Computer Networks 64 (2014) 55–70

Contents lists available at ScienceDirect

Computer Networks

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

Monitoring Virtual Nodes using mashups

Oscar Mauricio Caicedo Rendon [a,b,*], Carlos Raniery Paula dos Santos [a], Arthur Selle Jacobs [a], Lisandro Zambenedetti Granville [a,*]

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

ARTICLE INFO

Article history:
Received 1 May 2013
Received in revised form 6 December 2013
Accepted 5 February 2014
Available online 13 February 2014

Keywords:
Mashups
Monitoring mashups
Network virtualization
System virtualization
Virtual Node
Virtual Node Wrapper

ABSTRACT

The use of virtualization technologies is one of major trends in computer networks. Up to now, most of monitoring tasks on Virtual Nodes, made up of several system virtualization environments and network virtualization environments, require manual intervention via non-standardized interfaces. Although monitoring based on proprietary command lines and graphical user interfaces may be enough for homogeneous Virtual Nodes, it is certainly not suitable for monitoring, in an integrated way, Virtual Nodes in which, the aforementioned environments use heterogeneous virtualization technologies, both in networks and systems. In this paper, we demonstrate that the mashup technology can be used to carry out the integrated monitoring of heterogeneous Virtual Nodes. In this sense, we present a mashup-based architecture targeted to monitor such type of Virtual Nodes, we introduce a reference implementation of the mashup-based architecture, and we develop on it, three monitoring mashups. The quantitative assessment of these mashups corroborates that they generate low traffic and have short response time. Furthermore, their qualitative assessment reveals that it is feasible to provide flexible and extensible mashups for monitoring Virtual Nodes.

1. Introduction

Although the research on network virtualization is quite active today [1], little research was found concerning the integrated monitoring of physical and virtual resources that form part of Virtual Nodes. In order to monitor virtual resources of systems and networks, virtualization vendors are primarily providing proprietary and incompatible Command Line Interfaces (CLIs) and Graphical User Interfaces (GUIs). This lack of compatibility and standardization inevitably hinders the work of Virtual Infrastructure Administrators (including both network and system Administrator competences), who are many times forced to employ multiple monitoring tools, which may lead to serious consequences (*e.g.*, erroneous actions and increase of operating costs) for the organizations.

Even though monitoring based on proprietary and non-standardized CLIs and GUIs may be enough to homogeneous Virtual Nodes, it is certainly not suitable for monitoring, in an integrated way, Virtual Nodes formed by different Virtual Management Interfaces (VMIs), System Virtualization Environments (SVEs) [2], and Network Virtualization Environments (NVEs) [3]. For instance, Virtual Infrastructure Administrators are forced to employ multiple tools to monitor a Virtual Node made up of: (i) one or more virtual machines running on Xen, VMware, and VirtualBox (*e.g.*, Citrix XenCenter and VMware vCenter Operations Management Suite); and (ii) several virtual network elements, such as Open vSwitch (*e.g.*, sFlowTrend) and Vyatta Router (*e.g.*, NetFlow Analyzer). This

* Corresponding authors at: Institute of Informatics, Federal University of Rio Grande do Sul, Av. Bento Gonçalves, 9500 Porto Alegre, RS, Brazil. Tel.: +55 5130281095 (O.M.C. Rendon).

E-mail addresses: omcrendon@inf.ufrgs.br (O.M.C. Rendon), crpsantos@inf.ufrgs.br (C.R.P. dos Santos), asjacobs@inf.ufrgs.br (A.S. Jacobs), granville@inf.ufrgs.br (L.Z. Granville).

http://dx.doi.org/10.1016/j.comnet.2014.02.007
1389-1286/

multiplicity of tools leads to an overload on the monitoring tasks to be conducted by Virtual Infrastructure Administrators in the Virtual Nodes [4].

In our previous work [5], we analyzed the feasibility of using the mashup technology to manage traditional computer networks. It was observed that mashups are able to integrate information from multiple network resources, such as devices and services. We concluded that mashups enable network Administrators to accomplish very specific tasks (*e.g.*, botnet detection) [6] and to create customized management applications (*e.g.*, displaying the traffic of the border gateway protocol between two autonomous systems) [7]. Notwithstanding all the benefits of using mashups in network management, we have not observed their employment for monitoring the aforementioned Virtual Nodes yet.

In this paper, we extend our previous work in order to provide a mashup-based mechanism able to monitor heterogeneous Virtual Nodes. We argue that the composition model of mashups allows to deal with the heterogeneity, complexity, and stiffness of any VMI, SVE, and NVE. This model enables any Virtual Infrastructure Administrator to adapt, customize, and combine existing monitoring tools in order to improve system and network monitoring tasks on virtualized environments. In addition, the employment of mashups also supports the integrated monitoring of both system and network virtual elements, abstracting all technical details related to the interaction with these elements.

The key contributions of our research are:

- Demonstrate that it is feasible the use of the mashup technology for monitoring, in an integrated way, heterogeneous Virtual Nodes made up of several SVEs, NVEs, and their corresponding VMIs.
- Present a reference implementation of a mashup-based architecture for integrated monitoring of Virtual Nodes in which, the above mentioned environments use heterogeneous virtualization technologies, both in networks and systems.
- Demonstrate that, in realistic scenarios, monitoring mashups – built by using the implemented architecture – are flexible, extensible, do not consume bandwidth intensively, and have short response times.

The remainder of this paper is organized as follows. In Section 2, we present the mashup technology background. In Section 3, we review the related work about the virtual network monitoring. In Section 4, we introduce a mashup-based architecture for integrated monitoring of Virtual Nodes. In Section 5, we present the reference implementation of such an architecture. In Section 6, we describe and discuss the case study raised to evaluate our proposal. In Section 7, we provide some conclusions and implications for future work.

2. Mashups background

Mashups are Web applications created by combining different resources available on the Web [8]. They have been considered a fundamental piece of the Web 2.0, allowing end-users, who are not expert programmers, to create their own customized applications. Furthermore, mashups also encourage reusing pre-existing applications and cooperation among end-users.

Two important things contributed to dissemination of mashup technology usage. First, the number of available services and online APIs has increased, and second, new usability-oriented technologies (*e.g.*, AJAX and Macromedia Flash) allowed the creation of more dynamic applications and sophisticated GUIs [9,10]. Online APIs and usability-oriented technologies are the fundamental basis for supporting mashups creation.

The mashup technology is mainly characterized by a simple composition model, which enables customized applications to be easily and rapidly developed and executed by end-users [11,12]. The use of the mashup technology enables, for example, the integration of information from multiple sources at various levels (*i.e.*, data, presentation, and logic). The process of developing new mashups is conducted by mashup systems, which are also responsible for storing and executing these mashups. Mashup systems employ high-level abstractions in order to hide technical details for end-users. Another important characteristic of mashup systems is to support the reuse and extending of existing compositions for generating more sophisticated applications.

Nowadays, mashups are being used in many and distinct domains [13], ranging from simple weather reports [14] to project [15] and network management [5]. In the network management domain, for example, we have observed in a previous work [5] that network Administrators rely on several incompatible tools to manage their networks. Considering that such tools usually expose their results through Web interfaces, these results can be included as part of more complex management mashups. For example, graphs of the Multi Router Traffic Grapher (MRTG) could be displayed within a Google Maps Web page in order to create a monitoring tool able to display the current network status taking into account the geographical location of network elements. Mashups also enable network Administrators to address punctual needs, such as the exhibition of the border gateway protocol traffic exchanged between two autonomous systems [7], that otherwise would be very costly to resolve.

Despite all the benefits of using mashups in network management, their employment for monitoring virtual environments has not been observed yet. Thus, in this paper, we focus on analyzing the feasibility of using mashups to integrate disparate management information sources in virtualized environments of systems and networks. We highlight that our goal is not to observe how easy the employment of mashups for management is, because the easy-of-use is an intrinsic characteristic [11,12,16] of mashups. In order to define a mashup-based solution for monitoring Virtual Nodes, we review, in the next section, some of most important virtual network monitoring solutions found in the literature.

3. Virtual network monitoring

Although issues such as the heterogeneity, complexity, and stiffness of nodes monitoring on virtual environments

have not been directly addressed by mashups, some research has tackled one or more of these issues. Most significant development regarding research on virtual network monitoring are reviewed in this section.

The Web-based customer management system [17] is based on a two-layer architecture, targeted to monitor and control Virtual Private Networks (VPNs). The resource management layer hides the physical network elements through agents that use MIBLets (a logical part of a management information base). In the network management layer, by interacting with agents, a Customer Network Resource Management System (CNRMS) manages VPNs. Although CNRMS is not directly intended for monitoring virtual networks or systems, it is discussed because of its interesting proposal for hiding managed resources.

Lattice [18] is a framework for monitoring virtual network resources, such as virtual machines, virtual routers, and virtual service elements. This framework focuses on providing functionalities to properly monitor any virtual resource that moves from a virtual system to another. A Lattice prototype was developed for monitoring virtual machines that execute under hypervisor (*i.e.*, virtual machine manager) control. An important shortcoming of the Lattice framework is that it is centered on network programmers. Therefore, it was not conceived to allow its adaptation or customization by network Administrators. Moreover, features such as flexibility and extensibility were not considered in Lattice either.

Libvirt and Libvirtd [19] are aimed to allow the implementation of several architectures for remote management of arbitrary virtualization technologies. Drivers forming an abstraction layer of virtual environments are provided, as an Open Source API, written in the C Language. At first, commercial virtualization platforms, such as VMware ESX and Microsoft Hyper-V, were supported by the Libvirt library. Afterwards, other platforms as OpenVZ, VirtualBox, and KVM/QUEMU were also covered. Libvirt is centered on network programmers and not on network Administrators. As a consequence, the Libvirt customization and extensibility is constrained because it cannot be easily integrated or extended by network and system Administrators.

