Computer Networks 94 (2016) 285–306

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Rich dynamic mashments: An approach for network management based on mashups and situation management

Oscar Mauricio Caicedo Rendon [a,*], Felipe Estrada-Solano [a], Vinicius Guimarães [b], Liane Margarida Rockenbach Tarouco [b], Lisandro Zambenedetti Granville [b]

[a] Telematics Department - University of Cauca, Street 5 # 4–70 - Popayán, CA - Colombia
[b] Institute of Informatics - Federal University of Rio Grande do Sul, Av. Bento Gonçalves, 9500 - Porto Alegre, RS - Brazil

ARTICLE INFO

Article history:
Received 30 January 2015
Revised 14 August 2015
Accepted 5 October 2015
Available online 10 November 2015

Keywords:
Mashup
Network management
Situation management
Web-based network management

ABSTRACT

In network management, significant research efforts have been carried out to automate and facilitate the tasks conducted by network administrators. However, so far, none of these efforts has exploited the opportunities of jointly using the Situation Management discipline and the mashup technology for network management. This paper introduces an approach, called Rich Dynamic Mashments, to facilitate the daily work of network administrators when dealing with unexpected, dynamic, and heterogeneous situations. We have referred to as *nmsits* to such type of network management situations (*e.g.*, 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) and *mashments* to tunable mashups that use Situation Management for conducting network management tasks. The proposed approach is made up by the models of *nmsits* and *mashments*, the mechanisms to automatically recognize *nmsits* and dynamically compose *mashments*, and the architecture supporting these models and mechanisms. We further implement a prototype of our approach and conduct an extensive analysis on networks based on the Software-Defined Networking paradigm. The analysis results have provided directions and evidence that corroborate the feasibility of using Rich Dynamic Mashments as an effective approach for network management in terms of time-recognition, time-composition, time-consuming, time-response, and network traffic.

1. Introduction

The Situation Management (SM) discipline is intended to address situations happening or that might happen in dynamic systems [1]. SM aims to provide solutions that enable analyzing, correlating, and coordinating interactions among people, information, technologies, and actions targeted to overcome situations [2]. The basis of SM are [3]: (*i*) a situation modeled as a collection of entities in a domain, their attributes, and relationships in a time interval, (*ii*) the investigative aspect related to retrospective cause analysis of situations, (*iii*) the control aspect devised to change or preserve situations; and (*iv*) the predictive aspect aimed to foresee situations.

SM has been employed in diverse domains, such as disaster response [4], smart power grids [5], civil crisis [6], aviation [7], public health [8], electric power systems [9], and medical emergencies [10]. However, up to now, SM has not been used to deal with unexpected, dynamic, and heterogeneous situations that network administrators face in their daily work. Hereinafter, such type of situations are referred

* Corresponding author. Tel.: +57 3104918132.
E-mail addresses: omcaicedo@unicauca.edu.co, omcaicedo@gmail.com, omcrendon@inf.ufrgs.br (O.M. Caicedo Rendon), festradasolano@unicauca.edu.co (F. Estrada-Solano), vtguimaraes@inf.ufrgs.br (V. Guimarães), liane@penta.ufrgs.br (L.M. Rockenbach Tarouco), granville@inf.ufrgs.br (L.Z. Granville).

http://dx.doi.org/10.1016/j.comnet.2015.10.024
1389-1286/

to as *nmsits* [11]. Some examples of *nmsit* are: (*i*) a sudden packet loss in a core router of a network backbone, (*ii*) an unforeseen slowness in data transmission over a link between two virtual routers; and (*iii*) an unexpected increases in the number of corrupted packages transmitted by switches handled by an OpenFlow Controller.

Mashups are composite Web applications centered on end-users and created by combining resources (*e.g.*, data, application logic, and user interfaces) available along the Web [12]. In the previous definition, end-user centric means that mashups may be developed by users without advanced programming skills [13]. Mashups have been used in several domains, such as fire emergencies [14], telco services [15], and immersive mirror worlds [16]. Also, we have analyzed the mashup technology as a mechanism to compose network management applications [17] and accomplish specific tasks like virtual nodes monitoring [18].

Although a large number of research efforts [19–25] has been carried out to support management tasks, to the best of our knowledge, none of such efforts has jointly used SM and mashups to automate and facilitate the work of network administrators. In our previous work [11,26], we introduced the concept of *mashments* (*i.e.*, tunable mashups that automate the investigative and control aspects of SM for carrying out network management). We observed that *mashments* are a suitable approach to facilitate the tasks of network administrators when facing *nmsits*. Nevertheless, we have identified some features that are missing in current *mashments*: (*i*) they are not able to automatically recognize *nmsits*, constraining the analysis and resolution of such situations; and (*ii*) they are not dynamically composed, limiting the overcoming of recognized *nmsits*.

In this paper, we take a step further and introduce Rich Dynamic Mashments (RDM) to facilitate the work of network administrators when facing *nmsits*. We argue that the use of mechanisms to automatically recognize *nmsits* and dynamically compose *mashments* allows to make timely decisions in a less complex way. Our major contributions are

- Mechanisms to automatically recognize *nmsits* and dynamically compose *mashments*.
- An architecture that supports the above mentioned mechanisms and enables building up RDMs.
- A prototype that implements the proposed architecture.
- The demonstration, in a network that follows the Software-Defined Networking (SDN) paradigm, of the feasibility of using RDM to deal effectively with *nmsits* in terms of time-recognition, time-composition, time-consuming, time-response, and network traffic.

The remainder of this paper is organized as follows. In Section 2, we present scenarios and challenges of *nmsits*. In Section 3, we present related work. In Section 4, we introduce the RDM Architecture. In Section 5, we describe and discuss the proof-of-concept used to evaluate RDM. Finally, in Section 6, we provide conclusions and implications for future work.

2. Scenarios and challenges

In this section, we introduce motivating scenarios and their challenges. In the first scenario, let's consider network administrators manage a large company. In this company, the communication between the Pin Pads shops and the Enterprise Resource Planning system is provided by an outsourced Internet Service Provider (ISP). If a sudden failure in packet transmission in the ISP takes place or if an internal connection error in the border router of one or more shops occurs, the company might lose a significant amount of revenues because the payment by cards becomes inoperative.

In the second scenario, let's assume network administrators manage an SDN-based Network Operator. This operator provides network infrastructure to Small and Medium Enterprises (SMEs) using resources, such as OpenFlow controllers and virtual switches, supplied by several Virtual Network Providers. If an abrupt performance degradation in virtual links of one or more SMEs (*e.g.*, generated by unidentified errors in the virtual switches of providers) occurs, the operator might break the Service Level Agreements established with SMEs and, as a result, lose money.

In these both scenarios, network administrators need to easily and rapidly overcome *nmsits*. In particular, they face the following challenges: (*i*) conduct situational management operations (*e.g.*, collect, split, filter, add, and merge data) on multiple and heterogeneous devices/networks involved in *nmsits*, (*ii*) compose and tune solutions for *nmsits* in a less complex and time consuming way; and (*iii*) visualize information of *nmsits* in an understandable and friendly way.

The challenges of *nmsits* may be addressed, among other, with the following options. The first one is to use several mismatched network management solutions from the industry [27,28] and academy [24] [25], but it hinders and overloads the tasks of network administrators. Thus, this option is complex and time consuming.

A second option to cope with *nmsits*, it is to improve existing network management solutions, such as Nagios [29], ZenOSS [27], and OpenNMS [28], by deploying on them plugins that support SM. To the best of our knowledge, up to now, there is not a research that develops and adds plugins/packages based on the SM discipline to extend and enhance the above-referred solutions.

A third option to deal with *nmsits*, it is to use home-brewed scripts that integrate two or more network management solutions. The weaknesses of this option are, first, the skill required to write and run scripts (the development of scripts is a daunting and complex task for network administrators who usually do not have advanced programming knowledge). Second, the loss of focus and time of network administrators; instead of management the network itself, they are forced to acquire knowledge that is not necessarily relevant to their daily tasks.

A fourth option to address *nmsits*, it is to use our *mashments* [11,26]. In a broad sense, a mashment is a solution based on the SM discipline and the mashup technology for carrying out network management in an effective way. In particular, a mashment is defined as 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 work of network administrators.

We argue that *mashments* are closer and more appropriate to facilitate the daily needs of network administrators. Notwithstanding, as the network administrator is even

Table 1
Proposals on SM.

Ref	Description	Domain	Aspect		
			Investigative	Control	Predictive
[4]	An architecture based on Wireless Sensor Networks and Delay-Tolerant Networking to aid disaster responders in making decisions	Disaster response	✓	✓	✓
[5]	An architecture based on SOA that uses linked open data to meet the continuous changes in the electricity usage patterns of customers	Smart power grids	✓	✓	
[6]	A mobile agent platform to provide timely and protected access to situational information for on-site personnel and tactical center	Civil crisis	✓	✓	
[7]	An approach for deploying distributed systems that aim to assist a timely and correct making-decision during air security incidents	Aviation	✓	✓	
[8]	A rule-based platform supporting the development of situation-aware applications for monitoring suspicious cases of tuberculosis	Public health	✓	✓	✓
[9]	A situation awareness application aimed to face major disturbances (*e.g.*, snow storms) suffered by distribution system operators in Finland	Electric power systems	✓	✓	
[10]	A system based on finite-state machines and complex event processing to enhance information availability in emergency medical services	Medical emergencies	✓	✓	

responsible for identifying *nmsits* and creating the workflow of current *mashments*, the *mashment*-based approach still keeps some complexity and consumption of time. In this paper, the RDM-based approach is introduced to overcome the shortcomings and weaknesses of options previously described.

3. State-of-the-art

This section reviews the three top topics related to the RDM-based approach: situation management, mashup technology, and network management. To visit the related work on the SM literature, we focus on the SM fundamental aspects [3,30]: investigative, control, and predictive. The investigative aspect related to carrying out actions intended to comprehend situations (*i.e.*, what is happening?). The control aspect associated with creating plans for overcoming situations (*i.e.*, how to solve a situation?). The predictive aspect related to conducting actions for foreseeing future situations by taking into account past situations (*i.e.*, what is going to happen?).

Table 1 presents relevant proposals in SM by considering the application domain and the SM aspects. This table reveals that the most of SM-based proposals focuses on the investigative and control aspects. That is so because such aspects are the basis to carry out the predictive one. It is noteworthy that although SM has been used in different domains, ranging from disaster response to public health, none of SM-based proposals has been targeted to automate or facilitate tasks in network management.

Regarding mashups, it is important to mention that their use has been disseminated over the last decade thank mainly to two facts [31]: (*i*) the number of online services, widgets, and APIs rose significantly; and (*ii*) new usability-oriented technologies (*e.g.*, frameworks JavaScript, HTML5, and Web Sockets) arose to allow the creation of more dynamic applications and advanced Graphical User Interfaces (GUIs) by end-users.

Thanks the dissemination of the mashup technology, mashups mainly involving mapping services have been used in diverse domains, ranging from fire emergencies to network management. Table 2 presents relevant proposals in the mashup technology by considering its application domain and evaluated characteristics. This table reveals two facts: (*i*) the most of mashup-based proposals neither evaluates the complexity nor the time involved in the tasks supported by mashups; and (*ii*) none of proposals that uses mashups for network management employs the SM aspects.

We are pioneers in investigations that involve SM and mashups to automate and facilitate the daily tasks of network administrators. However, in the literature there are proposals for network management that uses other points-of-views. Table 3 presents relevant proposals in the network management domain by considering their main mechanism (*i.e.*, Agent, Rule, Policy, SOA, and BPM). This table reveals that in proposals such as COOLAID, NetOpen, Pyretic, and Procera, network administrators are in charge of writing/programming policies, rules, or basic services using specific languages or controllers. Thus, when using such proposals, the work of the network administrator remains complex and consumes a lot of time, hindering their use as situational solutions. In turn, OMNI and MEICAN are proposals that provide friendly GUIs to facilitate several network management tasks, but they were not conceived to be extended or improved by network administrators. Therefore, these proposals also are constrained when used as situational solutions.

Unlike the reviewed proposals, we consider concepts from SM and mashups to facilitate the handling of unexpected, dynamic, and heterogeneous situations happening in network management. It is important to highlight that in addition to the traditional metrics (*e.g.*, network traffic and time of response) evaluated in such proposals, we evaluate the

Table 2
Proposals on mashups.

Ref	Description	Evaluated characteristics				
		Domain	SM	Extensibility	Flexibility	Time of tasks
[14]	A mashup providing information of active fires, evacuation routes, and available hospitals	Fire emergencies		✓	✓	
[15]	An architecture that aims to facilitate the provisioning of telco mashups for end-users	Telco services		✓	✓	
[16]	A platform to create mashups aimed to show immersive street-level depictions of the physical world	Immersive mirror worlds		✓	✓	✓
[32]	A system that allows the creation of new musical content by developing multi-song mashups	Music		✓	✓	
[33]	A mashup-based approach to face the botnets security problem in a more flexible and extensible way	Network management		✓	✓	
[34]	A mashup system to qualitatively evaluate the feasibility of using Web 2.0 for network management	Network management		✓	✓	
[17]	A generic architecture to support the static composition of network management applications	Network management		✓	✓	

Table 3
Proposals on network management.

Ref	Description	Agent	Rule	Policy	SOA	BPM
[19]	The COnfiguring cOmpLex and dynamic networks AutomatIcally and Declarative (COOLAID) is a data-centric framework aimed to network configuration operations		✓			
[21]	The OpenFlow MaNagement Infrastructure (OMNI) proposes a multi-agent system that offers an specific API to build up applications for managing OpenFlow-based networks	✓				
[20]	NetOpen is a solution that permits to create network composite services, by combining networking primitives, aimed to monitor and configure OpenFlow-based networks				✓	
[22]	MEICAN is a solution that offers a Web-based friendly GUI to allow network administrators to participate in the provisioning process of virtual inter-domain circuits					✓
[23]	Pyretic is a domain-specific and imperative language built over Python, which supports the development of modular applications for managing OpenFlow-based networks			✓		
[24]	Procera is a control framework that provides a language and an engine for creating and executing policy-based applications intended to control and monitor SDN-based networks			✓		
[25]	A framework based on policy-controlled management patterns that offers abstractions for orchestrating OpenFlow applications by combining individual resilience services			✓		

time of recognition of *nmsits*, the time of composition of rich dynamic *mashments*, and the time spent by network administrators when facing situations.

4. Rich dynamic mashments

To detail our approach, initially, we present the model of *nmsits*. Afterwards, we introduce the model of RDMs. Finally, we present the RDM Architecture. Before introducing the models, Table 4 defines the most important abbreviations used to describe the proposed approach.

4.1. Fundamental models

We use JSON to represent the models of *nmsits* and RDMs. That is so because JSON is more lightweight than the eXtensible Markup Language (XML) [35]. In such models, capital and lowercase letters indicate names and values of JSON properties, respectively.

