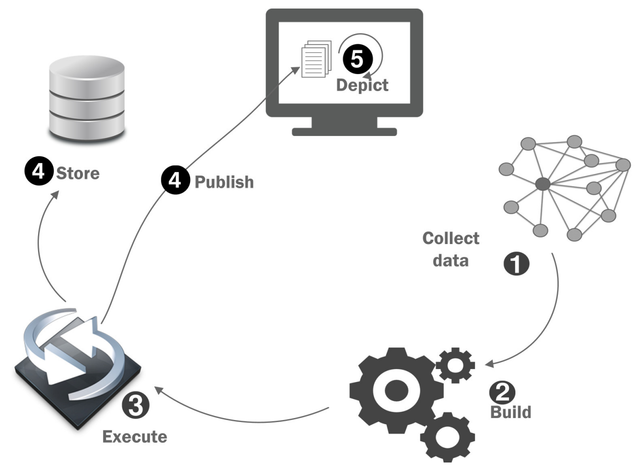

FIGURE 1. Overview - JANO functioning.

C. ARCHITECTURAL LAYERS AND ELEMENTS

Figure 2 introduces and details the JANO architecture that is located in the vertical SDN Management Plane proposed by [2] and [3] and modeled by us in [16]. This architecture has three layers: Adaptation, Planning, and Presentation. The Presentation Layer presents a Graphical User Interface (GUI) in the client-side to support the interaction between Administrators and JANO. The Planning Layer automatically recognizes network situations captured from the Managed SDN via the Adaptation Layer. Also, the Planning Layer automatically composes and executes plans to overcome such recognized situations and sends these plans to the Presentation Layer. Furthermore, the Planning Layer is responsible for storing and publishing plans. The Adaptation Layer is in charge of retrieving management information from the Managed Network and transferring it to the Planning Layer. To support the execution of plans, the Adaptation Layer receives management operations (*i.e.,* primitive tasks) from the Planning Layer to transfer them to the Managed SDN. In the next paragraphs, we describe in detail the layers and elements of JANO.

1) PRESENTATION LAYER

This layer via a Management GUI offers to Administrators mainly two functionalities. The first one is to visualize information about HTN-based solutions (*i.e.,* Plans, see Listing 9) that have already been executed or are currently running. This information is retrieved from the HTN Solution Repository located at the Planning Layer by using the Pr-Pl reference point. In JANO, a reference point is an interface that supports bi-directional interaction between two points located at different architectural layers. The second functionality is to customize HTN-based solutions for their reuse. This customization involves to retrieve and write data in the Planner Repository (containing the Planning Domains and Planning Problems, see Listing 7 and Listing 8) and the HTN Solution Repository by Pr-Pl.

The REST (REpresentational State Transfer), SOAP (Simple Object Access Protocol), and repositories drivers are options to instantiate Pr-Pl. Note that REST and SOAP are suitable models for providing request-response interactions over the HTTP (HyperText Transfer Protocol). In turn, specific drivers are helpful to directly accessing the repositories (back-end) from the Management GUI (front-end). Thus, if the Presentation Layer reads and writes directly by drivers, it provides its functionalities as follows. First, when the Administrator asks information about one or more plans (on running or executed, to visualization or customization) by the Management GUI, this layer sends requests to retrieve from the Planner Repository and the HTN Solution Repository the queried information. Second, this layer receives, decodes (from text-plain, JSON, or XML) and presents (in HTML) the corresponding responses that contain information about Plans, Planning Domains and Planning Problems.

2) PLANNING LAYER

This layer is in charge of automatically generating plans intended to manage SDN. This generation follows two phases: (*i*) automatic recognizing of situations to deal with; and (*ii*) automatic planning of solutions targeted to overcome the identified situations. This layer uses a Recognizer and a Planning Orchestrator to perform the phases mentioned above.

Figure 3 introduces the Recognizer that conducts the automatic recognition of *nmsits* by comparing the samples with the patterns. The Recognizer is formed by: Pattern Repository, Sensing, and Matching Mechanism. The Pattern Repository contains patterns (in JSON) written by networking experts. Patterns hold information representing *nmsits* that could arise in the Managed SDN. Every pattern follows the *nmsits* conceptual model that we proposed in [13]: $[\{NAMESIT : namesit, NMSIT : [\{nmsit_1\}, \{nmsit_n\}]\}]$. Where, $NAMESIT$ is the name of the $NMSIT$ set and $nmsit_n$ is given by: $[\{SITUATION : situation, EAC : [\{eac_1, eac_n\}]\}]$. Here, $SITUATION$ is the particular name of an *nmsit* and EAC represents the collection of entities, attributes, and constraints involved in such situation. The structure of eac_n is: $[\{ENTITY : entity_n, AC : [\{ac_1, ac_n\}]\}]$. Where, $ENTITY$ is any entity involved in a *nmsit* and AC represents the set of attributes and constraints of such an entity. Here, ac_n is: $[\{ATTRIBUTE : attribute_n, CONSTRAINT : constraint_n\}]$. Note that constraints serve to determine when a situation happens. An example of a simple *nmsit* is: *namesit = {drop of transmitted packages}, situation = {unexpected drop of sent packages in an OpenFlow-enabled virtual switch},* and *eac* = [{*ENTITY : cyscoSwitch2960, ac :* [{*ATTRIBUTE : sentPkg, CONSTRAINT : < 90%*}]}]. An example of an *nmsit* formed by two ones is: *namesit* = {*link has an unexpected behavior*}, $situation_1$ = {*a switch has an unforeseen overload in both memory and processor*}, $eac_1 = [\{ENTITY : cysco100, ac : [\{ATTRIBUTE : processor, CONSTRAINT :> 97\%\}, \{ATTRIBUTE : memory, CONSTRAINT :> 95\%\}]\}$, $situation_2$ = {*a switch has a sudden overload in both memory and processor*}, and