The Information Management Overlay (IMO) [20] system is aimed to allow the efficient and scalable collection of data about a running virtual network. This system is based on a decentralized architecture formed by Information Collection Points, Information Aggregation Points, and a Controller, which are able to monitor virtual network elements (*e.g.*, routers and switches running on a virtual machine). IMO is a low-level monitoring solution that does not offer a front-end. In fact, IMO Controller was conceived to be handled by network programmers and not by network Administrators. Thus, only network programmers can directly use, customize, extend, and enhance the IMO system.

In-Network Management (INM) is a clean-slate proposal for distributed management of future computer networks (*e.g.*, virtual and cloud networks). This proposal defines conceptual elements to facilitate the embedding of management functionalities inside the network and its devices. INM concepts were used to implement two prototypes: (i) the first aimed to supervise peer-to-peer (P2P) environments [21]; and (ii) the second to monitor, using a GUI, the performance of a network that supports the Network as a Service (NaaS) concept [22]. Up to now, there is no prototype of INM targeted to integrated monitoring of nodes on virtual environments.

It is important to highlight that traditional OpenSource monitoring solutions, such as Nagios [23], MonaLisa [24] and Ganglia [25], use diverse plug-ins to supervise different network elements like virtual switches and virtual machines. Unlike, we propose the use of the mashup technology to allow Administrators (networks, systems, and virtual infrastructures) to extend and improve their workspace by themselves.

4. Architecture for Virtual Nodes Monitoring based on mashups

Profiting the main characteristics of mashups, we present an architecture for monitoring heterogeneous Virtual Nodes that support the integration of management information from disparate sources. We define such an architecture by using a layered architectural pattern [26]. In this way, in our architecture, the lower layer provides services to the upper layer through decoupled interfaces, and the upper layer, in turn, consumes services from the lower layer. Fig. 1 depicts layers and elements of the mashup-based architecture that is structured as follows: a Managed Resources Layer, an Adaptation Layer, a Composition Layer, and a Presentation Layer. In a broad sense, the complexity of the Managed Resources Layer is hidden by the Adaptation Layer. The Composition Layer allows to build up monitoring mashups for Virtual Nodes, mainly through combining resources of the Adaptation Layer and the Interaction Elements. The monitoring mashups are displayed and executed in the Presentation Layer.

In the next subsections, we first present the actors that represent the roles played by human beings involved in the monitoring of Virtual Nodes. Second, we introduce the layers and elements used to accomplish such a monitoring.

4.1. Actors

There are six actors involved in the monitoring of Virtual Nodes. They are the Mashup Resource Builder, the Mashup Developer, the Mashup Analyst, the Network Administrator, the System Administrator, and the Virtual Infrastructure Administrator.

The *Mashup Resource Builder* is expected to be a T.I. Developer with significant knowledge about Web programming, network monitoring, and virtualization solutions. This actor is in charge of creating and publishing wrappers of virtual resources to be monitored. A wrapper [27] permits accessing a resource, retrieving and filtering data, and presenting such data in a well-known format. Hence, wrappers are used to deal with the heterogeneity of monitored resources. If a new virtualization technology arises, the Mashup Resource Builder must develop and deploy the corresponding wrapper, allowing such type of technology can be monitored through mashups.

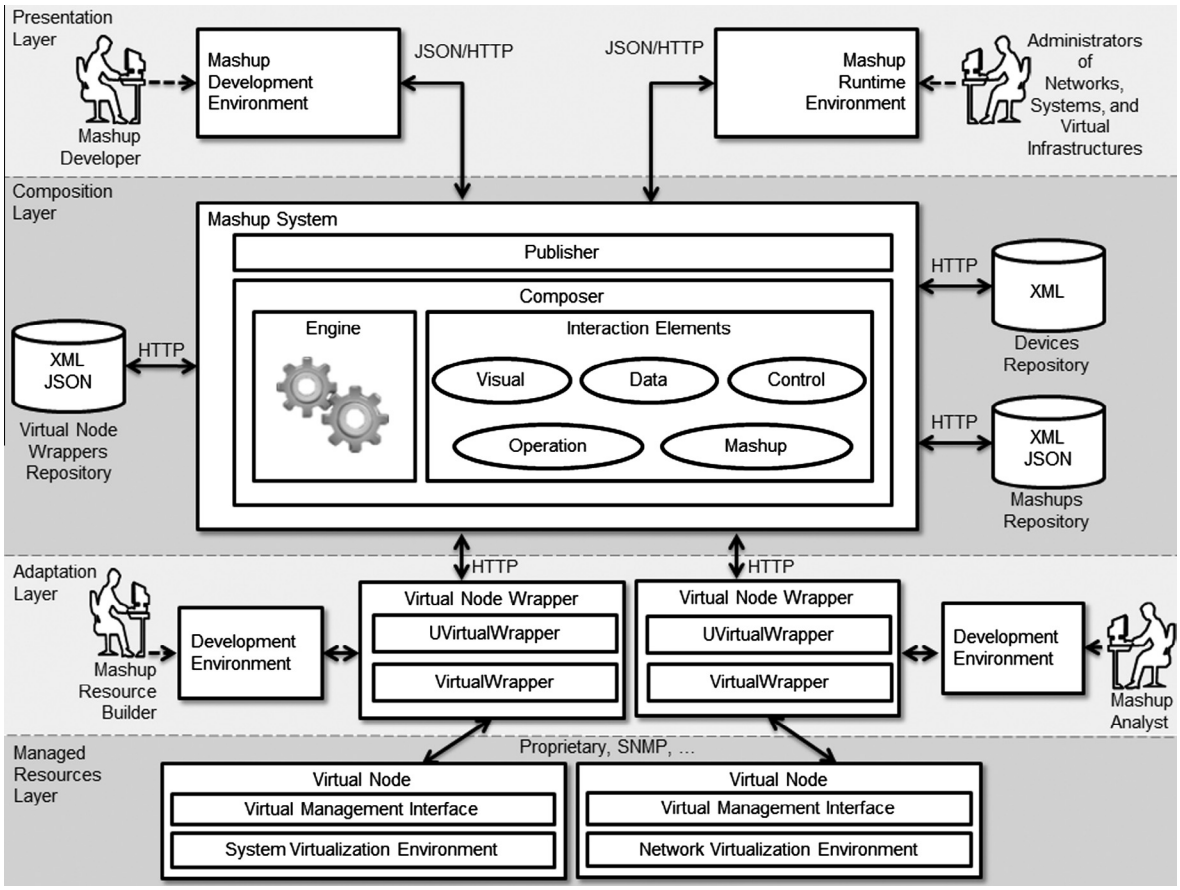

Fig. 1. Architecture for Virtual Nodes Monitoring based on mashups.

The *Mashup Analyst* is expected to be a T.I. Professional with excellent skills about software engineering and computer networks. This actor is in charge of defining requirements for monitoring Virtual Nodes by using mashups. The mashups Analyst translates the necessities of Administrators (networks, systems, and virtual infrastructures) for both the Mashup Resource Builder and the Mashup Developer. Some examples of monitoring requirements are: (i) adding a new system virtualization technology, (ii) incorporating a novel network virtualization technology, (iii) improving a supervision functionality; and (iv) appending a modern and integral GUI.

The *Mashup Developer* is responsible for creating and publishing monitoring mashups by combining resources: visual elements, data sources, control elements, and even mashups. These resources may be internal or external. A resource is external if it is located in a third-party, otherwise it is internal. An example of internal resource is a visual element that represents a specific virtualization technology. An external resource, for instance, is a Google or Yahoo library used online to display a visual interface, such as an organizational chart or a map. Accordingly, the Mashup Developer would need to have technical skills on Web programming.

The *Network Administrator* is in charge of both the monitoring of NVEs by using mashups, and the development of mashups to assist his/her daily activities. These mashups may be composed of basic resources (*e.g.*, datasources, GUIs, and Web Services) and mashups provided by the Mashup Developer. To accomplish that composition, the Network Administrator does not need technical skills about Web programming, since mashups tools function in a high-level abstraction.

The *System Administrator* is responsible for two things, the monitoring of SVEs by means of mashups and the creation of mashups to support his/her everyday work. Likewise to the Network Administrator, the System Administrator does not require technical skills to compose mashups.

The *Virtual Infrastructure Administrator* is in charge of both, the monitoring of Virtual Nodes (involving SVEs and NVEs) through the use of mashups and the building up of mashups to facilitate his/her quotidian work. The Virtual Infrastructure Administrator does not need technical knowledge to create monitoring mashups.

It is noteworthy that, later (Section 5), we provide to Mashup Developers, Network Administrators, System Administrators, and Virtual Infrastructure Administrators a customizable, user-friendly, and high-level development environment. Using such environment, these actors can quickly and conveniently create, adapt, and execute any monitoring mashup. In this sense, the above mentioned

actors are able to extend their monitoring solutions (improving their workspace) devoted to Virtual Nodes.

4.2. Layers and elements

Table 1 introduces the most important abbreviations used in the description of the proposed architecture. In the following paragraphs, the architectural layers and elements are described from bottom to top.

Fig. 2 depicts, using the Common Information Model (CIM), the Virtual Nodes that are located in the Managed Resources Layer. In a general way, Virtual Nodes are made up of SVEs/NVEs and their corresponding VMIs. According the Distributed Management Task Force (DMTF), a SVE [2] is formed by: one or more Host Computer Systems (HCSs), a Virtualization Layer (VL) (*e.g.*, Xen, VMware, VirtualBox, and OpenVZ), and Hosted Virtual Computer Systems (HVCSs). HCS supplies physical resources and VL manages the lifecycle of one or more HVCS. HVCS is composed of virtual resources allocated or assigned by VL from HCS.

In our architecture, we consider two types of Hosted Virtual Computer Systems: (i) NVEs such as the Vyatta Network OS that may be installed on VMware and XenServer, the Open vSwitch that may be operated on VirtualBox and Proxmox VE, and the Mininet OpenFlow VM that works on QUEMU and KVM; and (ii) Virtual Machines of traditional operating systems as Linux and Windows.