Model of nmsits. The *nmsits* are unexpected, dynamic, and heterogeneous situations that network administrators face in their daily work. Fig. 1(a) presents the model of *nmsits*.

The model of *nmsits* encoded in JSON is:

[{*NAMESIT*: *namesit*, *NMSIT*: [{$nmsit_1$}, {$nmsit_n$}]}]

Where, *NAMESIT* represents the global name of *NMSIT* that is formed by several $nmsit_n$. A single $nmsit_n$, in turn, is:

[{*SITUATION*: *situation*, *EAC*: [{*eac*}]}]

Where, *SITUATION* is the specific name of $nmsit_n$ and *EAC* is the collection of entities, attributes, and constraints involved in such $nmsit_n$. Each *entity* has a collection of attributes and their corresponding constraints. Note that these constraints serve to determine when $nmsit_n$ happens.

Examples of a single *nmsit* are:

$(-)$ $namesit = \{drop\ of\ received\ packages\}$,
$situation = \{sudden\ drop\ of\ received\ packages\ in\ virtual\ router\}$, and
$eac = \{ENTITY : vyattaRouter,$
$[\{ATTRIBUTE : receivedPkg, CONSTRAINT :< 95\%\}]\}$

Table 4
Abbreviations.

Abbreviation	Meaning	Explanation
EAC	Entity attribute constraint	A collection of entities, attributes, and constraints involved in a *nmsit*
NMR	Network management resource	An entity intended to conduct network management operations
NMRS	NMR as a service	A service that offers the functionalities provided by NMR
NMSIT	Network management situation	An unexpected, dynamic, and heterogeneous situation on network management
NSR	Network situational resource	An entity that provides functionalities aimed to automate SM aspects
NSRS	NSR as a service	A service that offers the functionalities provided by NSR
OpR	Operator resource	An entity suitable to combine resources and, so, to create RDMs
OpRS	OpR as a service	A service that offers the functionalities provided by OpR
RDM	Rich dynamic mashment	A solution based on SM and mashups for carrying out network management
SM	Situation management	A discipline to addresses situations in dynamic systems
SMR	Situation management resource	An entity that provides interaction to and from entities involved in *nmsits*
SMRS	SMR as a service	A service that offers the functionalities provided by SMR
WMR	Web-based network management resource	A Web entity conceived or that can be used for network management
WMRS	WMR as a service	A service that offers the functionalities provided by WMR

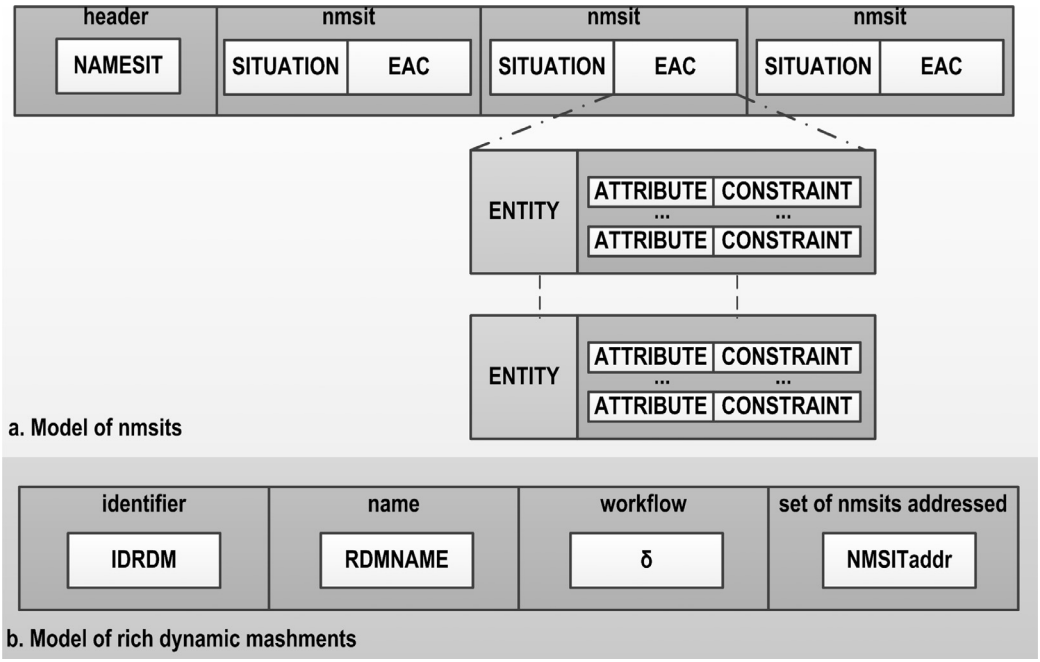

Fig. 1. Fundamental models.

(−) *namesit* = {*drop of sent packages*},
situation = {*unexpected drop of sent packages in a switch*}, and
eac = {*ENTITY* : *cysco*2960,
[{*ATTRIBUTE*: *sentPkg*, *CONSTRAINT*: <90%}]}

An example of a *nmsit* formed by two *nmsits* is:

(-) *namesit* = {*link has overload*},
$situation_1$ = {*switch has overload in memory and processor*},
eac_1 = {*ENTITY* : *cysco*1000,
[{*ATTRIBUTE*: *mem*, *CONSTRAINT*: >95%},
{*ATTRIBUTE*: *processor*, *CONSTRAINT*: >97%}]},
$situation_2$ = {*switch has overload in memory and processor*},
eac_2 = {*ENTITY* : *openvSwitch*,
[{*ATTRIBUTE*: *mem*, *CONSTRAINT*: >96%},
{*ATTRIBUTE*: *processor*, *CONSTRAINT*: >98%}]}.

Model of RDMs. RDMs are tunable composite solutions based on the SM discipline and the mashup technology for carrying out network management. An RDM is characterized by: (*i*) it recognizes automatically *nmsits*; and (*ii*) its workflow is generated without direct intervention of network administrators. Fig. 1 (b) presents the model of RDMs that encoded in JSON is:

[{*IDRDM*: *id*, *RDMNAME*: *name*, *δ*: [{*delta*}], $NMSIT_{addr}$: [{$nmsit_{addr}$}]}]

Where, *IDRDM*, *RDMNAME*, $NMSIT_{addr}$, and *δ* are the unique identifier, the friendly name, the set of *nmsits* addressed, and the workflow of an specific RDM, respectively. In turn, *δ* is:

[{*IDD*: *idd*, *RES*: [{res_1}, {res_n}], *CONN*: [{$conn_1$}, {$conn_n$}]}]

Where, *IDD* is the unique identifier of *δ*, *RES* is the set of resources that form a particular RDM, and *CONN* is the set

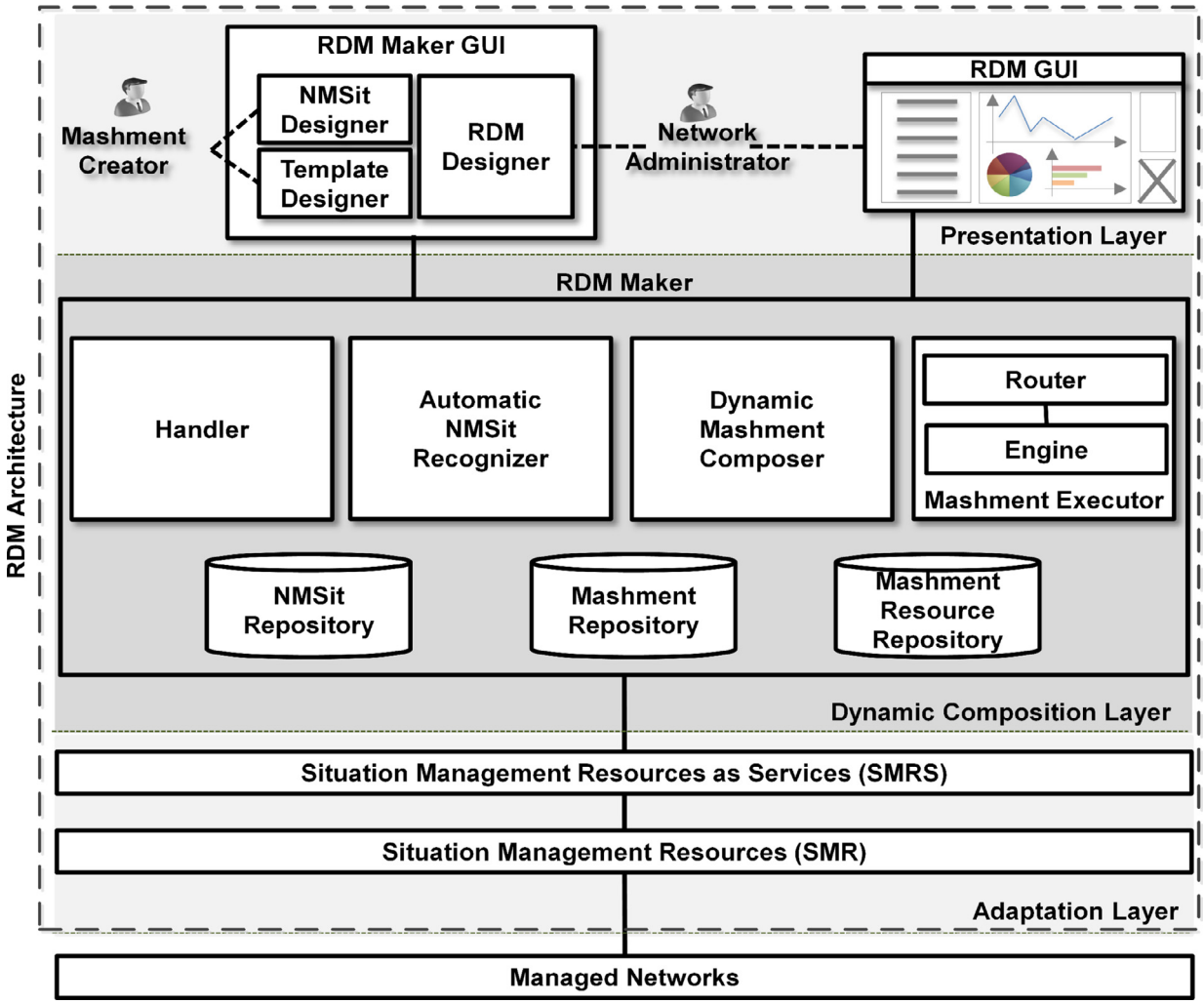

Fig. 2. Proposed architecture.

of logical connections created among these resources. Note that *RES* and *CONN* define how $NMSIT_{addr}$ are handled (*i.e.*, investigated and/or resolved).

Resources are helpful entities to interact (*e.g.*, to access, communicate, and send-retrieve information) with network elements involved in *nmsits* and for presenting, in an intelligible way, information about *nmsits*. A res_n is:

[{*IDRES*: *idres*, *RESNAME*: *resname*, *OPERATION*: [{op_1, op_n}]}]

Where, *IDRES* and *RESNAME* are the unique identifier and the name of the resource, respectively. *OPERATION* is the collection of operations offered by the resource. An op_n is:

[{*OPNAME*: *opname*, *PARAM*: [{par_1, par_n}], *PROD*: *produce*}]

Where, the name of the operation, the set of parameters need to invoke the operation, and the data type that the operation produces are, correspondingly, represented by *OPNAME*, *PARAM*, and *PRODUCE*.

Connections define the propagation of data between resources by specifying which outputs of a resource are inputs of other resources. A $conn_n$ is:

[{*IDC*: *idc*, *SRC*: [{*idresS*, *idparS*}], *DES*: [{*idresD*, *idparD*}]}]

Where, *IDC* is the unique identifier, *SRC* is the source resource, and *DES* is the destination resource of the connection, respectively. The resource (*idresS* or *idresD*) and the parameter (*idparS* or *idparD*) represents the connection among *SRC* and *DES*.

4.2. Architecture

The RDM Architecture (see Fig. 2) leverages the main foundations of the SM discipline and the mashup technology to facilitate the work of network administrators. In particular, this architecture offers: (*i*) the automatic recognition of *nmsits* by using matching pattern algorithms; and (*ii*) the dynamic creation of RDMs by using composition templates.

Two actors (*i.e.*, Mashment Creators and Network Administrators) are involved in tackling *nmsits*. The Creators are mainly responsible for: (*i*) creating models/patterns of *nmsits* that will be automatically recognized by the proposed approach, (*ii*) building up composition templates that will be dynamically customized by the RDM-based approach for

addressing *nmsits* recognized; and (*iii*) customizing and extending rich dynamic *mashments*. In turn, an Administrator is fundamentally in charge of coping with *nmsits* by using RDMs as well as customizing and extending them.

Fig. 2 depicts the RDM Architecture and the networks that it can manage. Three layers made up this architecture: (*i*) the Adaptation Layer that hides the intricacy and heterogeneity of Managed Networks (*e.g.*, SDN-based, NVE, and traditional networks), (*ii*) the Dynamic Composition Layer that recognizes *nmsits* and builds RDMs to face such situations; and (*iii*) the Presentation Layer that depicts the GUIs of RDMs and the RDM Maker. The next paragraphs present in detail these architectural layers.

4.2.1. Adaptation layer

This layer provides services to the Dynamic Composition Layer and offers as a mashable resource (*i.e.*, services) any entity that is useful to deal with *nmsits*. The elements that form the Adaptation Layer are the set of Situation Management Resources (SMR) and its respective representation as a service (SMRS). An SMR is an entity that provides interaction to and from network elements or entire networks involved in *nmsits*.

There are five types of SMR: Network Management Resource (NMR), Web-based Network Management Resource (WMR), Analytics Management Resource (AMR), Network Situational Resource (NSR), and Operator Resource (OpR). An NMR is an entity intended to conduct network management operations. Examples of NMR are Ganglia [36], Nagios [29], and ZenOSS [27] to manage traditional networks, Citrix Center [37] for monitoring virtual resources, NetOpen [20] and OMNI [21] to control OpenFlow-based networks, monitoring systems based on SNMP, and all APIs that provide interaction with network elements.

A WMR is a Web entity conceived or that can be used to perform network management tasks. Examples of WMR are the Multi Router Traffic Grapher (MRTG) [38] for generating Web pages with images presenting the traffic of network links, the RRDTool [39] for displaying over time the performance data of routers, the Yahoo Maps API [40] for showing the geographic location of several network devices, and the Google Chart API [41] for depicting the memory consumption of virtual switches.

An AMR is an entity intended to analyze network management information. Examples of AMR are the Management Traffic Analyzer [42] to interpret the functioning of network devices supporting SNMP, the Junos Network Analytics Suite [43] to understand what is happening on networks using Junos devices, and the Sandvine Network Analytics [44] to get right-time information of networks regardless of underlying technologies.

