UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

OSCAR MAURICIO CAICEDO RENDON

An Effective Approach for Network Management based on Situation Management and Mashups

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Advisor: Prof. Dr. Lisandro Zambenedetti
Granville

Porto Alegre
January 2015

CIP – CATALOGING-IN-PUBLICATION

Caicedo Rendon, Oscar Mauricio

An Effective Approach for Network Management based on Situation Management and Mashups / Oscar Mauricio Caicedo Rendon. – Porto Alegre: PPGC da UFRGS, 2015.

?? f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2015. Advisor: Lisandro Zambenedetti Granville.

1. Mashment. 2. Mashup. 3. Network Management Situation. 4. Situation Management. 5. Web-based Network Management. I. Granville, Lisandro Zambenedetti. II. Título.

“He aprendido que el mundo quiere vivir en la cima de la montaña,
sin saber que la verdadera felicidad está en subir la escarpada.”

— Gabo

ACKNOWLEDGEMENT

Acknowledgments are written in Spanish because of my family. En primer lugar quiero agradecer a Nina y Edgar por sus enseñanzas y apoyo incondicional en cada etapa de mi vida; su dedicación y sacrificio han sido la base para mi progreso. Sin lugar a duda estas palabras se quedan cortas para expresarles mi amor y gratitud. A Marce y Samuca por ser mi inspiración; ustedes son mi motivo para mejorar dia a dia. A Lore por ser el amor que facilita todo; sin ti lograrlo hubiese sido más difícil, gracias totales.

También quiero agradecer a todas las excelentes personas del grupo de redes con quienes compartí mis triunfos y derrotas en este duro camino doctoral. A Arthur y Felipe por ayudar en el desarrollo de mi investigación. A Ricardo, Wanderson, Cadori, y Javier por la amistad, el café y las innumerables conversaciones compartidas. A Esteves, Jedi, Weverton, Dalmazo, Roben, Lucas, Vinicius, Marcelo, Raniery, Jefferson, Juliano, Guilherme, y Glederson por facilitar mi adaptación al grupo y POA. A Ganso y Rodolfo por sus singulares ocurrencias. A los purinhos I, II, III y IV por proporcionar todo el café necesario para investigar.

Finalmente, a los profesores del grupo mi agradecimiento por la oportunidad brindada de pertenecer a tan selecto colectivo. Especialmente agradezco a mi tutor, Prof. Lisandro Zambenedetti Granville, por la confianza depositada, la transmisión de conocimiento y experiencia, los invaluables consejos y la exigencia de progreso continuo; indudablemente por su influencia hoy soy mejor investigador.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

AJAX	Asynchronous JavaScript and XML
AMR	Analytics Management Resource
AMRP	Analytics Management Resource Provider
AMRS	Analytics Management Resource as a Service
API	Application Program Interface
BGP	Border Gateway Protocol
BPM	Bussiness Process Management
CEP	Complex Event Processing
CLI	Command Line Interface
CSS	Cascading Style Sheet
CHS	Contextual Help System
DBMS	DataBase Management System
DRL	Drools Rule Language
DSO	Distribution System Operator
ERP	Enterprise Resource Planning
ForCES	Forwarding and Control Element Separation
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ISP	Internet Service Provider
ITIL	Information Technology Infrastructure Library
ITSM	IT Service Management

JSON	JavaScript Object Notation
KLM	Keystroke-Level Mode
MRTG	Multi Router Traffic Grapher
NMR	Network Management Resource
NMRP	Network Management Resource Provider
NMRS	Network Management Resource as a Service
NOS	Network Operating System
NSR	Network Situational Resource
NSRP	Network Situational Resource Provider
NSRS	Network Situational Resource as a Service
NVE	Network Virtualization Environment
OLAP	On-line Analytical Processing
OpR	Operator Resource
OpRP	Operator Resource Provider
OpRS	Operator Resource as a Service
PMM	Performance Monitoring Mashment
REST	REpresentational State Transfer
RSS	Rich Site Summary
SDN	Software-Defined Networking
SEP	Software Entity Provider
SLA	Service Level Agreement
SM	Situation Management
SME	Small and Medium Enterprise
SMR	Situation Management Resource

SMRP	Situation Management Resource Provider
SMRS	Situation Management Resource as a Service
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identificator
VNMM	Virtual Node Monitoring Mashment
WMR	Web-based Management Resource
WMRP	Web-based Management Resource Provider
WMRS	Web-based Management Resource as a Service
XML	eXtensible Markup Language

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The Situation Management discipline is intended to address situations happening or that might happen in dynamic systems. In this way, this discipline supports the provisioning of solutions that enable analyzing, correlating, and coordinating interactions among people, information, technologies, and actions targeted to overcome situations. Over recent years, the Situation Management has been employed in diverse domains ranging from disaster response to public health. Notwithstanding, up to now, it has not been used to deal with unexpected, dynamic, and heterogeneous situations that network administrators face in their daily work; in this thesis, these situations are referred to as network management situations.

The mashup technology also allows creating solutions, named mashups, aimed to cope with situations. Mashups are composite Web applications built up by end-users through the combination of Web resources available along the Internet. These composite Web applications have been useful to manage situations in several domains ranging from telecommunication services to water floods. In particular, in the network management domain, the mashup technology has been used to accomplish specific tasks, such as botnet detection and the visualization of traffic of the border gateway protocol.

In the network management domain, large research efforts have been made to automate and facilitate the management tasks. However, so far, none of these efforts has carried out network management by means of the Situation Management and the mashup technology. Thus, the goal of this thesis is to investigate the feasibility on using the Situation Management and mashups as an effective (in terms of complexity, consuming of time, traffic, and time of response) approach for network management.

To achieve the raised goal, this thesis introduces an approach formed by mashments (special mashups devised for coping with network management situations), the Mashment Ecosystem, the process to develop and launch mashments, the Mashment System Architecture, and the Mashment Maker. An extensive analysis of the approach was conducted on networks based on the Software-Defined Networking paradigm and virtual nodes. The results of analysis have provided directions and evidences that corroborate the feasibility of using the Situation Management and mashups as an effective approach for network management.

Keywords: Mashment. Mashup. Network Management Situation. Situation Management. Web-based Network Management.

1 INTRODUCTION

The Situation Management (SM) discipline is intended to address situations happening or that might happen in dynamic systems (JAKOBSON et al., 2005). In particular, this discipline aims to provide solutions that enable analyzing, correlating, and coordinating interactions among people, information, technologies, and actions targeted to overcome situations (KOKAR; MATHEUS; BACLAWSKI, 2009) (JAKOBSON, 2014). SM is based on the following foundations (JAKOBSON; BUFORD; LEWIS, 2007) (JAKOBSON, 2013): (*i*) a situation that is 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 satellite networking (GOPAL, 2007), disaster response (GEORGE et al., 2010), smart power grids (MAGOUTAS; MENTZAS; APOSTOLOU, 2011), aviation (KOELLE; TARTER, 2012), civil crisis (HEIN et al., 2012), public health (PEREIRA; COSTA; ALMEIDA, 2013), electric power systems (KROHNS-VALIMAKI; STRANDEN; SARSAMA, 2013), and emergency medical assistance (BRUNS et al., 2014). Notwithstanding, up to now, SM has not been used to deal with unexpected, dynamic, and heterogeneous situations that network administrators face in their daily work; in this thesis, these situations are referred to as network management situations. Some examples of network management situation are: (i) a sudden failure in the packet transmission of heterogeneous network devices; and (ii) an unforeseen slowness in a link formed by two virtual routers. It is important to highlight that this thesis does not affirm that network management situations have never been investigated, but such investigation has not been made from SM perspective.

The mashup technology also allows creating solutions, named mashups, aimed to cope with situations. Mashups are Web applications built up by end-users through the combination of Web resources available along the Internet (SIMMEN et al., 2008) (LAGA et al., 2012). Mashups have been useful to manage situations in several domains, such as project management (OZKAN; ABIDIN, 2009), water floods (TOSTI; SMARI, 2010), fire emergencies (MAJCHRZAK; MORE, 2011), telco services (GEBHARDT et al., 2012), data integration (HAN et al., 2013), immersive mirror worlds (STIRBU et al., 2013), and music (DAVIES et al., 2014). Furthermore, this technology has been analyzed as a feasible mechanism to accomplish specific tasks for network management (BEZERRA et al., 2010) (SANTOS et al., 2010b). Nevertheless, until now, the mashup technology has not been investigated to address network management situations.

In the network management domain, large research efforts (CHEN et al., 2010) (KIM; KIM, 2011) (MATTOS et al., 2011) (SANTANNA; WICKBOLDT; GRANVILLE, 2012) (MON-

SANTO et al., 2013) (KIM; FEAMSTER, 2013) (SMITH et al., 2014) have been made to automate and facilitate the management tasks. However, so far, none of these efforts has carried out network management by means of SM and mashups. Therefore, the goal of this thesis is to investigate the feasibility of using SM and mashups as an effective approach (*i.e.*, in terms of complexity, consuming of time, traffic, and time of response) for network management. To achieve this goal, this thesis raises the following hypothesis.

Hypothesis: **The employment of SM and mashups provides an effective approach for network management.**

The below **fundamental questions**, associated with the afore raised hypothesis, guide the investigation conducted in this thesis.

- What is the performance, in terms of the complexity and consuming of time, of solutions that use SM and mashups for network management?
- What is the performance, in terms of traffic and time of response, of solutions based on SM and mashups for network management?
- Which mechanisms could be employed to improve the performance of solutions that use SM and mashups for network management?

1.1 Contributions

The investigation about the feasibility of using SM and mashups as an effective approach for network management led to the following major contributions.

- The network management situation concept and its data model that introduce a novel way to characterize unexpected, dynamic, and heterogeneous situations in the network management domain.
- The mashment concept and its data model that present the use of mashups for carrying out the investigative and control aspects of SM in the network management domain.
- A new mashup ecosystem that defines the resources, stakeholders, software entities, activities, and interactions involved in addressing network management situations.
- A groundbreaking process formed by high-level tasks that presents how to deal with network management situations by developing and launching mashments.
- An assessment model that allows measuring the complexity and time of addressing network management situations with and without a mashment-based approach.

- A mashment system architecture that supports the carrying out of the ecosystem and the process aforementioned.
- A rule-based mechanism for automatic recognition of network management situations.
- A template-based mechanism for dynamic composition of mashments.

1.2 Methodology and organization

Figure 1.1 depicts the phases of the scientific research process followed in this thesis: Problem Definition, Hypothesis Construction, Experimentation, Conclusion, and Publish Fundings. In Problem Definition, the research question has been identified and defined. In Hypothesis Construction, the hypothesis and associated fundamental questions have been formulated. Furthermore, in such phase, the conceptual and technological proposals have been defined and carried out. In Experimentation, the hypothesis and evaluation results have been tested and analyzed, respectively. In Conclusion, conclusions and future works have been outlined. Note that Hypothesis Construction has been refed after Experimentation and Conclusion. In Publish Findings, papers for renowned conferences and journals have been submitted and published. This document was also written during such last phase.

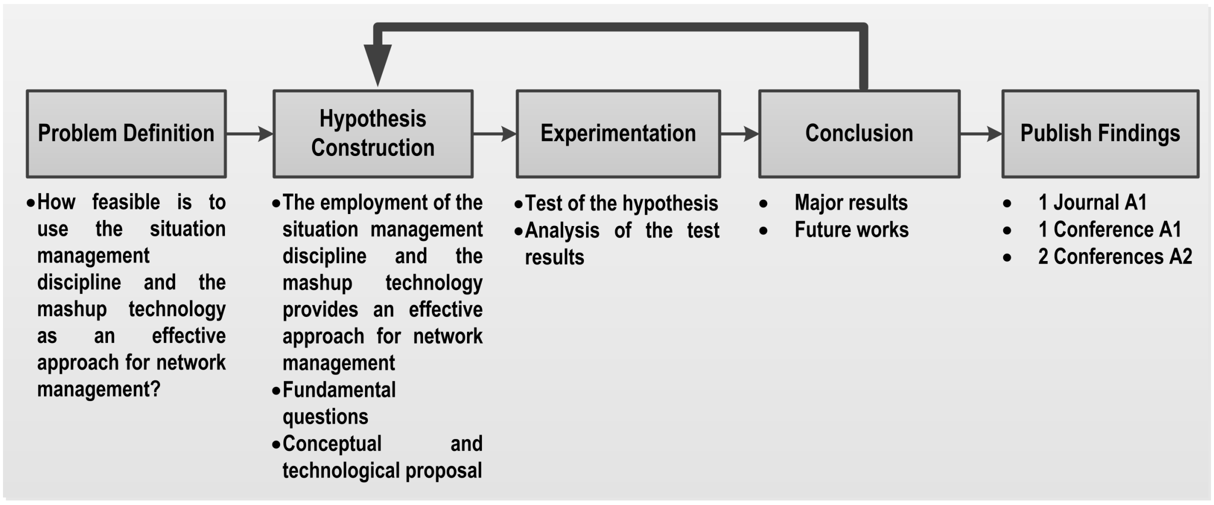

Problem Definition
Hypothesis Construction
Experimentation
Conclusion
Publish Findings
•How feasible is to use the situation management discipline and the mashup technology as an effective approach for network management?
•The employment of the situation management discipline and the mashup technology provides an effective approach for network management
•Fundamental questions
•Conceptual and technological proposal
•Test of the hypothesis
•Analysis of the test results
• Major results
• Future works
• 1 Journal A1
• 1 Conference A1
• 2 Conferences A2

Figure 1.1 – Thesis phases

The organization of this document reflects the phases outlined above.

- **This introductory chapter** presents the problem definition, raises the hypothesis, summarizes the contributions, and describes the overall structure of this thesis.
- **Chapter 2** reviews research about SM, mashups on situations and network management, and handling situations in the network management domain.
- **Chapter 3** introduces in detail the major contributions accomplished with this thesis.

- **Chapter 4** describes the experiments conducted to test the hypothesis, discusses the corresponding results, and presents implementation highlights.
- **Chapter 5** presents conclusions about the hypothesis and the fundamental questions as well as opportunities for future works.
- **Appendix** includes the list of papers in which the major results obtained during the development of this thesis have been published.

2 STATE OF THE ART

The goal of this chapter is to present the background of the main research topics touched in this thesis. In this way, this chapter starts presenting the SM discipline and its use on different application domains. After, this chapter reviews mashups and their use in diverse domains, including network management. This chapter finishes discussing some research works aimed to facilitate network management tasks.

2.1 Situation management

In a broad sense, SM is an emerging discipline, born in the US military, that provides a framework of concepts, models, and enabling technologies to design and develop situational solutions (KOELLE; TARTER, 2012). In a more specific definition, this discipline allows analyzing, correlating, and coordinating interactions among people, information, supporting tools, and actions targeted to overcome situations happening or that might happen in dynamic systems (JAKOBSON; BUFORD; LEWIS, 2007). In this sense, SM-based solutions deal with situations that are composite entities in a particular application domain whose components are other entities, their attributes, and relationships in a time interval (KOKAR; MATHEUS; BACLAWSKI, 2009) (JAKOBSON, 2014).

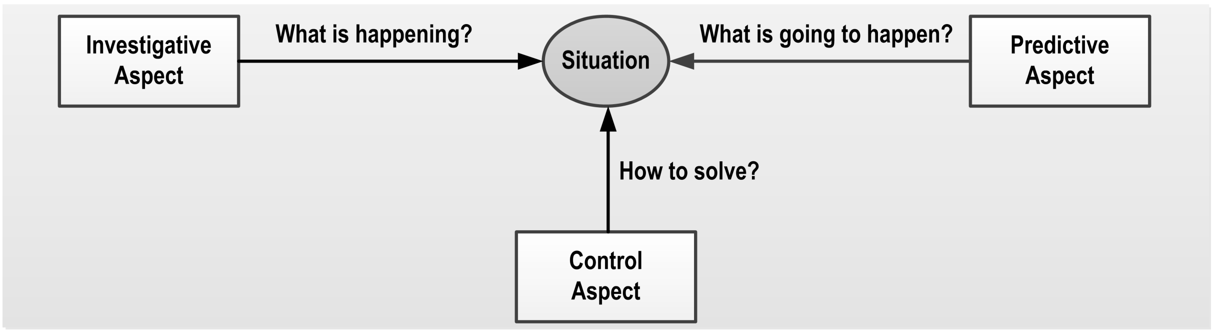


Figure 2.1 – Situation management aspects

SM includes three aspects (see Figure 2.1) always linked to the time axis (JAKOBSON; BUFORD; LEWIS, 2007) (JAKOBSON, 2013): investigative, control, and predictive. The investigative aspect aims to retrospective cause analysis of situations. Thus, this aspect is related to carrying out actions intended to comprehend situations (*i.e.*, to respond the question: what is happening?). For instance: (*i*) finding the root cause of transmission failure on voice packets in telecommunication networks; and (*ii*) monitoring of Service Level Agreements (SLAs) in network service providers.

The control aspect is directed to change or preserve situations (*i.e.*, to respond the question:

how to solve a situation?). In this way, this aspect is related to plan and implement actions aimed to overcome situations. For instance: (*i*) the planning of opening and closing hospitals during social emergencies; and (*ii*) rescheduling the job of personnel to improve its performance during the execution of technological projects.

The predictive aspect is related to project possible situations (*i.e.*, to respond the question: what is going to happen?). Therefore, this aspect is about the prediction of future situations taking into account past situations. For instance: (*i*) the launching of alerts notifying potential water floods, (*ii*) the throwing of warnings to announce possible cybernetic attacks; and (*iii*) the emission of alerts indicating when a server that hosts several virtual routers needs to be migrated.