A VMI represents one or more tailored APIs used to manage SVEs/NVEs and their constitutive elements. For instance, in the case of XenServer, the VMI can be proprietary (*e.g.*, XenSDK supplied by Citrix Systems) and/or open (*e.g.*, Libvirt, a free software available under the GNU Lesser General Public License – GPL). Another example of VMI is the Remote Access API of the Vyatta Network OS.

Fig. 3 presents the Adaptation Layer that hides the complexity and heterogeneity of the Managed Resources Layer, using a collection of Virtual Node Wrappers. Such collection is in charge of grouping, integrating, and homogenizing the VMIs. The interaction between each Virtual Node Wrapper and its VMI internally occurs in a Virtual Wrapper and depends on protocols (*e.g.*, SNMP, SOAP, HTTP, and Proprietary) provided by virtualization vendors of networks and systems for monitoring their solutions. The Adaptation Layer interacts via the HyperText Transfer Protocol (HTTP) and/or HTTP Secure (HTTPS) with the Composition Layer.

The Virtual Node Wrappers are structured like services, based on the Representational State Transfer (REST) architectural model, that communicate following a request–response model. In REST [28], services are represented by Uniform Resource Identifiers (URIs). These URIs are invoked through HTTP(S) requests, such as GET and POST. Likewise, the replies of every service are HTTP(S) responses.

The Adaptation Layer is able to reply to HTTP(S) requests from the Composition Layer. Specifically, these requests are targeted to URIs pointing to Virtual Wrappers (*i.e.*, implementations of REST-based services) that offer

Table 1
Abbreviations.

Abbreviation	Meaning	Explanation
HVCS	Hosted Virtual Computer System	It is any virtual machine, virtual network, or virtual network element
UHVCS	URI of Hosted Virtual Computer System	A URI that points to a monitoring operation on a HVCS
HCS	Host Computer System	A host supporting hosted virtual computer systems
UHCS	URI of Host Computer System	A URI that targets a monitoring operation on a HCS
NVE	Network Virtualization Environment	It is any network built by using virtualization techniques
SVE	System Virtualization Environment	It is any system built through the use of virtualization techniques
VMI	Virtual Management Interface	An interface to manage NVEs and/or SVEs
VL	Virtualization Layer	A virtualization solution, such as Xen, VMware, VirtualBox, and so on

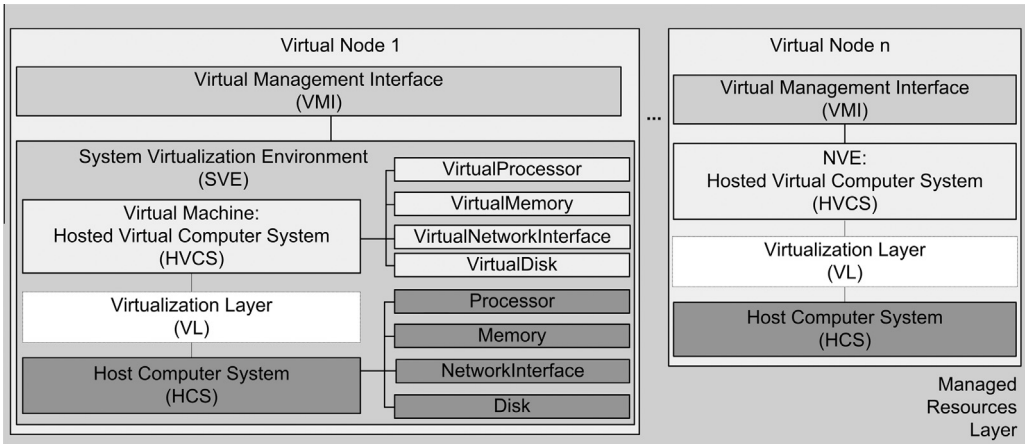

Fig. 2. Virtual Nodes in the Managed Resources Layer – Adapted from DMTF.

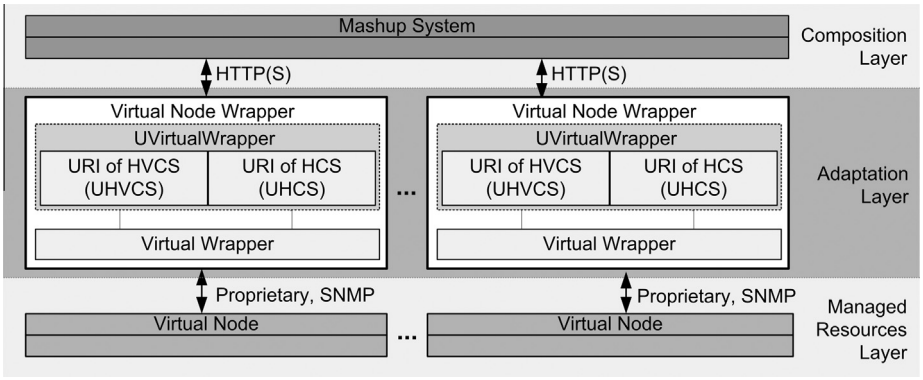

Fig. 3. Virtual Node Wrappers model.

monitoring operations. Each URI that targets a monitoring operation on a Host Computer Systems is called UHCS. An example of UHCS is the URL http://MashupSys/VNWrapper/typeSve/getHCS. With this URL, the Adaptation Layer offers to the Composition Layer the list of Host Computer Systems. The Virtual Wrapper, pointed by the above UHCS, provides such a list with the next structure codified on the JavaScript Object Notation (JSON): $[\{IDHCS : id, NAME : name, IP : ip, OS : os\}, \ldots, \{IDHCS : id, NAME : name, IP : ip, OS : os\}]$. Where, *IDHCS*, *NAME*, *IP*, and *OS* represent the identifier, the name, the IP address, and the Operating System of Host Computer System. Capital and lowercase letters indicate names and values of JSON object properties, respectively.

Each URI that points to a monitoring operation on a Hosted Virtual Computer System is named UHVCS. The set formed by UHCS and UVHCS is called UVirtualWrapper. An example of UHVCS is the URL http://MashupSys/VNWrapper/typeSve/getInfHVCS. Through this URL, the Adaptation Layer offers to the Composition Layer general information about a specific Hosted Virtual Computer System. The Virtual Wrapper, pointed by the above UHVCS, returns the information as a JSON object with the following structure: $[\{IDHVCS : id, NAME : name, OS : os, STATE : st, MEM : memory, CPU : ncpus, LASTUPD : dateupd\}]$. Here, capital and lowercase letters also indicate names and values of JSON object properties. Furthermore, *IDHVCS*, *NAME*, *OS*, *STATE*, *MEM*, *CPU*, and *LASTUPD* represent the identifier, the name, the Operating System, the State (running, turned off, and so on), the assigned static memory, the assigned CPUs number, and the date of last update of Hosted Virtual Computer System, respectively.

The Composition Layer is responsible for integrating, building, and offering Virtual Node monitoring resources to the Presentation Layer. The Composition Layer is formed by the Mashup System, the Virtual Node Wrappers Repository, the Mashups Repository, and the Devices Repository. The Mashup System is made up of the Composer, the Engine, the Publisher, and the Interaction Elements (*i.e.*, Visual, Data, Control, and Mashup). The Composer coordinates the invocation of the Interaction Elements and performs Operations (*e.g.*, sorting, filtering, and aggregating) over the information retrieved (JSON Objects) in such invocations. A Visual Element (*e.g.*, Google Chart and Yahoo Map libraries) is an API that provides mechanisms to build simple and efficient GUIs. A Data Element represents a mashup datasource (*e.g.*, a document containing monitoring information about a virtual router). A Control Element, for instance a management dashboard, determines when, where, and how the Data, Visual, Operation, and mashups are combined and triggered.

The Engine is the lifecycle manager of monitoring mashups and the Mashup System as a whole. In this way, when an initial request to execute a particular monitoring mashup is received, the Engine invokes the Composer (it is responsible for coordinating the functioning of such a mashup). Afterwards, the Engine waits indications from the Composer in order to create, cache, or delete instances of the corresponding mashup (including its elements). Furthermore, when Virtual Infrastructure Administrators require to develop monitoring mashups, the Engine is responsible for creating, caching and deleting instances of the Mashup Development Environment.

In a broad sense, the Publisher is in charge of providing the adaptation/publication of content (*i.e*, Interaction Elements – comprising the monitoring mashups – and the Mashup Development Environment) that is offered to Virtual Infrastructure Administrators by the Presentation Layer. In this sense, when a request to display content arrives from the Engine, first, the Publisher reads the HTTP-header of such a request to identify the features of client device in which the content is going to be presented. Second, the Publisher queries the Devices Repository to establish the corresponding device capabilities. Third, using these capabilities, the Publisher adapts and publishes the content. Fourth, the Publisher asks to the Mashup Runtime Environment to present the adapted content.

The Virtual Node Wrappers Repository stores metadata that describes and points each monitoring operation offered by Virtual Node Wrappers. For instance, the following stored metadata describes the operations (get a list of Host Computer Systems and get statistics of a Hosted Virtual Computer System) of a Virtual Node Wrapper that interacts with a SVE: $[\{PATH : /location/getHCS/, PAR : /ip/port/user/key/, PRODUCE : json\}, \{PATH : /location/statsHVCS/, PAR : /id/, PRODUCE : json\}]$. Thus, in this Repository, every Virtual Node Wrapper is represented by its operations (PATH indicating its location in the Mashup

System and its name), the parameters (PAR) needed to invoke such operations, and the data type that each operation produces/returns.