An NSR is an entity that provides functionalities aimed to automate the aspects of SM. These functionalities are: (*i*) collecting to retrieve information about *nmsits*, (*ii*) fusing&correlating to merge and correlate the information retrieved by collecting, supporting the creation of investigative plans (*i.e.*, workflows useful to determine the cause of *nmsits*); and (*iii*) resolving to enable conducting network management operations aimed to change or preserve *nmsits*, assisting the creation of resolutive plans (*i.e.*, workflows helpful to solve *nmsits*).

Examples of NSR are JESS [45], JBOSS Drools [46], and Apache Camel [47]. The JESS is a general-purpose platform that permits detecting and controlling situations by rules (defined using XML or the JESS Rule Language) and Java applications. The JBOSS Drools is a solution that allows recognizing and controlling generic situations by rules (defined using Drools Rule Language - DRL) and Java applications. The Apache Camel is a platform that enables processing events (*i.e.*, generic situations) from multiple sources employing a Complex Event Processor (CEP) based on Java.

An OpR is an entity that allows combining resources and, so, building up and generating RDMs. There are three OpR types: (*i*) control patterns (*e.g.*, sequential, parallel, conditional, and templates) that allow defining the workflow of RDMs, (*ii*) structures for configuring and invoking (*e.g.*, functionalities to set security credentials of virtual routers) the resources that form RDMs; and (*iii*) structures for receiving, sorting, and filtering (*e.g.*, functionalities to perform information selection on text-plain containing data of NVEs) the retrieved information from any type of resource.

SMRS are mashable entities that offer as a service network management operations of one or more NMR, WMR, AMR, NSR, and OpR, aiming to hide the complexity of these resources. The representation of resources as services consists in defining and providing a common data format to interchange information of resources, well-known interfaces to resources, and a common protocol to communicate with such interfaces.

Types of SMRS are NMR as a Service (NMRS), WMR as a Service (WMRS), AMR as a Service (AMRS), NSR as a Service (NSRS), and OpR as a Service (OpRS). An example of NMR is the Floodlight Controller API to handle switches OpenFlow, the associated NMRS is the Floodlight Service that allows, via requests-and-responses HTTP, to monitor these switches.

Internally, a Wrapper and a UWrapper made up an SMRS. A Wrapper is a service based on the REpresentational State Transfer (REST) [48] architectural model. In turn, a UWrapper is a URI pointing to one Wrapper. Since every Wrapper is based on REST, in the RDM architecture, the set of SMRS provides a mediator bus in which the communication is conducted by following the request-response model of HTTP. This bus enables the interaction between the layers of Adaptation and Dynamic Composition.

The Adaptation Layer responds to HTTP requests from the Dynamic Composition Layer as follows. First, the requests are targeted to UWrappers. Second, Wrappers invoke one or more SMR by using protocols (*e.g.*, SNMP, SOAP, HTTP, OpenFlow, and Proprietary) provided by network vendors for managing their solutions. Third, each invoked SMR carries out the requested functionalities by performing operations in the Managed Networks. Fourth, Wrappers receive responses from the invoked SMR. Fifth, Wrappers encode SMR results on JSON data and put such data on HTTP responses. Sixth, Wrappers send their HTTP responses to the Dynamic Composition Layer.

4.2.2. Dynamic composition layer

This layer offers RDMs to the Presentation Layer. In the Dynamic Composition Layer is the RDM Maker that supports: (*i*) the definition of *nmsit* patterns, (*ii*) the recognition of *nmsit* patterns, (*iii*) the specification of composition templates;

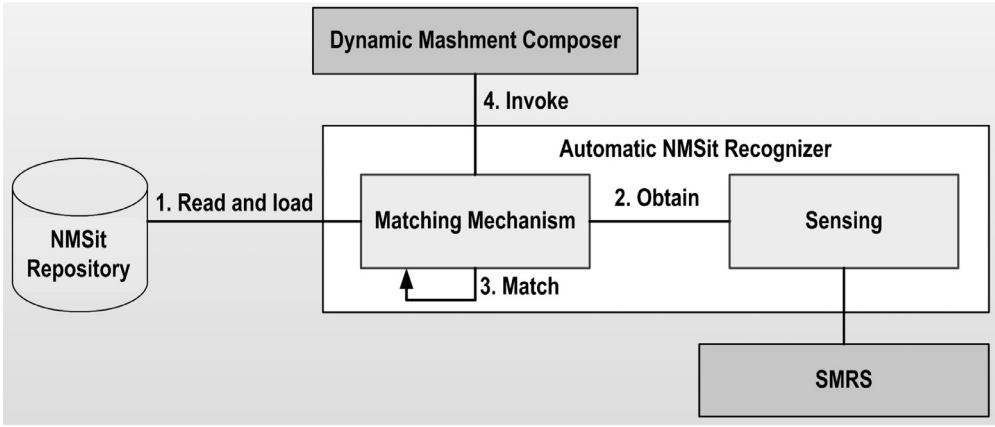

Fig. 3. Automatic recognition of nmsits.

and (*iv*) the generation from templates of RDMs that face *nmsits* recognized. The Mashment Resource Repository, NMSit Repository, Mashment Repository, Handler, Automatic NMSit Recognizer, Dynamic Mashment Composer, and Mashment Executor form the RDM Maker.

The Mashment Resource Repository stores metadata that describes and points out functionalities offered by SMRS. NMRS, AMRS, OpRS, and NSRS point out to services located on privative repositories accessible only by network administrators. In turn, WMRS by definition points out services available on the Web. A metadata of SMRS is an instance of res_n. The following example of metadada describes two operations provided by a particular NMRS. These operations are to get the list of virtual switches and get the statistics of a virtual switch by an OpenFlow Controller.

$(-)$ [{*IDRES*: $/path_1$, *RESNAME*: $nmrs_1$,
OPERATION : [{*OPNAME* : *SwitchList*,
PARAM : [{*IPCTRL* : *ipctrl*, *PORT* : *port*, *USER* :
user, *PWD* : *pwd*}],
PROD: *json*},
{*OPNAME* : *SwitchStat*,
PARAM : [{*IPCTRL* : *ipctrl*, *PORT* : *port*, *USER* :
user, *PWD* : *pwd*, *IDSWITCH* : *ids*}],
PROD: *json*}]}]

The NMSit Repository stores *nmsit* patterns defined by Mashment Creators. A pattern is an instance of the *nmsits* model that represents a collection of entities, attributes, and constraints. These constraints are conditions defined in attributes of entities for detecting *nmsits*. Thus, this repository contains the patterns that allows recognizing *nmsits*.

The Mashment Repository stores metadata of composition templates and RDMs that handle the recognized *nmsits*. A metadata of RDM stored in this Repository is an instance of the RDM model (see Section 4.1). It is relevant to point out that, first, if several RDMs constitute one RDM, its metadata includes the metadata of them. This inclusion means that a δ may encompass other δs. Second, several SMRS can be used and connected to form a δ that defines how to face one or more *nmsits*. Third, as our approach inherits the end-user composition model from mashups, network administrators can also customize and enhance RDMs.

The composition templates are useful skeletons to automatically compose RDMs. The metadata of templates is:

[{*IDTEMPLATE*: *id*, *NAMESIT*: *namesit*, δ: *delta*}]

Where, *IDTEMPLATE* is the unique identifier of template and δ is a predefined workflow (*i.e.*, an investigative and/or resolutive plan) to deal with $NMSit_{addr}$ identified by *NAMESIT*.

The Handler manages (*i.e.*, search, create, update, and delete) the metadata of *nmsits* and composition templates by manipulating the repositories of NMSit and Mashment. The Mashment Creator handles in the GUI of the NMSit Designer the *nmsits* metadata. The Mashment Creator manages in the GUI of the Template Designer the metadata of composition templates. The Handler also manages the Mashment Resource Repository to support the enhancement and improvement of RDMs that can be conducted by network administrators in the GUI of the RDM Designer.

The repositories described above support the generation of RDMs. Such generation follows two phases:

(*1*) *when* < *nmsits* > — mechanism to automatically recognize *nmsits*.
(*2*) *then* < *rdms* > — mechanism to dynamically compose *mashments*.

The first phase is the automatic recognition of *nmsits* by using matching pattern algorithms. The second phase is the dynamic composition of RDMs by composition templates.

The Automatic NMSit Recognizer (see Fig. 3) conducts the automatic recognition of *nmsits*. The modules Sensing and Matching Mechanism form the Recognizer. The Sensing is in charge of retrieving network management information by SMRS and delivering such information as streaming to the Matching Mechanism. The retrieving of information depends on communication model (pull or push) provided by SMRS.

The Matching Mechanism recognizes a *nmsit* as follows. First, it reads and loads *nmsit* patterns from the NMSit Repository. Second, it obtains information from Managed Networks and their devices by Sensing. Third, it conducts matching operations (*i.e.*, comparison of samples vs patterns) among the network information and the loaded *nmsit* patterns. RETE [45] and PHREAK [46] are algorithms that can be used to carry out this matching. Fourth, every time a *nmsit* is

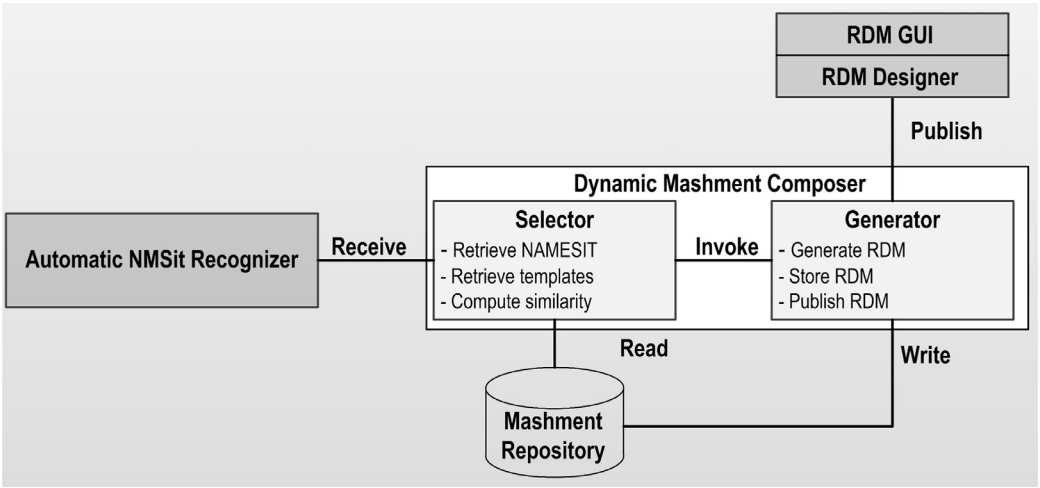

Fig. 4. Dynamic composition of mashments.

detected (*i.e.*, there is match), this mechanism defines $NMSIT_{addr}$ and invokes the Dynamic Mashment Composer.

The Dynamic Mashment Composer (see Fig. 4) performs the dynamic composition of RDMs. This Composer creates RDMs by automating our process to develop and launch *mashments* [26]. Specifically, it automates the tasks Select, Configure, and Combine of such process. Select is to define the resources (*i.e.*, *RES*) to be used to generate RDMs. Configure is to provide all functioning settings of resources selected. Combine is to define how a particular RDM will face an specific $NMSIT_{addr}$ by creating diverse connections (*i.e.*, *CONN*) among selected and configured resources.

The modules Selector and Generator form the Dynamic Mashment Composer. The Selector operates as follows. First, it receives $NMSIT_{addr}$ from the Automatic NMSit Recognizer. Second, it retrieves *NAMESIT* from $NMSIT_{addr}$. Third, it retrieves composition templates by reading the Mashment Repository. Fourth, it selects the δ (*RES* and *CONN*) for such $NMSIT_{addr}$. This selection is carried out by calculating the highest linguistic similarity among the *NAMESIT* of composition templates and the retrieved from $NMSIT_{addr}$. Such calculation is conducted by using the linguistic similarity algorithm [49] that is based on NGram [50], CheckSynonym [51], and ElementMatch [52]. Fifth, it invokes the Generator.

The Generator operates as follows: (*i*) it receives δ and $NMSIT_{addr}$ from the Selector, (*ii*) it uses such δ and $NMSIT_{addr}$ to generate an instance of the RDM model, (*iii*) it stores in the Mashment Repository the RDM generated by writing the corresponding metadata; and (*iv*) it publishes such RDM in the RDM Designer of the RDM Maker GUI. It is noteworthy that using such Designer, the network administrator can enhance, improve, and run RDMs.

The Mashment Executor executes RDMs. The Mashment Router and the Mashment Engine form the Executor. The Router is in charge of coordinating the execution of δs that are the core of RDMs. Thus, on runtime, the Router: (*i*) it receives invocations from the Engine, which means that the Router is called by the Engine to select RDMs to serve initial requests, (*ii*) it selects and links multiple resources (including one or more RDMs into another RDM) to attend invocations, by reading information from the repositories of Mashments and Resources; and (*iii*) it calls the Engine to request the instantiation of RDMs and their underlying resources.

The Mashment Engine is a lifecycle manager responsible for creating, deleting, and caching instances of RDMs. When initial requests to execute RDMs arrive from a browser, the Engine invokes the Router. Afterwards, the Engine waits indications from the Router to manage the instances of RDMs and their constitutive resources.

4.2.3. Presentation layer

This layer executes and presents, in the client-side, the RDM Maker GUI and the RDM GUI. The NMSit Designer, the Template Designer, and the RDM Designer form the RDM Maker GUI. All these GUIs are accessible by Web browsers.

The NMSit Designer is a Web-based friendly GUI in which Mashment Creators define *nmsit* patterns to be recognized. The Template Designer is another GUI where Mashment Creators specify composition templates used to cope with *nmsits*. The RDM Designer is the GUI in which network administrators can enhance, save, delete, and run RDMs.

The RDM GUI represents the visualizations of compositions generated by the RDM Maker. Examples of visualizations useful for network management are [53]: (*i*) a 2D scatterplot chart to present information about packet errors in a virtual router involved in a *nmsit*; and (*ii*) a link/node representation to display a network topology (*e.g.*, a glyph-based representation) and 2D charts (*e.g.*, line and bar charts) to provide additional information of corresponding nodes and links.

5. Proof-of-concept

To assess our approach, first, we implemented the RDM System prototype that is an instance of the architecture described in the previous chapter. Second, we built a test environment. Third, we conducted a late evaluation of our architecture. A late evaluation of a software architecture takes place when its implementation (*e.g.*, the RDM System prototype) is complete [54].