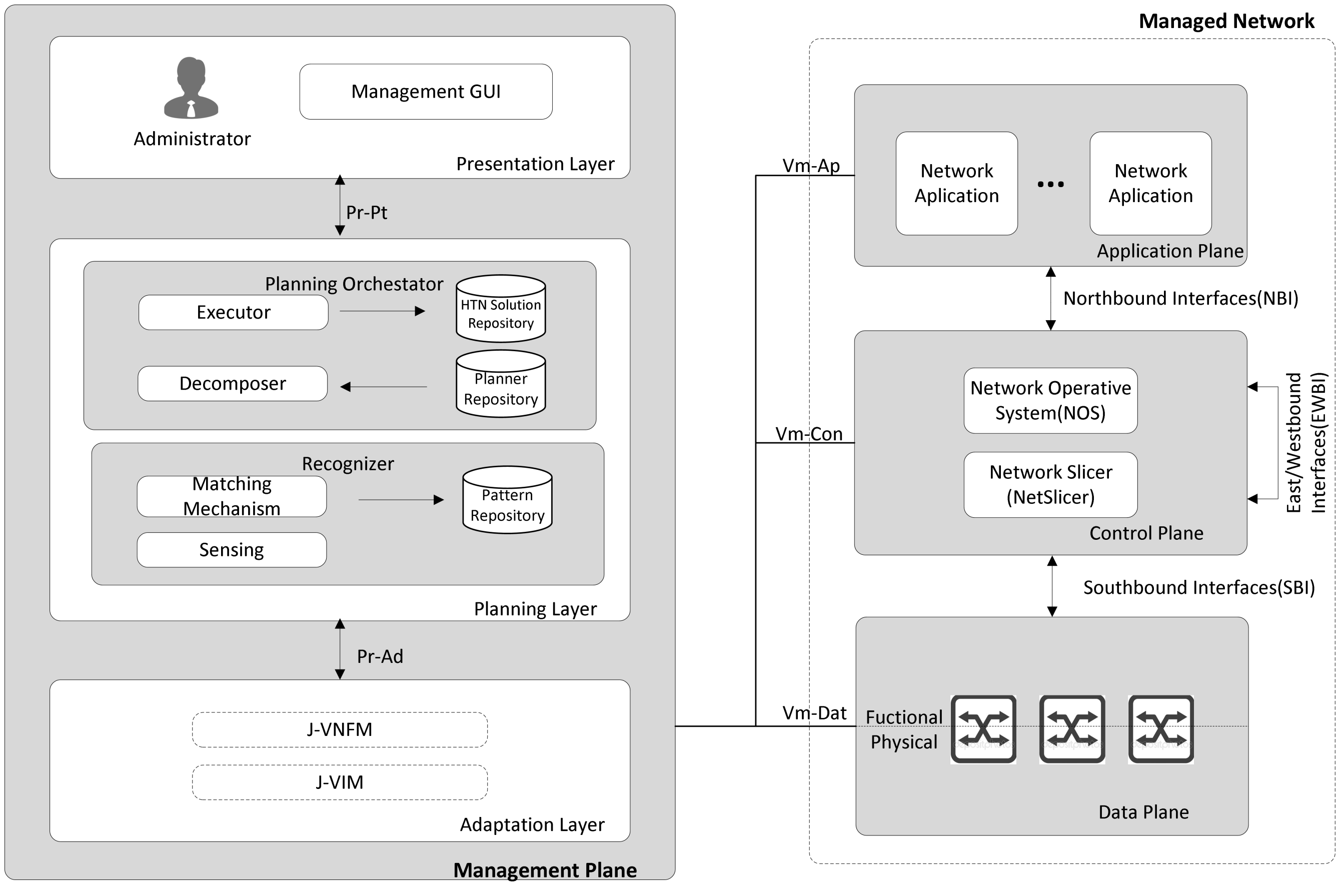

FIGURE 2. **JANO architecture.**

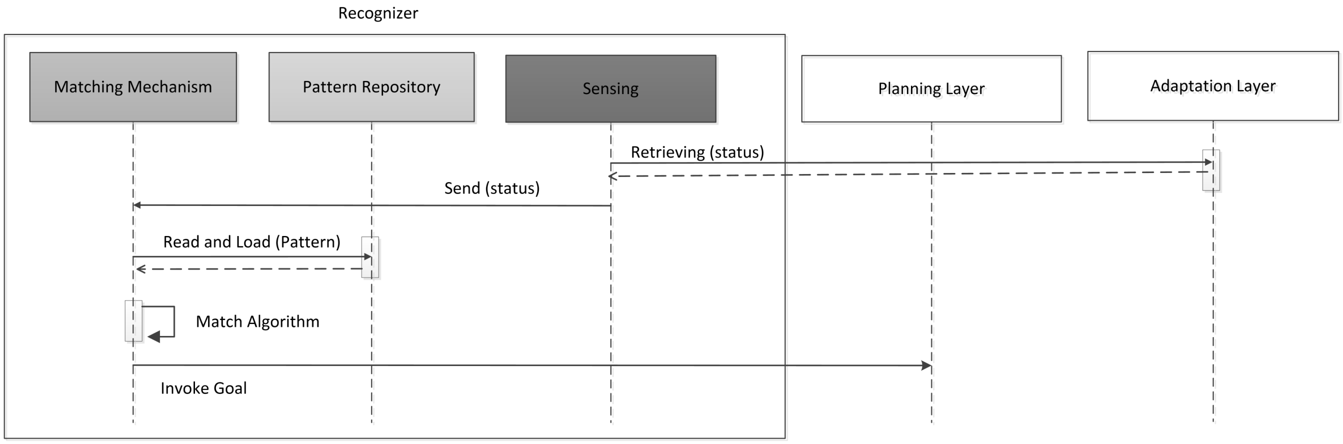

FIGURE 3. **Automatic recognition.**

$eac_2 = [\{ENTITY : openvswitch, ac : [\{ATTRIBUTE : proces, CONSTRAINT :> 91\%\}, \{ATTRIBUTE : mem, CONSTRAINT :> 91\%\}]\}]$.

In the Recognizer, the Sensing is in charge of retrieving network management information from the Managed SDN and delivering it as streaming to the Matching Mechanism.

The Matching Mechanism automatically recognizes a situation as follows (see Figure 3). First, it receives samples (*i.e.*, management information) coming from the Sensing. Second, it loads and reads situation patterns from the Pattern Repository. Third, it conducts matching operations by comparing samples with patterns. Matching operations can be carried out by using different algorithms, such as PHREAK [43] and RETE [44]. Fourth, if a situation is detected (*i.e.*, there is a match), the Planning Orchestrator is invoked to build and execute an automatic plan that aids to overcome such a situation.

The Planning Orchestrator is in charge of composing plans automatically and executing them for coping with the network situations. Figure 4 depicts the automatic composition of a plan to overcome a particular *nmsit* that has been identified by the Recognizer. The Planning Orchestrator has three elements: the Decomposer, Executor, HTN Solution Repository, and Planner Repository. When the Planning Orchestrator receives an invocation, the Decomposer retrieves a planner from the Planner Repository. A planner uses two descriptors: Planning Domain and Planning Problem. Experts in automated planning use HTN notation to write these descriptors. The Planning Domain is to define the useful operators and methods to compose plans automatically. The Planning Problem is to establish the initial state and the final state that a plan must reach during its execution.

Algorithm 1 HTN Planning

```
SHOP2($s, T, D$)
$P = empty - plan$
$T0 \leftarrow \{t \in T\}$
LOOP
if $T0 = \oslash$ then
    return $P$
end if
choose any $t \in T0$
if $t$ is a primitive task then
    $A \leftarrow \{a : a$ is an instance of an operator in $D\}$
    if $A = \oslash$ then
        return failure
    end if
    append $a$ to $P$
    modify $T$ by removing $t$
    $T0 \leftarrow \{t \in T$  no task in $T$ is constrained to preced $t\}$
else
    $M \leftarrow \{m : m$ is an instance of a method in $D\}$
    if $M = \oslash$ then
        return failure
    end if
    choose a $m \in M$
    modify $T$ by removing $t$
end if
REPEAT
```

The Decomposer follows the Simple Hierarchical Ordered Planner (SHOP2) planning algorithm described in [38]. SHOP2 is an automated system for planning HTN tasks in the same order that they will be executed later [36]. The Algorithm 1 depicts a simplified version of the SHOP2 planning procedure. The inputs of the algorithm are the initial state s, a partially ordered set of tasks T, and a domain description D. The output is the generated plan that is initially empty. First, a task $t \in T$ is selected. If t is primitive (*i.e.*, if t can be accomplished directly using an action that is an instance of a planning operator), then, SHOP2 finds an action a that matches t and whose preconditions are satisfied in s. If t is compound (*i.e.*, a method needs to be applied to t to decompose it into subtasks), then, SHOP2 chooses a method m that will decompose t into subtasks. The process repeats until achieving the goal state.