In order to carry out the above aspects, usually, SM-based solutions conduct the following general management process (JONES et al., 2006): (*i*) they collect information of state and time about situations from many and heterogeneous sources, (*ii*) they correlate and fuse multi-source information to assist the recognition and comprehension of situations and, so, support the timely and correct making-decision when situations happen, (*iii*) they analyze past situations in order to predict future ones, (*iv*) they reason, plan, and implement actions to tackle situations within some predefined constraints; and (*v*) they present situational information aiming at the human comprehension maximization. In the literature, if a solution implements one or more SM aspects, it is considered as a situational solution.

In consonance with the aforedescribed aspects, the core theory of SM includes: Situation Modeling, Situation Recognition, and Situational Reasoning (JAKOBSON et al., 2005) (JAKOBSON, 2014). The Situation Modeling is to represent situations by considering the involved entities and the attributes, relations, and behavior of such entities. Examples of languages for modeling situations are the Web Ontology Language (GRAU et al., 2008) and the Unified Modeling Language (RUMBAUGH; JACOBSON; BOOCH, 2004). The Situation Recognition is to detect when situations happen in dynamic systems. The Situational Reasoning is related to plan, control, predict, and explain situations that happen or might happen in these systems. Examples of approaches that allow to implement both Situation Recognition and Situational Reasoning are the Case-Based Reasoning (SQUICCIARINI et al., 2014) and the Complex Event Processing (CEP) (RAYMUNDO et al., 2014).

Although research on SM has been mainly focused on military applications, over recent years, SM-based solutions have been used in diverse domains, such as satellite networking, disaster response, smart power grids, civil crisis, aviation, public health, electric power systems, and emergency medical assistance. For instance, in the domain of polyester film base manufacturing, a complex situational management application employing expert systems was proposed for monitoring the non-steady state events and assisting human operators with the event tasks (ADAMS; REYNOLDS, 2000).

In the satellite networking domain, a model-based framework for implementing situation management infrastructure was proposed to monitor and control satellite networks (GOPAL, 2007). This solution is based on the On-line Analytical Processing (OLAP), the DataBase Management Systems (DBMS), and the Simple Object Acces Protocol (SOAP). OLAP is used to generalize and summarize the large amount of data generated by satellite networks. DBMS is employed to store terminal configuration, payload configuration, and network administrator requests. SOAP is used to offer service interfaces targeted to facilitate the interaction among the proposed infrastructure and network management external systems.

In the disaster response domain, the architecture DistressNet was proposed to assist the disaster responders in correct making-decision (GEORGE et al., 2010). DistressNet is formed by wireless sensor networks and a set of delay-tolerant networking protocols. The wireless sensors allow to collect in site data. In turn, the delay-tolerant protocols enable to transfer critical messages from a network core to disconnected networks. The joint use of these sensors and protocols allows sensing, collecting, merging, and timely delivering high volumes of accurate information to the responders.

In the domain of smart power grids, a dynamic and flexible architecture was presented to meet proactively and intelligently the continuous changes in the electricity usage patterns of customers (MAGOUTAS; MENTZAS; APOSTOLOU, 2011). This architecture is based on the Service Oriented Architecture (SOA), the linked open data, and the publish-subscribe paradigm. The SOA concepts are used to facilitate the flexible integration of services. The linked open data is employed to connect heterogeneous information sources referenced by Uniform Resource Identifiers (URIs). The publish-subscribe paradigm is used to conduct push of information from linked data.

In the civil crisis management domain, a secure mobile agents platform was introduced to provide timely and protected access to situational information for emergency responders, such as the on-site personnel, the tactical crisis command, and the off-site strategic command centre (HEIN et al., 2012). This platform is based on: (*i*) a secure agent infrastructure to construct modular cooperative services in charge of collecting crisis information, (*ii*) a trusted docking station to enforce integrity guarantees; and (*iii*) a secure docking module to authenticate emergency responders and verify software.

In the aviation domain, an approach was proposed to support the design and development of distributed systems intended to support the timely and correct making-decision during air security incidents (*e.g.*, sudden flight maneuvers, trajectory deviation, and terrorist attacks) in SESAR and NextGen (KOELLE; TARTER, 2012). SESAR and NextGen are european programs intended to reduce air traffic delays and improve air safety. This approach is based on ontologies and software agents. Ontologies are used to model incidents as situations, situational elements, actors and the relations between all them. In turn, software agents are employed to collect, fuse, and share heterogeneous information about the modeled situations.

In the public health domain, a rule-based platform for situation management was introduced to monitor suspicious cases of tuberculosis (PEREIRA; COSTA; ALMEIDA, 2013). This platform, named SCENE, supports the development of situation-aware applications by means of situational design artifacts and a situation runtime. These artifacts are Situation Classes and Situation Rules written in the Drools Rule Language (DRL). Classes and Rules are used to define/specify the tuberculosis disease and its symptoms. The situation runtime manages the lifecycle of situation-aware applications by controlling artifacts in a CEP-based engine.

In the domain of electric power systems, a situation awareness solution was proposed to face major disturbances (*e.g.*, snow storms and fires) suffered by different distribution system operators (DSOs) in Finland (KROHNS-VALIMAKI; STRANDEN; SARSAMA, 2013). This solution is a Web composite application that presents information about disturbances in the electric power supply by combining data from several DSOs, public actors (*i.e.*, municipality and police), and map services. It is important to mention that such application was built by expert programmers and not devised to be extended or improved by end-users.

In the domain of emergency medical assistance, an approach based on the Finite State Machines and CEP was proposed for reducing the wait time of patients by improving the effectiveness in the ambulance fleet coordination (BRUNS et al., 2014). The Finite State Machines were used to model the operational states of ambulances and their corresponding state transitions. CEP was used to provide situation awareness by supporting the detection of changes in states of ambulances. Such detection was carried out using intelligent mechanisms of fusion and analysis of data streams emitted by the ambulance fleet.

Table 2.1 presents the aforecited works about the SM discipline, revealing that most of them focuses on the investigative and control aspects; it is because these aspects are foundations to conduct the predictive aspect. Furthermore, it is important to note that none of these works has employed the fundamentals of the SM discipline to cope situations in the network management domain. Consequently, there is a chance to innovate in the research gap located at the intersection of SM and network management. This gap is later leveraged in this thesis.

In order to close this brief introduction to the SM discipline, it is to noteworthy that in the literature, there are diverse management frameworks that despite using different terminology could be used for achieving the goals raised by SM. One of such frameworks is the Information Technology Infrastructure Library (ITIL) (OGC, 2011) that proposes a set of high-level guidelines for IT Service Management (ITSM). In particular, ITIL introduces several IT processes, such as the Event Management, Incident Management, Problem Management, and Change Management, that could be employed for managing generic situations.

Table 2.1 – Research on situation management

		Aspect		
Research Work	**Domain**	**Investigative**	**Control**	**Predictive**
(GOPAL, 2007)	Satellite networking	✓	✓	✓
(GEORGE et al., 2010)	Disaster response	✓	✓	✓
(MAGOUTAS; MENTZAS; APOSTOLOU, 2011)	Smart power grids	✓	✓	
(HEIN et al., 2012)	Civil crisis	✓	✓	
(KOELLE; TARTER, 2012)	Aviation	✓	✓	
(PEREIRA; COSTA; ALMEIDA, 2013)	Public health	✓	✓	✓
(KROHNS-VALIMAKI; STRANDEN; SARSAMA, 2013)	Electric power systems	✓	✓	
(BRUNS et al., 2014)	Emergency med assistance	✓	✓	

According ITIL definitions (OGC, 2007a), the Event Management is the service operation process responsible for dealing with events. An event is any detectable or discernible ocurrence that has significance for the management of IT infrastructures or the delivery of IT services. The Incident Management is the service operation process in charge of coping with incidents. An incident is any event causing unplanned interruption of an IT infrastructure/service or reduction in the quality of such infrastructure/service. Considering these definitions, when the Event Management is used to detect events that trigger the registration of major incidents (*i.e.*, incidents with shorter timescale and greater urgency) and hence the Incident Management, the Event Management could be matched to the Situation Recognition. In the same direction, if the Incident Management is used to address major incidents, this process might be assimilated to the part of the Situational Reasoning responsible for planning and controlling of situations.

The Problem Management is defined as the service operation process in charge of dealing with problems (OGC, 2007a). A problem is a cause of one or more incidents. In this way, when the Problem Management is used to diagnose the root cause of major problems and to determine the resolution of underlying incidents, this process might be equated to the part of the Situational Reasoning in charge of explaining/investigating situations.

The Change Management is the service transition process responsible for controlling the lifecycle of changes (OGC, 2007b). A change is the addition, modification, or removal of anything that could affect an IT service/infrastructure. As this process can be triggered by the

Incident Management when a plan built for addressing a major incident includes emergency changes (*i.e.*, changes that are time sensitives and hence must be implemented as soon as possible) to be carried out in any element of an IT infrastructure or an IT service, the Change Management could be also involved in the part of planning and controlling of the Situational Reasoning.

Although the above paragraphs reveal that some goals of several ITIL processes (or disciplines) could be assimilated to diverse aims of the SM discipline, there are also important differences among these processes (and ITIL as a whole) and SM. Unlike ITIL that defines concepts like event, incident, and problem, SM just considers the concept of situation. In this direction, it is to accentuate that the Event Management, Incident Management, and Problem Management are generic processes and thus they can be used for managing any type of event, incident, and problem, respectively. In contrast, SM is focused on managing situations that could be equated to major incidents and their associated problems.

The Event Management, Incident Management, and Problem Management do not model, in a formal way, events, incidents, or problems as the SM discipline does with situations. In particular, in SM, the Situation Modeling defines the model of situations (simple and compound) that includes their structure and dynamism as well as their context. Regarding dynamism, it is to note that a distinguishing characteristic of SM is the issue of time: (*i*) the aspects of investigation, control, and prediction are always linked to the time axis; and (*ii*) the evolution of situations (including their attributes, constraints, and relationships) over time.

The above mentioned ITIL processes consider several activities for carrying out the recognition, reasoning, and learning of events, incidents, and/or problems. However, such processes do not center in these activities as SM does. In this sense, it is significant to underline that the SM discipline includes in its core theory and general management process: the Situation Recognition and Situation Reasoning for recognizing, learning, and reasoning about situations.

2.2 Mashup technology

Mashups are composite Web applications centered in end-users and created by combining different resources available along the Web (MAXIMILIEN; RANABAHU; TAI, 2007) (CAPPIELLO et al., 2010). End-user centric means that mashups may be developed by users who usually do not have advanced programming skills (LAGA et al., 2012). Furthermore, mashups encourage the cooperation among end-users and the reuse of existing Web applications (SIMMEN et al., 2008) (HUANG; FAN; TAN, 2012).

The mashup technology is fundamentally characterized by resource abstraction and service composition models (YU et al., 2008). The resource abstraction model hides the technical details of underlying resources for end-users. This hiding is mainly carried out by means of Application Programming Libraries (APIs) and widgets. APIs (WEISS; GANGADHARAN, 2010) provide well-defined interfaces to bidirectionally interact with Web resources, such as Web feeds using the Rich Site Summary (RSS) format and Web services based on SOAP. Widgets (LAGA et al., 2012) operate in a more high-abstraction level than APIs and usually offer graphical mechanisms to interact, in a simple and efficient way, with APIs and specific content. Examples of widgets are visual boxes offering specific content about network speed and memory consumption on personal computers.

The service composition model allows end-users to develop mashups by blending resources represented as services. These resources are offered by multiple companies, such as providers of Web services and widgets. Furthermore, such resources are available at various levels (*e.g.*, data, presentation, and logic) of Web applications. For instance, the housingmaps application that combines geographic location from Google Maps with house prices from Crailglist is a classical mashup, which involves integration at levels of data and presentation.

The service composition model of mashups is assisted by systems that offer visual tools (*e.g.*, widgets and wire features) to combine, store, and execute services in a high-abstraction level. Moreover, these systems are responsible for supporting the reuse and improvement of existing compositions, promoting the generation of more sophisticated and innovative mashups. It is relevant to mention that, in the literature, such systems are indistinctly known as mashup systems, mashup tools, mashup makers, and mashup development environments.

Over the last decade, two important things have contributed to disseminate the use of mashups (MAXIMILIEN; RANABAHU; TAI, 2007): (*i*) the number of available services, widgets, and online APIs has significantly increased; and (*ii*) new usability-oriented technologies, such as the Asynchronous JavaScript and XML (AJAX) and Macromedia Flash, emerged to allow the creation of more dynamic applications and advanced Graphical User Interfaces (GUIs).

In the literature, if a mashup is developed for coping rapidly with an immediate need of one or a small set of end-users, it is considered as a situational solution (LATIH et al., 2011). Nevertheless, in this regard, it is important to highlight that such definition of mashups does not formally consider the SM discipline. In fact, such definition does not take into account any foundation of SM (*i.e.*, situation, aspects - investigative, control, and predictive -, Situation Modeling, Situation Recognition, and Situation Reasoning).

The mashup technology has been used as situational solution in many diverse domains, such as project management, water floods, fire emergencies, telco services, data integration, immersive mirror worlds, and music. For instance, in the project management domain, a mashup system was proposed to support management tasks on situational projects (OZKAN; ABIDIN, 2009) that involve a small number of users and have a short lifespan. Using such mashup system, project managers were enabled to quickly visualize, filter, and share information about their projects by developing mashups.

A software architecture was introduced to facilitate the carrying out of emergency management operations in water floods (TOSTI; SMARI, 2010). This architecture uses Web 2.0 technologies, SOA concepts, and wireless sensor networks to support the estimation of the water speed and timing of possible floods. A flood monitoring mashup was used to illustrate the functioning of such an architecture. This mashup collected, correlated, and presented data from multiple wireless sensors deployed in a simulated environment.

During a fire emergency happened in the city of San Diego (California, USA), a mashup was developed by volunteers to provide to population the information about active fires, evacuation routes, working hospitals, and available refuges (MAJCHRZAK; MORE, 2011). This mashup retrieved, integrated, and presented information from multiple services, such as maps, geographic information systems, and wikis, offered by civil organizations, private companies, and the government.

In the domain of telecommunication advanced services, a reference architecture was proposed to facilitate the provisioning of telco mashups for end-users (GEBHARDT et al., 2012). A telco mashup is a composite service that combines functionalities from telecomunication networks (*e.g.*, streaming, quality of service, and billing) and device capabilities (*e.g.*, geographic location and health information) with services (*e.g.*, instant messaging and IP telephony) available along the Web. Until now, in the literature, there is not evidence about a prototype of a mashup system or a telco mashup that implements/follows such reference architecture.

In the domain of data integration, a mashup development environment called Mashroom was introduced to solve the transient and ad-hoc data integration problem of bussiness users (HAN et al., 2013). Such development environment is based on the resource abstraction model of mashups and the spreadsheet programming paradigm (BURNETT et al., 2001). Using Mashroom, bussiness users (*i.e.*, an end-user working in a company) who have little programming expertise can develop situational data mashups. The spreadsheet-based development is performed on the fly and interactively by aggregating heterogeneous data sources represented as services.

In the immersive mirror worlds domain, a lightweight platform for mashups was presented to enable Web developers to create, in a mashup manner, realistic and immersive street-level representations of the physical world (STIRBU et al., 2013). The Acme Tours & Travels Inc built by using this plaftorm a mashup called Cloud City Scene. This mashup integrates geographical tagged data with street-level panorama images to provide live and three dimensional information about routes, stops, arrivals, and departures of buses in several cities around the world.

In the ITSM domain, a methodology, a model, and a set of mashup patterns (SANTOS et al., 2011b) (SANTOS et al., 2013) were introduced for measuring, preventing, and eliminating inefficiencies and errors caused by humans in the Request Fulfillment process. This process is part of service operation processes defined by ITIL and deals with service requests (*i.e.*, requests that do not represent a disruption to a service) from the users (OGC, 2007a). The referred inefficiencies (*i.e.*, suboptimal execution of activities due to their intrinsic complexity and manual execution) were quantitatively evaluated and analyzed from a time productivity perspective. In turn, human errors (*i.e.*, activity defects introduced by operators) were assessed and studied in a quantitative way from a probabilistic point of view.

In the music industry domain, the AutoMashUpper system was introduced to offer new possibilities for musical creation (DAVIES et al., 2014). Using AutoMashUpper, music enthusiasts and professional DJs are able to build up novel musical content (*i.e.*, music mashups) by mixing different songs at different regions of an input song. In this system, a music mashup can be built considering harmonic compatibility, rhythmic compatibility, and spectral balance of each component song.

Table 2.2 – Research on mashups

		Evaluated characteristic					
Research Work	**Domain**	**Extensibility**	**Flexibility**	**Complexity**	**Time of task**	**Time of response**	**Error of task**
(OZKAN; ABIDIN, 2009)	Project management	✓	✓	✓			
(TOSTI; SMARI, 2010)	Water floods	✓	✓				
(MAJCHRZAK; MORE, 2011)	Fire emergencies	✓	✓				
(SANTOS et al., 2011b)	ITSM	✓	✓		✓		
(GEBHARDT et al., 2012)	Telco services	✓	✓				
(HAN et al., 2013)	Data integration	✓	✓		✓		
(STIRBU et al., 2013)	Immer mirror worlds	✓	✓				
(SANTOS et al., 2013)	ITSM	✓	✓				✓
(DAVIES et al., 2014)	Music	✓	✓				
This thesis	Netw management	✓	✓	✓	✓	✓	

Table 2.2 presents the aforedescribed works about the mashup technology, revealing diverse facts, first, most of such works was more concerned about qualitative aspects than quantitative ones. In this way, these works mainly studied characteristics such as flexibility and extensibility. Second, unlike this thesis, such works did not quantitatively analyze the complexity that users perceive when conducting tasks assisted by mashups. Third, some research on the mashup technology assessed the time that users spend to carry out tasks by using mashups but such research did not evaluate the time of response of mashups themselves. It is to point out that this thesis is not interested in measuring errors that a network administrator might make in his/her daily work.