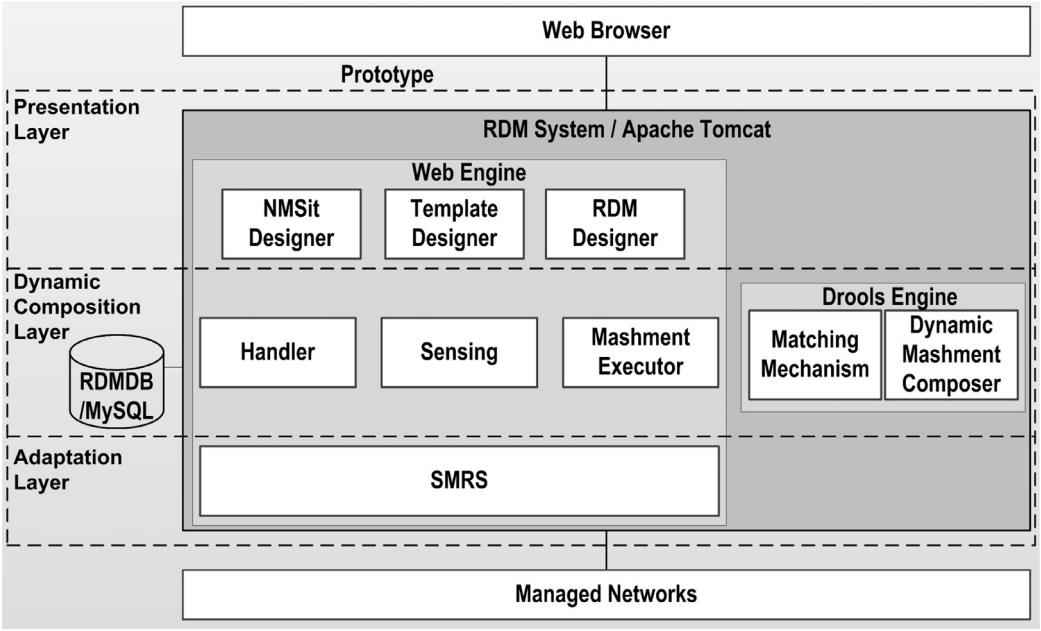

Fig. 5. RDM system prototype.

The late evaluation of the proposed approach is performance-based [55] and intended to determine its feasibility in terms of time-recognition, time-composition, time-consuming, time-response, and network traffic. The time-recognition and time-composition related to the mechanisms for automatic recognition of *nmsits* and dynamic composition of *mashments*, respectively. The time-consuming associated with the process that network administrators need to carry out for facing *nmsits*. The time-response and traffic related to the behavior during runtime of solutions used to handle *nmsits*.

5.1. Prototype

Fig. 5 depicts the RDM System prototype and the Managed Networks. The prototype was built upon the Mashment Maker [11] [26] and deployed by using a MySQL Server and an Apache-Tomcat Server (running a Web Engine and a Drools Engine). The Maker is a browser-based mashup development environment that provides functionalities for assisting Mashment Creators and network administrators in the creation, reuse, and launching of traditional *mashments*.

Presentation layer. The Web Engine deploys the NMSit Designer, the Template Designer, and the RDM Designer. These designers were built using the Yahoo User Interface (YUI) API and the Google Chart API. These APIs are based on the Asynchronous Javascript and XML (AJAX), granting dynamic and interactive interaction of all GUIs of designers with SMRS.

Dynamic composition layer. The Web Engine also deploys the modules Sensing, Handler, and Mashment Executor. These modules are services implemented by using the Java Language and following the REST architectural model [56]. REST was used because it is the de-facto model for developing mashups. Specifically, we implemented: (*i*) Sensing (*i.e.*, SensorService) for interacting with Managed Networks, (*ii*) Handler (*i.e.*, HandlerService) for managing the Mashment Repository, the NMSit Repository, and the Mashment Resource Repository; and (*iii*) Mashment Executor (*i.e.*, ExecutorService) for controlling the execution and lifecycle of RDMs.

The Drools Engine deploys the Matching Mechanism and the Dynamic Mashment Composer. The Matching Mechanism is a Java-based application that uses for recognizing *nmsits* the implementation of the PHREAK algorithm offered by Drools. Furthermore, this mechanism translates from JSON to DRL the *nmsit* patterns stored in the RDMDB. This translation is needed because Drools only understand DRL.

The Dynamic Mashment Composer is also a Java-based application that customizes composition templates. It is to note that, first, the Mashment Creator defines these templates in the Template Designer that saves them in JSON format in RDMDB. Second, once a template has been customized, it is also stored using JSON in RDMDB and, further, automatically exposed in the RDM Designer like a visual element.

RDMDB is a unique database that implements the Mashment Resource Repository, the Mashment Repository, and the NMSit Repository. A MySQL Server deploys such database.

Adaptation layer. SMRS are also services that follow the REST architectural model. In particular, REST-based services (*i.e.*, POXService, BeaconService, and FloodlightService) enables the interaction with the Managed Networks of this proof-of-concept.

5.2. Managed networks

In the proof-of-concept, we chose to manage SDN-based networks because of their commercial and investigative significance. SDN proposes an architecture for future networks

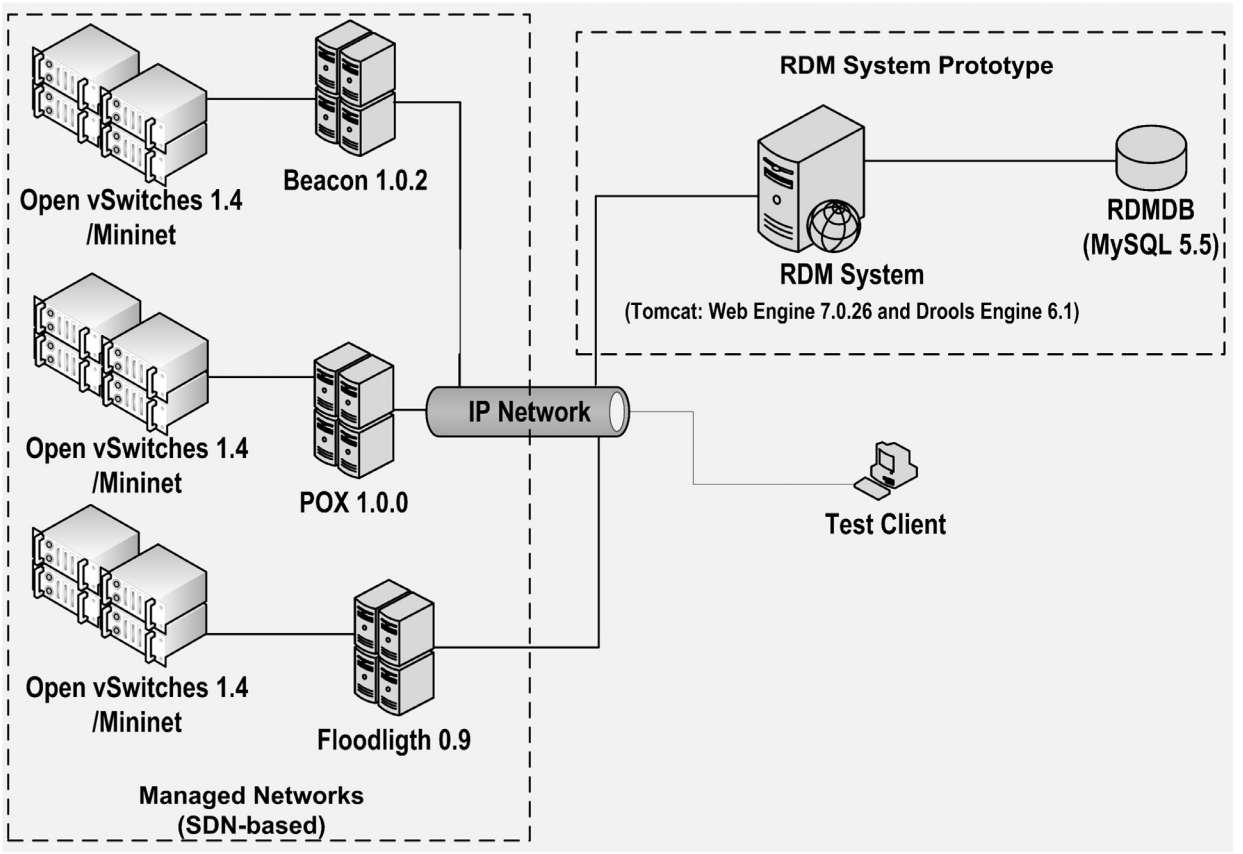

Fig. 6. Test environment.

[57], which separates data and decision policies to simplify the network operation. SDN-based networks follow three layers [58–60]: (*i*) the packet forwarding datapath (*e.g.*, switches and routers passing packets), (*ii*) the Network Operating System (NOS) that controls such datapath by using a vendor-independent protocol; and (*iii*) the Network Services (*e.g.*, a new routing protocol) running on the top of NOS.

There are different proposals for deploying SDN like the Forwarding and Control Element Separation (ForCES) framework [61] and OpenFlow [62]. In OpenFlow, the Controller (*i.e.*, NOS), such as POX [63], Beacon [64], and Floodlight [65], handles network devices (*i.e.*, the datapath) by the OpenFlow protocol. Furthermore, the Controller supports deploying new and centralized Network Applications (*i.e.*, Network Services), such as groundbreaking applications to path selection and novel multicasting protocols.

5.3. Test environment

To evaluate the proposed approach, we conducted a proof-of-concept in a test environment (see Fig. 6). In such environment, the RDM System prototype was executed on a Web engine 7.0.26 and a Drools engine 6.1. RDMDB was executed on a MySQL 5.5. The RDM System and RDMDB were deployed on a machine with Linux Ubuntu O.S., 2.53 GHz Intel Core i5 processor, 4 GBytes RAM, and 250 GBytes hard disk. In turn, the applications used to evaluate the proposed approach were executed on a Test Client with 2 GBytes RAM and 2.53 GHz core 2 duo processor.

In the test environment, we managed three SDN-based networks controlled by Beacon 1.0.2, POX 1.0.0, and Floodlight 0.9. These controllers handled a lot of Open vSwitches 1.4; later in each evaluation, we will define the exact quantity of switches per network. Each OpenFlow controller was executed on a machine with 2.33 GHz Core 2 Duo processor, 2 GBytes RAM, and 160 GBytes hard disk. In turn, the Open vSwitches were executed on Mininet 2.2.1 and deployed on a server with 8 GBytes RAM and 3.4 GHz Core i7 processor. Mininet [58] is a software useful for emulating OpenFlow-based networks.

Fig. 7 presents the high-level operation of our test environment: (*i*) the RDM System gets the network information of SDN-based networks by the modules Sensing and SMRS, (*ii*) the above-mentioned modules return the network information to the RDM System, (*iii*) it retrieves the *nmsit* patterns from RDMDB, (*iv*) the RDM System conducts a match operation among the information and patterns retrieved, (*v*) if there is a match, it retrieves from RDMDB a composition template to address the recognized *nmsit*, (*vi*) it uses the retrieved template to generate the RDM that will address the detected *nmsit*; and (*vii*) it executes network management operations in the SDN-based networks when the network administrator launches the generated RDM by the Mashment Maker GUI.

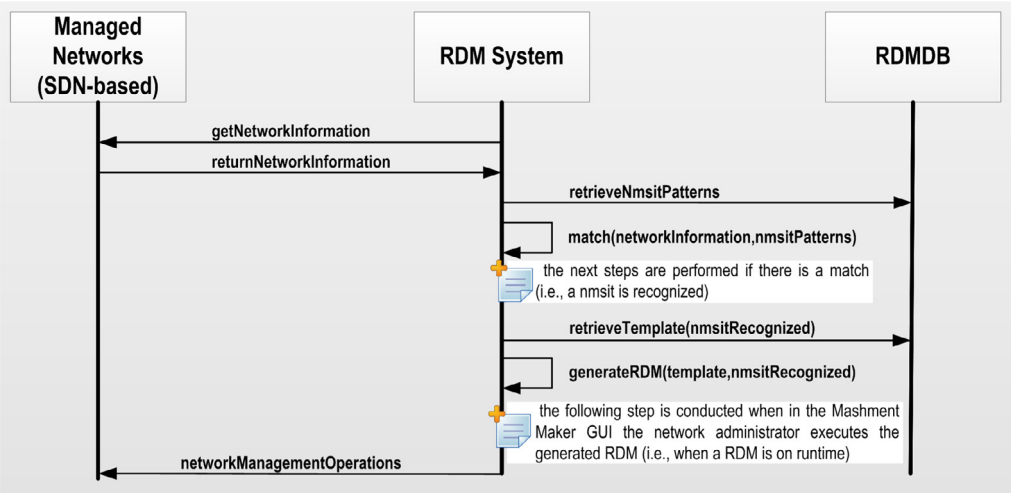

Fig. 7. Test environment high-level operation.

5.4. Time-recognition

The time-recognition is the time that the RDM System takes for recognizing *nmsit* patterns. To measure time-recognition, we used two OpenFlow-based networks handled by POX and Floodlight (see Fig. 6). Each controller handled a datacenter network topology with 259 switches distributed in 4 levels of depth (*i.e.*, layers of access, aggregation, core, and edge) and 6 servers per rack. Thus, in total, we used 2 controllers, 518 switches, and 3626 ports. In this and the following evaluations involving average time in milliseconds (*ms*), we took 30 measurements with 95% confidence level.

Using the NMSit Designer, we defined a *nmsit* pattern to detect when any port of any switch handled by POX or Floodlight had more than 5% of dropped packets. Fig. 8 depicts such pattern encoded on JSON and DRL. It is important to highlight that, first, the RDM System performs the translation from JSON to DRL. Second, for network administrators and Mashment Creators such translation is always hidden.

In the time-recognition evaluation, first, we loaded only a *nmsit* pattern (*i.e.*, just a rule in the Drools Engine) and varied the number of generated *nmsits* from 250 to 1750 in each OpenFlow-based network. Thus, the total number of generated *nmsits* in each evaluation was from 500 to 3500. Second, we varied the number of loaded rules from 1 to 400 and once again the amount of generated *nmsits* from 500 to 3500.

Fig. 9 presents the time-recognition results. These results reveal that the RDM System is able to recognize *nmsit* patterns in a short time; in the worst behavior approximately 30.3 *ms* to identify 3500 *nmsits* having 400 loaded rules/patterns. Furthermore, the time-recognition is increased linearly in a negligible way with the growth of *nmsits* and loaded rules. Consequently, we can state that, in terms of time-recognition, it is feasible to use our approach to dealing effectively with *nmsits*.

5.5. Time-composition

The time-composition is the time that the RDM System takes for dynamically customizing composition templates and, so, generating RDMs. To measure time-recognition, we use three OpenFlow-based networks handled by POX, Floodlight, and Beacon (see Fig. 6). Each controller was in charge of handling a datacenter network topology with 259 switches distributed in 4 levels of depth and 6 servers per rack. Thus, in total, we used 3 controllers, 777 switches, and 4662 ports.

Using the Template Designer, we defined composition templates for monitoring when ports of switches handled by POX, Floodlight, or Beacon had more than 5% of dropped packages. Fig. 10 depicts a snippet of a composition template and the corresponding dynamic *mashment* generated by the RDM Maker.