To detail the functioning of the Planning Orchestrator, let us consider two situations. In the first one (see Figure 5), an SDN provider needs to perform a live migration of its controller because of: (*i*) technological obsolescence, (*ii*) new features required; or (*iii*) workload overload. Here, it is needed an HTN planner to migrate an SDN controller, involving turning on the new controller and turning off the running controller; the network switches must start to be handled by the new controller without losing continuity of services. In this situation, first, it is needed to define the Planning Problem (see Listing 1) that contains both the initial state of controllers (*i.e.*, the situation to face) and the desired state to reach (*i.e.*, the goal: live migration of controller by providing service continuity). Second, it is needed to define the Planning Domain (see Listing 2) useful to achieve the goal, containing a set of all possible tasks (*i.e.*, operators) and ways of decomposing them (*i.e.*, methods). In this Domain, the method allows modifying the states of controllers by calling operators. Once operators are invoked, they are executed to carry out the live migration of SDN controller and, so, achieve the goal.

```
1. when current_Ctrl needs to be changed //initial state
2. then turn_off(current_Ctrl) and turn_on(new_Ctrl) //
     desired goal
```

Listing 1. **Planner problem - live migration of controllers.**

```
1. turn_off //operator to disable a controller
2. turn_on  //operator to enable a controller
3. modify switches control //method to change the ctrl of
     switches
   3.1 precondition //check current states
   3.2 call operator //perform the changing operation
```

Listing 2. **Planner domain - live migration of controllers.**

In the second situation, an SDN provider offers network services to a prestigious bank. To grant an excellent service availability, the provider uses two links that transport OpenFlow messages, one as main and other as backup. Here, it is needed an HTN planner for switching the state (*i.e.*, on and off) of these links every time a fail happens. In particular, first, it is needed to define the Planning Problem (see Listing 3) that contains both the initial state (*i.e.*, the situation to overcome)

```
1. when main_OFlink is in fail //initial state
2. then turn_off main_OFlink and turn_on backup_OFlink //
     desired goal
```

Listing 3. **Planner problem - links situation.**

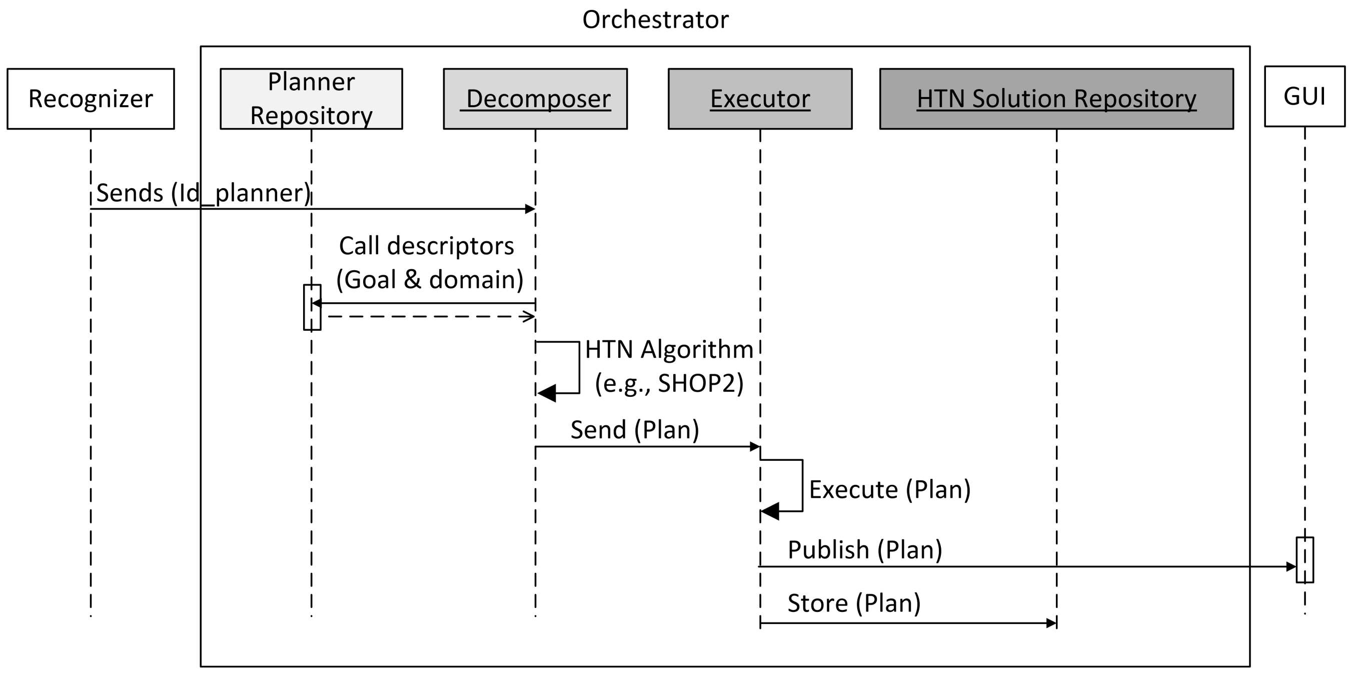

FIGURE 4. Automatic composition.

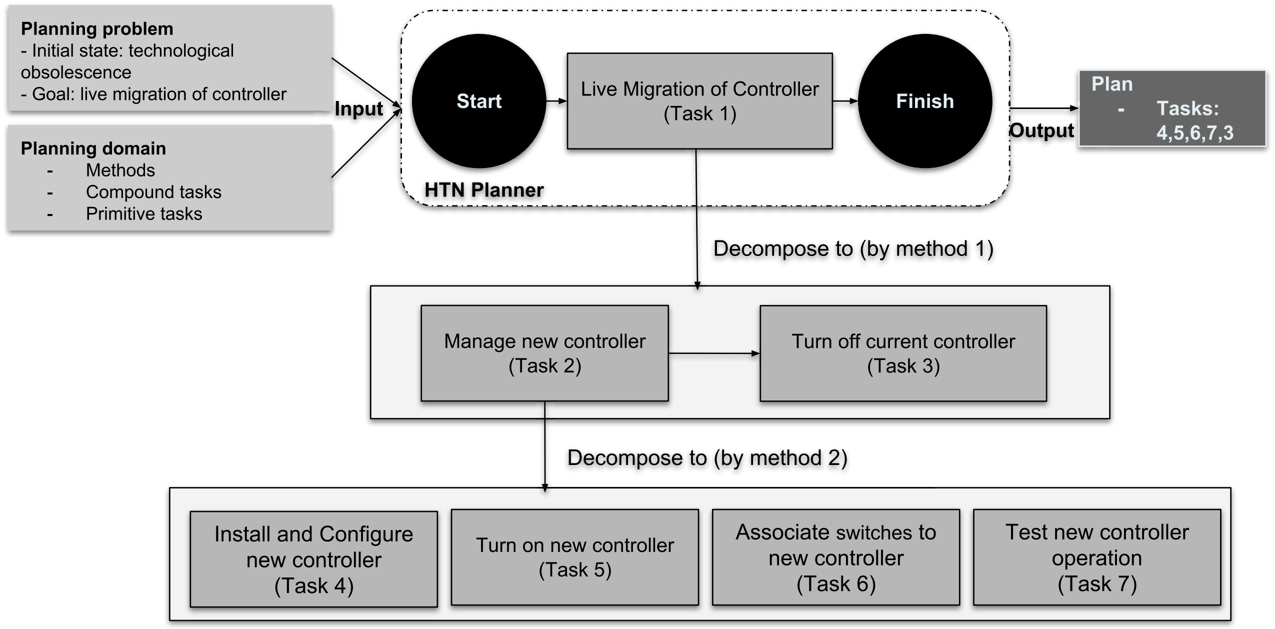
Planning problem
- Initial state: technological obsolescence
- Goal: live migration of controller
Planning domain
- Methods
- Compound tasks
- Primitive tasks
Input
Start
Live Migration of Controller (Task 1)
Finish
HTN Planner
Output
Plan
- Tasks: 4,5,6,7,3
Decompose to (by method 1)
Manage new controller (Task 2)
Turn off current controller (Task 3)
Decompose to (by method 2)
Install and Configure new controller (Task 4)
Turn on new controller (Task 5)
Associate switches to new controller (Task 6)
Test new controller operation (Task 7)