In the network management domain, a mashup-based approach was proposed to deal with the security problem of botnets, in an more flexible, extensible, and usable way (SANTOS et al., 2010a). This approach was carried out in a mashup. The mashup built integrates dinamically botnet information collected from existing mitigation tools, such as sandboxes and antiviruses, with geographic location retrieved from online mapping and geolocation APIs.

In the same direction of the work about botnets, a mashup system was introduced to evaluate qualitatively the feasibility of using Web 2.0 technologies on the network management domain (BEZERRA et al., 2010). In the prototype of proposed system, a mashup was built in order to monitor the traffic of the Border Gateway Protocol (BGP) among two autonomous systems by integrating traffic router information. This integrated information was collected by using the Simple Network Management Protocol (SNMP) and presented by combining images retrieved from a generator of traffic graphs and maps generated from a mapping service.

As a continuation of the last two works above cited, a generic architecture was proposed to support the composing of network management applications (SANTOS et al., 2010b). This architecture allows network administrators to design their own management applications through the composition of external resources. The qualitative evaluation of the proposed architecture was carried out by means of a mashup system prototype. In such prototype were developed the BGP peering mashup and the botnet mashup previously referred.

As an extension of the formerly mentioned generic architecture, the Maestro architecture (SANTOS et al., 2011a) was presented for providing data confidentiality in mashups that are developed to assist daily activities of network administrators. An instance of Maestro was implemented in a proof-of-concept prototype in which was built the network traffic monitoring mashup. A quantitative evaluation was conducted to measure the overhead of time caused in such mashup by the modules added in Maestro for achieving data confidentiality. It is to highlight that such evaluation was not focused on measure the overall time of response of the built mashup.

Table 2.3 reveals several facts, first, none of the aforecited research works focused on network management by jointly using SM aspects and mashup concepts. Actually, research on mashups for network management did not consider the investigative, control, and/or predictive aspects of SM. Second, similarly to research that employs mashups in other domains, research that uses mashups to manage networks neither evaluates the complexity nor the time involved in the conducting of management tasks on situational scenarios. Three, the traffic generated by mashups conducting network management tasks was also not assessed. Fourth, the time of response of mashups for network management was not deeply evaluated; just a concise evaluation of time-overhead was performed. It is to noteworthy that the above facts externalize a research gap located at the intersection of the SM discipline, the mashup technology, and the network management. This thesis leverages such gap to propose an innovator approach aimed to carry out network management by SM and mashups.

Table 2.3 – Mashups for network management

		Evaluated characteristic					
Research Work	**SM**	**Extensibility**	**Flexibility**	**Traffic**	**Complexity**	**Time of task**	**Time of response**
(SANTOS et al., 2010a)		✓	✓				
(BEZERRA et al., 2010)		✓	✓				
(SANTOS et al., 2010b)		✓	✓				
(SANTOS et al., 2011a)		✓	✓				✓
This thesis	✓	✓	✓	✓	✓	✓	✓

2.3 Network management

So far, in the literature of network management, there is not an approach that uses the SM discipline and the mashup technology with the goal of facilitating the management tasks. However, there is a lot of research works that address such goal from other points of view. The next paragraphs review the most relevant of such works.

The COOLAID (CHEN et al., 2010) is a data-centric framework focused on automatic and declarative configuration of complex and dynamic networks. This framework automates operations related to setting networks and their devices, aiming to reduce the involvement of network administrators. COOLAID is formed by a data abstract model, a distributed database, and a database query language. The data abstract model represents a network of interconnected devices. The distributed but centrally managed database stores such a model. With COOLAID, network administrators can conduct configuration operations by using specific database queries (*e.g.*, select, insert, and update), which are called primitives.

Lattice (CLAYMAN; GALIS; MAMATAS, 2010) is a framework for monitoring virtual network resources, such as virtual machines, virtual routers, and virtual links. 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 control. An important shortcoming of this framework is that it is centered on network programmers. Thus, it was not conceived to allow its adaptation or customization by network administrators.

Libvirt and Libvirtd (BOLTE et al., 2010) are solutions 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 administrators.

The Information Management Overlay (IMO) (CLAYMAN et al., 2011) 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.

NetOpen (KIM; KIM, 2011) is a solution that employs composite applications to monitor and configure OpenFlow-based networks. Such composite applications are based on SOA and referred as to networking services. In this solution, expert network programmers are responsible for developing networking services by combining primitives (*i.e.*, networking basic services). NetOpen considered, among others, the following primitives: (*i*) to retrieve information of switches, link states, and flow tables; and (*ii*) to configure network devices. An example of NetOpen networking service is the L2-Switching Service. This service provides network connectivity between end-hosts connected to OpenFlow networks.

The OpenFlow MaNagement Infrastructure (OMNI) (MATTOS et al., 2011) is a solution intended to control and monitor OpenFlow-based networks. OMNI is formed by a set of NOX controller applications, a multi-agent system, and a friendly Web user interface. The NOX-based applications, released as APIs, provide functionalities to: (*i*) reconfigure, after migration, a flow going across a set of switches to another group of switches; and (*ii*) to retrieve statistics about OpenFlow switches. The multi-agent system offers features to autonomously conduct management on OpenFlow networks. The Web interface allows network administrators to access easily functionalities of NOX-based applications. It is to highlight that only advanced network programmers can extend or improve the OMNI applications.

The Management Environment of Inter-domain Circuits for Advanced Networks (MEICAN) (SANTANNA; WICKBOLDT; GRANVILLE, 2012) is a solution devised to support the provisioning of virtual inter-domain circuits. MEICAN is formed by a Bussiness Process Management (BPM) system, a network middleware, and a front-end system. The BPM system allows network administrators to take part in the decision-making process to establish and manage the above mentioned circuits. The middleware is responsible for configuring network resources by using information read from BPM workflows. The configuration of these resources results in the creation of virtual inter-domain circuits. The front-end system is a set of friendly Web user interfaces that allow to design, administrate, and run the workflows.

Pyretic (MONSANTO et al., 2013) is an imperative and domain-specific language embedded in Python that aims to support the building up of solutions for managing OpenFlow networks. Pyretic is based on three elements: an abstract packet model, an algebra of high-level policies, and network objects. The abstract model and the algebra assist to compose policies for monitoring and controlling. These policies allow to create modular applications that are executed on network objects, such as switches and ports. When developing these applications, advanced network programmers are able to manage OpenFlow-based networks in a higher-abstraction level than when programming rules on particular controllers.

Procera (KIM; FEAMSTER, 2013) is a control framework aimed to manage networks that follow the Software-Defined Networking (SDN) paradigm. This framework is formed by a policy language based on functional reactive programming and a policy engine. Such a policy language provides for expert network programmers a set of control domains. These domains support the creation of monitoring and configuration policies. This creation of policies operates in a higher-abstraction level than the development of rules intended to particular network operating systems. The policy engine is in charge of translating high-level policies to rules to be executed on specific SDN-based networks.

OrchSec (ZAALOUK et al., 2014) is an architecture intended to enhance network security by leveraging the SDN paradigm. The main components of this architecture are: Network Monitor and Orchestrator. The Network Monitor offers functionalities to inspect network traffic, apply traffic filters, and trigger events when a particular traffic pattern happens. The Orchestrator supports the development of composite applications for coping with security attacks such as spoofing of the Address Resolution Protocol and Distributed Denial of Service. These applications are built by combining, in an API level, SDN capabilities (*e.g.*, network-visibility and centralized management) and Network Monitor functionalities.

Table 2.4 – Research on network management

Research Work	**Mashup**	**SM**	**Agent**	**Rule**	**Policy**	**SOA**	**BPM**
(CHEN et al., 2010)				✓			
(CLAYMAN; GALIS; MAMATAS, 2010)						✓	
(BOLTE et al., 2010)						✓	
(MATTOS et al., 2011)			✓				
(CLAYMAN et al., 2011)						✓	
(KIM; KIM, 2011)						✓	
(SANTANNA; WICKBOLDT; GRANVILLE, 2012)						✓	✓
(MONSANTO et al., 2013)					✓		
(KIM; FEAMSTER, 2013)					✓		
(ZAALOUK et al., 2014)						✓	
This thesis	✓	✓		✓		✓	

Table 2.4 presents the aforedescribed research works, revealing that none of them has used SM and mashups for carrying out network management as this thesis does. In COOLAID (CHEN et al., 2010), Lattice (CLAYMAN; GALIS; MAMATAS, 2010), Libvirt (BOLTE et al., 2010), IMO (CLAYMAN et al., 2011), NetOpen (KIM; KIM, 2011), Pyretic (MONSANTO et al., 2013), Procera (KIM; FEAMSTER, 2013), and OrchSec (ZAALOUK et al., 2014), the network administrator is responsible for writing or programming policies, queries, rules, or basic services in specific languages, APIs, and/or controllers. Therefore, when using such proposals the work of network administrator remains complex and consumes a lot of time, which makes difficult their use as situational solutions. In turn, OMNI (MATTOS et al., 2011) and MEICAN (SANTANNA; WICKBOLDT; GRANVILLE, 2012) provide friendly GUIs to facilitate several network management operations. However, these proposals were not conceived to be extended or enhanced by network administrators. Thus, OMNI and MEICAN also offer a constrained response capacity to be used as situational solutions.

Finally, regarding all the aforecited proposals, it is important to point out that they were evaluated using metrics, such as traffic, time of response, and/or code lines. However, none of them has jointly assessed the complexity and the consuming of time of network management tasks fulfilled daily by network administrators.

2.4 Final remarks

Initially, in this chapter, the main concepts of the SM discipline and relevant research employing SM in diverse application domains were described. Subsequently, fundamental concepts about the mashup technology and important research that uses mashups in different domains were introduced. Afterwards, several research works that aim to facilitate the daily work of network administrator were presented.

Unlike the works presented in this chapter, this thesis considers concepts from the SM discipline and the mashup technology for proposing an approach (see Chapter 3) that focuses on carrying out network management tasks in an effective way. Furthermore, complementarily to these works that have been evaluated using metrics, such as traffic, time of response, and/or code lines, this thesis is also concerned with the complexity and the consuming of time of the work conducted by network administrators when coping with unexpected, dynamic, and heterogeneous situations (see Chapter 4).

3 CARRYING OUT NETWORK MANAGEMENT USING SITUATION MANAGEMENT AND MASHUPS

This chapter provides an extensive study about how to network administrators participating in a service ecosystem are able to address network management situations. In this sense, this chapter starts proposing the innovative concepts of network management situation and mashment. Next, this chapter presents a new service ecosystem based on such concepts. After, this chapter introduces and discusses a novel process that network administrators must follow in the ecosystem to deal with network management situations by mashments. This chapter finishes presenting a new architecture that supports the operation of the ecosystem and process above mentioned.

3.1 Fundamental concepts

This thesis defines two fundamental concepts for carrying out network management using SM and mashups. The following subsections detail the network management situation concept and its datamodel, describe several motivating scenarios, and present the mashment concept.

3.1.1 Network management situation

A network management situation is an unexpected, dynamic, and heterogeneous situation happening or that might happen in the network management domain. Hereinafter, for the sake of brevity, network management situations are called nmsits. Examples of basic nmsits include: (*i*) a router has a sudden loss of received packages; and (*ii*) a switch has an unforenseen loss of transmitted packages. Two or more nmsits form a complex one. For instance, first, a link formed by two virtual switches has an abrupt and intermittent performance degradation. Second, a set of routers/switches has an unexpected traffic overload.

Figure 3.1 presents the conceptual model of nmsits. Such model encoded in the JavaScript Object Notation (JSON) is: $[\{NAMESIT : namesit, NMSIT : [\{nmsit_1\}, \{nmsit_n\}]\}]$. Where, $NAMESIT$ is the generic name of the set of $NMSIT$ and $nmsit_n$ is given by: $[\{SITUATION : situation, EAC : [\{eac_1, eac_n\}]\}]$. Here, $SITUATION$ is the specific name of nmsit and EAC represents the collection of entities, attributes, and constraints involved in such nmsit. The structure of eac_n is: $[\{ENTITY : entity_n, AC : [\{ac_1, ac_n\}]\}]$. Where, $ENTITY$ is any entity involved in a nmsit and AC represents the set of attributes and constraints of such entity. Here, ac_n is: $[\{ATTRIBUTE : attribute_n, CONSTRAINT : constraint_n\}]$. In the rest of this thesis, JSON is used to represent data and metadata because it is more lightweight than the eXtensible Markup Language (XML) (PAUTASSO; ZIMMERMANN; LEYMANN, 2008). In such data and metadata, capital and lowercase letters indicate names and values of JSON properties, respectively.

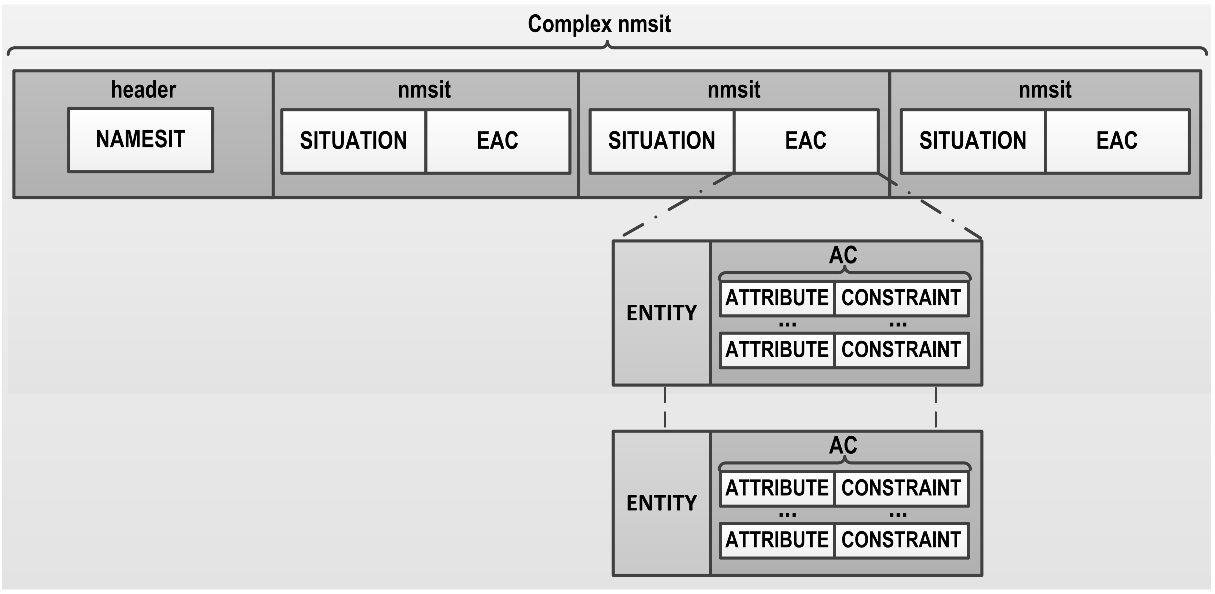


Figure 3.1 – Network management situation model

Examples of nmsits encoded in JSON are:
(-) $namesit$ = {*drop of received packages*}, $situation$ = {*sudden drop of received packages in a virtual router*}, and $eac = [\{ENTITY : vyattaRouter, ac : [\{ATTRIBUTE : receivedPkg, CONSTRAINT :< 95\%\}]\}]$
(-) $namesit$ = {*drop of transmitted packages*}, $situation$ = {*unexpected drop of sent packages in a virtual switch*}, and $eac = [\{ENTITY : cyscoSwitch2960, ac : [\{ATTRIBUTE : sentPkg, CONSTRAINT :< 90\%\}]\}]$.

An example of a nmsit formed by two ones is:
$namesit$ = {*link has an unexpected overload in memory and processor*}, $situation_1$ = {*a switch has an unforeseen overload in both memory and processor*}, $eac_1 = [\{ENTITY : cysco100, ac : [\{ATTRIBUTE : processor, CONSTRAINT :> 97\%\}, \{ATTRIBUTE : memory, CONSTRAINT :> 95\%\}]\}$, $situation_2$ = {*a switch has a sudden overload in both memory and processor*}, and $eac_2 = [\{ENTITY : openvswitch, ac : [\{ATTRIBUTE : proces, CONSTRAINT :> 91\%\}, \{ATTRIBUTE : mem, CONSTRAINT :> 91\%\}]\}]$.

A first example of scenario in which one or more nmsits happen is: let's suppose that a team of network administrators is responsible for managing a large company. In this company, the communication between the Pin Pads shops and the Enterprise Resource Planning (ERP) system is provided by an outsourced Internet Service Provider (ISP). If a sudden failure on packet transmission occurs in the ISP or an internal connection error happens in the border router of one or more shops, the company might lose a huge amount of revenues, since the payment by cards becomes inoperative.

A second example of scenario in which one or more nmsits happen is: let's suppose that a group of network administrators is in charge of managing a Network Operator. This operator provides network infrastructure to Small and Medium Enterprises (SMEs) using resources, such as virtual switches and virtual routers, supplied by several Virtual Network Providers (*e.g.*, VNP_a, VNP_b, and VNP_c). If an abrupt performance degradation occurs in virtual links of one or more SMEs, generated by unidentified errors in the virtual switches that connect VNP_a, VNP_b, and VNP_c, the operator might infringe the SLAs established with SMEs and, as result, lose a lot of money.