In the time-composition evaluation, first, we varied the number of resources forming the composition templates from 2 to 8. It is noteworthy that more than 60% of mashups consist of $3-8$ components/resources [66]. Second, we modified the number of templates from 10 to 50; note that the number of templates defines the number of simultaneously generated RDMs.

Fig. 11 presents the time-composition results. These results reveal that the RDM System generates RDMs by customizing composition templates in a short time. In the worst behavior, approximately 14500 *ms* to dynamically compose 50 RDMs formed by 8 resources. Furthermore, the time-composition increases linearly with the growth of generated RDMs and resources per composition template. Considering, first, the above results. Second, the mechanism for generating RDMs has similar time-composition behavior than the composition proposals introduced on other application domains [67]. We can state that, in terms of time-composition, it is feasible to use the proposed approach to coping effectively with *nmsits*.

5.6. Time-consuming

The time-consuming (*i.e.*, T_{cons}) is the time that network administrators spend to cope with *nmsits*. In this evaluation, we raise a *nmsit* called NMSit-AS. Let's consider that a network administrator manages three Autonomous Systems, called AS_1, AS_2, and AS_3. AS_1, AS_2, and AS_3 are handled by Beacon, Floodlight, and POX, respectively. When there is

```
{"SITUATION":"test",
"EAC":[{"ENTITY":"openflowController","PROPERTY":[{"ATTRIBUTE":"ip","CONSTRAINT":"192.168.210.45 or 192.168.210.74"},
{"ATTRIBUTE":"port","CONSTRAINT":"8082 or 8083"},{"ATTRIBUTE":"type","CONSTRAINT":"pox or floodlight"},
{"ATTRIBUTE":"openflowElement","CONSTRAINT":"openflowSwitch"}]},
{"ENTITY":"openflowSwitch", "PROPERTY":[{"ATTRIBUTE":"dpid", "CONSTRAINT":"all"},
{"ATTRIBUTE":"openflowSwitchComponent", "CONSTRAINT":"openflowPort"}]},
{"ENTITY":"openflowPort", "PROPERTY":[{"ATTRIBUTE":"number","CONSTRAINT":"all"},
{"ATTRIBUTE":"percentTransmittedDropped","CONSTRAINT":"> 5"}]}]}
```


JSON

Drools Rule Language

```
package net.mashment.drools.rules
import net.mashment.drools.entities.*
import net.mashment.drools.DynamicMashmentComposer
rule "test"
   when
      $e0 : OpenflowController(ip == "192.168.1.2" || == "192.168.1.3", port == "8082" || == "8083", type == "pox" || == "floodlight")
      $e1 : OpenflowSwitch(openflowController == $e0)
      $e2 : OpenflowPort(openflowSwitch == $e1, percentTransmittedDropped > 5)
   then
      DynamicMashmentComposer($e0,$e1,$e2)
end
```

Fig. 8. Test nmsit.

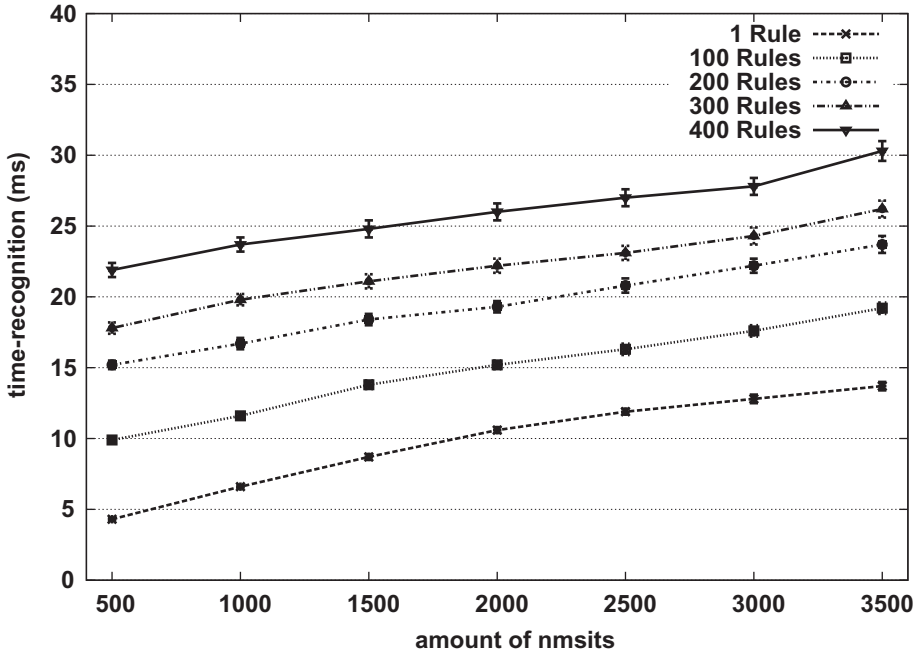

Fig. 9. Time-recognition behavior.

unexpected degradation in the performance of links that connect these *ASs*, he/she needs to identify in each *AS* which are the Open vSwitches that are causing such performance. In this way, he/she requires to rapidly and easily obtain a situational solution that presents, in an integrated, visual, and intelligible way, information about links and Open vSwitches.

To deal with NMSit-AS, the network administrator tests several options: (*i*) without RDM by the *Situational Script*, (*ii*) without RDM by the Performance Monitoring Mashment (*PMM*); and (*iii*) with the proposed approach by the RDM of Performance (*RDMP*). In general terms, the *Situational Script* is an application programmed and executed by the network administrator in a low-abstraction level. *PMM* is a composite solution developed and launched by the network administrator in the Mashment Maker [11] [26]. *RDMP* is a *mashment* automatically generated by the RDM System and offers the same functionalities as *PMM*.

To assess T_{cons}, we use the Keystroke–Level Model (KLM) [68] because it is useful to estimate the time that network

{"RES":
[{"config":{"position":[474,173]},"name":"OpenflowMonitor","value":{"graphTool":"[wired]","nos1":"[wired]","nos1params":"","nos2":
"[wired]","nos2params":"","nos3":"[wired]","nos3params":""},
{"config":{"position":[340,10]},"name":"OpenflowController","value":{"ip":"","port":"","type":""}},
{"config":{"position":[126,16]},"name":"OpenflowController","value":{"ip":"","port":"","type":""}},
{"config":{"position":[44,80]},"name":"OpenflowController","value":{"ip":"","port":"","type":""}},
{"config":{"position":[253,418]},"name":"RRDTool","value":{"refreshTime":""}}],"properties":{"desc":"","name":"template1","nmsit-test":""},
"CON":
[{"src":{"moduleId":1,"terminal":"out"},"des":{"moduleId":0,"terminal":"nos1"}},
{"src":{"moduleId":2,"terminal":"out"},"des":{"moduleId":0,"terminal":"nos2"}},
{"src":{"moduleId":3,"terminal":"out"},"des":{"moduleId":0,"terminal":"nos3"}},
{"src":{"moduleId":4,"terminal":"out"},"des":{"moduleId":0,"terminal":"graphTool"}}]}

Composition template
Generated mashment

{"RES":
[{"config":{"position":[474,173]},"name":"OpenflowMonitor","value":{"graphTool":"[wired]","nos1":"[wired]","nos1params":
{"openflowSwitch":{"dpid":"00:00:00:00:00:00:01:15","openflowPort":{"number":"1","percentTransmittedDropped":"5"}}},
"nos2":"[wired]","nos2params":{"openflowSwitch":{"dpid":"00:00:00:00:00:00:01:15","openflowPort":{"number":"1",
"percentTransmittedDropped":"5"}}}, nos3":"[wired]","nos3params":{"openflowSwitch":{"dpid":"00:00:00:00:00:00:01:15","openflowPort":
{"number":"1", "percentTransmittedDropped":"5"}}},
{"config":{"position":[340,10]},"name":"OpenflowController","value":{"ip":"192.168.1.10","port":"8082","type":"pox"}},
{"config":{"position":[126,16]},"name":"OpenflowController","value":{"ip":"192.168.1.9","port":"8081","type":"floodlight"}},
{"config":{"position":[44,80]},"name":"OpenflowController","value":{"ip":"192.168.1.7","port":"8083","type":"beacon"}},
{"config":{"position":[253,418]},"name":"RRDTool","value":{"refreshTime":""}}],"properties":{"desc":"","name":"template1","nmsit-test":""},
"CON":
[{"src":{"moduleId":1,"terminal":"out"},"des":{"moduleId":0,"terminal":"nos1"}},
{"src":{"moduleId":2,"terminal":"out"},"des":{"moduleId":0,"terminal":"nos2"}},
{"src":{"moduleId":3,"terminal":"out"},"des":{"moduleId":0,"terminal":"nos3"}},
{"src":{"moduleId":4,"terminal":"out"},"des":{"moduleId":0,"terminal":"graphTool"}}]}

Fig. 10. Snippet of template and generated mashment.

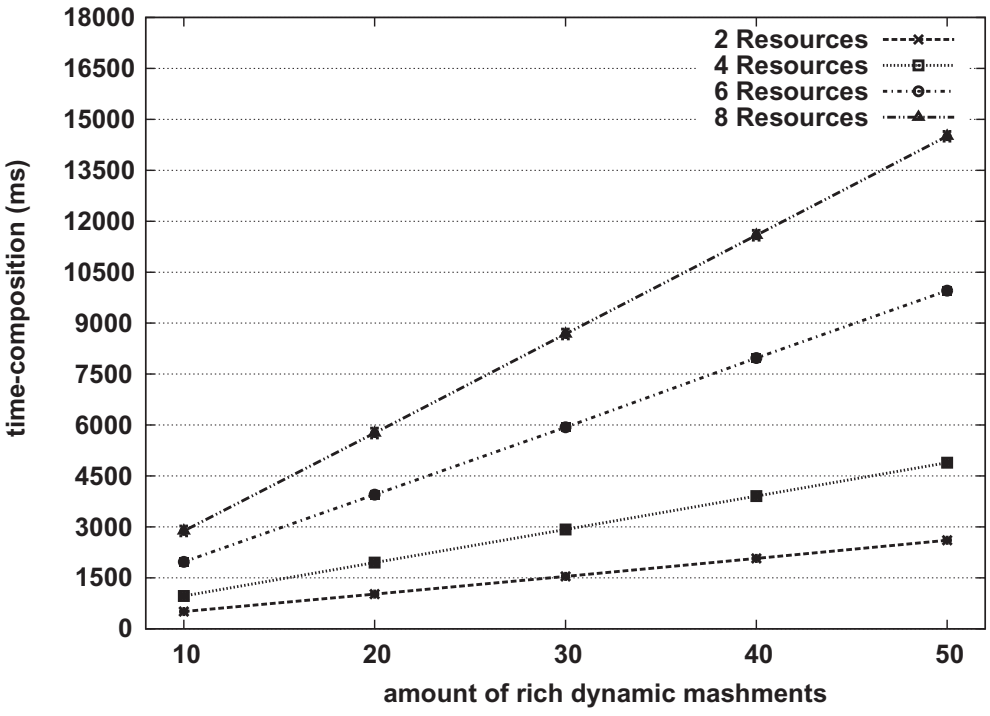

Fig. 11. Time-composition behavior.

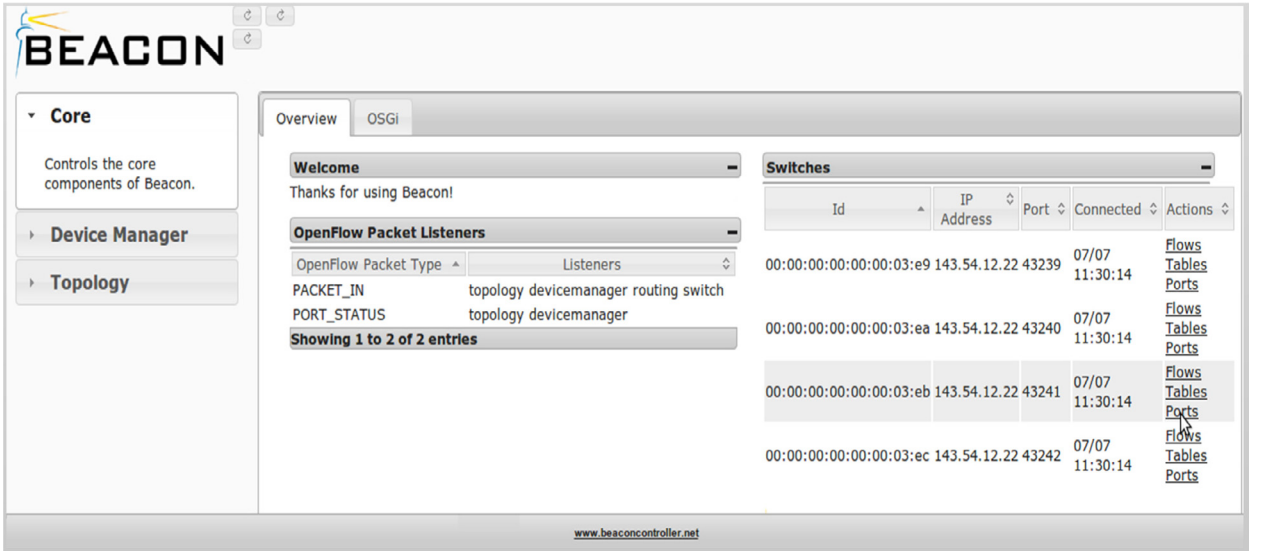

Fig. 12. Beacon Web Tool [64].

administrators spend to carry out tasks supported on computer keyboard and mouse. In KLM, each task is modeled as a sequence of actions. The original KLM-actions [68] and some helpful extensions [69] are: (*i*) press and release a key – $k = 0.2$ s, (*ii*) type a string – $nk * 0.2$ s, (*iii*) point the mouse – $p = 1.1$ s, (*iv*) hold or release the mouse – $b = 0.1$ s, (*v*) move the hand from mouse to keyboard – $h = 0.4$ s, (*vi*) drag-and-drop a visual element – $dnd = 1.3$ s; and (*vii*) wire two visual elements – $wire = 4.1$ s.

Addressing with Situational Script. $T_{cons:\,script}$ is the spent time by the network administrator for addressing NMSit-AS with the *Situational Script*. Without the mashment-based approach, for retrieving situational information, he/she uses a monitoring Web-based tool per controller, such as Beacon Web Tool [64], POX Web Tool [63], and Floodlight Web Tool [65]. As these tools operate similarly, $T_{cons:beacon} = T_{cons:pox} = T_{cons:\,floodlight}$.

The network administrator retrieves information in HTML tables about the packet traffic of a switch from the Beacon Web Tool [64], by the actions (see Fig. 12): (*i*) point the mouse to Core Components tab, (*ii*) press and release the mouse to select the Core Components tab, (*iii*) point the mouse to the Overview tab that presents a switches list, (*iv*) press and release the mouse to select the Overview tab, (*v*) point the mouse to select a switch, (*vi*) press and release the mouse to select a switch, (*vii*) point the mouse to the Ports link; and (*viii*) press and release the mouse to select Ports of switch. Considering these actions, $T_{cons:beacon} = h + 4p + 8b = 5.6$ s and the time of separately use the above mentioned tools is $T_{cons:nonIntegrated} = 16.8$ s.