FIGURE 5. Live Migration of Controllers by HTN.

of links and the desired state (*i.e.*, the goal). Second, a set of all possible tasks (*i.e.,* operators) and ways of decomposing them (*i.e.*, methods) must be defined in the Planning Domain (see Listing 4) for achieving the desired goal. In this Domain, the method allows modifying the states of links by calling the operators. Again, once operators are invoked, they are executed to achieve the desired goal.

After plans generation, they are automatically executed. During such an execution, the Planning Layer sends management operations (*i.e.*, primitive tasks) to the Adaptation Layer by the Pl-Ad that, like Pr-pl, may be instantiated by using REST or SOAP. In turn, the Adaptation Layer

```
1. turn_off  //the operator to disable a link
2. turn_on   //the operator to enable a link
3. modify    //the method to modify states of links
   3.1 precondition //check current states
   3.2 call operator //perform the action
```

Listing 4. Planner domain - links situation.

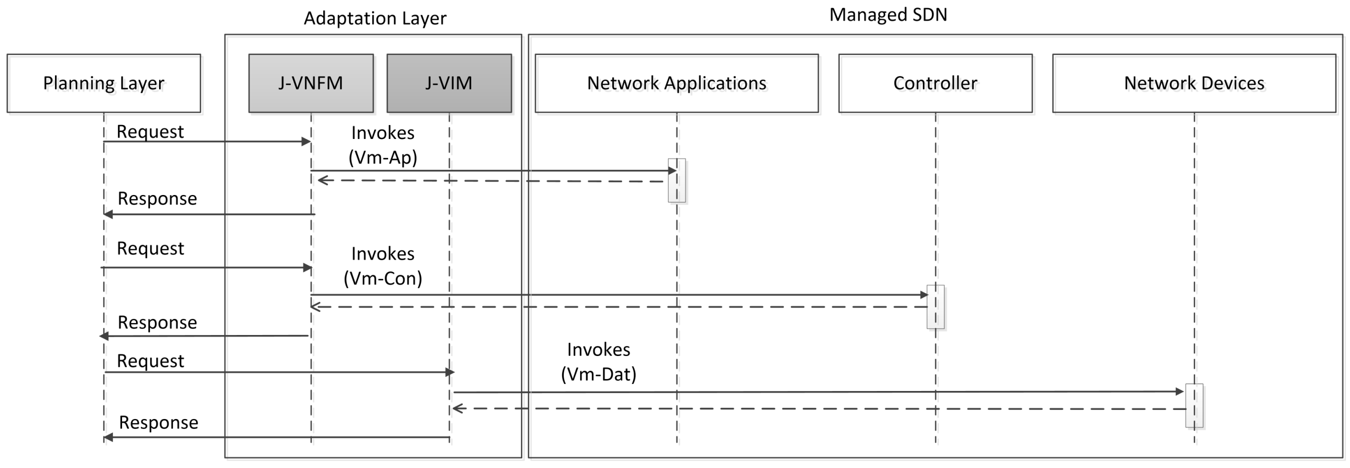

FIGURE 6. **Adaptation layer example.**

sends these primitives to the Managed SDN that is responsible for their running. The executed and on running plans are automatically stored in the HTN Solution Repository and published for their depicting in the Presentation Layer. Besides, these plans can be used by Administrators for further reference when they need to know previous actions carried out on the Managed SDN.

3) ADAPTATION LAYER

This layer is responsible for hiding the complexity and heterogeneity of the Managed SDN. In this sense, the Adaptation Layer is in charge of retrieving information from the planes of Application, Control, and Data to send this information to the Planning Layer. Furthermore, the Adaptation Layer receives management operations from the Planning Layer to forward them to the Managed SDN. The Adaptation Layer has two entities, called J-VNFM and J-VIM. These entities are based on VNF Manager and Virtualised Infrastructure Manager, respectively. In turn, J-VIM is in charge of virtualizing and managing the network infrastructure (physical and virtual) including the three traditional SDN planes.

The Adaptation Layer and the Managed SDN interact by three reference points, named Vm-Dat, Vm-Con, and Vm-Ap, that allow retrieving management information and forwarding management operations. Vm-Dat is to interact with the Data Plane, Vm-Con is to communicate with the Control Plane, and Vm-Ap is to interact with the Application Plane. REST/SOAP are also options to instantiate Vm-Dat, Vm-Con, and Vm-Ap.

Figure 6 presents an example of using REST to instantiate Vm-Dat, Vm-Con, and Vm-Ap. First, HTTP-requests are sent to J-VNFM or J-VIM (*i.e.,* depending on the SDN Plane to manage) from the Planning Layer. Second, J-VNFM or J-VIM sends HTTP-requests to URIs that point to the Managed SDN. An example of HTTP-request encoded on JSON is $http : //MashmentSys/Wrapper/Beacon/getSwitches$ that offers to the Planning Layer, a list containing basic information about switches handled by a Beacon OpenFlow Controller. Third, the Managed SDN carries out the requested management tasks. Fourth, J-VNFM or J-VIM receives responses containing data encoded on JSON from the Managed SDN. An example of data received is $[\{IPCTRL : ipCtrl, LIST : [\{IDSWITCH : id_1, IPSWITCH : ip_1\}, \{IDSWITCH : id_n, IPSWITCH : ip_n\}]\}]$. In this structure, $IPCTRL$ is the IP address of the Beacon Controller, and $LIST$ is the corresponding switches list. Such list has a set of identifiers ($IDSWITCH$) and IP addresses ($IPSWITCH$) of switches. Fifth, J-VNFM or J-VIM sends such data via HTTP-responses to the Planning Layer. Finally, concerning the Adaptation Layer, it is relevant to highlight that Administrators never access it directly. The Planning Layer always conducts this access.