The Mashups Repository keeps metadata of mashups built in the Mashup System. This metadata is structured by means of JSON notation as follows. [{*IDRESOURCE* : *id*, *NAME* : *name*, *PAR* : *parameters*, *TO* : *reSrc*, *FROM* : *reDes*}, *ldots*, {*IDRESOURCE* : *id*, *NAME* : *name*, *PAR* : *parameters*, *TO* : *resSrc*, *FROM* : *resDes*}]. Therefore, in the Mashups Repository, a mashup is represented by one or more resources identified by *IDRESOURCE* and *NAME*, the functioning parameters of these resources defined by *PAR*, and the relationships among resources defined by *TO* and *FROM*. In this metadata, lowercase letters indicate values of properties.

The Devices Repository is based on the User Agent Profile [29] and the Wireless Universal Resource File [30]. This repository stores, by using the eXtensible Markup Language (XML), the information about capabilities of mobile devices. These capabilities are used by the Publisher to facilitate the presentation of monitoring mashups on diverse terminals as smartphones and tablets.

The Presentation Layer permits to build and extend Virtual Nodes Monitoring solutions, by means of the Mashup Development Environment and the Mashup Runtime Environment. Using, the Mashup Development Environment that is based on visual and drag-and-drop mechanisms, the Network Administrators, System Administrators, Virtual Infrastructure Administrators, and Mashup Developers can combine Interaction Elements in order to create, adapt, and customize monitoring mashups. The Mashup Runtime Environment represents browsers, which are software entities in charge of showing and executing, anywhere and anytime, the monitoring mashups.

5. Reference implementation

The reference implementation is composed of four elements: the Managed Resources, the Mashups System Server, the Mashup Development Environment, and the Mashup Runtime Environment. These elements are detailed in the following subsections.

5.1. Managed resources

SVEs, located in the Managed Resources Layer, include: (i) XenServer used as Host Computer System and Virtualization Layer. XenServer [31] is a virtualization platform distributed by Citrix Systems, Inc., that executes directly on server hardware, without requiring an operating system (*i.e.*, standalone monitor mode), (ii) VirtualBox used as Virtualization Layer. VirtualBox [32] is a virtualization system, distributed by Oracle under the terms of GPL, that runs on operating systems (*i.e.*, hosted monitor mode) such as Windows, Linux, and Mac OS X. Here, we use Linux Debian as Host Computer System of VirtualBox, (iii) Open vSwitch (*i.e.*, a virtual switch) is a Hosted Virtual Computer System; and (iv) Linux-based virtual machines are also Hosted Virtual Computer Systems.

NVEs, located in the Managed Resources Layer, include: (i) Floodlight [33] is an OpenFlow Controller developed in the Java Language and deployed as Hosted Virtual Computer System, (ii) Open vSwitch is a Hosted Virtual Computer System handled by Floodlight, (iii) Virtual Links communicate Open vSwitches and are managed by Floodlight, (iv) Flows contain rules to control the communication in OpenFlow-based networks and are handled by Floodlight; and (v) Mininet used as Host Computer System of Open vSwitches. The Mininet [34] is a software to emulate OpenFlow networks and use VirtualBox as Virtualization Layer. Concerning Links and Flows, it is important to point out that they are particular elements of the Hosted Virtual Computer Systems forming part of NVEs.

In the Managed Resources Layer, VMIs are: (i) the XenSDK (*xensdk*), a virtual machine that enables the comprehensive remote management (including Hosted Virtual Computer Systems) of the XenServer hosting it, (ii) the VirtualBox Web Service (*vboxws*) that lets to thoroughly manage, in a remote way, any VirtualBox server (including Hosted Virtual Computer Systems); and (iii) The Floodlight Web Service (*floodlightws*) that permits the management of an OpenFlow network controlled by Floodlight, including flows, links, and virtualized Open vSwitches. In the Reference Implementation, the VMIs were only used for monitoring purposes.

5.2. Mashups system server

The Mashups System Server includes the Adaptation and the Composition Layers, the Virtual Node Wrappers Repository, and the Mashups Repository. These repositories are hereinafter named *mashupsdb*. The Adaptation Layer is formed by three services: *xenservice*, *vboxservice*, and *floodlightservice*. The *xenservice* implements the Virtual Node Wrapper for SVEs based on XenServer, using RESTful that is a REST implementation on the Java Language. Similarly, *vboxservice* is the RESTful implementation of the Virtual Node Wrapper for SVEs based on VirtualBox. Monitoring operations of *xenservice* are innerly performed by the XenSDK API (version 6.0). Likewise, monitoring functionalities of *vboxservice* are internally carried out by the VirtualBoxSDK API (version 4.1). The *floodlightservice* implements the Virtual Node Wrapper for NVEs based on the Floodlight Controller. The monitoring operations of *floodlightservice* are conducted by the Floodlight REST API (version 1.0). It is important to note that by developing Virtual Node Wrappers as services the complexity of Virtual Nodes and their virtualization technologies is hidden for Administrators of networks, systems, and virtual infrastructures.

The Composition Layer is formed by: *composer* and *publisher*. The *composer* is a Web application, based on Java Servlets and AJAX, that implements both the Composer and the Engine of our architecture. Internally, the *composer* is in charge of invoking, via HTTP, the Virtual Node Wrappers (*i.e.*, *xenservice*, *vboxservice*, and *floodlightservice*) needed to build the workflow and to execute the mashups defined by Network Administrators, System Administrators, Virtual Infrastructure Administrators, and Mashup Developers. The results of *composer* are offered to *publisher* by means of JSON objects transported in HTTP responses.

The *publisher* is also a Web application based on AJAX and Java Servlets, that sends for the Mashup Runtime Environment the results of the composition conducted by *composer*. These results are displayed by using JavaScript and the Google Visualization API. This API renders its visual components through the HyperText Markup Language (HTML) version 5. Accordingly, the Reference Implementation provides cross-browser and cross-platform compatibility to run mashups on smartphones, tablets, desktops, and laptops.

The metadata of monitoring mashups and their resources (*e.g.*, one URL targeting *floodlightservice*) is stored into the *mashupsdb* for promoting their reuse and allowing the extension and improvement of the Reference Implementation. The *mashupsdb* was implemented using a MysqlServer version 5.1.

5.3. Mashup Development Environment

The Mashup Development Environment is a Web application based on the frameworks: YUI and WireIt. The former is an Open Source, JavaScript and Cascading Style Sheets (CSSs) framework for building highly interactive applications. The latter is a set of Open Source JavaScript libraries used to create wirable interfaces for dataflow applications, visual programming languages, graphical modeling, and graph editors.

Fig. 4 depicts the Mashup Development Environment that is formed by six visual components: *xen*, *vbox*, *dashboard*, *floodlight*, *ofmonitor*, *integrator*, and *designer*. The first two components, *xen* and *vbox*, are the visual representation (*i.e.*, a high-level encapsulation of system virtualization technologies) of SVEs based on XenServer and VirtualBox, respectively. The *dashboard* is a collection of views and operations used for graphically presenting information about Host Computer Systems and guests (*i.e.*, a common term used to refer a Hosted Virtual Computer Systems) running on them. Every *dashboard* operation can easily be applied to every Virtual Node by drawing visual links into the *designer*. In this way, the complexity of monitoring operations is hidden for Mashup Developers, Network Administrators, System Administrators, and Virtual Infrastructure Administrators. The operations encapsulated by *dashboard* are, mainly, to retrieve the Host Computer System list, retrieve the guest list for each Host Computer System, retrieve statistics for each Host Computer System and its guests, and start, stop, and resume guests.

The *floodlight* is the visual representation of a NVE (*i.e.*, a high-level encapsulation of network virtualization technologies) based on Floodlight Controller running on a Virtualization Layer, such as VirtualBox and Xen. The *ofmonitor* is a collection of views and operations used to present, in a graphic way, information about Floodlight-based NVEs. The operations provided by *ofmonitor* are, primarily, to retrieve list of virtual flows, virtual Open vSwitches, and virtual links, to retrieve details about each of these virtual elements, and to retrieve packet traffic information on each Open vSwitch. The *integrator* is a visual component that allows to integrate two monitoring mashups in a GUI-level.

The *designer* allows Network Administrators, System Administrators, Virtual Infrastructure Administrators, and Mashup Developers to create, save, load, delete, and run monitoring mashups. It is important to stand out that, first, the building up of mashups is based on a development process aided by dragging-and-dropping and wiring of available elements on the *designer*. Second,

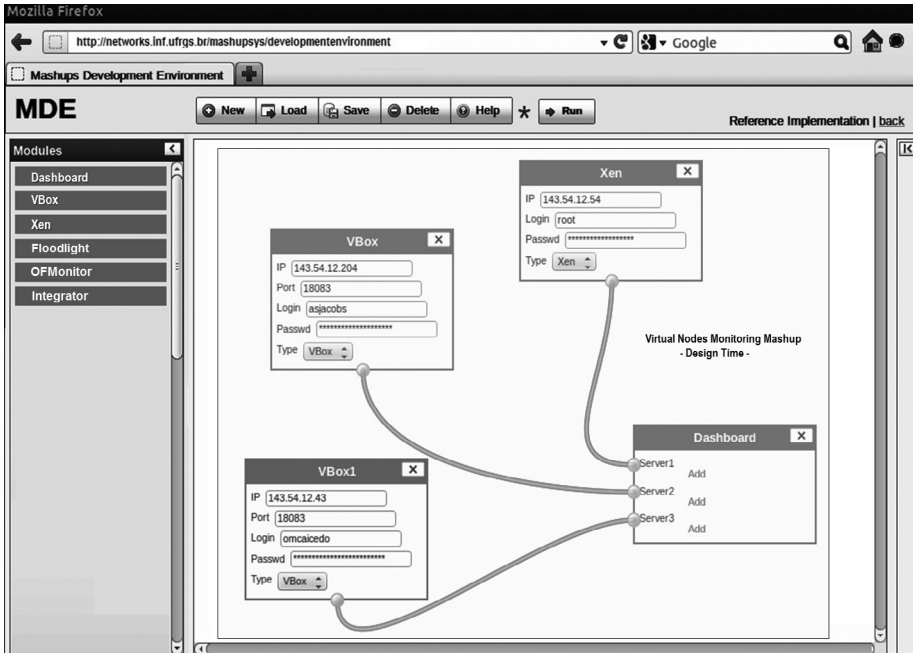

Fig. 4. Mashup Development Environment and Virtual Nodes Monitoring Mashup on runtime.

all *designer* operations (*e.g*, save, load, and delete) are supported by the *mashupsdb*. Third, the combination of these elements assists the enhancement of monitoring mashups. Fourth, such an enhancement supports the improvement of the *designer* and, consequently, the upgrade of workspace used by the above mentioned Administrators.