To obtain in a unique GUI, the retrieved information by the Web-based tools, the network administrator develops and launches the *Situational Script*. As this script generates HTML tables and RRDTool images, $T_{cons:dev} = T_{table} + T_{rrdImages}$. Considering only the time to type the code for generating the tables and images, $T_{cons:dev} = (h + 290k) + (h + 1200k) = 298.8$ s. In turn, $T_{cons:lau} = h + p + 2b + 11k = 3.9$ s. Thus, $T_{cons:script} = T_{cons:nonIntegrated} + T_{cons:dev} + T_{cons:lau} = 319.5$ s.

Addressing with PMM. $T_{cons:\,pmm}$ is the time spent by the network administrator for handling NMSit-AS with *PMM*. Here, $T_{cons:pmm} = T_{cons:dev} + T_{cons:lau} + T_{cons:use}$. As *PMM* is developed in the Mashment Maker [11] [26], $T_{cons:dev} = T_{sel} + T_{con} + T_{com}$.

To develop *PMM* (see Fig. 13), the network administrator initially selects Visual Resources by dragging-and-dropping *Beacon*, *POX*, *Floodlight*, *RRDTool*, and *OF Monitor*. Thus, $T_{sel} = 5 * dnd = 6.5$ s. Afterwards, he/she configures the selected resources by providing the functioning parameters of *Beacon*, *POX*, *Floodlight*, and *RRDTool*. As he/she manually writes these parameters, $T_{con:beacon} = T_{con:pox} = T_{con:\,floodlight} = [4 * (p + h + 2b) + (16 + 8 + 8 + 5) * k] = 14.2$ s and $T_{con:rrd} = p + h + 2b + 3k = 2.3$ s. Therefore, $T_{con} = 44.9$ s. Finally, he/she creates connections among the selected and configured resources by wiring *Beacon - OF Monitor*, *POX - OF Monitor*, *Floodlight - OF Monitor*, and *RRDTool - OF Monitor*. Thus, $T_{com} = 4 * wire = 16.4$ s.

Before requesting the execution of *PMM*, the network administrator saves it: (*i*) point the mouse in the Save button and click it, (*ii*) point the mouse in the dialog that asks for the *mashment* name and click it; and (*iii*) type the string "*PMM*". Once *PMM* has been saved, he/she launches it by clicking the button Run. Thus, $T_{cons:lau} = 3(h + p + 2b) + 3k = 5.7$ s.

On runtime, *PMM* (see Fig. 14) allows the network administrator to retrieve information about NMSit-AS. He/she carries out in *PMM* the following actions to retrieve generic information of switches/links in three different controllers: (*i*) points the mouse to the Controllers list, (*ii*) presses and releases the mouse to select three distinct controllers, (*iii*) points the mouse to the button Switches/Links; and (*iv*) presses and releases the mouse to click the button Switches/Links. Furthermore, to retrieve information about flows, tables, ports, or traffic of three switches: (*i*) presses and releases the mouse to select three switches in

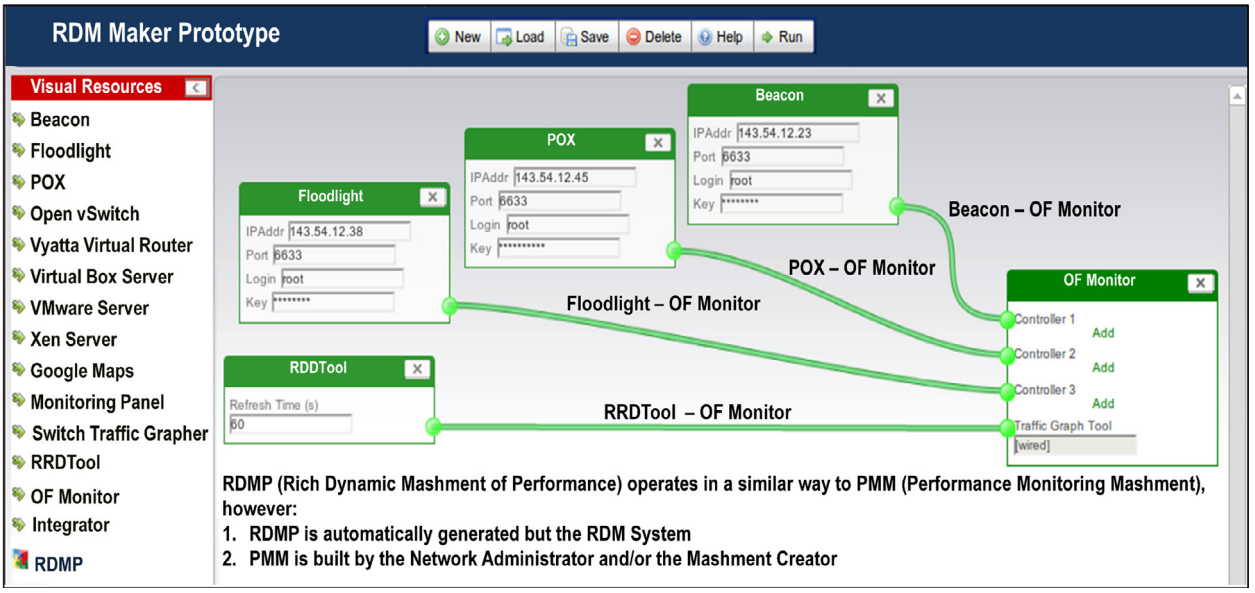

Fig. 13. Rich Dynamic Mashment of Performance.

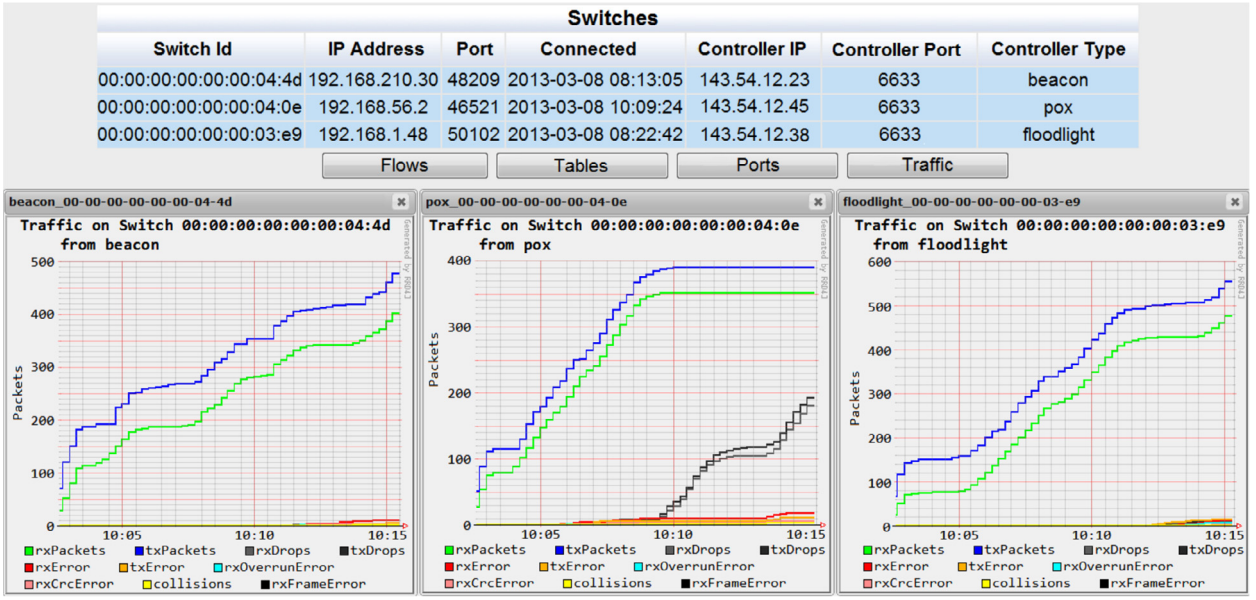

Fig. 14. RDMP and PMM on runtime.

different controllers, (*ii*) points the mouse to the button Flows, Tables, Ports, or Traffic; and (*iii*) presses and releases the mouse to click the button Flows, Tables, Ports, or Traffic. Thus, $T_{cons:use} = h + 3p + 16b = 5.3$ s.

Since we have already calculated $T_{cons:dev}$, $T_{cons:\,lau}$, and $T_{cons:\,use}$, we can calculate $T_{cons:\,pmm}$. As a result, it is expected that a network admistrator takes 78.8 s to deal with NMSit-AS by using *PMM*.

Addressing with RDMP. $T_{cons:\,rdmp}$ is the time spent by the network administrator for facing NMSit-AS with *RDMP*. As *RDMP* is generated by the RDM System (see Figs. 10 and 13), $T_{cons:rdmp} = T_{cons:lau} + T_{cons:use}$. As *RDMP* is a *mashment* dynamically composed, the network administrator launches it by clicking the button Run. Therefore, $T_{lau} = h + p + 2b =$ 1.7 s. Furthermore, since on runtime *RDMP* and *PMM* provide identical functionalities and show the same GUI (see Fig. 14), $T_{cons:use} = 5.3s$. Thus, $T_{cons:rdmp} = 6s$.

Fig. 15 depicts the time-consuming results that reveal: (*i*) the time that the network administrator spends for developing *RDMP* is zero, attained by mechanisms for automatic recognition of *nmsits* and dynamic composition of *mashments*; and (*ii*) because every RDM is ready to be launched by

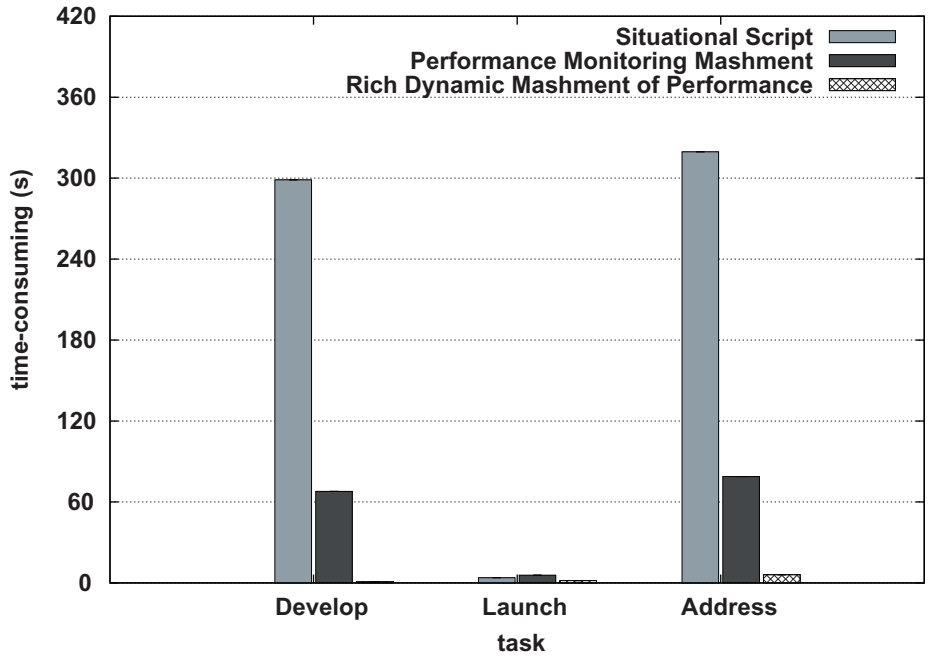

Fig. 15. Time-consuming behavior.

the RDM Designer, the time for launching *RDMP* ($T_{cons:lau} = 1.7$ s) is less than for *SituationalScript* ($T_{cons:lau} = 3.9$ s) and *PMM* ($T_{cons:lau} = 5.7$ s).

Summing up, the time for addressing NMSit-AS with RDM ($T_{cons:rdmp} = 6$ s) is less (about 92.3%–98.1%) than without our approach ($T_{cons:pmm} = 78.8$ s and $T_{cons:script} = 319.5$ s). This global result and the results per task demonstrate that, in terms of time-consuming, it is feasible to use our approach to handling effectively *nmsits*.

5.7. Time-response

To continue the evaluation, it is measured the time-response of *RDMP* and *Beacon Web Tool* when conducting the operations *SwitchesList* and *LinksList* over the networks of the test environment (see Fig. 6). These operations offer visual information of Open vSwitches and their links, which is useful to tackle the NMSit-AS.

In the time-response evaluation of *SwitchesList*, the number of Open vSwitches was varied from 20 to 100 per OpenFlow-based network. Thus, the total number of switches in each evaluation was 60, 120, 180, 240, and 300. Fig. 16 presents the corresponding results. Considering that the time-response (r in *ms*) of Web systems can be ranked as optimal ($r \leq 100$), good ($100 < r \leq 1000$), admissible ($1000 < r \leq 10000$), and deficient ($r > 10000$) [70], the time-response results reveal: (*i*) *SwitchesList* of *RDMP* has a good r that grows negligibly (less than 1 *ms* per switch) when the number of switches is increased in linear and tree topologies; and (*ii*) r is ranked as optimal for *Beacon Web Tool* and as good for *RDMP*; this result was expected because *Beacon Web Tool* operates with one type of controller and *RDMP* with three different types of controller.

In the time-response evaluation of *LinksList*, the number of links was varied from 50 to 250 per OpenFlow-based network. Therefore, the total number of links in each evaluation was 150, 300, 450, 600, and 750. Fig. 17 depicts the corresponding results that reveal: (*i*) *LinksList* of *RDMP* has a good r that grows negligibly (less than 1 *ms* per link) when the number of links is increased in linear and tree topologies; and (*ii*) r is ranked as optimal for *Beacon Web Tool* and as good for *RDMP*; again, this result was expected because *Beacon Web Tool* works with one type of controller and *RDMP* with three different types.

Although, at runtime, *RDMP* uses several software modules (*e.g.*, NMRS like BeaconService and Visual Resources like OF Monitor) to integrate and present monitoring information from different controllers, its behavior on time-response is good for the most of operations and regardless of controllers, topologies, and number of switches and links. Such behavior is because the Adaptation Layer hides the heterogeneity of controllers and, in turn, their centralized nature handles the number of network elements. Summing up, the time-response evaluation results demonstrate that, in terms of such metric, it is feasible to use the proposed approach to dealing with *nmsits* like the raised NMSit-AS.

5.8. Network traffic

To continue the evaluation, we measured the network traffic generated by *RDMP* and *Beacon Web Tool* when carrying out *SwitchesList* and *LinksList* in the networks of the test environment (see Fig. 6). In this and the following evaluations, the network traffic is expressed in *Bytes* or *KBytes*.