A third example of scenario in which one or more nmsits happen is: let's suppose that a team of network administrators is responsible for managing the network of a University. Such network is composed of several campus networks handled each one by a different controller. If an unforeseen loss/duplication of packets occurs in the Open vSwitches that communicate the campus networks, the network as a whole may collapse. This collapse might generate severe and unexpected failures in the financial and academic systems of the University that are located at different campus.

In the aforedescribed scenarios, network administrators require investigating and comprehending nmsits as easy and fast as possible. In particular when coping with nmsits, network administrators face mainly the following challenges: (*i*) the intricacy and heterogeneity to conduct situation management operations (*e.g.*, to collect, split, filter, add, and merge information) on multiple devices/networks involved in nmsits, (*ii*) the need by functionalities that enable to rapidly and easily create tunable and composite situation solutions; and (*iii*) the necessity by visualization functionalities that provide situational information in a very understable way.

Currently nmsits and their associated challenges can be addressed by following, among other, the next approaches. The first one is to use several mismatched network management solutions, such as ZenOSS (BADGER, 2008), OpenNMS (CHIANG et al., 2009), and proprietary Command Line Interfaces (CLIs), but it hinders and overloads the work of network administrators. In this sense, a shortcoming of this approach is that it is more complex and requires more time than using an integrated solution. The second approach is to use home-brewed scripts that integrate two or more network management solutions. The shortcomings of this approach are the difficulty and time required to program and run these scripts. These shortcomings are because the development of scripts is a daunting and complex task for non-programmers as network administrators who usually do not have deep knowledge in programming. Since when using the above approaches, the work of network administrators remains complex and time consuming, this thesis introduces an effective approach based on SM and mashups for network management.

3.1.2 Mashment

In a broad sense, a mashment (see Figure 3.2) 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. In this definition, first, tunable means that mashments are extendible and customizable. Second, as mashments are based on the mashup technology, they can be created/composed and launched by a network administrator because they inherit the abstraction model, composition model, and focus on end-user of mashups. Third, facilitate is associated with the complexity/difficulty perceived and the time spent by network administrators to carry out situational management tasks.

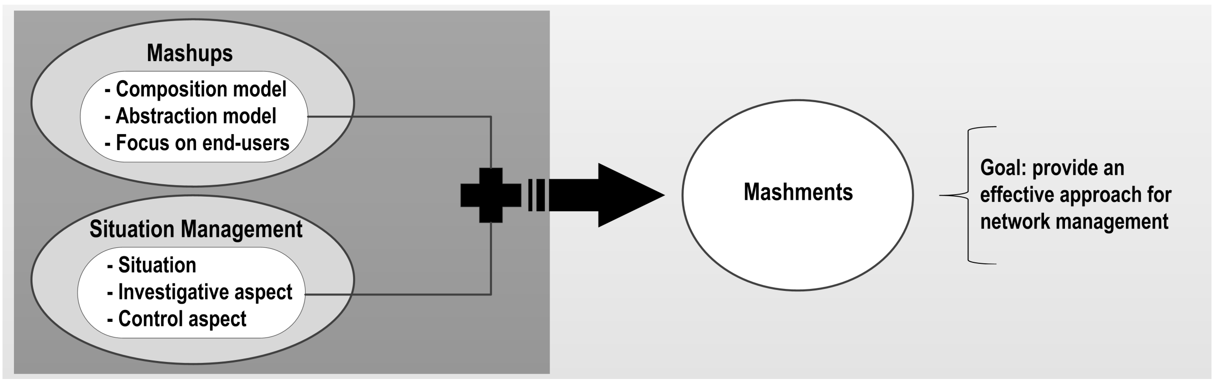


Figure 3.2 – Mashment concept

Regarding the mashment concept is worth noting, it is innovative because, first, the conduction of network management tasks has not been up to now investigated or addressed using foundations of the SM discipline and the mashup technology. Second, it leads network management towards a novel environment focused on network administrators in which they are able to satisfy their situational needs by themselves. Third, unlike previous solutions used to handle situations in the network management domain that are concerned about traffic and/or time of response, the solution proposed in this thesis further considers the complexity perceived and the time needed by network administrators to conduct tasks aimed to face nmsits.

3.2 Mashment ecosystem

The concept of ecosystem in the software industry arose with the increasing number of companies that embraced the deployment of SOA-based architectures to obtain new composite applications from the combination of several atomic services (HUANG; FAN; TAN, 2012). In a broad sense, a service ecosystem is formed by services, providers of services, and relationships among providers and services (BARROS; DUMAS, 2006). In particular, when acting as a single unit, providers supplying resources (*e.g.*, data, GUIs, and application logic), end-users and developers building up mashups by combining resources, and end-users using mashups form a service ecosystem, called mashup ecosystem (HUANG; FAN; TAN, 2012) (ENDRES-NIGGEMEYER, 2013). In this sense, the natural way to present the mashment-based approach introduced in this thesis is from a service ecosystem point of view.

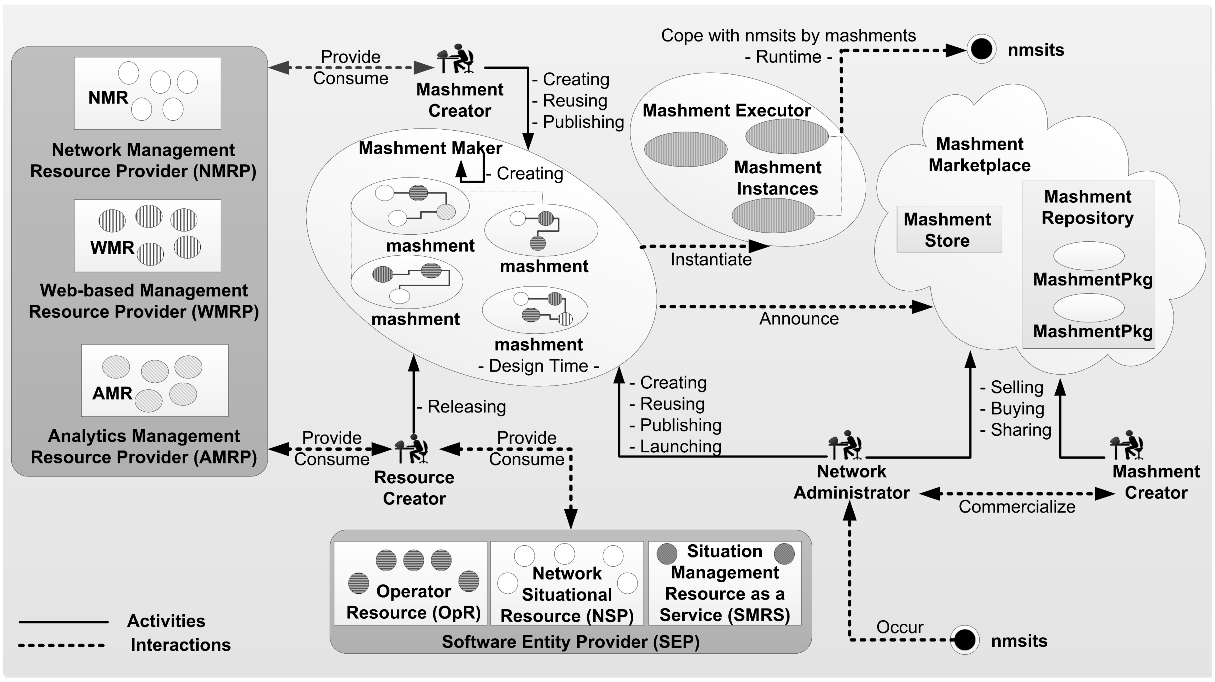


Figure 3.3 – Mashment ecosystem

The Mashment Ecosystem (see Figure 3.3) aims to provide an effective approach for network management and is based on: (*i*) the abstraction model, composition model, and focus on end-users of mashups; and (*ii*) the investigative and control aspects of SM. This ecosystem is formed by: resources (including mashments), stakeholders, software entities, activities performed by stakeholders and software entities, and interactions happening among stakeholders and software entities. In a general way, in the proposed ecosystem, Network Administrators and Mashment Creators build up mashments by using the Mashment Maker. This Maker is also responsible for automatically creating mashments. These mashments are made up of resources from different providers and are executed in the Mashment Executor. Such resources are released by the Resource Creator. In the Marketplace, Network Administrators and Mashment Creators share, sell, and purchase mashments.

From a high-level point of view, Network Administrators participating in the Mashment Ecosystem can address nmsits as follows. First, they obtain mashments in four ways: (*i*) buying mashments in the Marketplace, (*ii*) getting mashments free of charge in the Marketplace, (*iii*) creating mashments in the Mashment Maker; and (*iv*) selecting mashments generated by the Maker. Second, they launch mashments by the Maker. On runtime, a launched mashment copes with one or more nmsits. It is important to highlight that the proposed ecosystem can evolve over time because of the emergence and perishing of resources, the sharing and commercialization of mashments, and the dynamic interactions among stakeholders. The following subsections explain in detail the Mashment Ecosystem.

3.2.1 Resources

In the Mashment Ecosystem, the Situation Management Resources (SMR) are entities clearly identifiable in a time interval that provide access and communication to and from network elements or entire networks involved in nmsits. There are seven types of SMR: Network Management Resource (NMR), Web-based Network Management Resource (WMR), Analytics Management Resource (AMR), Network Situational Resource (NSR), Operator Resource (OpR), Situation Management Resource as a Service (SMRS), and Mashment.

A NMR is any entity intended to conduct network management operations. Examples of NMR are Ganglia (MASSIE; CHUN; CULLER, 2003), Nagios (BARTH, 2008), and ZenOSS (BADGER, 2008) to manage traditional networks, Citrix Center (WANG et al., 2011) for monitoring virtual resources, NetOpen (KIM; KIM, 2011) and OMNI (MATTOS et al., 2011) to control OpenFlow-based networks, network monitoring systems based on SNMP, and all APIs that provide interaction with network elements.

A WMR is any entity, available along the Web, conceived or that can be used to perform network management tasks. Examples of WMR are the Multi Router Traffic Grapher (MRTG) (OETIKER, 1998) to generate Web pages with images presenting the traffic of network links, the RRDTool (OETIKER, 2001) to display over time the performance data of routers, the Yahoo Maps API (LIN; GAO; XU, 2009) and the Google Maps API (TERESCO, 2012) to show the geographic location of several network devices, and the Google Chart API (RUTHKOSKI, 2013) to show the memory consumption of virtual switches.

An AMR is any entity intended to analyze network management information. Examples of AMR are the Management Traffic Analyzer (SALVADOR; GRANVILLE, 2008), the Junos Network Analytics Suite (NETWORKS, 2014), and the Sandvine Network Analytics (ULC, 2014). The Management Traffic Analyzer is useful to interpret the functioning of network devices supporting SNMP. The Junos Network Analytics Suite is helpful to understand what is happening on networks using Junos proprietary devices. The Sandvine Network Analytics allows getting right-time information of networks regardless of underlying technologies.

A NSR is any entity that provides functionalites aimed to automate and carry out the aspects of SM. These functionalities are of three types: (*i*) Collecting that is to retrieve information about nmsits, (*ii*) Fusing&Correlating that is to merge and correlate the retrieved information by Collecting, Fusing&Correlating and Collecting support the creation of investigative plans that are useful to determine the cause of nmsits; and (*iii*) Resolving that enables to perform network management operations aimed to control (change/preserve) nmsits, consequently, Resolving supports the creation of resolutive plans. Examples of NSR are the JESS (HILL, 2003), JBOSS Drools (BROWNE, 2009), and Apache Camel (IBSEN; ANSTEY, 2010). The JESS is a general-purpose platform that permits to detect and control situations by rules (defined using XML or the JESS Rule Language) and Java applications. The JBOSS Drools is a generic platform that allows to recognize and control generic situations by rules (defined using DRL) and Java applications. The Apache Camel enables to process events (*i.e.*, general situations) from mutiple sources by means of a CEP-engine based on the Java Language.

An OpR is any entity suitable to combine resources and, so, to build up and generate mashments. There are three types of OpR: (*i*) control patterns (*e.g.*, sequential, parallel, conditional, and templates) that allow defining the control flow of mashments, (*ii*) structures for configuring (*e.g.*, functionalities to set the security credentials needed to monitor a virtual router) and invoking the resources that form mashments; and (*iii*) structures for receiving, sorting, and filtering (*e.g.*, functionalities to perform information selection on text-plain containing data of network devices running on virtualized environments) the retrieved information from any type of resource.

A SMRS is any entity that offers 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 on defining and providing a common data-format to interchange information of resources, a well-known interface to each resource, and a common protocol to communicate with such interfaces. There are five types of SMRS: NMR as a Service (NMRS) provides functionalities of NMR, WMR as a Service (WMRS) offers operations of WMR, AMR as a Service (AMRS) supplies functionalities of AMR, NSR as a Service (NSRS) provides operations of NSR, and OpR as a Service (OpRS) offers functionalities of OpR. An example of NMR is the Floodlight Controller (FLOODLIGHT, 2013) that handles switches OpenFlow-enabled, the associated NMRS is the Floodlight REST API that provides, via requests-and-responses HTTP, monitoring information of these switches including their flows and links.

A mashment is also a resource of the Mashment Ecosystem, which means that mashments can be used to create another ones. It is to note that by considering mashments as a type of resource, their reuse is promoted and, as a consequence, the growth of the proposed ecosystem itself is also encouraged. Mashments can be static or dyamic. Static ones are characterized by: (*i*) they are not able to recognize nmsits; and (*ii*) their plans (*i.e.*, execution flows to investigate and control nmsits) are defined by Network Administrators in the Mashment Maker. Unlike static mashments, dynamic ones are able to recognize nmsits and their plans are automatically generated (*i.e.*, without direct intervention of Network Administrators) by the Maker.

3.2.2 Stakeholders

In the Mashment Ecosystem, a stakeholder affects and is affected by activities and interactions carried out by other one. There are six types of stakeholders: Network Management Resource Provider (NMRP), Web-based Management Resource Provider (WMRP), Software Entity Provider (SEP), Resource Creator, Mashment Creator, and Network Administrator.

NMRP, WMRP, and AMRP are in charge of supplying NMR, WMR, and AMR, respectively. Examples of NMRP are, first, the Citrix Systems Inc that offers solutions and programming interfaces to manage virtual servers. Second, the Cisco Systems Inc that provides tools and libraries to monitor and configure network devices. An example of WMRP is a big player as the Yahoo Inc that provides visualization libraries and map services useful to present network management information. Another example of WMRP is the Oetieker&Partners Inc that provides Web solutions intended for network monitoring, such as RRDTool and MRTG. An example of AMRP is the Juniper Networks Inc that offers the Junos Network Analytics Suite.

A SEP is responsible for providing one or more software entities. In general, the Mashment Maker (containing visual representations of resources), Mashment Executor, OpR, NSR, and SMRS are provided, in an unified way, by the same SEP. In turn, the Mashment Repository and Mashment Store are usually offered, in a distributed way, by different SEP. It is to point out that in an ecosystem as the proposed can exist several Makers, Executors, Repositories, and Stores.

Before being used in the building up of mashments, many WMR, NMR, AMR, OpR, and NSR need of adaption, in data format and/or communication protocol. The Resource Creator is responsible for carrying out this adaptation called releasing. Since such a releasing requires strong programming skills, Resource Creators are usually software companies and professional developers. An example of Resource Creator is the Open Software community that provides APIs to interact via standardized protocols with network devices and servers containing virtual routers.

The Mashment Creator is in charge of: (*i*) creating, reusing, and publishing mashments by the Mashment Maker, (*ii*) defining nmsits (*i.e.*, nmsit patterns that are instances of nmsit model) that will be automatically recognized, (*iii*) creating composition templates intended to dynamically generate mashments that will be responsible for dealing with the recognized nmsits; and (*iv*) sharing, selling, and buying mashments in the Marketplace. The Creator can obtain economic benefits by commercializing (selling and buying) mashments. Examples of this stakeholder are software companies and professional developers with deep knowledge in network management.

The Network Administrator is responsible for, first, coping with nmsits by using mashments. Second, he/she can create, reuse, publish, and launch mashments by the Mashment Maker and the Mashment Executor. The creation, reuse, and publication of mashments support the continuous improvement of workspaces used in the network management domain. Third, he/she can commercialize mashments in the Marketplace for getting profits. It is noteworthy that as mashments are a special type of mashup, they inherit its ease of development (LIU et al., 2007)(CAPPIELLO et al., 2011)(SANTOS et al., 2013). As a result, Network Administrators do not need advanced technical skills about Web programming for developing mashments.

3.2.3 Activities and interactions

In the Mashment Ecosystem, activities are actions conducted by stakeholders and software entities in order to deal with nmsits. The stakeholders perform activities by using software entities. In turn, software entities are able to automatically carry out some activities. Eight activities are considered in the proposed ecosystem: Releasing, Creating, Reusing, Publishing, Launching, Selling, Buying, and Sharing.

When carrying out the activity of Releasing, Resource Creators convert SMR (by adapting data format and communication protocol of NMR, WMR, AMR, NSR, and OpR) in mashupable resources called SMRS. Such an adaptation enables the combination of resources and it is needed because there is not a common format neither a standardized interface/protocol to retrieve and/or bidirectionally interact with data, application logic, and user interfaces of SMR involved in nmsits. After Releasing, the adapted resources can be used to build up mashments.

The Mashment Maker, Mashment Creator, and Network Administrator perform Creating. When conducting this activity, first, Creators and Administrators can build up static mashments. From a high-level point of view, the static creation involves the next steps: (*i*) discover the available resources, (*ii*) select the suitable resources to address nmsits; and (*iii*) orchestrate plans (*i.e.*, the execution flows of mashments) targeted to cope with nmsits, by combining the previously selected resources. Second, when Creators carry out this activity, they can build up nmsit models and composition templates that are key elements for dynamically generating mashments.