IV. PROOF-OF-CONCEPT

To test the feasibility of JANO, first, we implemented a prototype. Second, we evaluated such a prototype regarding time-planning (*i.e.,* time required by the Planning Orchestrator for building an automatic plan), time-consuming (*i.e.,* time needed by Administrators for dealing with situations) and network traffic. The network traffic evaluation is to measure the additional traffic generated by JANO.

A. PROTOTYPE

Figure 7 depicts the deployment of JANO. The Presentation Layer was implemented as a typical Web-based GUI. The Planning Orchestrator was developed by using JSHOP2 [38] that is a Java-based implementation of SHOP2. The Matching Mechanism was built using the PHREAK algorithm provided by JBoss Drools [43]. It is important to highlight that we deployed JANO in two ways. The first one by using OpenStack [45]. Thus, we provided JANO as a Service (JaaS) and leveraged important features from OpenStack like flexibility, elasticity, and scalability. The second deployment was performed in conventional servers.

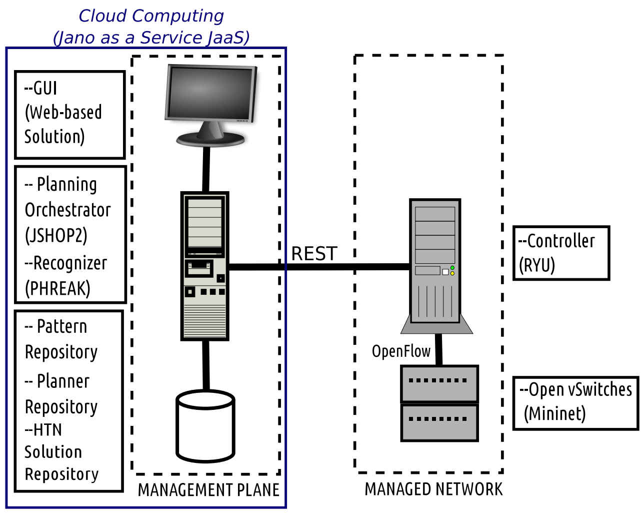

FIGURE 7. JANO prototype.

In our test environment, the Managed SDN has a controller, virtual switches, and hosts. Virtual switches (*i.e.,* OVSK) and hosts were deployed on Mininet. Mininet allows emulating networks based on OpenFlow that run on Oracle VM VirtualBox [46]. The controller was deployed by using Ryu [47]; a platform that provides software components with well-defined APIs to create SDN applications. It is important to mention that we implemented and deployed in a Java-based application the communication from JANO to Ryu by following the REST architectural model.

1) OPERATIONAL EXAMPLES

In order to detail the operation of the JANO prototype, we returned to *nmsits* (planner problems) presented in Listings 1, 2, 3, and 4. Listing 5 presents the Planning Problem, by using HTN notation, of the *nmsit* where the live migration of a controller (see Listing 1) is performed because of technological obsolescence. This Planning Problem includes the description of the initial state where old_controller is on and, also, the description of the goal that is the replacement/change of such a controller by a new one. In turn, Listing 6 presents the Planning Domain, by using HTN notation, containing operators and methods that allow decomposing complex tasks in primitive ones for reaching the desired goal. It is important to highlight that Planning Domains are fundamental for the Planning Orchestrator because these Domains include all actions useful for addressing any *nmsit*.

```
(defproblem problem basic
 ((on old_controller))
 ((replace old_controller with new_controller))
);defproblem
```

Listing 5. HTN problem - live migration of controllers.

```
(defdomain basic(
 (:operator (!replace ?a) ()() ((on ?a)))
 (:operator (!disable ?a) ((on ?a)) ())
 (:method (replace ?x ?y)
 ((on ?x) (not (on ?y)))
 ((!disable ?x) (!enable ?x)))
 ); basic
); defdomain
```

Listing 6. HTN domain - live migration of controllers.

```
(defproblem problem basic
 ((on link2))
 ((exchange link2 link1))
);defproblem
```

Listing 7. HTN problem for links situation.

Listing 7 presents the Planning Problem, by using HTN notation, of the situation where the state of two links (see Listing 3) that transport OpenFlow messages are exchanged to enable a link and disable the other one. This Planning Problem includes the description of the initial state where link2 is on and, also, the description of the goal where the status of links must be exchanged. Listing 8 presents the Planning Domain, by using HTN notation, containing operators and methods that allow decomposing tasks. Again, note that Planning Domains are fundamental for the Planning Orchestrator because it includes all necessary actions to face *nmsits*.

```
(defdomain basic(
 (:operator (!enable ?a) ()() ((on ?a)))
 (:operator (!disable ?a) ((on ?a)) ())
 (:method (exchange ?x ?y)
 ((on ?x) (not (on ?y)))
 ((!disable ?x) (!enable ?x)))
 ); basic
); defdomain
```

Listing 8. HTN domain for links situation.

```
1 plan(s) was found:
Plan #1:
Plan cost: 2.0 //This plan has 2 primitive tasks.
(!disable link2)
(enable link1)
────────────────
Time used (milliseconds)=0.004
```

Listing 9. HTN solution for links situation.

Listing 9 presents the plan built automatically by the Planning Orchestrator. This plan uses two primitive tasks to overcome the links situation described in the operational examples. The primitives are to disable link2 and enable link1 that was initially disabled. The Planning Orchestrator also depicts: (*i*) the cost of the plan that is the sum of operators cost; and (*ii*) the time (in milliseconds) spent to generate the plan automatically. Note that the cost may be calculated by using different metrics like the number of tasks. The estimated cost could be useful to select the best plan according to a ranking.

B. EVALUATION AND ANALYSIS

We evaluate the behavior of JANO regarding time-planning, time-consuming, and network traffic. It is important to mention that in all evaluations, we took 32 measurements with 95% confidence level.

During the **time-planning** evaluation, we generated the plans described in the operational examples (see Listings 1, 2, 3, and 4) and measured the corresponding time-planning when JANO is deployed in a conventional Web server and in a cloud environment (*i.e.,* JaaS) based on OpenStack. Table 3 presents the time-planning results when the number of primitive tasks included per plan was 2, 10, 20, 40, 80, and 100. The test results reveal, first, that time-planning increases linearly with the number of tasks. Second, the planners described in the operational examples have equal time-planning behavior because they have the same number of primitive tasks. Third, as expected, the time-planning in JaaS is lower than in traditional JANO.

TABLE 3. Time-planning for a single plan.

Amount of primitive tasks	Time-planning (ms) traditional way	Time-planning (ms) JaaS
2	9.2	8.6
10	43.2	19.2
20	68	32.5
40	105.2	54.1
80	149.9	116.8
100	171.2	149.7

In the time-planning evaluation, we also conducted tests for 2, 4, 8, and 16 automatic plans (50% of plans for the links situation and 50% of plans for the live migration situation). In these tests, we varied the number of primitive tasks, using the same two primitives repetitively, per plan from 4 to 28. Figures 8 and 9 present the time-planning results for JANO deployed in a traditional way and as a service, revealing that such time slightly and linearly increases with the number of primitives. Furthermore, as expected, the time-planning in JaaS is lower than in traditional JANO.