In the traffic evaluation of *SwitchesList*, the number of Open vSwitches was varied from 20 to 100 per OpenFlow-based network. Thus, the total number of switches in each

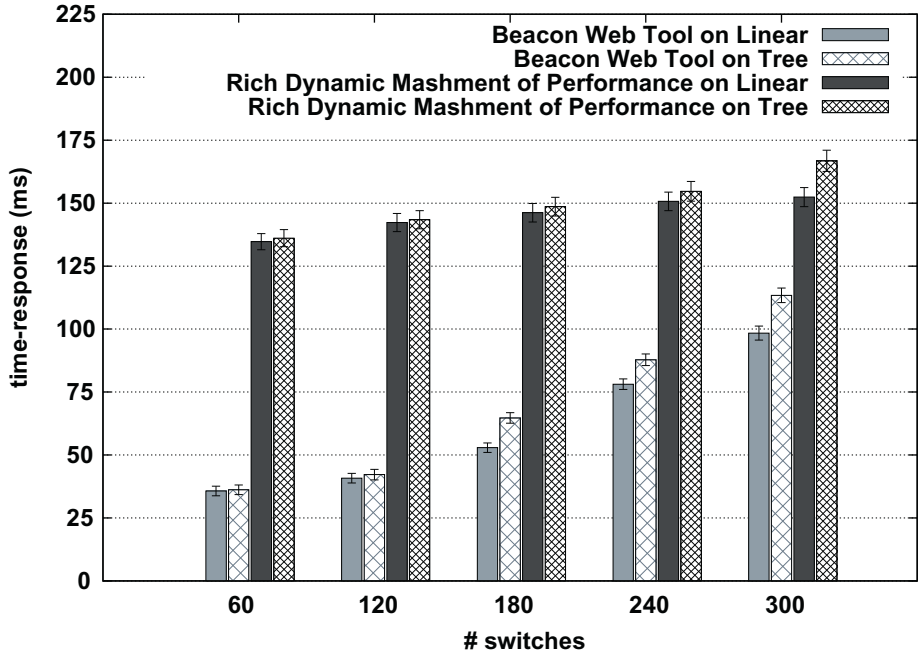

Fig. 16. Time-response on SwitchesList.

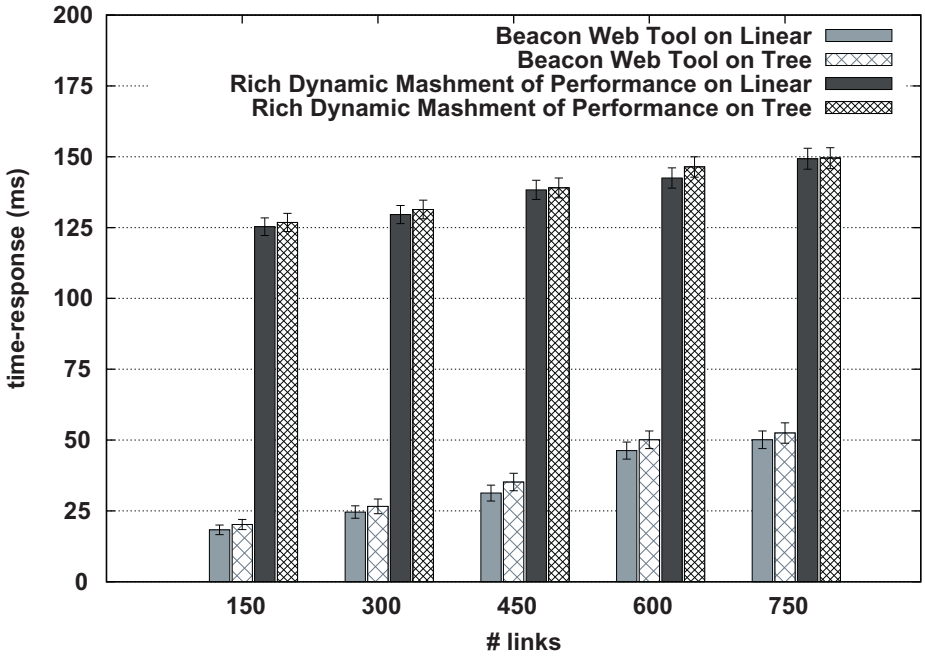

Fig. 17. Time-response on LinksList.

evaluation was 60, 120, 180, 240, and 300. Fig. 18 presents the corresponding results in which there is not discrimination by topology because, the traffic generated by *SwitchesList* of *RDMP* and *Beacon Web Tool* was independent of topologies (linear and tree) tested. In addition, these results reveal: (*i*) the traffic generated by *SwitchesList* of *RDMP* grows negligibly (approx 112 *Bytes* per switch) when the number of switches is increased, (*ii*) in relation to this operation, *RDMP* generates more traffic than *Beacon Web Tool*; and (*iii*) the additional traffic generated by *RDMP* is always less than 10%. Considering that *Beacon Web Tool* operates with just one type of controller and *RDMP* integrates data from three different types, the above facts corroborate that *SwitchesList* of *RDMP* has a good behavior on network traffic.

In the traffic evaluation of *LinksList*, the number of links was varied from 50 to 250 per OpenFlow-based network.

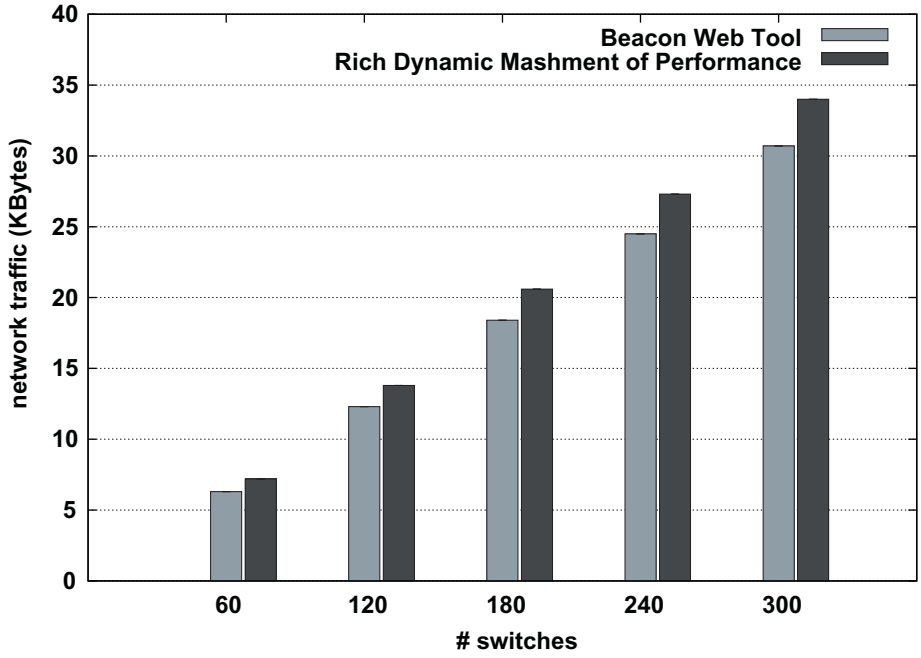

Fig. 18. Traffic on SwitchesList.

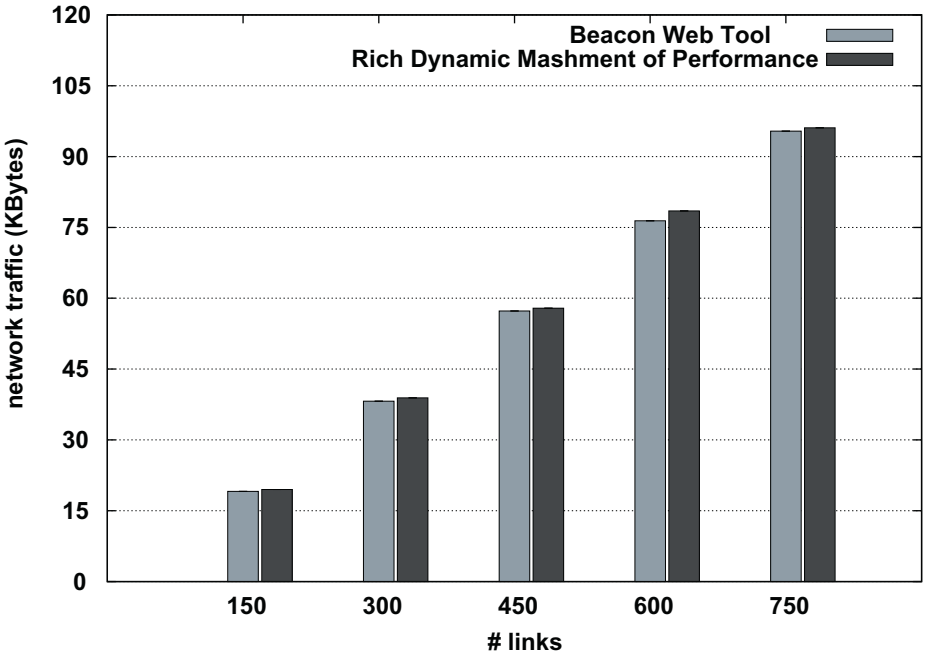

Fig. 19. Traffic on LinksList.

Therefore, the total number of links in each evaluation was 150, 300, 450, 600, and 750. Fig. 19 depicts the corresponding results in which there is not discrimination by topology because the traffic generated by *LinksList* of *RDMP* and *Beacon Web Tool* was independent of topologies tested. Furthermore, these results reveal: (*i*) the traffic generated by *LinksList* of *RDMP* grows negligibly (approx 129 *Bytes* per link) when the number of links is increased, (*ii*) regarding this operation, *RDMP* generates more traffic than *Beacon Web Tool*; and (*iii*) the additional traffic generated by *RDMP* is always less than 5%. Since the *Beacon Web Tool* operates with just one type of controller and *RDMP* integrates data from three different types, the above facts corroborate that *LinksList* of *RDMP* has a good behavior on network traffic.

Regarding the results of the network traffic evaluation of *RDMP*, it is important to mention: (*i*) JSON was used to

decrease the size of information exchanged between the layers of Adaptation, Dynamic Composition, and Presentation because JSON is less verbose than XML; and (*ii*) the size of GUIs is too small to impact the quantity of traffic generated by *RDMP*. Furthermore, although *RDMP* integrates monitoring information from different controllers by using several additional software modules, its extra traffic is always less than 10% (worst operation - *SwitchesList*). Summing up, the above results corroborate that, in terms of network traffic, it is feasible to use our approach to coping with nmsits like the raised NMSit-AS.

5.9. Final remarks

The addressing with and without RDM of several *nmsits* was evaluated and analyzed in terms of: (*i*) time-recognition and time-composition related to mechanisms of automatic recognition of *nmsits* and dynamic composition of *mashments*, (*ii*) time-consuming related to the process followed by network administrators to face *nmsits*; and (*iii*) time-response and network traffic associated with the runtime behavior of solutions used to handle *nmsits*.

The evaluation results revealed several facts. First, the RDM Architecture allows recognizing automatically and composing dynamically rich dynamic*mashments* in a short time. Second, if network administrators cope with *nmsits* (*e.g.*, *NMSit-AS*) by developing and launching static *mashments* (*e.g.*, *PMM*), the time-consuming decreases. Third, such decreasing is greater when mechanisms to automatically recognize nmsits and dynamically compose *mashments* are used (*e.g.*, *RDMP*). Fourth, although an RDM uses extra software layers to face *nmsits*, such layers generate few additional time-response and network traffic in relation to Web-based network management tools (*e.g.*, Beacon Web Tool).

Summing up, the evaluation results demonstrate that, in terms of time-recognition, time-composition, time-consuming, time-response, and network traffic, it is feasible to use the proposed approach to deal with *nmsits* in an effective way. Therefore, such results confirm the relevance of the models of rich dynamic *mashments* and *nmsits*, the mechanisms to automatically recognize *nmsits* and dynamically compose *mashments*, the RDM Maker, and the RDM Architecture as a whole.

From a qualitative point of view, our approach provides mainly flexibility and extensibility. The flexibility refers to that RDM allows network administrators by themselves to customize and improve their workspace. They do not require a lot of Web programming skills to create situational management capabilities (*e.g.*, *PMM* and *RDMP*) because RDM provides a high-level abstraction (*i.e.*, mashable components) of network management technologies as well as of situational management operations.

The extensibility refers to that with our approach, network administrators can create (by conducting a simple process and using existing *mashments*) novel, advanced, and complex situational composite services targeted to overcome *nmsits*. It is possible because RDM leverages the composition, abstraction, and reusing models from mashups as well as allows implementing the investigative and control aspects of SM. Furthermore, in our approach the Mashment Creators can extend the RDM Maker by aggregating *mashments*, *nmsit* patterns, and composition templates. Such extension leads to improving the workspace of network administrators.

According to the evaluation results and the qualitative characteristics of RDM, it can be considered as a step forward in the network management, the SM discipline, and the mashup technology. In this regard, the network management is driven towards an environment focused on situations, composite situational solutions, and network administrators. The mashup foundations are brought up to SM to carrying out its investigative and control aspects. The mashup technology is led to a novel application domain located at the intersection of SM and network management.

6. Conclusions and future work

In this paper, we investigated the feasibility of using SM and mashups as an effective approach to facilitate the daily work of network administrators. The more significant contributions achieved by such investigation are: (*i*) the *nmsits* model that presented a way to characterize unexpected, dynamic, and heterogeneous situations in the network management domain by SM, (*ii*) the rich dynamic *mashments* model that introduced how to use mashups to conduct the investigative and control aspects of SM on network management, (*iii*) the mechanism to automatically recognize *nmsits* that presented how to detect such situations by rules and matching algorithms, (*iv*) the mechanism to dynamically compose *mashments* that introduced how to generate situational solutions by composition templates, and (*v*) the architecture that supported the proposed approach as a whole.

We also presented a prototype that carried out our approach and its evaluation in an SDN-based realistic scenario. In this scenario, we raised diverse *nmsits* and analyzed the feasibility of using rich dynamic *mashments* as an effective approach for network management. Considering the evaluation results, we can state about our approach: (*i*) it recognizes *nmsits* and composes rich dynamic *mashments* in a short time, (*ii*) it decreases the consumption of time of daily tasks performed by network administrators when facing *nmsits*, (*iii*) it has a good behavior in terms of time of response; and (*iv*) it has a good behavior in terms of network traffic because its additional entities and layers generate few extra traffic (less than 10%) in relation to the solutions currently used to cope with *nmsits*.

In the next research steps, we plan to extend and enhance the prototype to support other management tasks (*e.g.*, configuration and accounting) on traditional, SDN-based, and virtual networks. Finally, we also are interested in evaluating the productivity of network administrators that use rich dynamic *mashments*.

Acknowledgments

The CAPES (Brazil) and the University of Cauca (Colombia) supported the research of the professor Caicedo.