The Mashment Maker automatically carries out Creating in order to generate dynamic mashments. In a general way, the dynamic creation involves: (*i*) recognize automatically nmsits by using models/patterns defined by Mashment Creators, (*ii*) select resources involved in the recognized nmsits, (*iii*) define plans (*i.e.*, customize the composition templates defined by Mashment

Creators) in an automatic way by using the previously selected resources, (*iv*) monitor if the resources used in these plans are available and flawless; and (*v*) reconfigure such plans if any resource is in flaw or unavailable.

The Mashment Creator and Network Administrator also conduct the activities: Reusing, Publishing, Sharing, Selling, and Buying. When carrying out the activity of Reusing, Creators and Administrators take advantage of existing mashments, aiming to create more complex and innovative mashments. Furthermore, this activity supports the growth over time of the proposed ecosystem.

When conducting the activity of Publishing, the Mashment Creator and Network Administrator package and put mashments in the Mashment Repository. After Publishing, mashments can be shared, sold, and purchased. The activity of Sharing enables to offer and get mashments free of charge. The activities of Selling and Buying allow commercializing mashments. Sharing, Selling, and Buying promote the reuse of mashments and encourage the evolution over time of the proposed ecosystem.

The Network Administrator conducts the activity of Launching. When performing this activity, he/she requests to the Mashment Maker for executing static and/or dynamic mashments. After a mashment is launched, on runtime, it is called Mashment Instance. Note that mashments are always launched by the Network Administrator, it is because the proposed ecosystem is not intended to substitute him/her else to support his/her daily work.

In the Mashment Ecosystem, the interactions take place in the relationships stakeholder / stakeholder, software entity / software entity, and stakeholder / software entity. Seven interactions are considered in the proposed ecosystem: Provide, Consume, Instantiate, Announce, Commercialize, Occur, and Cope.

Provide and Consume are interactions that occur from the need of consumption and supply of NMR, WMR, AMR, NSR, OpR, and SMRS, during: (*i*) the building up of the Mashment Maker; and (*ii*) the carrying out of Releasing, Creating, and Reusing. Provide and Consume take place among, first, Resource Providers, Resource Creators, Mashment Creators, and Network Administrators. Second, Resource Providers and the Maker.

Instantiate is an interaction that happens among the Mashment Maker and the Mashment Executor when a Network Administrator requests the execution of one or more mashments. Announce is an interaction that takes place among the Maker and the Marketplace when a Network Administrator or a Mashment Creator asks for publishing one or more mashments that can be later shared, purchased, and sold in Commercialize. This commercialization occurs among Mashment Creators and Network Administrators.

Occur and Cope are special interactions defined to represent the happening of nmsits and the corresponding responses offered by Mashment Instances. When one or more nmsits Occur, Network Administrators obtain (Buying, Creating, or Reusing) and run (Launching) mashments. During Cope, to deal with nmsits, Mashment Instances delegate their situational management operations to SMRS that, in turn, delegates these operations to SMR.

3.2.4 Software entities

In the Mashment Ecosystem, the software entities are responsible for supporting and automating the activities and interactions aforedescribed. The proposed ecosystem considers three types of software entities: Mashment Maker, Mashment Executor, and Mashment Marketplace.

The Mashment Maker assists the activities Creating, Reusing, Publishing, and Launching. Furthermore, it is involved in the interactions Provide, Consume, Instantiate, and Announce. To aid handling of nmsits, the Maker supports: (*i*) a lightweight process for developing and launching mashments (see Section 3.3), (*ii*) mechanisms to automatically recognize nmsits and dynamically generate mashments (see Section 3.4); and (*iii*) high-level programming tools (see Section 4.1) providing visual functionalities to create and run mashments.

The Mashment Executor is involved in the interactions Instantiate and Cope. The Executor is responsible for controlling the execution flow of Mashment Instances that handle nmsits. As mashments are formed by two or more resources that function in back-end or front-end, the Executor is also a lifecycle manager that enables to create, destroy, and cache Mashment Instances into Web servers and Web/mobile clients. The Section 3.4 details the Executor.

The Mashment Marketplace allows to establish a new value chain in which revenues are not shared only by WMRP, AMRP, NMRP, and SEP but all stakeholders. Marketplaces involving end-users (as Network Administrators) and professional developers (as Mashment Creators) have proved valuable to promote the evolution over time of service ecosystems (as the Mashment Ecosystem). The Android Market (BUTLER, 2011) and the Apple Store (COGET, 2011) are successful examples of application marketplaces.

The Marketplace is made up of the Mashment Store and the Mashment Repository. The Store supports the carrying out of activities Selling, Sharing, and Buying, assisting in that way to the Commercialize interaction. In turn, the Repository is involved in the Announce interaction. As result of Announce and in order to be sold or shared, MashmentPkgs are stored in the Mashment Repository. It is to point out that the Repository (see Section 3.4) and the Store encourage the expansion over time of the proposed ecosystem.

3.3 Process for addressing nmsits by mashments

The process to develop and launch mashments defines how to address nmsits in the proposed ecosystem. The following subsections describe the overall functioning, complexity, and consuming of time of such process.

3.3.1 Overall functioning

Considering the ecosystem above described, the set of mashments is expressed as follows. $MASHMENT = \{mashment_x | mashment_x = (R_{used}, r_{root}, \delta, NMSIT_{addr})\}$. Where: (*i*) R_{used} is the set of resources used to build up the $mashment_x$, (*ii*) r_{root} is the root resource ($\in R_{used}$) that starts the execution of such $mashment_x$, (*iii*) δ is the execution flow (*i.e.*, an investigative/resolutive plan that defines how R_{used} are logically combined/linked to handle $NMSIT_{addr}$) of the $mashment_x$; and (*iv*) $NMSIT_{addr}$ is the set of one or more nmsits addressed by the $mashment_x$.

Regarding the set $MASHMENT$, it is relevant to mention, first, $NMSIT_{addr} \subseteq$ of all unexpected, dynamic, and heterogeneous situations that may happen in the network management domain. Second, if the $mashment_x$ is static, the Network Administrator is responsible for developing it by using the Mashment Maker. Such development implies the definition of R_{used}, r_{root}, δ, and $NMSIT_{addr}$. Third, if the $mashment_x$ is dynamic, the Maker is in charge of generating it by customizing a composition template that predefines a specific δ for dealing with a particular $NMSIT_{addr}$. Such generation also involves the definition of R_{used}, r_{root}, δ, and $NMSIT_{addr}$.

The process to develop and launch the $mashment_x$ is formed by the following high-level tasks (see Figure 3.4): Select, Configure, Combine, Launch, and Tune. Select, Configure, Combine, and Tune can be conducted by the Mashment Maker, Network Administrator, and Mashment Creator. Launch can be carried out by the Network Administrator. The tasks of this process are related to activities of the Mashment Ecosystem as follows: Select, Configure, Combine, and Tune that form Develop are associated with the activities of Creating and Reusing. In turn, Launch corresponds to the Launching activity.

- ***Select Resources***. This task is to define R_{used} by selecting resources available in the Mashment Ecosystem. This selection may include the choosing of, first, a $mashment_z$ ready to be used (*i.e.*, there is a mashment that handles $NMSIT_{addr}$). Second, one or more mashments (*i.e.*, there is a $mashment_y$ that faces a similar $NMSIT_{addr}$) to be modified/enhanced. Third, resources to create the $mashment_x$ from scratch. Fourth, a composition template (*i.e.*, there is a predefined δ for coping with a particular $NMSIT_{addr}$)

that will be customized for generating the $mashment_x$.

- ***Configure Resources***. This task is to define the set of resources configured, called R_{conf}. This set is defined by providing all functioning settings of one or several resources belonging to R_{used}. Note that $R_{conf} \subseteq R_{used}$.
- ***Combine Resources***. This task is to create (from scratch) or customize (from a composition template) the δ of the $mashment_x$ by combining/linking selected and configured resources. Such creation or customization includes, first, the choosing of r_{root}. Second, the specification of the data propagation between the resources involved in the building up of the $mashment_x$ by correlating their outputs and inputs.
- ***Tune Mashment***. If the $mashment_x$ needs to be modified/enhanced, its δ can be tunned. Such tuning may imply the selection, configuration, and combination of new resources or simply the rearrangement of previously selected resources.
- ***Launch Mashment***. This task is to request the execution of the $mashment_x$. As a result, an instance of the $mashment_x$ is created. On runtime, such an instance copes with $NMSIT_{addr}$.

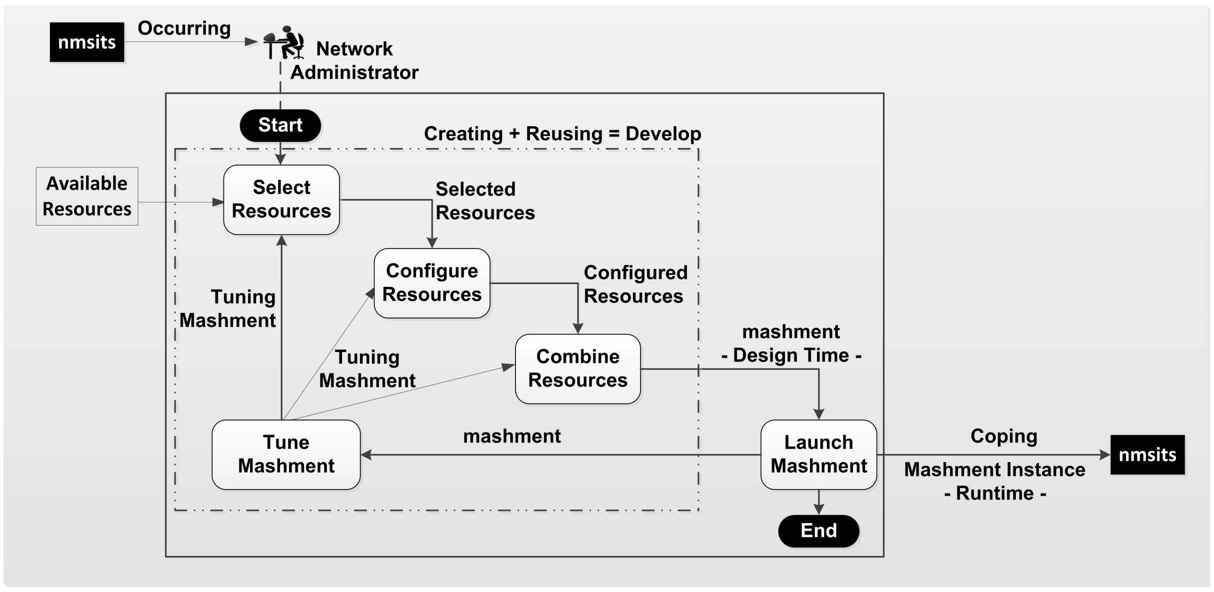


Figure 3.4 – Process to develop and launch mashments

3.3.2 Complexity

The complexity (*i.e.*, ζ) to develop and launch the $mashment_x$ is computed by summing the complexity of tasks forming the process aforedescribed. It is because the tasks Select, Configure, Combine, and Launch are independent. In the equation 3.1, ζ_{sel}, ζ_{con}, ζ_{com}, and ζ_{lau} represent the complexity of Select, Configure, Combine, and Launch, respectively. In turn, i,

j, k, and o denote the number of times that such tasks are conducted, allowing to consider the complexity of Tune.

$$\zeta = \sum_{1}^{i} \zeta_{sel} + \sum_{1}^{j} \zeta_{con} + \sum_{1}^{k} \zeta_{com} + \sum_{1}^{o} \zeta_{lau} \tag{3.1}$$

The following paragraphs define ζ_{sel}, ζ_{con}, ζ_{com}, and ζ_{lau}. These definitions are based on per-task metrics for IT Service Management (ITSM) processes (DIAO; KELLER, 2006). Such metrics are used as foundation because, first, they allow quantifying the complexity perceived by human beings (as Network Administrators) participating in a particular process formed by several tasks. Second, they permit identifying tasks to be automated. Third, they enable to establish basis for measuring improvements between versions of a process.

The complexity of Select is expressed as follows:

$$\zeta_{sel} = \sum_{m=1}^{M} \varsigma_m + (nAvailableResources - 1) * gF * cF \tag{3.2}$$

In the equation 3.2, M is the total number of elements being part of R_{used} and ς_m is the complexity of selecting the m-resource. ς_m can take one of three values depending on the automation of m-selection: 0 - if m-selection is fully automated (*e.g.*, there is a tool that selects automatically the resources needed to build up the $mashment_x$). 1 - if m-selection is manual but tool-assisted (*e.g.*, there is an integrated development tool to select the resources required to create the $mashment_x$). 2 - if m-selection is manual (*e.g.*, writing code). $nAvailableResources$ is the number of resources available to build up the $mashment_x$ (*i.e.*, more available resources result in higher complexity of selection). gF is the grade of guidance provided to select the resources needed to form the $mashment_x$, which can take one of three values: 1 - if correct guideline about resources to be selected is offered. 2 - if general information about each available resource is supplied. 3 - if information is not provided. cF represents the impact of wrong selection of resources, its value can be: 0 - if negligible impact. 1 - if moderate impact. 2 - if severe impact.

The complexity of Configure is defined in the following way:

$$\zeta_{con} = \sum_{n=1}^{N} \varsigma_n . \tag{3.3}$$

In the equation 3.3, N is the total number of resources belonging to R_{conf} (note that as $R_{conf} \subseteq R_{used}$, so, $N \leq M$) and $\varsigma_n = \sum_{p=1}^{P} sourceParameter(p)$ is the complexity of configuring the n-resource. In ς_n, P is the total number of parameters to be configured and

$sourceParameter(p)$ can take one of seven values: 0 - if the p-parameter value is produced from automation (*e.g.*, there is a tool that automatically generates the configuration parameters). 1 - if the p-parameter value may be chosen freely (*e.g.*, a new password). 2 - if the p-parameter value is taken from task documentation (*e.g.*, set up $port = 8080$ for a HTTP server). 3 - if the p-parameter value is extrapolated from task documentation (*e.g.*, define a range of IP addresses). 4 - if the p-parameter value is not trivial for unexperienced Network Administrators (*e.g.*, set up the $url = http://IPAddressOfXenServer/rrdUpdates?host = true$ to retrieve statistics of virtual machines running on a determined XenServer). 5 - if the p-parameter is fixed by the environment to a specific value that is defined by Network Administrators after additional research (*e.g.*, set up the $smnpOid = 1.3.6.1.4.1.9.9.91.1.1.1.1.4$ to obtain the temperature of the Catalyst Cisco Switch). 6 - if the p-parameter value is constrained by the environment to a limited set of possible choices where Network Administrators need to infer the right choice (*e.g.*, set up the type of server virtualization technology to be monitored: $virtTech = VMware$).

The complexity of Combine is expressed as follows:

$$\zeta_{com} = \sum_{l=1}^{L} \varsigma_l + (M-1) * goF * coF \tag{3.4}$$

In the equation 3.4, L is the total number of links (logical connections) created to build up the $mashment_x$ and ς_l represents the complexity of creating the l-link that connects two elements of R_{used}. ς_l can take one of four values: 0 - if the l-link is automatically created (*e.g.*, there is a tool that generates automatically the δ of the $mashment_x$). 1 - if the l-link is manually created by a support tool and data transferred among resources connected must not be adapted. 2 - if the l-link is manually created and data transferred among resources connected must not be adapted. 3 - if the l-link is manually built and data transferred among resources connected must be adapted. M is the total number of elements of R_{used} (*i.e.*, more selected resources result in higher complexity of combination). goF is the grade of guidance provided to link the selected resources, it can take one of three values: 1 - if correct guidance is provided. 2 - if general information about links that can be established is offered. 3 - if information is not supplied. coF represents the impact of wrong combination of resources, its value can be: 0 - if negligible impact. 1 - if moderate impact. 2 - if severe impact.

The complexity of Launch (ζ_{lau}) can take one of three values depending on the automation for launching the $mashment_x$: 0 - if entirely automated, for instance, an autonomous mashment system that launches on demand the $mashment_x$. 1 - if manual but tool-assisted, for example, using a graphical launching environment to start the $mashment_x$. 2 - if manual, for instance, programming or customizing a script every time the $mashment_x$ needs to be executed.

Regarding the above complexities (*i.e.*, ζ_{sel}, ζ_{con}, ζ_{com}, and ζ_{lau}) is important to note that they are defined in a general way. This generic definition aims to allow quantifying the complexity of conducting the proposed process in an automatic, tool-assisted, and manual way.

3.3.3 Time-consuming

In the same way that the calculation of complexity, the consuming of time to develop and launch the $mashment_x$, hereinafter called time-consuming (*i.e.*, T), is computed by summing the time of conducting the tasks forming the process aforedescribed. In the equation 3.5, T_{sel}, T_{con}, T_{com}, and T_{lau} represent the time-consuming of Select, Configure, Combine, and Launch, respectively. In turn, i, j, k, and o indicate the number of times that such tasks are carried out, permitting to take into account the time of Tune.

$$T = \sum_{1}^{i} T_{sel} + \sum_{1}^{j} T_{con} + \sum_{1}^{k} T_{com} + \sum_{1}^{o} T_{lau} \tag{3.5}$$

The following paragraphs define T_{sel}, T_{con}, T_{com}, and T_{lau}. These time-consuming definitions are based on the Keystroke-Level Model (KLM) (KIERAS, 2001). KLM is used because it is useful to estimate the time that end-users (as Network Administrators) spend to carry out tasks supported on computer keyboard and mouse. In KLM, each task is modeled as a sequence of actions. Table 3.1 presents the original KLM-actions (KIERAS, 2001) and some helpful extensions (TIAN; WEBER; LUTTEROTH, 2011) found in the literature.