5.4. Mashup Runtime Environment

Every standardized Web browser that supports HTML version 5 and AJAX can be used as Mashup Runtime Environment to access and execute: (i) the dynamic GUI of the Mashup Development Environment that can be customized and extended during the mashup creation (design phase) by Mashup Developers, (ii) the monitoring mashups that can be invoked by Administrators (networks, systems, and virtual infrastructures) from the Mashup Development Environment or directly using URIs; and (iii) the Virtual Node Wrappers that can be requested by Mashup Resource Builders and Mashup Developers via URIs (*e.g*, UHCS and UHCVS), for instance, during the integration of a new virtualization technology.

It is to stand out that the Mashup Runtime Environment exchanges JSON data with both the Mashup Development Environment and the Mashups System Server by using a synchronous and/or asynchronous communication model. For instance, in the GUI of the Mashup Development Environment, the HTTP synchronous model is used to save a mashup. Instead, during the performing of all monitoring mashup operations (*e.g.*, to retrieve and show a Host Computer System list), the AJAX asynchronous model is used to avoid the blocking of GUIs.

6. Case study

Fig. 5 presents the test environment of the case study used to evaluate our proposal. This environment supports the deployment of the Reference Implementation, the virtual infrastructure (SVEs and NVEs), and the mashups developed to monitor such an infrastructure. Every Xen, VirtualBox, and Repository of Xen Guests was executed on a machine with 2.33 GHz core 2 duo processor, 2 GBytes RAM, and 160 GBytes hard disk. Both the Mashups System Server and the MashupsDB (*i.e.*, the *mashupsdb* instantiation) 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. The virtualized Floodlight and the virtual Open vSwitches handled by it, were deployed on a server with 8 GBytes RAM and 3.4 GHz core i7 processor. The Client was run on a personal computer with 2 GBytes RAM and 2.53 GHz core 2 duo processor.

Fig. 6 depicts the relationships between the Managed Resources Layer, the Adaptation Layer, the Virtual Nodes Monitoring Mashup (VNMM), the Floodlight Monitoring Mashup (FMM), and the Integrated Monitoring Mashup (IMM). VNMM allows System Administrators to monitor three heterogeneous SVEs. The first one, a *xenpool* formed by: two XenServers that support seven virtual machines of Linux Ubuntu OS and three Open vSwitch (deployed on a virtual machine of Linux Debian OS). The other two SVEs are based on *vbox*/Debian. Each *vbox* supports two guests: one Open vSwitch (deployed on a virtual machine of Linux Debian OS) and a virtual machine of Linux Ubuntu OS. The VMIs of *xenpool* and *vbox* are *xensdk* and *vboxws*, respectively. FMM permits Network Administrators to monitor a NVE based on Floodlight/*vbox* and Open vSwitch/Mininet/*vbox*. This NVE is an OpenFlow-based virtual network that uses Floodlight in the control layer and Open vSwitch in the datapath. The VMI of such a NVE is *floodlightws*. IMM allows Virtual Infrastructure Administrators to monitor the aforementioned SVEs and NVEs, by integrating VNMM and FMM.

In the next subsections, VNMM, FMM, and IMM are detailed and evaluated. Furthermore, these three mashups are quantitatively and qualitatively analyzed.

6.1. Virtual Nodes Monitoring Mashup

In the case study, we used the Reference Implementation to build up VNMM (Fig. 4) that allows to monitor non-homogeneous SVEs. On runtime, VNMM offers the following operations (Fig. 7): (i) `nodeList` retrieves a list with basic features (*e.g., name, node type, ip address*) of

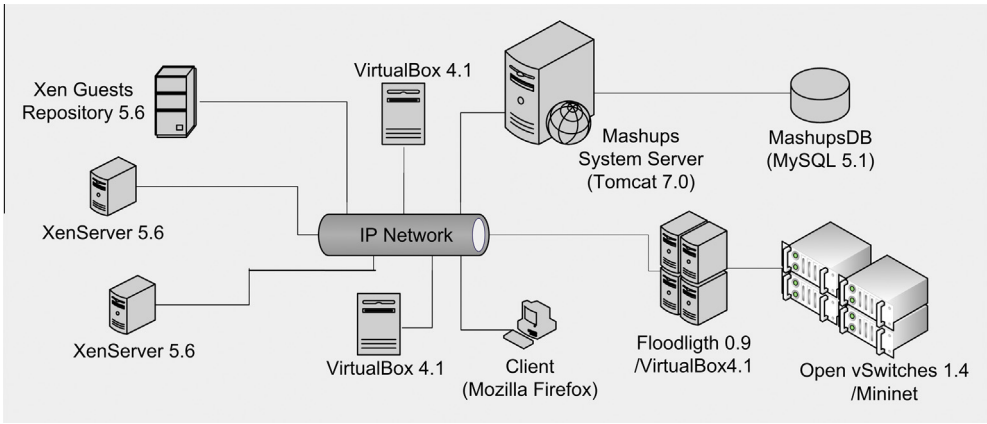

Fig. 5. Test environment.

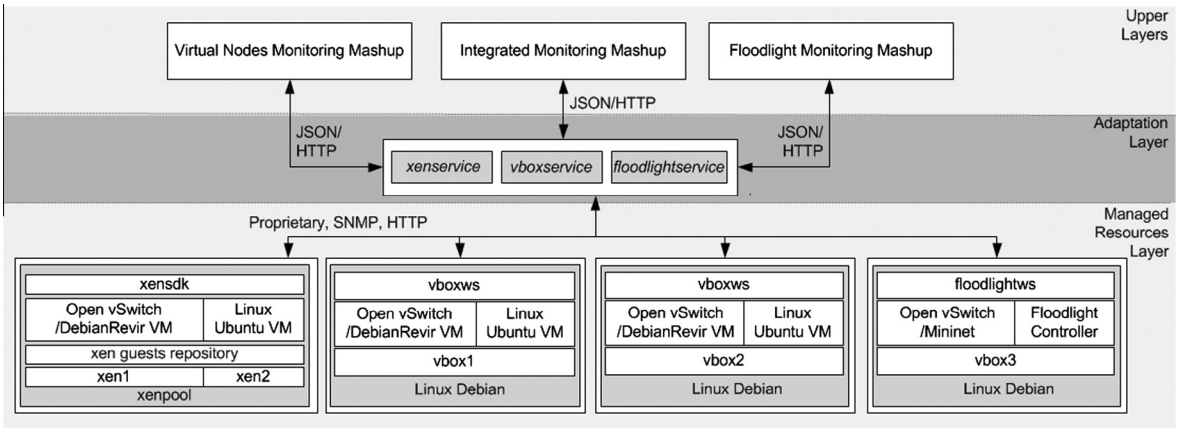

Fig. 6. VNMM, FMM, and IMM on the architecture.

SVEs forming a Virtual Node, shown in the welcome page, (ii) `nodeStructure` retrieves the structure of a selected Host Computer System, shown by an organizational chart, (iii) `guestFeaturesXen` retrieves basic features of a selected Hosted Virtual Computer System (*e.g.*, Open vSwitch/ DebianRevir) running on XenServer, presented in a data table, (iv) `guestFeaturesVB` retrieves basic features of a selected Hosted Virtual Computer System running on VirtualBox, presented in a data table, (v) `guestStatsXen` retrieves statistics (in the last hour) about performance of memory, processor, and network interfaces for a guest on XenServer, depicted by using a line chart, (vi) `guestStatsVB` retrieves the behavior (instantaneous) of memory and processor for a guest on VirtualBox, displayed through a bar chart; and (vii) `control` starts, stops, and resumes guests from XenServer and VirtualBox.

In VNMM, initially, we evaluated both the response time and the network traffic of three operations (*i.e.*, `nodeList`, `guestFeatures (Xen/VB)`, and `guestStats (Xen/VB)`. In such an evaluation, we performed the measurements of time and traffic with all virtual switches and virtual machines running on. In this and the following evaluations that involved the average response time in seconds, we took 30 measurements with 95% confidence level.