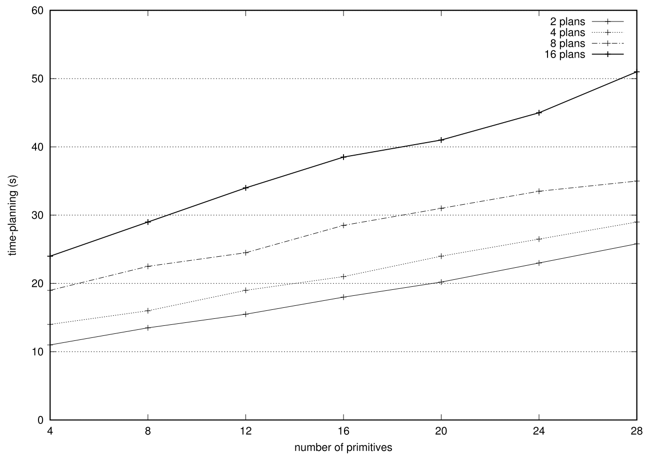

FIGURE 8. Time-planning for several plans in traditional JANO.

JANO is a feasible solution regarding time-planning because, first, it can automatically generate plans to manage

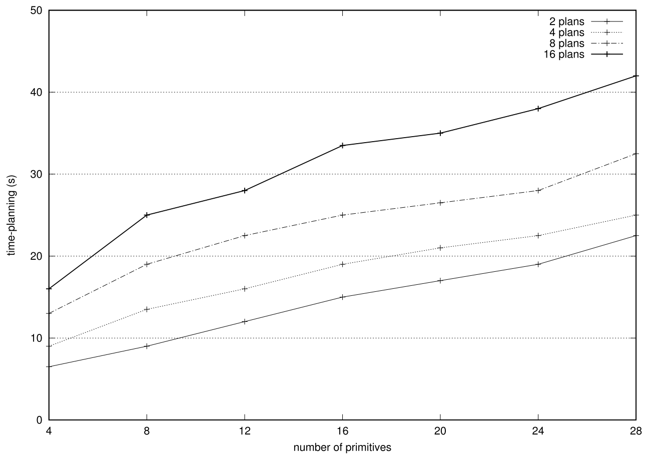

FIGURE 9. Time-planning for several plans in JaaS.

network situations in SDN. Second, it spends a negligible time to create these plans taking as reference the Web Applications optimal time (($r \leq 100ms$ [48]). Consequently, concerning time-planning, we have corroborated the importance of using HTN and NFV to accomplish SDN management.

For the **time-consuming** evaluation, we use the Keystroke-Level Model (KLM) because it allows to estimate the time that Administrators spend to carry out management tasks supported on computer mouse and keyboard. In KLM [49], each task is modeled as a sequence of actions: (i) Hold or release the mouse $\rightarrow B = 0.1s$, (ii) Press and release a key $\rightarrow K = 0.2s$, (iii) Type a string $\rightarrow nk * 0.2s$, (iv) Move the hand from mouse to keyboard or vice-versa $\rightarrow H = 0.4s$; and (v) Point the mouse $\rightarrow P = 1.1s$.

In JANO, to manage network situations, the Administrator does not need to perform any action. He only needs to launch the JANO Web-based GUI to see the plans that are on running. The corresponding actions sequence is as follows: ***(1)*** Point the mouse to select the direct link to the JANO Web application $\rightarrow P$, ***(2)*** Hold and release the mouse to access such an application $\rightarrow 2B$. ***(3)*** Point the mouse to select a particular plan that is on running $\rightarrow P$; and ***(4)*** Hold and release the mouse to see details of the selected plan $\rightarrow 2B$. According to this sequence, the estimated time for addressing a network situation is $\mathbf{T} = \mathbf{2P} + \mathbf{4B}$. Then, we expect that, by using JANO, Administrators spend around $2.6s$ to cope with a network situation. We also conducted, an experimental study to measure the time-consuming. In the study participated 30 people whose age ranged from 18 to 28 and had frequently used web tools. In the experiment, the result of the average time was $2.7s$ and, therefore, KLM predictions were corroborated.

Figure 10 compares JANO and Rich Dynamic Mashments (RDM) [13] regarding time-consuming when they are used to address the *nmsit* presented in the operational example. An RDM is a tunable mashup that combines diverse types of resources from multiple providers and automates the investigative and control aspects of SM, aiming to facilitate the

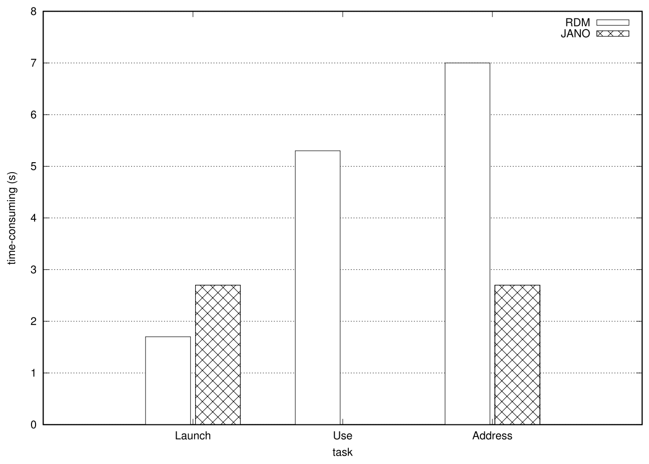

FIGURE 10. **Time-consuming: JANO vs RDM.**

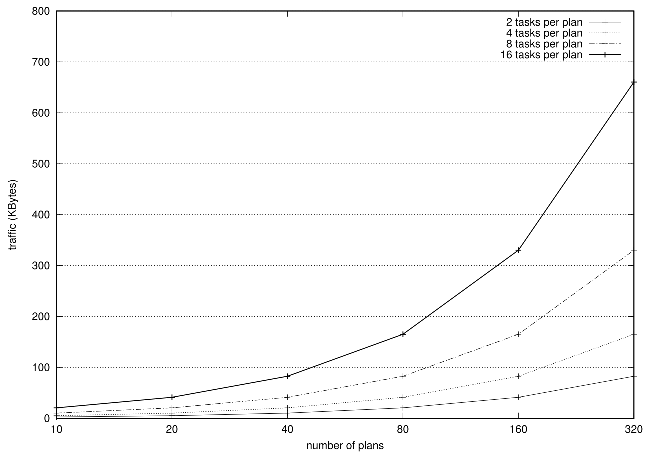

FIGURE 11. **Traffic versus plans launched.**

work of network administrators. The time-consuming for addressing *nmsits* by RDM is $T_{con:rdm} = T_{lau} + T_{use}$. Where T_{lau} and T_{use} are the time spent by Administrators for launching and using RDM, respectively. As $T_{lau} = 1.7s$ and $T_{use} = 5.3s$, $T_{con:rdm} = 7s$. It is remarkable that this time is more than twice the JANO time-consuming (approximately $2.7s$).

JANO is a feasible solution regarding time-consuming because, first, HTN and NFV support the automation of SDN management tasks. In fact, Administrator intervention is only necessary to get information about plans on running (or executed). This intervention is short since it involves few graphical elements. Furthermore, the time response to obtaining such information is quick, and it is only related to read and write in the repositories (deployed as databases). Consequently, we can state that JANO aids to solve the low-automation problem in SDN management.