Table 3.1 – KLM actions

Action	**Description**
k	Press and release a key
$nc * k$	Type a string
p	Point the mouse
b	Hold or release the mouse
h	Move the hand from mouse to keyboard
dnd	Drag-and-drop a visual element
$wire$	Wire two visual elements

The time-consuming of Select is defined in the following way:

$$T_{sel} = \sum_{m=1}^{M} \tau_m \tag{3.6}$$

In the equation 3.6, M is the total number of elements being part of R_{used} and τ_m is the time required to select the m-resource. Such τ_m can take one of three values depending on the automation of m-selection: 0 - if m-selection is completely automated, for instance, a tool that automatically defines R_{used}. dnd - if m-selection is manual but tool-assisted, for example, a development integrated tool supporting the definition of R_{used}. $h + p + 2b + nc * k$ - if m-selection is manual, for instance, writing code in a specific programming language.

The time-consuming of Configure is expressed as follows:

$$T_{con} = \sum_{n=1}^{N} \tau_n \tag{3.7}$$

In the equation 3.7, N is the total number of elements belonging to R_{conf} and τ_n is the time required to configure the n-resource. Such τ_n can take one of two values depending on the automation of n-configuration: 0 - if the n-configuration value is produced from automation. $h + p + 2b + nc * k$ - if operational settings must be manually written.

The time-consuming of Combine is defined in the following way:

$$T_{com} = \sum_{l=1}^{L} \tau_l \tag{3.8}$$

In the equation 3.8, L is the total number of links created to build up the $mashment_x$ and τ_l represents the time required to create the l-link that connects two elements being part of R_{used}. Such τ_l can take one of three values depending on the automation of l-link creation: 0 - if the l-link is automatically created (*e.g.*, there is a tool that generates the δ of the $mashment_x$). $wire$ - if the l-link is manually created by means of a support tool. $h + p + 2b + nc * k$ - if the l-link is manually created/codified.

The time-consuming of Launch (T_{lau}) can take one of three values depending on the automation of requesting the execution of the $mashment_x$: 0 - if entirely automated, for instance, an autonomous mashment system that launches $mashments$ on demand. $h + p + 2b$ - if manual but tool-assisted, for example, using an execution environment to start the $mashment_x$. $h + p + 2b + nc * k$ - if manual, for instance, programming/customizing a home-brewed script every time the $mashment_x$ must be launched.

Regarding T_{sel}, T_{con}, T_{com}, and T_{lau} is important to note that they are defined in a general way. This generic definition aims to allow quantifying the time required to carry out the process to develop and launch mashments in an automatic, tool-assisted, and manual way.

3.4 Mashment system architecture

The Mashment System Architecture is intended to support: (*i*) the carrying out of the Mashment Ecosystem; and (*ii*) the process to develop and launch mashments. This architecture (see Figure 3.5) uses a layered architectural pattern (KESHAV, 1997) that operates as follows. The lower layer ($Layer_n$) provides services to the upper layer ($Layer_{n+1}$) through decoupled interfaces. In turn, the upper layer consumes services from the lower layer.

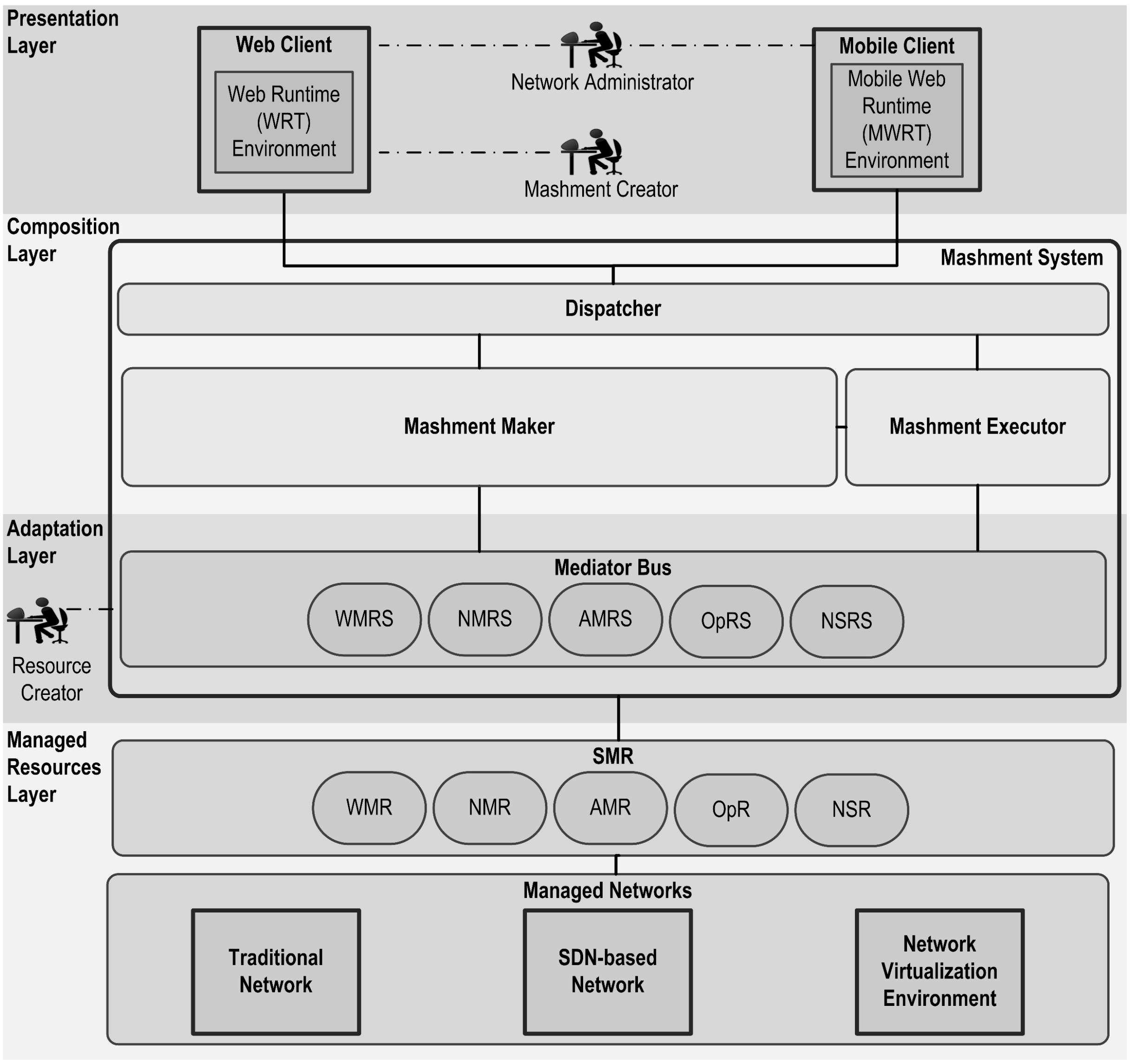


Figure 3.5 – Mashment system architecture

The layers and elements of the Mashment System Architecture are: The Managed Resources Layer made up of networks and SMR, the Adaptation Layer comprised of a collection of SMRS, the Composition Layer formed by the Mashment Maker, Mashment Executor, and Dispatcher, and the Presentation Layer made up of runtime environments. In a broad sense, in the proposed architecture: (*i*) the Adaptation Layer hides the complexity and heterogeneity of the Managed Resources Layer, (*ii*) the Composition Layer supports the build up of static and dynamic mashments; and (*iii*) the Presentation Layer assists the client-side execution and display of mashments. The following subsections present in detail the proposed architecture from bottom to top.

3.4.1 Managed resources layer

Figure 3.6 depicts the Managed Resources Layer. This layer is formed by a collection of SMR, traditional networks, SDN-based networks, and Network Virtualization Environments (NVEs). Considering that, first, this thesis already discussed SMR (see Subsection 3.2.1). Second, traditional networks are widely known. The next paragraphs introduce just concepts about SDN-based networks and NVEs, these concepts are useful to understand the case studies conducted to evaluate the mashment-based approach (see Chapter 4).

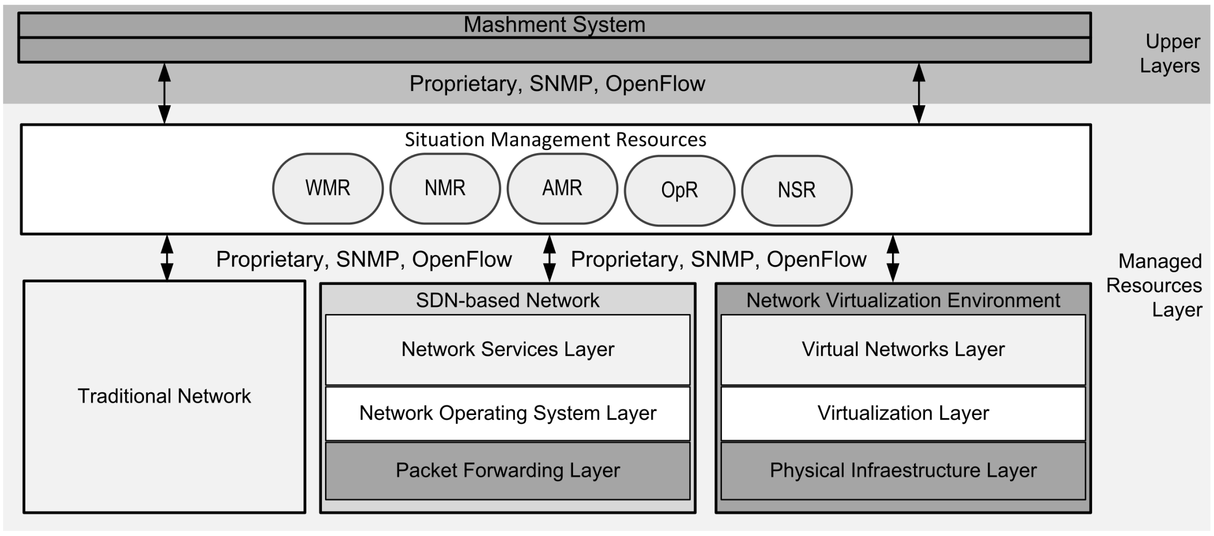

Mashment System
Upper Layers
Proprietary, SNMP, OpenFlow
Situation Management Resources
WMR
NMR
AMR
OpR
NSR
Proprietary, SNMP, OpenFlow
Proprietary, SNMP, OpenFlow
Managed Resources Layer
Traditional Network
SDN-based Network
Network Services Layer
Network Operating System Layer
Packet Forwarding Layer
Network Virtualization Environment
Virtual Networks Layer
Virtualization Layer
Physical Infraestructure Layer

Figure 3.6 – Managed resources layer

In the field of computer networks, the SDN paradigm has emerged as an important trend that proposes an architecture for future networks in which data and decision policies are separated in order to simplify the network operation (NATASHA et al., 2008). In a general way, SDN-based networks are formed by three architectural layers (LANTZ; HELLER; MCKEOWN, 2010) (LARA; KOLASANI; RAMAMURTHY, 2013): the Packet Forwarding datapath (*e.g.*, switches and routers passing packets), the Network Operating System (NOS) that controls such datapath by using a vendor-independent protocol, and the Network Services (*e.g.*, a new routing protocol) running on the top of NOS.

There are different proposals for deploying the SDN paradigm, such as the Forwarding and Control Element Separation (ForCES) framework (DORIA et al., 2010) and OpenFlow (MCKEOWN et al., 2008). In the ForCES framework, the Control Elements (*i.e.*, NOS) and Forwarding Elements (*i.e.*, the datapath) communicate by the ForCES protocol. In such a framework, the Network Services can be developed as distributed features running on the Control Elements.

In OpenFlow, the Controller (*i.e.*, NOS), such as POX (POX, 2013), Beacon (ERICKSON, 2013), NOX (NOX, 2013), and Floodlight (FLOODLIGHT, 2013), uses the OpenFlow protocol to handle network devices (*i.e.*, the datapath). Furthermore, the Controller supports deploying new-centralized Network Applications (*i.e.*, the Network Services), such as groundbreaking applications to path selection and novel multicasting protocols.

In the context of SDN-based networks, examples of NMR (*i.e.*, a specific type of SMR) are tailored libraries, such as the Java Beacon API and the NOX C++ API. The Java Beacon API allows conducting management operations on the Beacon Controller. In turn, the NOX C++ API allows carrying out network management tasks on the NOX Controller.

The Network Virtualization is also an important topic trend in the computer networks, which fundamentally proposes sharing a network physical infrastructure among several virtual networks (CHOWDHURY; BOUTABA, 2009). A NVE is formed by a Physical Infrastructure Layer, a Virtualization Layer, and a Virtual Networks Layer. The Physical Infrastructure Layer provides resources (*e.g.*, servers, routers, and links). The Virtual Networks are formed by virtual resources (*e.g.*, virtual routers/switches and network slices) allocated by the Virtualization Layer (*e.g.*, Xen, VMware, OpenVZ, and Flow Visor) from the Physical Infrastructure Layer.

In the context of NVE-based networks, examples of NMR are libraries such as the Vyatta Remote Access API (VYATTA, 2013) and Libvirt (BOLTE et al., 2010). The Vyatta Remote Access API allows conducting network management operations on the Vyatta Virtual Router. In turn, Libvirt enables to manage different virtualization servers (*e.g.*, Citrix XenServer and VMware ESX) and virtual networks using bridging.

3.4.2 Adaptation layer

Figure 3.7 presents the Adaptation Layer. This layer is responsible for hiding the complexity and heterogeneity of the Managed Resources Layer by a collection of SMRS called Mediator Bus. SMRS is in charge of grouping, integrating, and homogenizing one or more SMR involved in the investigation and resolution of nmsits. The interaction between SMRS and SMR (*i.e.*, NMR, WMR, AMR, OpR, and NSR) internally occurs in a Wrapper accessed via UWrapper and depends on protocols (*e.g.*, SNMP, SOAP, HTTP, OpenFlow, and Proprietary) provided by network vendors for managing their solutions.

The Wrappers are services based on the Representational State Transfer (REST) architectural model. In REST (FIELDING; TAYLOR, 2002), services are represented by URIs. These URIs are invoked through HTTP(S) requests, such as GET and POST. The replies of services are HTTP(S) responses, such as 200 OK and 404 Not Found. Therefore, in the proposed architecture, the collection of SMRS provides a Mediator Bus in which the communication is based on the request-response model of HTTP(S). This bus enables the interaction between the layers of Adaptation and Composition.

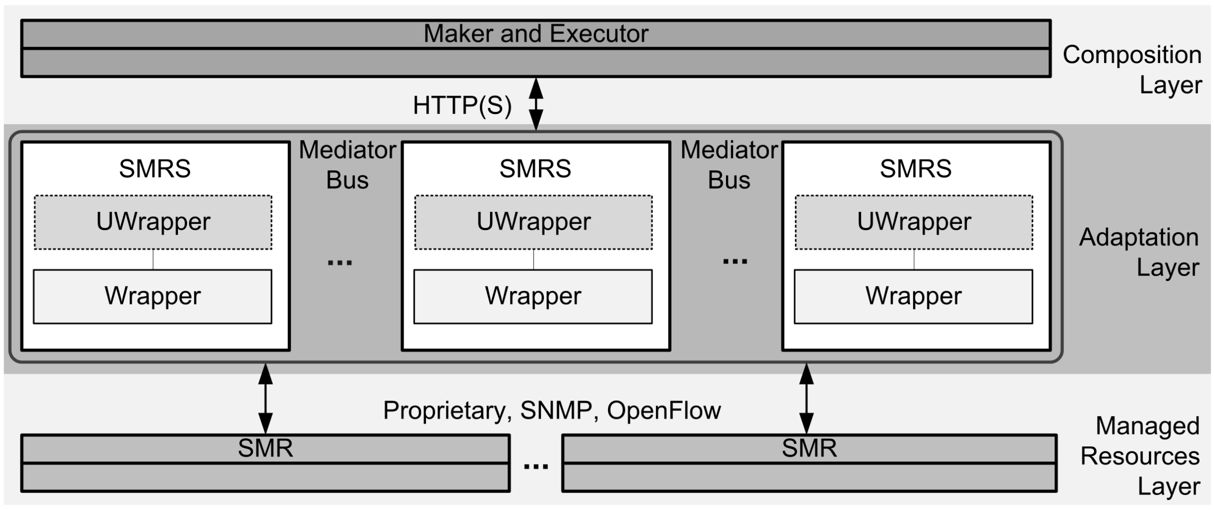


Figure 3.7 – Adaptation layer

The Adaptation Layer responds to HTTP(S) requests from the Composition Layer as follows. First, the requests are targeted to UWrappers that are URIs pointing to Wrappers. Second, Wrappers invoke one or more SMR. Third, SMR carries out the requested functionalities. Fourth, Wrappers receive responses from SMR. Fifth, Wrappers encode SMR results on JSON data and put such data on HTTP-responses. Sixth, Wrappers send their HTTP-responses to the Composition Layer.

An example of UWrapper is $http : //MashmentSys/Wrapper/Beacon/getSwitches$ that offers to the Composition Layer, a list of switches handled by a Beacon OpenFlow Controller. The Wrapper, pointed by the afore exemplified UWrapper, provides the switches list by the next JSON structure: $[\{IPCTRL : ipCtrl, LIST : [\{IDSWITCH : id_1, IPSWITCH : ip_1\}, \{IDSWITCH : id_n, IPSWITCH : ip_n\}]\}]$. In this structure, $IPCTRL$ is the IP address of the Beacon Controller and $LIST$ is the corresponding switches list. Such list is formed by a set of identifiers ($IDSWITCH$) and IP addresses ($IPSWITCH$) of switches.