According the performance analysis of java Web sites [35], the response time (r in *seconds*) of this kind of system can be ranked as: optimal when $r \leq 0.1$, good for $0.1 < r \leq 1$, admissible for $1 < r \leq 10$, and deficient if $r > 10$. Considering these thresholds, Table 2 reveals that `nodeList`, `guestStats`, and `guestFeatures` had a good or admissible response time average for SVEs based on

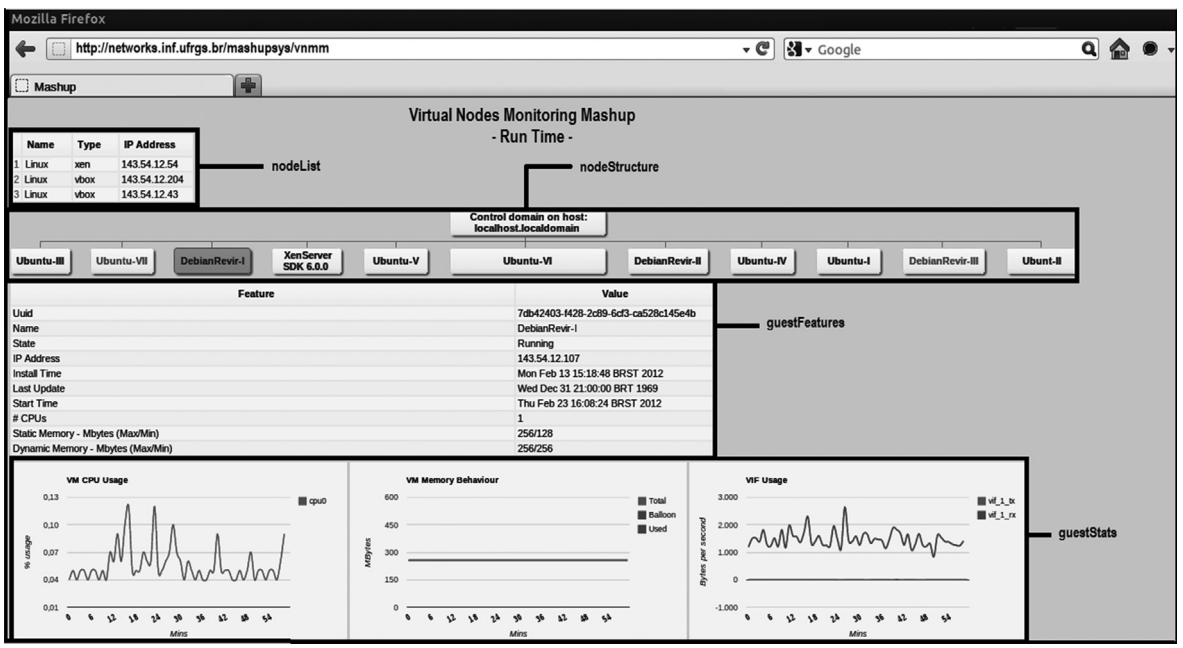

Fig. 7. Virtual Nodes Monitoring Mashup on runtime.

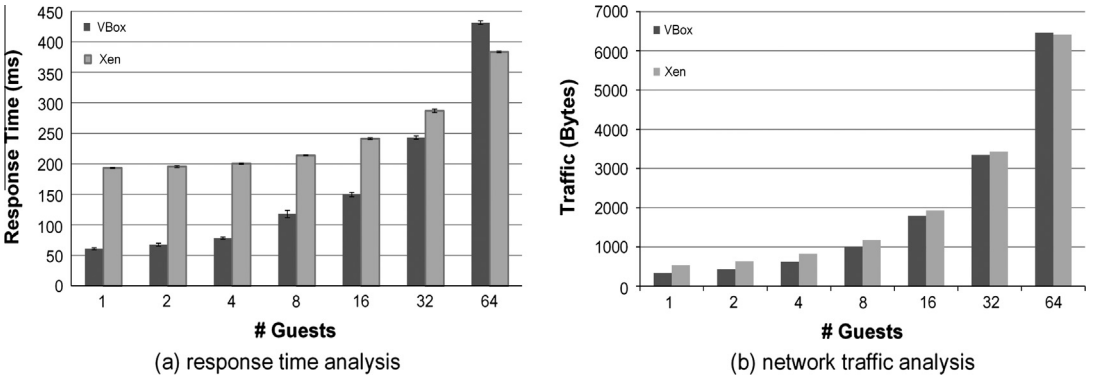

Fig. 8. VNMM – nodeStructure.

XenServer and VirtualBox. As expected, `nodeList` had the highest response time. This operation integrates information of all SVEs rather than, for instance, `guestStats` (*i.e.*, `guestStatsXen` or `guestStatsVB`) that retrieves information from only one SVE. The operations `guestFeaturesXen` and `guestFeaturesVB` behaved like `guestStats`.

Table 2 also discloses that proprietary scripts (executed by command line) had better response time than `nodeList`, `guestStats`, and `guestFeatures`. Since the extra time of `guestStats` and `guestFeaturesXen` is less than 10%, we strengthen the statement that they have a good behavior on response time. As expected, the response time of `nodeList` was the highest because mashups use additional layers to collect, aggregate, and present monitoring information from different SVEs. Being that the additional time on `nodeList` is closing to 48%, in a future implementation of the Mashups System Server, we must carry out information collection in a more efficient way. We consider that such additional time is not a relevant constraint of VNMM (admissible behavior), which aims to instantiate the mashups monitoring concept and non-present a commercial solution.

Regarding network traffic, Table 3 reveals that VNMM generated low traffic to retrieve information about hosts (*i.e.*, `nodeList`) and features of guests (*i.e.*, `guestFeaturesXen` and `guestFeaturesVB`). In `guestStats`, because of implementation differences in XenSDK and VirtualBox Web Service, the statistics for a specific guest in *xenpool* are formed by using a large number of JSON objects that show the guest behavior during the last hour. However, in Nodes based on VirtualBox, this operation only shows a snapshot, so few objects are required.

Table 3 also discloses that `nodeList`, `guestStats`, and `guestFeatures` generated more network traffic than proprietary scripts (developed and executed on Linux Ubuntu OS). Since, the additional network traffic is less than 10%, we bolster the statement that such operations have a good behavior on network traffic. In this sense, it is important to highlight that we used JSON in order to decrease the size of information exchanged between the Adaptation, Composition, and Presentation Layers.

In VNMM, we also conducted the assessment of the response time on the operation `nodeStructure`, modifying at each SVE, the number of guests from 2^0 to 2^6. If the number of guests was $\leqslant 8$, we used all guests in active state

Table 2
VNMM and proprietary scripts – response time.

Operation	XenServer-based SVE		VirtualBox-based SVE	
	VNMM (ms)	Script (ms)	VNMM (ms)	Script (ms)
`nodeList`	1408 ± 110	725 ± 35	1408 ± 110	725 ± 35
`guestFeatures`	190 ± 3.69	174 ± 4	63 ± 3	61 ± 2
`guestStats`	372 ± 4.71	340 ± 13	600 ± 15	550 ± 14

Table 3
VNMM and proprietary scripts – network traffic.

Operation	XenServer-based SVE		VirtualBox-based SVE	
	VNMM (Bytes)	Script (Bytes)	VNMM (Bytes)	Script (Bytes)
`nodeList`	487	450	487	400
`guestFeatures`	459	456	351	318
`guestStats`	3061	2800	155	153

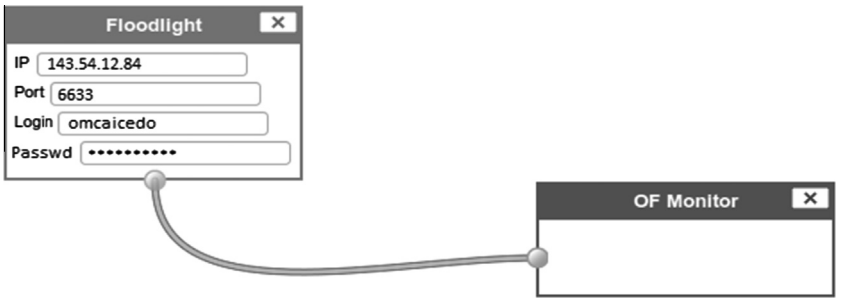

Fig. 9. FMM on design time.

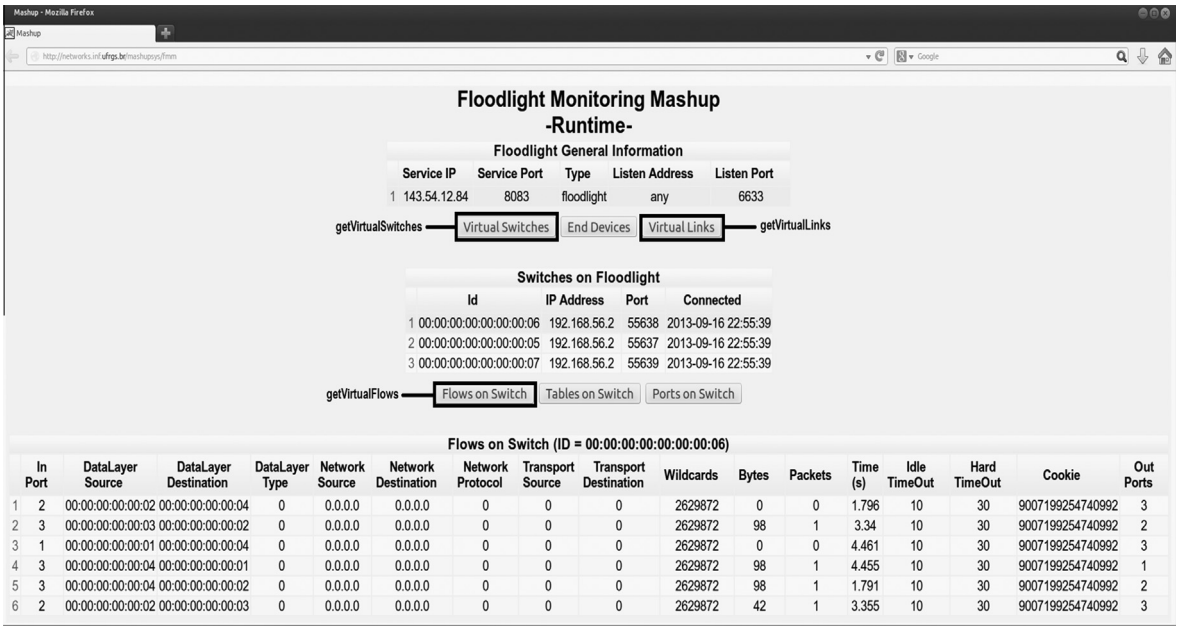

Fig. 10. FMM on runtime.