The **network traffic** evaluation aims to analyze the impact of JANO in an SDN. Here, we measured the traffic on the controller because it is a core component of SDN. Furthermore, we only tested traditional JANOS since the traffic behavior from and to the controller is regardless the JANO deployment. Specifically, we measured the traffic when the controller deals with requests coming from 10, 20, 40, 80, 160 and 320 plans with 2, 4, 8, and 16 primitive tasks. Figure 11 depicts the network traffic evaluation results, revealing the OpenFlow messages increment linearly with the number of primitives. In fact, one primitive task generated just one OpenFlow message (approximately 0.129 kB). Therefore, we have a relation one to one that allows knowing in advance the traffic that an automatic plan can generate (*e.g.,* xin the example of links, two primitives represent 0.258 kB).

JANO is a feasible solution regarding network traffic because, first, it can perform SDN management tasks with a low traffic overhead. It is noteworthy that JANO does not affect the traffic between switches, it only increases the number of OpenFlow messages between the controller and switches.

As we already analyzed JANO quantitatively, subsequently, we perform some brief comments about its flexibility, workability, and automation. Flexibility is the ability to adapt to changes; this involves scalability and elasticity [50]. In this sense, scalability is the ability to handle a growing amount of work without suffering significant degradation in quality [51]. In turn, elasticity is the automatic adaptation of resources according to the current demand [52]. Scalability and elasticity (involving flexibility) are mainly leveraged by JaaS (*i.e.,* JANO deployed in OpenStack). Workability refers to the ability to work in existing networks [53]. As JANO performs an instantiation of NFV Orchestrator, VNFM, and VIM, it can be easily adapted to work in SDN or traditional networks. Automation is the capability to operate independently [54]. JANO presents a high level of automation because it generates plans by itself.

V. CONCLUSIONS

Considering that SDN management is a significant challenge, in this paper, we introduced JANO. JANO is an approach for managing SDN based on HTN, NFV, and a vertical management plane. The time-planning evaluation corroborated that JANO is a promissory alternative to manage a network situation in a short time and it helps to overcome the problem of low management tasks automation presented in related work. We solved this problem by using an HTN planning technique (SHOP2) that allows building and executing solutions automatically. Human intervention is only needed to get information about such solutions at runtime. By an experimental and KLM evaluation, we observed that JANO reduces the time that Administrators spend to manage a network situation compared to solutions based on mashups and situational management. Furthermore, this reduction corroborated that the automation of tasks, introduced by HTN, facilitates the work of Administrators. The network traffic evaluation also allows stating that JANO is a promising solution because the automatic plans built by JANO do not affect the traffic between switches, these plans only increase the OpenFlow messages between the controller and switches. From a qualitative outlook, it is noteworthy that flexibility and workability are implicit features of JANO because it offers high-automation level by SHOP2 used to instantiate NFV MANO. Besides, JANO can operate in different kind of networks by creating particular instances of VNFM and VIM.

As future work, we are interested in enabling JANO to support Natural Language Processing and Intends to make easier the daily work of Administrators by allowing interaction from any device connected to the Internet. Finally, we desire to develop a mechanism able to generate descriptors automatically every time unknown network situations are detected.

REFERENCES

[1] Y. Zhang, X. Gong, Y. Hu, W. Wang, and X. Que, "SDNMP: Enabling SDN management using traditional NMS," in *Proc. IEEE ICCW*, Jun. 2015, pp. 357–362.

[2] J. A. Wickboldt, W. P. D. Jesus, P. H. Isolani, C. B. Both, J. Rochol, and L. Z. Granville, "Software-defined networking: Management requirements and challenges," *IEEE Commun. Mag.*, vol. 53, no. 1, pp. 278–285, Jan. 2015.

[3] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[4] S. Sezer *et al.*, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, Jul. 2013.

[5] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "FlowCover: Low-cost flow monitoring scheme in software defined networks," in *Proc. IEEE GLOBECOM*, Dec. 2014, pp. 1956–1961.

[6] T. Choi, S. Kang, S. Yoon, S. Yang, S. Song, and H. Park, "SuVMF: Software-defined unified virtual monitoring function for SDN-based large-scale networks," in *Proc. ACM Conf. Future Internet Technol.*, 2014, Art. no. 4.

[7] P. H. Isolani, J. A. Wickboldt, C. B. Both, J. Rochol, and L. Z. Granville, "Interactive monitoring, visualization, and configuration of openflow-based SDN," in *Proc. IFIP/IEEE IM*, May 2015, pp. 207–215.

[8] J. Lin, R. Ravichandiran, H. Bannazadeh, and A. Leon-Garcia, "Monitoring and measurement in software-defined infrastructure," in *Proc. IFIP/IEEE IM*, May 2015, pp. 742–745.

[9] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for software defined networks," in *Proc. IEEE NOMS*, May 2014, pp. 1–9.

[10] O. M. C. Rendon, F. Estrada-Solano, and L. Z. Granville, "A mashup-based approach for virtual SDN management," in *Proc. IEEE Comput. Softw. Appl. Conf.*, Jul. 2013, pp. 143–152.

[11] O. M. C. Rendon, F. Estrada-Solano, and L. Z. Granville, "A mashup ecosystem for network management situations," in *Proc. IEEE GLOBECOM*, Dec. 2013, pp. 2249–2255.

[12] O. M. C. Rendon, C. R. P. dos Santos, A. S. Jacobs, and L. Z. Granville, "Monitoring virtual nodes using mashups," *Comput. Netw.*, vol. 64, pp. 55–70, May 2014.

[13] O. M. C. Rendon, F. Estrada-Solano, V. Guimarães, L. M. R. Tarouco, and L. Z. Granville, "Rich dynamic mashments: An approach for network management based on mashups and situation management," *Comput. Netw.*, vol. 94, pp. 285–306, Jan. 2016.

[14] *Recommendation y.3300 Framework of Software-Defined Networking*, document Y.3300, ITU-T, 2014. [Online]. Available: http://www.itu.int/rec/T-REC-Y.3300-201406-I/en

[15] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, Eds., *Software-Defined Networking (SDN): Layers and Architecture Terminology*, document RFC 7426, Internet Requests for Comments, RFC Editor, Jan. 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7426, doi: 10.17487/RFC7426.

[16] F. Estrada-Solano, A. Ordonez, L. Z. Granville, and O. M. C. Rendon, "A framework for SDN integrated management based on a CIM model and a vertical management plane," *Comput. Commun.*, vol. 102, pp. 150–164, Apr. 2017.

[17] A. Mestres *et al.*, "Knowledge-defined networking," *SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 3, pp. 2–10, Sep. 2017, doi: 10.1145/3138808.3138810.

[18] S. Ayoubi *et al.*, "Machine learning for cognitive network management," *IEEE Commun. Mag.*, vol. 56, no. 1, pp. 158–165, Jan. 2018.

[19] M. A. Gironza-Ceron, W. F. Villota-Jacome, A. Ordonez, F. Estrada-Solano, and O. M. C. Rendon, "SDN management based on hierarchical task network and network functions virtualization," in *Proc. ISCC*, Jul. 2017, pp. 1360–1365.

[20] C. Ullrich and E. Melis, "Pedagogically founded courseware generation based on htn-planning," *Expert Syst. Appl.*, vol. 36, no. 5, pp. 9319–9332, 2009.