Another example of UWrapper is $http : //MashmentSys/Wrapper/serv/getInf$. By means of this UWrapper, the Adaptation Layer offers to the Composition Layer, general information about servers that host virtual network elements. After invoking one or more SMR (*e.g.*, Libvirt), the Wrapper - pointed by the above exemplified UWrapper - returns general server information by the next structure encoded in JSON: $[\{IDSERV : id, NAME : name, OS : os, STATE : st, MEMORY : mem, CPU : ncpus\}]$. In this structure, $IDSERV$, $NAME$, OS, $STATE$, $MEMORY$, and CPU represent the identifier, the name, the operating system, the state (*e.g.*, running, turned off, and interrupted), the available memory, and the number of processors of the hosting server, respectively.

Regarding the Adaptation Layer is relevant to highlight, first, Network Administrators never access it directly. This access is always conducted by the layers of Presentation and Composition. Second, the Resource Creator is responsible for providing SMRS by using different development environments. Since these environments are out of the scope of the Mashment System Architecture, they are not presented in this thesis.

3.4.3 Composition layer

In order to support the building up of static and dynamic mashments, the Composition Layer consumes services from the Adaptation Layer and offers services to the Presentation Layer. The Composition Layer is architecturally formed by the Mashment Maker, Mashment Executor, and Dispatcher. These architectural elements are described below.

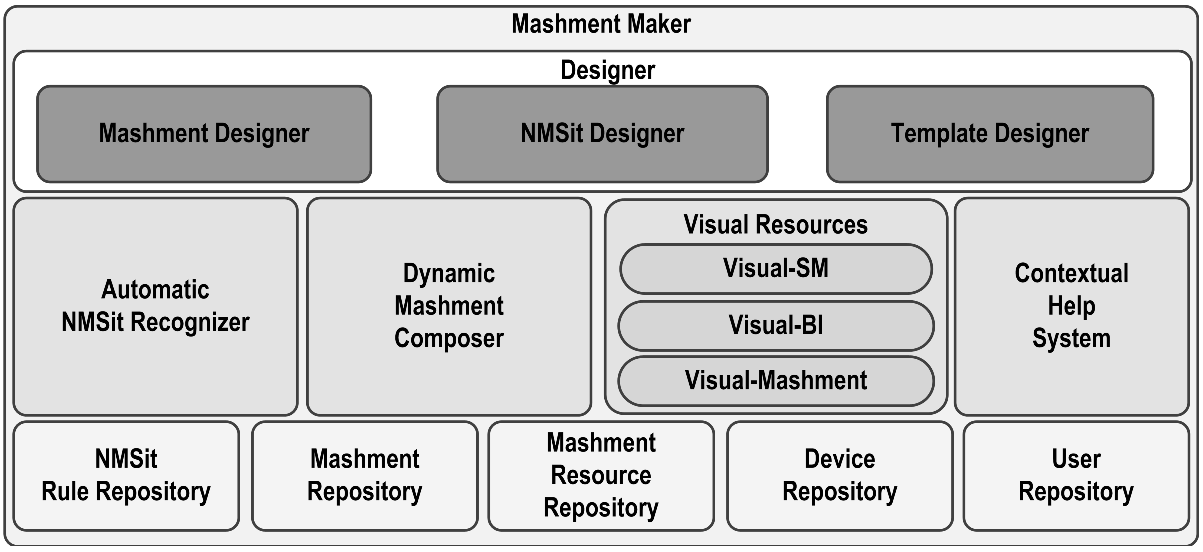


Figure 3.8 – Maker modules

From a high-level point of view, the Mashment Maker offers two functionalities: (*i*) it allows Mashment Creators and Network Administrators to create mashments (*i.e.*, static composition of mashments); and (*ii*) it automatically generates mashments (*i.e.*, dynamic composition of mashments). The Maker (see Figure 3.8) provides its functionalities by using the following modules: User Repository, Device Repository, NMSit Rule Repository, Mashment Repository, Mashment Resource Repository, Automatic NMSit Recognizer, Dynamic Mashment Composer, Visual Resources, Designer, and Contextual Help System (CHS).

Mashment Maker - modules for overall functioning. The modules supporting the general functioning of the Maker are the User Repository, Device Repository, Mashment Resource Repository, Mashment Repository, and CHS. The User Repository stores data of Network Administrators and Mashment Creators. Such a data is used to conduct the access control to the Maker.

The Device Repository stores information about capabilities of devices in which mashments can be presented. In this way, this repository assists in the presentation of mashments on multiple terminals, such as smartphones and tablets. Examples of repositories useful to carry out the Device Repository are the User Agent Profile (MATSUYAMA et al., 2004) and the Wireless Universal Resource File (SALOMONI et al., 2007).

The Mashment Resource Repository stores metadata that describes and points functionalities offered by SMRS (*i.e.*, NMRS, WMRS, AMRS, OpRS, and NSRS). The metadata of a SMRS is: $[\{IDRES : idres, RESNAME : resname, OPERATION : [\{op_1, op_n\}]\}]$. Here, $IDRES$ and $RESNAME$ are the unique identifier and the name of the resource, respectively. In turn, $OPERATION$ is the collection of operations offered by the resource. An op_n is: $[\{OPNAME : opname, PARAM : [\{par_1, par_n\}], PROD : produce\}]$. Where, $OPNAME$ is the name of the operation, $PARAM$ is the set of parameters need to invoke such operation, and $PROD$ is the data type that the operation returns/produces.

The following metadata, for instance, describes two operations (*i.e.*, get the list of virtual switches and get the statistics of a virtual switch) provided by a specific NMRS to retrieve situational management information from a NVE (*i.e.*, a virtualized OpenFlow-based environment): $[\{IDRES : /path_1/, RESNAME : nmrs_1, OPERATION : [\{OPNAME : SwitchList, PARAM : [\{IPCTRL : ipcontrol, PORT : port, USER : user, PWD : pwd\}], PROD : json\}, \{OPNAME : SwitchStatistics, PARAM : [\{IDSWITCH : ids, USER : u, PWD : p\}], PROD : json\}]\}]$.

The Mashment Repository stores metadata of mashments. This metadata is: $[\{IDMASH : id, MASHNAME : name, \delta : [\{delta\}], NMSIT_{addr} : [\{nmsit_{addr}\}]\}]$. Here, $IDMASH$, $MASHNAME$, $NMSIT_{addr}$, and δ are the unique identifier, the friendly name, the set of nmsits addressed, and the execution flow of the mashment, respectively. In turn, δ is: $[\{IDD : idd, RES : [\{res_1\}, \{res_n\}], CONN : [conn_1, conn_n]\}]$. Where, IDD is the unique indentifier of δ, RES is the set of resources forming the mashment, and $CONN$ is the set of logical connections/links created among these resources for handling $NMSIT_{addr}$.

Connections define the propagation of data between resources by specifying which outputs of a resource are supplied to inputs of other one. A $conn_n$ is given by: $[\{IDC : idc, SRC : [\{idresS, idparS\}], DES : [\{idresD, idparD\}]\}]$. Where, IDC is the unique identifier, SRC is the source, and DES is the destination of the connection, respectively. In turn, SRC and DES are represented by the identifiers of the resource ($idresS$ and $idresD$) and the parameter ($idparS$ and $idparD$) connected.

CHS is in charge of providing contextual guidance to support the selection, configuration, and combination of resources available in the Mashment Maker. Contextual guidance means that help information is presented in accordance to current action being performed by Network Administrators and Mashment Creators. Furthermore, CHS is responsible for offering to these stakeholders general guidelines about functioning the Maker.

Mashment Maker - modules for static composition. The Network Administrator and the Mashment Creator can carry out static composition of mashments by using the Designer and the Visual Resources exposed on it. Visual Resources represent to GUI, SMRS, and even mashments in a high-level abstraction, aiming to diminish the complexity perceived and the time required by Administrators and Creators to build up new mashments or enhance existing ones. The metadata of Visual Resources is stored in the Mashment Resource Repository and follows the next structure: $[\{IDVISRES : idvisres, TYPE : type, VISRESNAME : name, IDRES : idrepres\}]$. Where, $IDVISRES$, $TYPE$, and $VISRESNAME$ are the identifier, the type, and the name of the Visual Resource, respectively. In turn, $IDRES$ is the identifier of the resource represented by the Visual Resource.

There are three types of Visual Resources (see Figure 3.9): Visual-SM, Visual-BI, and Visual-Mashment. Visual-SM represents SMRS in a graphic way. There are five types of Visual-SM: *(i)* Visual-NMRS (*e.g.*, a box offering management functionalities of a Vyatta Virtual Router) represents NMRS, *(ii)* Visual-WMRS (*e.g.*, a box providing functional features of the RRDTool) depicts WMRS, *(iii)* Visual-AMRS (*e.g.*, a box supplying services of the Management Traffic Analyzer) represents AMRS, *(iv)* Visual-MO (*e.g.*, a box providing a dashboard that hides the collection, correlation, and fusion of monitoring information from heterogeneous NOS) depicts OpRS; and *(v)* Visual-NSRS (*e.g.*, a box offering functionalities of the Apache Camel) represents NSRS.

Visual-BI represents basic GUIs that are useful to define the composite and advanced GUIs of mashments. For instance, the GUI of a mashment can be created by inserting network traffic images built using JavaScript Charts into a sandbox implemented with YUI Yahoo. Finally, regarding the Visual Resources is important to note that after a mashment has been created, it is represented as a Visual-Mashment. Such representation aims to facilitate the enhancement, improvement, and launching of mashments.

The Designer allows Network Administrators and Mashment Creators to develop and launch mashments. To achieve this goal, the Designer supplies the Mashment Designer (see Figure 3.9): (*i*) it provides the Dragging-and-Dropping service to select Visual Resources (*i.e.*, to define R_{used}), (*ii*) it offers the Wiring service to combine Visual Resources (*i.e.*, to create δ), (*iii*) it uses the Helping service to invoke CHS that offers guidance about the selection, configuration, and combination of Visual Resources (*i.e.*, to define R_{conf}), (*iv*) it provides the Launching service to invoke the Mashment Executor; and (*v*) it offers the services Saving, Loading, and Deleting that aim to facilitate the reuse, tuning, and enhancement of mashments. In particular, Saving enables to write metadata of mashments in the Mashment Repository. Loading is in charge of reading such metadata and exposing Visual-Mashments. Deleting allows removing the metadata of mashments. It is important to highlight that the above services of the Mashment Designer are intended to allow Network Administrators to customize and improve their workspace when addressing nmsits.

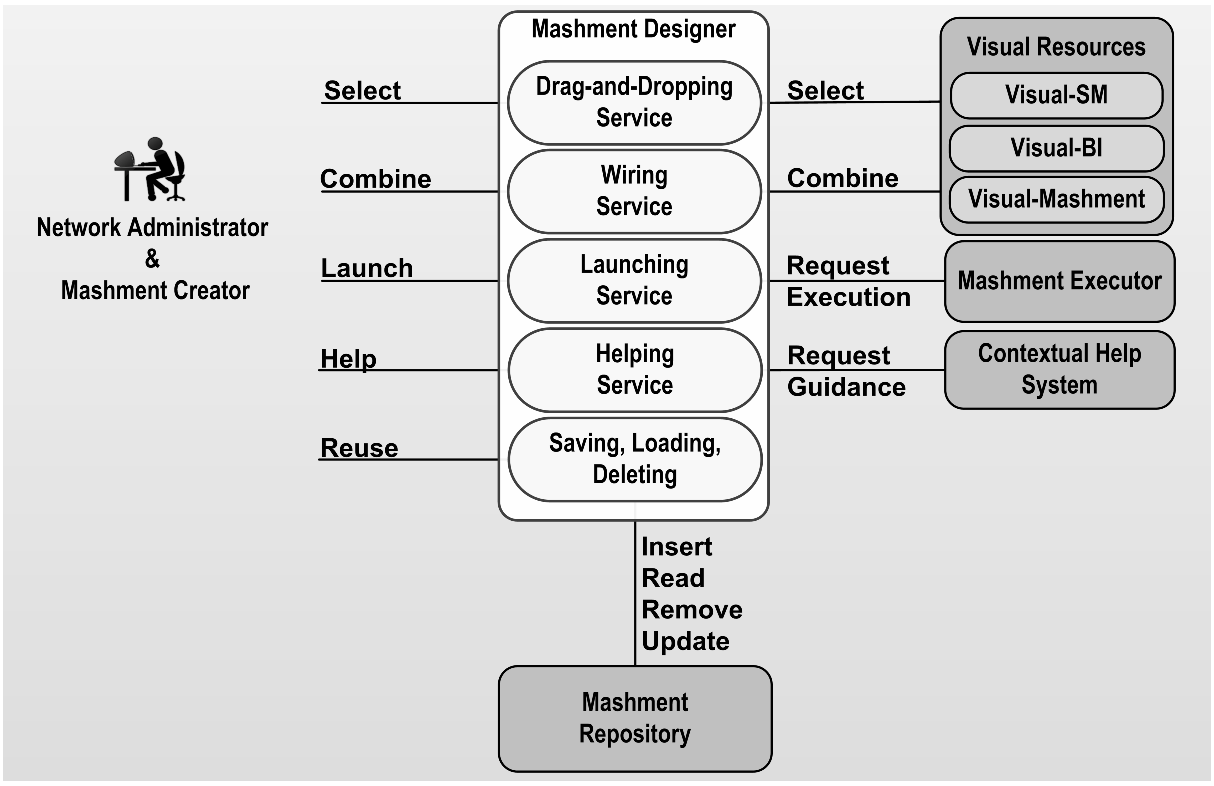


Figure 3.9 – Designer services for static mashments

Mashment Maker - modules for dynamic composition. The automatic generation of mashments operates in two phases: (*1*) $when < nmsits >$ - mechanism for automatic recognition of nmsits; and (*2*) $then < mashments >$ - mechanism for dynamic composition of mashments. In a broad sense, these phases involve the following functionalities: (*i*) the definition of nmsits that will be recognized, (*ii*) the recognition of nmsits, (*iii*) the specification of composition templates for generating mashments; and (*iv*) the dynamic composition of mashments that will face nmsits. These functionalities are provided by the modules: NMSit Rule Repository, Mashment

Repository, Designer, Automatic NMSit Recognizer, and Dynamic Mashment Composer.

The NMSit Rule Repository stores nmsit patterns that are instances of the nmsit conceptual model described in the Subsection 3.1.1. Each one of these patterns defines a collection of entities, attributes, and constraints in which constraints are conditions (*i.e.*, rules) defined in attributes of entities for detecting nmsits. In this way, this repository stores nmsits that will be recognized and later handled by dynamic mashments. An snippet of rule encoded in JSON is: $[\{Router, [\{PkDrop, >= 5\%\}, \{PktError, >= 3\%\}]\}]$. Another snippet of rule is given by: $[\{VSwitch, [\{Processor, < 5\%\}, \{Mem, < 5\%\}, \{PktDrop, >= 7\%\}, \{PktError, >= 5\%\}]\}]$.

The Mashment Repository in addition to store metadata of mashments also stores metadata of composition templates. The composition templates are skeletons useful to dynamically compose mashments. A composition template is: $[\{IDTEMPLATE : id, NAMESIT : namesit, \delta : delta\}]$. Where, $IDTEMPLATE$ is the unique identifier of template and δ is a predefined workflow (*i.e.*, an investigative and/or resolutive plan formed by combining mashupable situational resources) to deal with one or more nmsits identified by $NAMESIT$ (see Figure 3.1).

The Designer besides to support the development and launching of mashments allows Mashment Creators to define: (*i*) nmsits that will be recognized/detected; and (*ii*) composition templates that will be used for generating mashments and, ultimately, for addressing recognized nmsits. In this way, the Designer is also responsible for supporting the management (*i.e.*, create, read, update, and delete) of metada of nmsits and composition templates by handling the NMSit Rule Repository and the Mashment Repository, respectively (see Figure 3.10). Specifically, the NMSit Designer is to manage nmsits patterns, the Template Designer is to manipulate templates, and the above described Mashment Designer aids to enhance mashments.

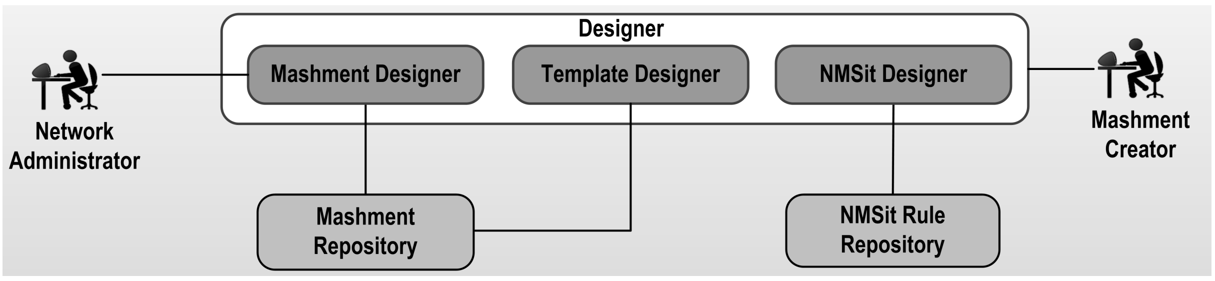


Figure 3.10 – Designer elements for dynamic mashments

The mechanism for automatic recognition of nmsits is conducted by the Automatic NMSit Recognizer (see Figure 3.11). The Recognizer detects when nmsits happen by using the modules: Sensing and Matching Mechanism. The Sensing is in charge of retrieving network management information by the Mediator Bus and delivering this information as a streaming to the Matching Mechanism. Such retrieving of information depends on communication (pull-based or push-based) provided by SMRS forming the Mediator Bus.