References

[1] G. Jakobson, L. Lewis, C. Matheus, M. Kokar, J. Buford, Overview of situation management at SIMA 2005, in: MILCOM, 3, IEEE, Atlantic City, USA, 2005, pp. 1630–1636.

[2] G. Jakobson, On modeling context in situation management, in: CogSIMA, IEEE, San Antonio, USA, 2014.1600–166
[3] G. Jakobson, On conceptualization of eventualities in situation management, in: CogSIMA, IEEE, San Diego, USA, 2013, pp. 75–82.
[4] S. George, W. Zhou, H. Chenji, M. Won, Y.O. Lee, A. Pazarloglou, R. Stoleru, P. Barooah, DistressNet: a Wireless ad hoc and Sensor Network Architecture for Situation Management in Disaster Response, IEEE Commun. Mag. 48 (3) (2010) 128–136.
[5] B. Magoutas, G. Mentzas, D. Apostolou, Proactive situation management in the future internet: the case of the smart power grid, in: DEXA, IEEE, Toulouse, France, 2011, pp. 267–271.
[6] D.M. Hein, R. Toegl, M. Pirker, E. Gatial, Z. Balogh, H. Brandl, L. Hluchý, Securing mobile agents for crisis management support, in: STC, ACM, New York, USA, 2012, pp. 85–90.
[7] R. Koelle, A. Tarter, Towards a distributed situation management capability for SESAR and NextGen, in: ICNS, IEEE, Herndon, USA, 2012 061–O6–12.
[8] I. Pereira, P. Costa, J. Almeida, A rule-based platform for situation management, in: CogSIMA, IEEE, San Diego, USA, 2013, pp. 83–90.
[9] H. Krohns-Valimaki, J. Stranden, J. Sarsama, Improving shared situation awareness in disturbance management, in: CIRED, IET, Stockholm, Sweden, 2013, pp. 1–4.
[10] R. Bruns, J. Dunkel, H. Billhardt, M. Lujak, S. Ossowski, Using complex event processing to support data fusion for ambulance coordination, in: FUSION, 2014, pp. 1–7.
[11] O. Caicedo Rendon, F. Estrada-Solano, L. Zambenedetti Granville, A mashup ecosystem for network management situations, in: GLOBECOM, IEEE, New York, USA, 2013, pp. 2249–2255.
[12] C. Cappiello, F. Daniel, M. Matera, C. Pautasso, Information quality in mashups, IEEE Internet Comput. 14 (4) (2010) 14–22.
[13] N. Laga, E. Bertin, R. Glitho, N. Crespi, Widgets and composition mechanism for service creation by ordinary users, IEEE Commun. Mag. 50 (3) (2012) 52–60.
[14] A. Majchrzak, P.H.B. More, Emergency! Web 2.0 to the rescue!, Commun. ACM 54 (2011) 125–132.
[15] H. Gebhardt, M. Gaedke, F. Daniel, S. Soi, F. Casati, C. Iglesias, S. Wilson, From mashups to telco mashups: a survey, IEEE Internet Comput. 16 (3) (2012) 70–76.
[16] V. Stirbu, Y. You, K. Roimela, V. Mattila, A lightweight platform for web mashups in immersive mirror worlds, IEEE Pervasive Comput. 12 (1) (2013) 34–41.
[17] C. dos Santos, R. Bezerra, J. Ceron, L. Granville, L. Rockenbach Tarouco, On using mashups for composing network management applications, IEEE Commun. Mag. 48 (12) (2010) 112–122.
[18] O.M.C. Rendon, C.R.P. dos Santos, A.S. Jacobs, L.Z. Granville, Monitoring virtual nodes using mashups, Comput. Netw. 64 (2014) 55–70.
[19] X. Chen, Y. Mao, Z.M. Mao, J. Van der Merwe, Declarative Configuration Management for Complex and Dynamic Networks, in: Co-NEXT, ACM, New York, USA, 2010, pp. 6:1–6:12.
[20] N. Kim, J. Kim, Building NetOpe Networking Services over OpenFlow-based Programmable Networks, in: ICOIN, 2011, pp. 525–529.
[21] D. Mattos, N. Fernandes, V. da Costa, L. Cardoso, M. Campista, L. Costa, O. Duarte, OMNI: OpenFlow MaNagement Infrastructure, in: NOF, IEEE, Paris, France, 2011, pp. 52–56.
[22] J. de Santanna, J. Wickboldt, L. Granville, A BPM-based Solution for Inter-domain Circuit Management, in: NOMS, 2012, pp. 385–392.
[23] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, Composing Software-defined Networks, in: NSDI, USENIX Association, Berkeley, CA, USA, 2013, pp. 1–14.
[24] H. Kim, N. Feamster, Improving Network Management with Software Defined Networking, IEEE Commun. Mag. 51 (2) (2013) 114–119.
[25] P. Smith, A. Schaeffer-Filho, D. Hutchison, A. Mauthe, Management patterns: SDN-enabled network resilience management, in: NOMS, 2014, pp. 1–9.
[26] O. Caicedo Rendon, F. Estrada Solano, L. Zambenedetti Granville, An Approach to Overcome the Complexity of Network Management Situations by Mashments, in: AINA, IEEE, Victoria, Canada, 2014, pp. 875–883.
[27] M. Badger, Zenoss Core Network and System Monitoring, Packt, Birmingham, UK, 2008.
[28] C. Chiang, G. Levin, S. Li, C. Serban, M. Wolberg, R. Chadha, G. Hadynski, L. LaBarre, Enabling Distributed Management for Dynamic Airborne Networks, in: POLICY 2009, IEEE, London, UK, 2009, pp. 102–105.
[29] W. Barth, Nagios: System and Network Monitoring, 2nd, No Starch Press, San Francisco, USA, 2008.
[30] G. Jakobson, J. Buford, L. Lewis, Situation Management: Basic Concepts and Approaches, in: Information Fusion and Geographic Information Systems, Lecture Notes in Geoinformation and Cartography, Springer Berlin Heidelberg, New Yor, USA, 2007, pp. 18–33.
[31] E.M. Maximilien, A. Ranabahu, S. Tai, Swashup: Situational Web Applications Mashups, in: OOPSLA, ACM, Montreal, Canada, 2007, pp. 797–798.
[32] M. Davies, P. Hamel, K. Yoshii, M. Goto, AutoMashUpper: Automatic Creation of Multi-Song Music Mashups, IEEE/ACM Trans. Audio, Speech, Language Proces. 22 (12) (2014) 1726–1737.
[33] C. dos Santos, R. Bezerra, J. Ceron, L. Granville, L. Tarouco, Botnet master detection using a mashup-based approach, in: CNSM, 2010, pp. 390–393.
[34] R. Bezerra, C. dos Santos, L. Bertholdo, L. Granville, L. Tarouco, On the Feasibility of Web 2.0 Technologies for Network Management: A Mashup-based Approach, in: NOMS, IEEE, Osaka, Japan, 2010, pp. 487–494.
[35] C. Pautasso, O. Zimmermann, F. Leymann, Restful Web Services vs. Big Web Services: Making the Right Architectural Decision, in: Proceedings of International Conference on World Wide Web, ACM, New York, USA, 2008, pp. 805–814.
[36] M.L. Massie, B.N. Chun, D.E. Culler, The ganglia distributed monitoring system: design, implementation and experience, Paral. Comput. 30 (2003) 2004.
[37] J. Wang, L. Yang, M. Yu, S. Wang, Application of server virtualization technology based on CitriX XenServer in the information Center of the Public Security Bureau and Fire Service Department, in: ISCCS, IEEE, Kota Kinabalu, Malaysia, 2011, pp. 200–202.
[38] T. Oetiker, MRTG: the multi router traffic grapher, in: LISA, USENIX, 1998, pp. 141–148.
[39] T. Oetiker, Monitoring your it gear: the MRTG story, IT Professional 3 (6) (2001) 44–48.
[40] S. Lin, Z. Gao, K. Xu, Web 2.0 Traffic Measurement: Analysis on Online Map Applications, in: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video, ACM, New York, USA, 2009, pp. 7–12.
[41] T.L. Ruthkoski, Google Visualization API Essentials, Packt, Birmingham, UK, 2013.
[42] E. Salvador, L. Granville, Using Visualization Techniques for SNMP Traffic Analysis, in: ISCC, IEEE, Marrakech, Morocco, 2008, pp. 806–811.
[43] I. Juniper Networks, Juniper Home, 2014, [Accessed November 2015].
[44] S.I. ULC, SandvineHome, 2014, [Accessed November 2015].
[45] E.F. Hill, Jess in Action: Java Rule-Based Systems, Manning Publications Co., Greenwich, USA, 2003.
[46] P. Browne, JBoss Drools Business Rules, Packt, Birmingham, UK, 2009.
[47] C. Ibsen, J. Anstey, Camel in Action, 1st, Manning Publications Co., Greenwich, CT, USA, 2010.
[48] R.T. Fielding, R.N. Taylor, Principled Design of the Modern Web Architecture, ACM Trans. Internet Technol. 2 (2) (2002) 115–150.
[49] D. Grigori, J. Corrales, M. Bouzeghoub, A. Gater, Ranking BPEL Processes for Service Discovery, IEEE Trans. Services Comput. 3 (3) (2010) 178–192.
[50] R.C. Angell, G.E. Freund, P. Willett, Automatic Spelling Correction Using a Trigram Similarity Measure, Infor. Proces. Manag. 19 (4) (1983) 255–261.
[51] G.A. Miller, WordNet: a Lexical Database for English, Commun. ACM 38 (11) (1995) 39–41.
[52] A.A. Patil, S.A. Oundhakar, A.P. Sheth, K. Verma, Meteors Web Service Annotation Framework, in: Proceedings of International Conference on World Wide Web, ACM, New York, USA, 2004, pp. 553–562.
[53] M. Bostock, V. Ogievetsky, J. Heer, D^3 data-driven documents, IEEE Trans,. Visual. Comput. Graphics 17 (12) (2011) 2301–2309.
[54] P. Clements, R. Kazman, M. Klein, Evaluating Software Architectures: Methods and Case Studies, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
[55] P. Shanmugapriya, R.M. Suresh, Article: software architecture evaluation methods - a survey, Int. J. Comput. Appl. 49 (16) (2012) 19–26.
[56] R.T. Fielding, R.N. Taylor, Principled design of the modern web architecture, ACM Trans. Internet Technol. 2 (2) (2002) 115–150.
[57] G. Natasha, K. Teemu, P. Justin, P. Ben, C. Martin, M. Nick, S. Scott, NOX: towards an operating system for networks, Comput. Commun. Rev. ACM 38 (3) (2008) 105–110.
[58] B. Lantz, B. Heller, N. McKeown, A network in a laptop: rapid prototyping for software-defined networks, in: Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networks, ACM, New York, USA, 2010, pp. 19:1–19:6.
[59] A. Lara, A. Kolasani, B. Ramamurthy, Network innovation using openflow: a survey, IEEE Commun. Surveys Tutorials PP (99) (2013) 1–20.

[60] O.M.C. Rendon, F. Estrada-Solano, L.Z. Granville, A mashup-based approach for virtual SDN management, in: Proceedings of Annual International Computer, Software & Applications Conference (COMPSAC), 2013, pp. 143–152.
[61] A. Doria, J. Hadi, R. Salim, H. Haas, W. Khosravi, W. Wang, L. Dong, J. Gopal, J. Halpern, Forwarding and control element separation (ForCES) protocol specification, 1 (2010) 5–29. (RFC 5810)
[62] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, Comput. Commun. Rev. ACM 38 (2) (2008) 69–74.
[63] POX, POX Home, 2014, [Accessed November 2015].
[64] D. Erickson, Beacon Home, 2013, [Accessed November 2015].
[65] FloodLight, Floodlight Home, 2014, [Accessed November 2015].
[66] G. Huang, Y. Ma, X. Liu, Y. Luo, X. Lu, B. Blake, Assisting navigation and complementary composition of complex service mashups, IEEE Trans. Services Comput. PP (99) (2014).1–1
[67] A. Ordonez, V. Alcazar, J.C. Corrales, P. Falcarin, Automated context aware composition of advanced telecom services for environmental early warnings, Expert Syst. Appl. 41 (13) (2014) 5907–5916.
[68] D. Kieras, Using the Keystroke-Level model to estimate execution times, in: University of Michigan, 2001.
[69] S. Tian, G. Weber, C. Lutteroth, A tuplespace event model for mashups, in: OzCHI, ACM, New York, USA, 2011, pp. 281–290.
[70] S. Joines, R. Willenborg, K. Hygh, Performance Aanalysis for Java Websites, Addison–Wesley Longman Publishing Co., Inc., Boston, USA, 2002.

Oscar Mauricio Caicedo Rendon is full professor at the Telematics Department of the University of Cauca, Colombia. He received his Ph.D. degree in Computer Science from the Institute of Informatics (INF) of the Federal University of Rio Grande do Sul (UFRGS), Brazil, in 2015. He also received a Master degree in Telematics (2006) and a degree in Electronics and Telecommunications Engineering (2001) from the University of Cauca (UNICAUCA). His research interests include network management, Software-Defined Networking, and Network Function Virtualization.

Felipe Estrada-Solano is a Master student in Telematics at the University of Cauca (UNICAUCA). He received his degree in Electronics and Telecommunications Engineering (2010) from UNICAUCA. His topics of interest include network and service management, Web 2.0/3.0 technologies, and Software-Defined Networking.

Vinicius Guimarães is professor at the Sul-Rio-Grandense Federal Institute of Education, Science, and Technology (IFSul), Brazil, and a Ph.D. student in Computer Science at the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), Brazil. He received his B.Sc. (2005) degree in Computer Science from Catholic University of Pelotas (UCPEL), Brazil and his M.Sc degree from Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil. His research interests include network and service management, Software-Defined Networking, and information visualization.

Liane Margarida Rockenbach Tarouco is full professor at the Center for New Technologies on Education of the Federal University of Rio Grande do Sul (UFRGS), Brazil. She received his M.Sc. degree in computer science, from UFRGS in 1976 and Ph.D. degree from USP in 1990. She was member of IFIP WG 6.5 (Messaging Systems) and WG 6.6 (Network Management) and has served as a TPC member for many important events in the area of computer networks.

Lisandro Zambenedetti Granville received the M.Sc. and Ph.D. degrees in Computer Science from the Institute of Informatics (INF) of the Federal University of Rio Grande do Sul (UFRGS), Brazil, in 1998 and 2001, respectively. Currently, he is a professor at INF-UFRGS. Lisandro is co-chair of the Network Management Research Group (NMRG) of the Internet Research Task Force (IRTF) and vice-chair of the Committee on Network Operations and Management (CNOM) of the IEEE Communications Society (COMSOC). He was also technical program committee co-chair of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010) and 18th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2007). His research interests include network and services management, Software-Defined Networking, network virtualization, information visualization, and network programmability.