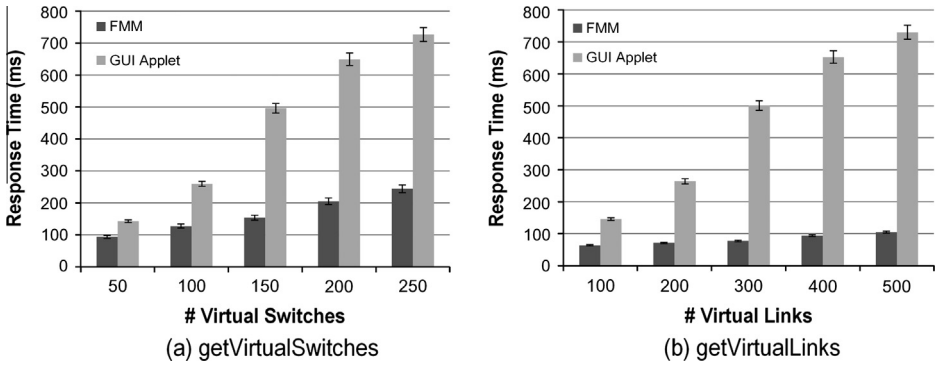

Fig. 11. FMM – response time.

(running), otherwise, we used 8 guests in such a state and the others turned off. The results (Fig. 8(a)) depict that for SVEs based on XenServer and VirtualBox, `nodeStructure` had a good response time average ($r \leq 1$). This operation had better response time average for *xenpool* than *vbox*/Debian when the number of guests was increased from 32 to 64. The response time average of `nodeStructure` for *xenpool* increases 3.1 *milliseconds* per guest, and, for *vbox*/

Debian, it grows 5.9 *milliseconds*. Then, if the number of guests is equal or greater than 48, the response time average for *xenpool* is better than for *vbox*/Debian. This behavior of response time average occurs because, as there is a large number of guests in the Managed Resources Layer, HTTP connections of the XenSDK are more efficient than Simple Object Access Protocol (SOAP)/HTTP connections of the VirtualBox Web Service. Regarding visual elements, it is meaningful to state that their response time is too small to impact the performance of VNMM as a whole.

In VNMM, finally, we evaluated the network traffic generated by the operation `nodeStructure`, varying at each SVE, the number of guests from 2^0 to 2^6. If the number of guests was $\leqslant 8$, we used all guests in active state, else, we used 8 guests running and the rest turned off. Fig. 8(b) depicts that this operation generated more traffic for *vbox*/Debian than *xenpool* when the number of guests was increased from 32 to 64. The traffic generated by `nodeStructure` for *xenpool* increases 93.2 Bytes per guest, and, for *vbox*/Debian, it grows 97.2 Bytes. Then, if the number of guests is equal or greater than 52, the traffic generated for *xenpool* is less than for *vbox*/Debian. This traffic behavior occurs, because for a large number of guests, the XML codification, used by Web Services based on SOAP in the Managed Resources Layer, is more verbose than JSON codification used on RESTful Services. Regarding visual elements, it is relevant to point out that their size is too small to impact the quantity of traffic generated by VNMM.

6.2. Floodlight Monitoring Mashup

In the case study, we also used the *designer* to build up FMM (Fig. 9), by dragging-and-dropping and wiring *floodlight* and *ofmonitor*. FMM allows the integrated monitoring of switches, links, and flows that are part of a Floodlight-based NVE by providing the following operations (Fig. 10): (i) `getVirtualSwitches` retrieves a list with information (*e.g., mac address -id-, ip address, and port*) about Open vSwitches handled by Floodlight, (ii) `getVirtualLinks` retrieves a list that includes information (*e.g., source id, source port, destination id, and destination port*) of virtual links established among virtual switches; and (iii) `getVirtualFlows` retrieves a list that contains information (*e.g., network source, network destination, network protocol, and outports*) associated to flows responsible for controlling the behavior of a specific Open vSwitch. FMM uses HTML tables to present the information retrieved by the above mentioned operations.

In FMM and the Floodlight GUI Applet (*i.e.*, a Web application for Floodlight monitoring), we assessed the response time average of operations `getVirtualSwitches` and `getVirtualLinks`. In this assessment, we varied the number of Open vSwitches from 50 to 250 and the quantity of virtual links from 100 to 500. Fig. 11 depicts the assessment results, disclosing that FMM had a behavior on the response time average: optimal for `getVirtualLinks` and good for `getVirtualSwitches`. This behavior is always better than the behavior reached by the GUI Applet. As a consequence, we can state that it is feasible to use FMM for monitoring NVEs based on Floodlight.

In FMM and the GUI Applet, we also evaluated the network traffic generated by operations `getVirtualSwitches` and `getVirtualLinks`, modifying the number of Open vSwitches from 50 to 250 and the quantity of virtual links from 100 to 500. Fig. 12 depicts the evaluation results of these operations, in which, FMM and the GUI Applet use about 8.7 KBytes and 13.75 KBytes per 50 virtual switches, respectively. Also, FMM and the GUI Applet use about 19.06 KBytes and 27.49 KBytes per 100 virtual links. Thus, such results reveal that FMM had better behavior on network traffic than the GUI Applet. As a consequence, we strengthen the assertion that it is feasible to use FMM for monitoring NVEs based on Floodlight. In the same direction, the evaluation results of VNMM and FMM demonstrates the practicality of applying mashups for monitoring Virtual Nodes.

6.3. Integrated Monitoring Mashup and qualitative analysis

In the case study, we also used the *designer* to build up IMM (Fig. 13). In this sense, first, we dragged-and-dropped three visual components: *vnmm*, *fmm*, and *integrator*. It is important to mention that existing monitoring mashups (VNMM and FMM) are represented as visual components in order to facilitate their reuse. Second, we wired two links *vnmm – integrator* and *fmm – integrator*. On runtime, IMM offers to Virtual Infrastructure Administrators the operations of VNMM and FMM: `nodeList`, `nodeStructure`, `guestFeatures`, `guestStats`, `controlGuests`,

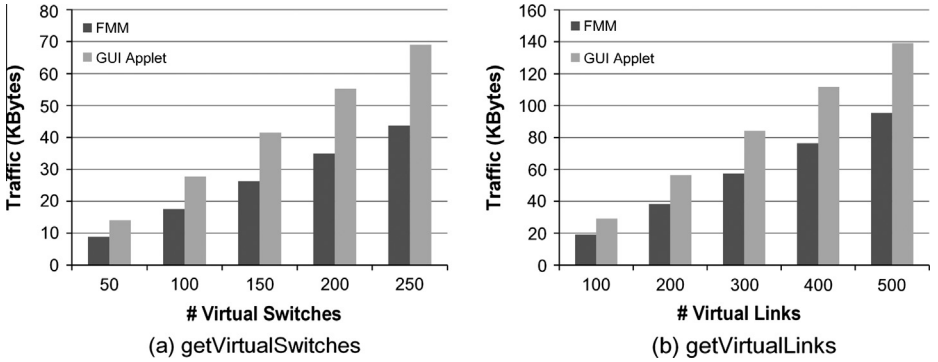

Fig. 12. FMM – network traffic.

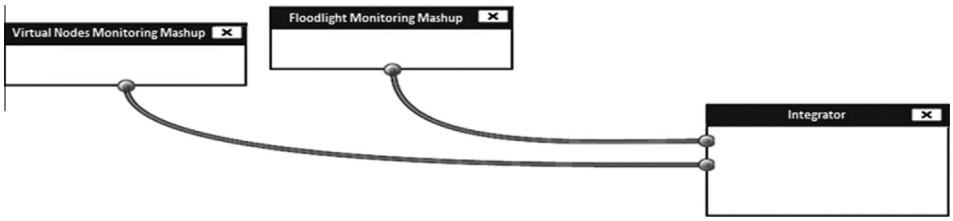

Fig. 13. IMM on design time.

`getVirtualSwitches`, `getVirtualLinks`, and `getVirtualFlows`. Therefore, using IMM, such Administrators are able to monitor heterogeneous Virtual Nodes formed by both, the SVEs based on Xen/VirtualBox and NVEs supported on Floodlight.

As we already analyzed quantitatively the operations provided by VNMM and FMM, which compose IMM. Next, we analyze IMM in a qualitative way in order to define the main characteristics (*i.e.*, flexibility and extensibility) provided by monitoring mashups. These characteristics are described in the following paragraphs, in which, Network Administrators, System Administrators, and Virtual Infrastructure Administrators are simply referred like Administrators.

Flexibility. Monitoring mashups allow Administrators to customize and improve their workspace by themselves. Administrators do not require a lot of Web programming skills to create monitoring capabilities, like IMM, targeted to SVEs and/or NVEs because, *designer* provides a high-level abstraction of system/network virtualization technologies and monitoring tasks; these technologies and tasks are represented in a visual way as mashup-able components. Furthermore, unlike traditional composition technologies, such as the Business Process Execution Language and the Web Service Conversation Language, that are developer-centric, the mashup technology provides a flexible and easy-of-use way for user-centric service composition [12,16].

Extensibility. Leveraging the composition, abstraction, and reusing models inherited from the mashup technology, Administrators can create, using existing monitoring mashups (like VNMM and FMM), novel, advanced, and complex virtual monitoring composite services (like IMM) devoted to SVEs and/or NVEs. As a consequence, Administrators who develop monitoring mashups are able to extend the Mashup System (specifically, *designer*) and, therefore, enhance/improve their workspace. In a general way, building up a mashup from existing applications is easier than developing an application from scratch [36,37].

On the other hand, doing a qualitative comparison with the Reference Implementation, Lattice is less extensible and flexible because it does not permit its customization by Administrators themselves. The major difference from the IMO system is that our proposal works in a high-level abstraction, regardless of system and network virtualization technologies. Libvirt is a low-level API that facilitates the building up of monitoring systems directed to SVEs but not to NVEs. Thus, in the Reference implementation, Libvirt is constrained to be used like VMI of SVEs. For monitoring NVEs, it is necessary the use of specific APIs like the Floodlight REST API employed in the case study. Regarding traditional OpenSource monitoring solutions, our proposal must be considered as an alternative, based on mashups and non-on plugins, devoted to add system/network virtualization support and a complement to reach integration and not like a surrogate.