[21] K. Giotis, Y. Kryftis, and V. Maglaris, "Policy-based orchestration of NFV services in software-defined networks," in *Proc. IEEE NETSOFT*, Apr. 2015, pp. 1–5.

[22] J. Matias, J. Garay, N. Toledo, J. Unzilla, and E. Jacob, "Toward an SDN-enabled NFV architecture," *IEEE Commun. Mag.*, vol. 53, no. 4, pp. 187–193, Apr. 2015.

[23] G. Monteleone and P. Paglierani, "Session border controller virtualization towards 'service-defined' networks based on NFV and SDN," in *Proc. IEEE SDN4FNS*, Nov. 2013, pp. 1–7.

[24] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 1955–1980, 4th Quart., 2014.

[25] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou, "Adaptive resource management and control in software defined networks," *IEEE Trans. Netw. Service Manage.*, vol. 12, no. 1, pp. 18–33, Mar. 2015.

[26] F. Al-Turjman and S. Alturjman, "Context-sensitive access in industrial Internet of Things (IIoT) healthcare applications," *IEEE Trans. Ind. Inform.*, vol. 14, no. 6, pp. 2736–2744, Jun. 2018.

[27] F. M. Al-Turjman, M. Imran, and S. T. Bakhsh, "Energy efficiency perspectives of femtocells in Internet of Things: Recent advances and challenges," *IEEE Access*, vol. 5, pp. 26808–26818, 2017.

[28] A. Malik, B. Aziz, M. Adda, and C. H. Ke, "Optimisation methods for fast restoration of software-defined networks," *IEEE Access*, vol. 5, pp. 16111–16123, 2017, doi: 10.1109/ACCESS.2017.2736949.

[29] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Netw.*, vol. 32, no. 2, pp. 92–99, Mar./Apr. 2018.

[30] L. Mamatas, S. Clayman, and A. Galis, "A flexible information service for management of virtualized software-defined infrastructures," *Int. J. Netw. Manage.*, vol. 26, no. 5, pp. 396–418, 2016.

[31] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 4, pp. 725–739, Dec. 2016.

[32] *ETSI GS NFV 002 V1.2.1: Network Functions Virtualisation (NFV); Architectural Framework*, document GS NFV 002 (V1.2.1), 2014.

[33] *ETSI GS NFV 003 v1.2.1: Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV*, document GS NFV 003 (v1.2.1), 2014.

[34] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[35] K. Chen, J. Xu, and S. Reiff-Marganiec, "Markov-HTN planning approach to enhance flexibility of automatic Web service composition," in *Proc. IEEE ICWS*, Jul. 2009, pp. 9–16.

[36] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "HTN planning for Web Service composition using SHOP2," *Web Semantics, Sci., Services Agents World Wide Web*, vol. 1, no. 4, pp. 377–396, 2004.

[37] I. Georgievski and M. Aiello. (2014). "An overview of hierarchical task network planning." [Online]. Available: https://arxiv.org/abs/1403.7426

[38] A. Ordonez, J. C. Corrales, and P. Falcarin, "Hauto: Automated composition of convergent services based in htn planning," *Ingeniería Investigación*, vol. 34, no. 1, pp. 66–71, 2014.

[39] P. A. P. R. Duarte, J. C. Nobre, L. Z. Granville, and L. M. R. Tarouco, "A P2P-based self-healing service for network maintenance," in *Proc. IFIP/IEEE IM*, May 2011, pp. 313–320.

[40] G. Yang, K. Lee, W. Jeong, and C. Yoo, "Flo-v: Low overhead network monitoring framework in virtualized software defined networks," in *Proc. Int. Conf Future Internet Technol.*, 2016, pp. 90–94.

[41] S. Biundo and B. Schattenberg, "From abstract crisis to concrete relief—A preliminary report on combining state abstraction and HTN planning," in *Proc. 6th Eur. Conf. Planning*, 2014, pp. 1–8.

[42] G. Milliez, R. Lallement, M. Fiore, and R. Alami, "Using human knowledge awareness to adapt collaborative plan generation, explanation and monitoring," in *Proc. IEEE HRI*, Mar. 2016, pp. 43–50.

[43] P. Browne, *JBoss Drools Business Rules*. Birmingham, U.K.: Packt, 2009.

[44] E. Friedman-Hill, *Jess in Action: Rule-based Systems in Java*, vol. 46. Greenwich, U.K.: Manning, 2003.
[45] K. Jackson, *OpenStack Cloud Computing Cookbook*. Birmingham, U.K.: Packt, 2012.
[46] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. ACM SIGCOMM Workshop Hot Topics Netw.*, 2010, Art. no. 19.
[47] RYU. (2011). *Welcome to RYU the Network Operating System*. [Online]. Available: https://ryu.readthedocs.io/en/latest/
[48] S. Joines, R. Willenborg, and K. Hygh, *Performance Analysis for Java Web Sites*. Boston, MA, USA: Addison-Wesley, 2002.
[49] D. Kieras, "Using the keystroke-level model to estimate execution times," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Univ. Michigan, Ann Arbor, MI, USA, 2001.
[50] C. Yang and Q. Huang, *Spatial Cloud Computing: A Practical Approach*. Boca Raton, FL, USA: CRC Press, 2013.
[51] J. Y. Lee and S. D. Kim, "Software approaches to assuring high scalability in cloud computing," in *Proc. IEEE ICEBE*, Nov. 2010, pp. 300–306.
[52] D. M. Shawky and A. F. Ali, "Defining a measure of cloud computing elasticity," in *Proc. IEEE ICSCS*, Aug. 2012, pp. 1–5.
[53] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.
[54] M. Ghallab, D. S. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Amsterdam, The Netherlands: Elsevier, 2004.

WILLIAM VILLOTA received the degree in electronics and telecommunications engineering from the Universidad del Cauca, Colombia, in 2016. He is currently pursuing the master's degree with the University of Campinas, Brazil. His research interests include network and service management, network function virtualization, software-defined networking, and Web 2.0/3.0.

MARIO GIRONZA received the degree in electronics and telecommunications engineering from the Universidad del Cauca, Colombia, in 2016. His research interests include network and service management, network function virtualization, software-defined networking, and Web 2.0/3.0.

ARMANDO ORDOÑEZ received the Ph.D. degree in telematics engineering from the Universidad del Cauca, Colombia, in 2014. He is currently a Researcher with the Foundation University of Popayán, Colombia. His research fields are in the applications of artificial intelligence in telecommunications.

OSCAR MAURICIO CAICEDO RENDON received the bachelor's degree in telecommunications and the master's degree in telematics from the University of Cauca (Unicauca) in 2006 and 2001, respectively, and the Ph.D. degree in computer science from the Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Brazil, in 2015. He is currently a Full Professor with the Telematics Department, Unicauca. He is a member of the Telematics Engineering Group, Unicauca, and the Computer Networks Group, UFRGS. He has published in prominent journals, such as *Communications Magazine*, JISA, *Computer Networks*, and *Computer Communications*, and in relevant conferences, such as the IEEE Globecom, AINA, COMPSAC, CNSM, and ISCC. He was a recipient of an IETF Fellowship and a Traveler Grant from ACM Sigcomm.