7. Conclusions and future work

In this paper, we proposed the monitoring of Virtual Nodes by mashups. Using concepts like composite applications, Virtual Node Wrappers, monitoring mashups, and Mashup Development Environment, we demonstrated that it is feasible to build a flexible and extensible system intended to monitor SVEs and/or NVEs. Virtual Node Wrappers provide the abstraction of SVEs, VMIs, and NVEs. This abstraction offers the flexibility to add new network and system virtualization technologies as they arise. Furthermore, the Mashup Development Environment allows Network Administrators, System Administrators, and Virtual Infrastructure Administrators without advanced programming skills, to create, in a high-level abstraction and through a user-centric service composition model, their monitoring mashups. Consequently, these Administrators can themselves customize, extend, enhance, and integrate their monitoring solutions. This is essential to solutions of monitoring devoted to Virtual Nodes because of continuous changes occurring in virtual environments.

We also presented realistic monitoring scenarios, in which, multiple Interaction Elements were combined to build up monitoring mashups, namely VNMM, FMM, and IMM. These mashups were aimed to meet three particular challenges: the overall monitoring of non-homogeneous SVEs based on Xen and VirtualBox, the whole monitoring of a NVE supported on the Floodlight OpenFlow Controller, and the integrated monitoring of such SVEs and NVE. The Reference Implementation and the mashups built were able to overcome the raised challenges, demonstrating the relevance of our proposal. Considering the results of quantitative and qualitative evaluations performed in these scenarios, we can state, first, monitoring mashups have short response time, good on NVEs and good or admissible on SVEs, and generate low traffic. Second, monitoring mashups are flexible and extensible.

From an implementation point of view: (i) to reduce the encode and decode time of HTTP messages, we developed

Virtual Node Wrapers as RESTFul Web Services handling and generating JSON objects, (ii) we also used simple and light JSON objects to decrease the transferred information between the Adaptation, Composition, and Presentation Layers, (iii) in order to enrich the mashups interaction, we developed the Mashup Development Environment by using programming tools supported by AJAX; and (iv) we built VNMM, FMM, and IMM, by means of visual elements, drag-and-drop, and wiring capabilities offered by the Mashup Development Environment, illustrating the easiness of creating the monitoring mashups.

In next research steps, we plan to extend and enhance the Reference Implementation to support other integrated management tasks (*e.g.*, faults, configuration, and performance) on traditional and/or virtual networks. Finally, we also are interested in add a recommender system in order to expedite the development of monitoring mashups.

Acknowledgements

The research of PhD (c) Caicedo is supported by the PECPG (Agreement Program Students Graduate) of the CAPES (Brazil) and the University of Cauca (Colombia).

References

[1] P. Jianli, S. Paul, R. Jain, A survey of the research on future internet architectures, Commun. Mag. 49 (7) (2011) 26–36.
[2] Distributed Management Task Force, CIM System Virtualization Model White Paper, November 2007.
[3] N.M.M.K. Chowdhury, R. Boutaba, Network virtualization: state of the art and research challenges, Commun. Mag. 47 (7) (2009) 20–26.
[4] F.F. Daitx, R.P. Esteves, L.Z. Granville, On the use of SNMP as a management interface for virtual networks, in: IM, 2011, pp. 177 – 184.
[5] C. dos Santos, R. Bezerra, J. Ceron, L. Granville, L. Rockenbach Tarouco, On using mashups for composing network management applications, Commun. Mag. 48 (12) (2010) 112–122.
[6] 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.
[7] 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, 2010, pp. 487–494.
[8] C. Cappiello, F. Daniel, M. Matera, C. Pautasso, Information quality in mashups, Internet Comput. 14 (4) (2010) 14–22.
[9] J. Yu, B. Benatallah, F. Casati, F. Daniel, Understanding mashup development, Internet Comput. 12 (5) (2008) 44–52.
[10] H. Gebhardt, M. Gaedke, F. Daniel, S. Soi, F. Casati, C. Iglesias, S. Wilson, From mashups to telco mashups: a survey, Internet Comput. 16 (3) (2012) 70–76.
[11] C. dos Santos, L. Zambenedetti Granville, L. Shwartz, N. Anerousis, D. Loewenstern, Quality improvement and quantitative modeling – using mashups for human error prevention, in: IM, 2013, pp. 143–150.
[12] C. Cappiello, M. Matera, M. Picozzi, G. Sprega, D. Barbagallo, C. Francalanci, DashMash: a mashup environment for end user development, in: S. Auer, O. DÃaz, G. Papadopoulos (Eds.), Web Engineering, Lecture Notes in Computer Science, vol. 6757, Springer, Berlin, Heidelberg, 2011, pp. 152–166.
[13] K. Huang, Y. Fan, W. Tan, An empirical study of programmable web: a network analysis on a service-mashup system, in: ICWS, 2012, pp. 552–559.
[14] A. Majchrzak, P.H.B. More, Emergency! web 2.0 to the rescue!, Commu ACM 54 (4) (2011) 125–132.
[15] S. Mohammadi, A. Khalili, S. Ashoori, Using an enterprise mashup infrastructure for just-in-time management of situational projects, in: ICEBE, 2009, pp. 3–10.
[16] X. Liu, Y. Hui, W. Sun, H. Liang, Towards service composition based on mashup, in: IEEE Congress on Services, 2007, pp. 332–339.
[17] R. Boutaba, W. Ng, A. Leon-Garcia, Web-based customer management of VPNs, J. Netw. Syst. Manage. 9 (1) (2001) 67–87.
[18] S. Clayman, A. Galis, L. Mamatas, Monitoring virtual networks with lattice, in: NOMS, 2010, pp. 239–246.
[19] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehorster, A. Brinkmann, Non-intrusive virtualization management using libvirt, in: DATE, 2010, pp. 574–579.
[20] S. Clayman, R. Clegg, L. Mamatas, G. Pavlou, A. Galis, Monitoring, aggregation and filtering for efficient management of virtual networks, in: CNSM, 2011, pp. 1–7.
[21] D. Dudkowski, M. Brunner, G. Nunzi, C. Mingardi, C. Foley, M. de Leon, C. Meirosu, S. Engberg, Architectural principles and elements of in-network management, in: IM2009, 2009, pp. 529–536.
[22] D. Dudkowski, B. Tauhid, G. Nunzi, M. Brunner, A prototype for in-network management in NaaS-enabled networks, in: IM, 2011, pp. 81–88.
[23] W. Barth, Nagios: System and Network Monitoring, Second ed., No Starch Press, San Francisco, CA, USA, 2008.
[24] H.B. Newman, I. Legrand, P. Galvez, R. Voicu, C. Cirstoiu, MonALISA: A Distributed Monitoring Service Architecture, CoRR cs.DC/0306096.
[25] M. Massie, The ganglia distributed monitoring system: design, implementation, and experience, Parallel Comput. 30 (7) (2004) 817–840.
[26] S. Keshav, An Engineering Approach to Computer Networking, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.
[27] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati, Data integration in data warehousing, IJCIS: Int. J. Coop. Inform. Syst. 10 (3) (2001) 237–271.
[28] R.T. Fielding, R.N. Taylor, Principled design of the modern web architecture, ACM Trans. Internet Technol. 2 (2) (2002) 115–150.
[29] K. Matsuyama, M. Kraus, K. Kitagawa, N. Saito, A path-based RDF query language for CC/PP and UAProf, in: PERCOMW, 2004, pp. 3–7.
[30] P. Salomoni, S. Mirri, S. Ferretti, M. Roccetti, Profiling learners with special needs for custom e-learning experiences, a closed case?, in: W4A, ACM, 2007, pp 84–92.
[31] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, SIGOPS Operat. Syst. Rev. 37 (5) (2003) 164–177.
[32] J. Watson, VirtualBox: bits and bytes masquerading as machines, Linux J. 2008 (166).
[33] A. Lara, A. Kolasani, B. Ramamurthy, Network innovation using OpenFlow: a survey, Commun. Surv. Tutor. PP (99) (2013) 1–20. IEEE.
[34] B. Lantz, B. Heller, N. McKeown, A network in a laptop: rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, ACM, New York, NY, USA, 2010, pp. 19:1–19:6.
[35] S. Joines, R. Willenborg, K. Hygh, Performance Analysis for Java Websites, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
[36] J. Tatemura, A. Sawires, O. Po, S. Chen, K.S. Candan, D. Agrawal, M. Goveas, Mashup feeds: continuous queries over web services, in: SIGMOD, ACM, New York, NY, USA, 2007, pp. 1128–1130.
[37] R. Hasan, M. Winslett, R. Conlan, B. Slesinsky, N. Ramani, Please permit me: stateless delegated authorization in mashups, in: ACSAC, IEEE Computer Society, Washington, DC, USA, 2008, pp. 173–182.

Oscar Mauricio Caicedo Rendon is a Ph.D. candidate in computer science at the Institute of Informatics (INF) of the Federal University of Rio Grande do Sul (UFRGS), Brazil. In 2006 he received a M.Sc. degree in telematics of the University of Cauca, Colombia, from which he also held a degree in electronics and telecommunications engineering (2001). His topics of interest include network management, Web services-based management, and Web 2.0/3.0 technologies.

Carlos Raniery Paula Dos Santos 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 2008 and 2013, respectively. In 2005 he received a degree in telematics from the Federal Center of Technological Education of Ceará (CEFET-CE). His topics of interest include network management, Web services-based management, P2P-based systems, and Web 2.0/3.0 technologies.

Arthur Selle Jacobs is a B.Sc. student in computer science at the Institute of Informatics (INF) of the Federal University of Rio Grande do Sul (UFRGS), Brazil. His topics of interest include network management and Web 2.0/3.0 technologies.

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 network, network virtualization, information visualization, and network programmability.