The Matching Mechanism recognizes nmsits as follows, first, it reads rules (nmsit patterns) from the NMSit Rule Repository. Second, it obtains information about managed networks and their devices by Sensing. Third, it constantly conducts a matching operation (*i.e.*, in a general way, comparison of samples with patterns) among network information and rules to determine if one or more nmsits happen. This matching operation can be carried out by using RETE (DOORENBOS, 1995) (HILL, 2003) and PHREAK (BROWNE, 2009) that are recognition pattern algorithms currently offered by rule-based engines, such as JBOSS Drools and JESS. Fourth, it defines $NMSIT_{addr}$ and invokes the Dynamic Mashment Composer every time a nmsit is detected (*i.e.*, there is a matching).

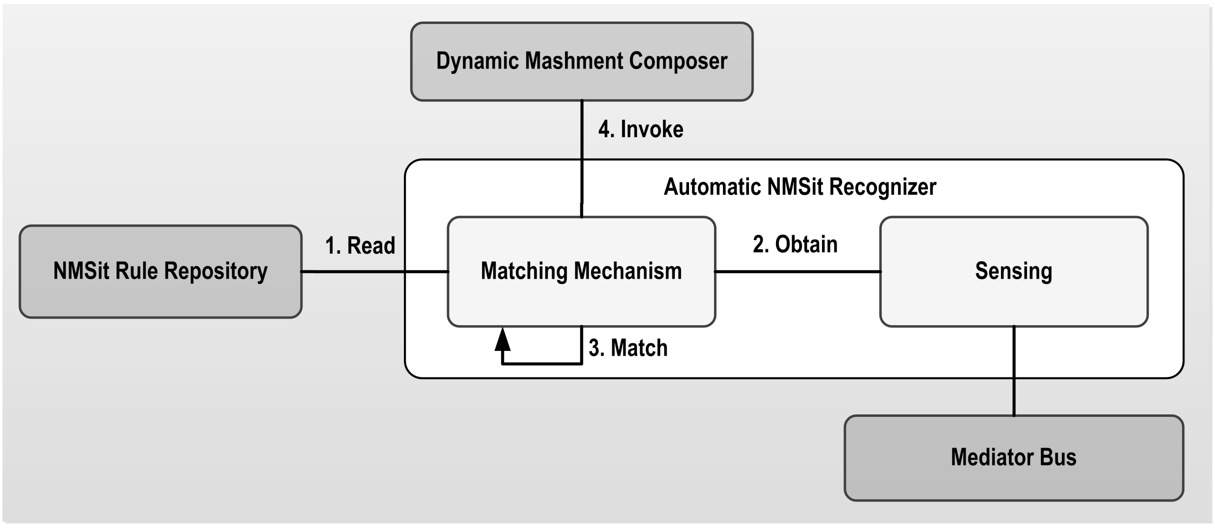


Figure 3.11 – Automatic recognizer of nmsits

The mechanism for dynamic composition of mashments is conducted by the Dynamic Mashment Composer that automates the tasks Select, Configure, and Combine. These tasks are part of the proposed process to develop and launch mashments (see Subsection 3.3.1). Summarizing, Select is to define the resources (*i.e.*, RES formed by SMRS and Visual Resources) to be used to generate a mashment. Configure is to provide all functioning settings of selected resources. Combine is to define how a particular mashment will deal with a specific $NMSIT_{addr}$ by creating connections (*i.e.*, $CONN$) among selected and configured resources.

The Dynamic Mashment Composer (see Figure 3.12) is formed by: Selector and Generator. 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 a composition template, which includes a specific 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 with the linguistic similarity algorithm (GRIGORI et al., 2010) that is based on NGram (ANGELL; FREUND; WILLETT, 1983), CheckSynonym (MILLER, 1995), and ElementMatch (PATIL et al., 2004). Fifth, it invokes the Generator.

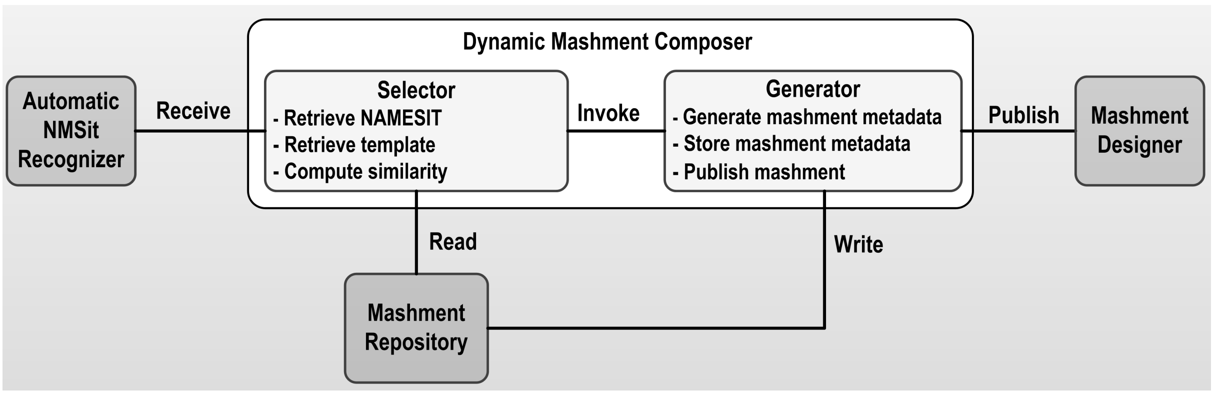


Figure 3.12 – Dynamic composer of mashments

The Generator functions as follows: (*i*) it receives δ and $NMSIT_{addr}$ from the Selector, (*ii*) it generates a mashment (*i.e.*, an instance of mashment metadata) by customizing the selected δ with the information of entities involved in nmsits as well as their attributes and constraints, (*iii*) it stores in the Mashment Repository the generated mashment by writing the corresponding metadata; and (*iv*) it publishes such a mashment in the Mashment Designer as a Visual-Mashment. It is to noteworthy that the Network Administrator can further improve and run the generated mashment by using the Mashment Designer.

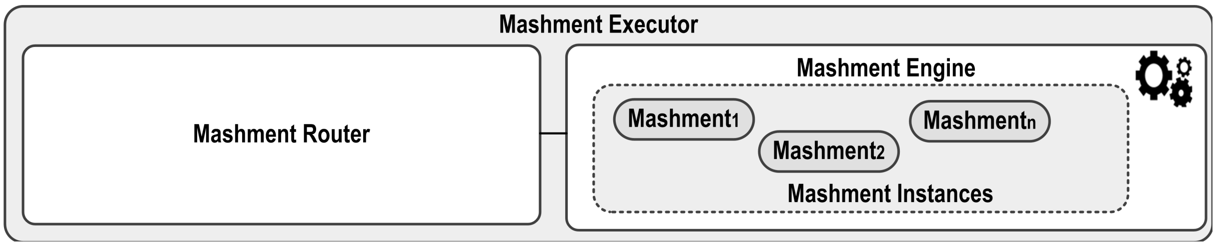


Figure 3.13 – Executor modules

In addition to the Mashment Maker, the Composition Layer is formed by the Mashment Executor and the Dispatcher. The Executor (Figure 3.13) is made up of the modules: Mashment Router and Mashment Engine. The Router is responsible for coordinating the execution of δs that are the core of mashments. Thus, the Router on runtime: *(i)* it receives mashments invocations from the Engine, which means that the Router is called by the Engine to select a mashment to service an initial request, *(ii)* it selects and links multiple resources (including mashments into other mashment) to attend invocations, by reading information from the repositories of Mashments, Resources, and Users; and *(iii)* it calls the Engine to request the instantiation of mashments and their elements. In the literature of service composition, software entities providing similar functionalities to the Router are referred as to Application Routers (CHEUNG; PURDY, 2008) and Web Service Orchestrators (ESCOBEDO et al., 2010).

The Mashment Engine is a lifecycle manager responsible for creating, deleting, and caching instances of mashments. When initial requests to execute mashments are received from Web and/or Mobile Clients, the Engine invokes the Router. Afterwards, the Engine waits indications from the Router in order to create, cache, or delete instances of mashments and their resources. These resources can be one or more NMRS, WMRS, AMRS, OpRS, NSRS, and their corresponding Visual Resources. Furthermore, when Network Administrators and Mashment Creators need to develop and request the execution of mashments by Web Clients, the Engine is in charge of creating, caching and deleting instances of the Mashment Maker.

The Dispatcher is responsible for adapting and sending content (*i.e*, mashments and the Mashment Maker) to the Presentation Layer (see Subsection 3.4.4). When requests to display content arrive from devices Client (*e.g.*, tablets, laptos, and smartphones) via the Engine, the Dispatcher: (*i*) it reads the HTTP-header of each one of these requests in order to identify devices in which content will be presented, (*ii*) it queries the Device Repository to establish the capabilities of devices that carry out these requests, (*iii*) it adapts content by using such devices capabilities; and (*iv*) it sends adapted content to the Presentation Layer.

3.4.4 Presentation layer

Figure 3.14 depicts the Presentation Layer that communicates with the Composition Layer by JSON/HTTP(S). This layer is responsible for executing and presenting, in the client-side, GUIs of the Mashment Maker and mashments; called Mashment Maker GUI and Mashments GUI, respectively. It is important to highlight that the Mashment Maker GUI includes the GUIs of the Mashment Designer, NMSit Designer, and Template Designer.

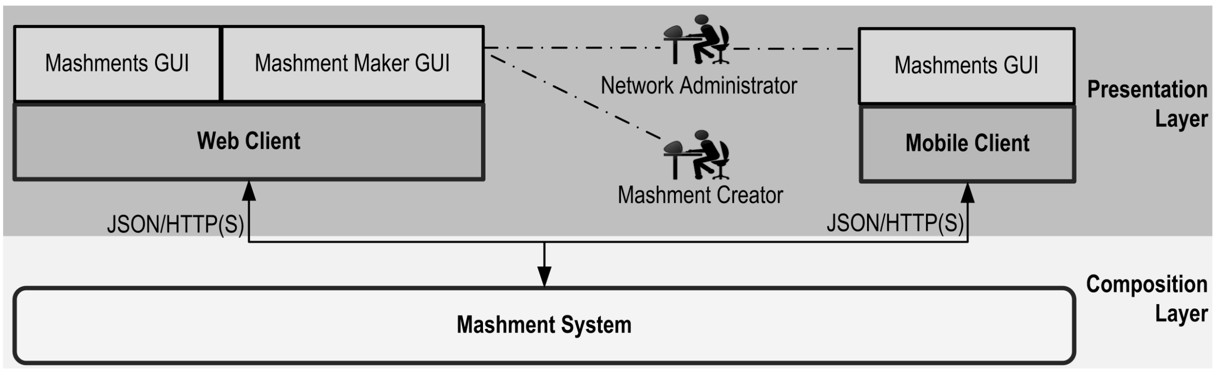


Figure 3.14 – Presentation layer

The Presentation Layer is formed by the Web Client and the Mobile Client. The Web

Client has a runtime environment in charge of presenting the Mashment Maker GUI and the Mashments GUI to Network Administrators and Mashment Creators. The Mobile Client has a runtime responsible for showing the Mashments GUI to Network Administrators. As the Mashment Maker GUI is not enabled to be presented in the Mobile Client, mashments cannot be developed by using this type of client.

3.5 Final remarks

This chapter presented a groundbreaking approach that uses the SM discipline and the mashup technology for carrying out network management in an effective way. The contributions achieved in this thesis with the introduced approach can be divided into: conceptual and specific ones.

- Conceptual contributions are:
    - The nmsit concept that introduces a novel way to characterize unexpected, dynamic, and heterogeneous situations in the network management domain by using SM.
    - The mashment concept that presents how to employ the SM discipline and the mashup technology for carrying out network management.

- Specific contributions are:
    - The Mashment Ecosystem that presents the resources, stakeholders, software entities, activities, and interactions related to handling nmsits.
    - The process to develop and launch mashments that introduces how to deal with nmsits by conducting a simple set of high-level tasks (*i.e.*, Select, Configure, Combine, Launch, and Tune).
    - The model of complexity and time-consuming that permits assessing the complexity as well as the consuming of time of addressing nmsits with and without the mashment-based approach.
    - The Mashment System Architecture that supports the making of both the ecosystem and the process aforementioned and, thus, the making of the mashment concept too.
    - The mechanism for nmsits automatic recognition that presents how to detect nmsits on the fly by using rules and matching algorithms.
    - The mechanism for mashments dynamic composition that introduces how to generate mashments in an automatic way by using composition templates.

4 EVALUATING THE MASHMENT-BASED APPROACH

This chapter provides an extensive evaluation about the feasibility of using the mashment-based approach for effectively carrying out network management. Specifically, this chapter presents the Reference Implementation and three case studies carried out for evaluating the addressing of specific nmsits with and without using the proposal introduced in this thesis.

4.1 Reference implementation

The Reference Implementation of the mashment-based approach is formed by: Managed Resources, Mashment System Server, Mashment Maker, and Runtime Environments. The following subsections describe these elements.

4.1.1 Managed resources

In the Reference Implementation, the Managed Resources are elements of virtual nodes, SDN-based networks, and the corresponding SMR. The elements of virtual nodes are Xen servers, VirtualBox servers, virtualized Open vSwitches, and Linux-based virtual machines. Xen (BARHAM et al., 2003) is a virtualization platform, distributed by Citrix Systems, Inc., that directly executes on server hardware without requiring an operating system (*i.e.*, standalone monitor mode). Xen was used as Host Computer System (supplier of physical resources) and Virtualization Layer (lifecycle manager of Hosted Virtual Computer Systems). VirtualBox (WATSON, 2008) is a virtualization system, distributed by Oracle, Inc., that runs on operating systems (*i.e.*, hosted monitor mode) such as Windows, Linux, and Mac OS X. Linux Debian was used as Host Computer System of VirtualBox that, in turn, was used as Virtualization Layer. It is important to note that in virtual nodes, virtual network elements (*e.g.*, virtualized Open vSwitches) and virtual machines (*e.g.*, Linux-based VMs) are considered as Hosted Virtual Computer Systems.

The Reference Implementation includes the following elements of SDN-based networks: (*i*) Floodlight (FLOODLIGHT, 2013) that is an OpenFlow Controller developed in the Java Language and deployed as Hosted Virtual Computer System, (*ii*) Beacon (ERICKSON, 2013) that is an OpenFlow Controller implemented in the Java Language and deployed as Hosted Virtual Computer System, (*iii*) POX (POX, 2013) that is an OpenFlow Controller developed in the Python Language and deployed as Hosted Virtual Computer System, (*iv*) Open vSwitch that is a Hosted Virtual Computer System handled by Floodlight, Beacon, and POX, (*v*) Virtual Links that communicate Open vSwitches, (*vi*) Flows that contain rules to control the communication in OpenFlow-based networks; and (*vii*) Mininet (LANTZ; HELLER; MCKEOWN, 2010) that is a software used to emulate OpenFlow networks and deployed on VirtualBox.

The Managed Resources Layer of the Reference implementation includes the following SMR: (*i*) XenSDK that enables the comprehensive remote management (including Hosted Virtual Computer Systems) of Xen servers, (*ii*) VirtualBox Web Service that lets to thoroughly manage, in a remote way, VirtualBox servers (including Hosted Virtual Computer Systems); and (*iii*) Floodlight Web Service, Beacon Web Service, and POX Web Service that permit the remote management of OpenFlow-based networks (including flows, links, and Open vSwitches) controlled by Floodlight, Beacon, and POX, respectively. In the Reference Implementation, all SMR were only used for monitoring purposes.

4.1.2 Mashment system server

The Mashment System Server deploys the Adaptation Layer and several elements of the Composition Layer. The Adaptation Layer is formed by XenService, VBoxService, FloodlightService, BeaconService, and POXService. These services implement SMRS (specifically NMRS) for XenSDK, VirtualBox Web Service, Floodlight Web Service, Beacon Web Service, and POX Web Service. Each SMRS was created by using RESTful that is a REST implementation on the Java Language. In particular: (*i*) XenService was implemented by using the XenSDK API 6.0, (*ii*) VBoxService was developed with the VirtualBox SDK API 4.1; and (*iii*) FloodlightService, BeaconService, and POXService were built by using the Java Jersey API 2.3 and the Java Socket API 1.0. It is important to note that by developing SMRS the complexity of Managed Resources and their underlying technologies is hidden for Network Administrators and Mashment Creators.

The Mashment System Server deploys the following elements of the Composition Layer: Automatic NMSit Recognizer, Dynamic Mashment Composer, Mashment Executor, Dispatcher, Mashment Resource Repository, Mashment Repository, and NMSit Rule Repository. The Automatic NMSit Recognizer is a Java-based application that uses for detecting nmsits the implementation of the PHREAK algorithm offered by the JBOSS Drools engine. It is relevant to point out that, first, nmsit patterns are defined by the Mashment Creator in the NMSit Designer and, thereupon, stored in JSON format in the NMSit Rule Repository. Second, as nmsit patterns are in JSON, before being passed to matching, they are translated to DRL by the Recognizer.

The Dynamic Mashment Composer is a Java-based application that customizes the composition templates defined in the Template Designer by Mashment Creators. It is important to stand out that, first, these templates are stored in JSON format in the Mashment Repository. Second, once a template has been customized, it is stored as a mashment metadata in the Mashment Repository and automatically exposed in the Mashment Maker as a Visual-Mashment.

The Executor is a Web application, based on Java Servlets and AJAX, that implements the Mashment Router as well as the Mashment Engine. Internally, the Executor is in charge of invoking by HTTP, SMRS (*e.g.*, FloodlightService, POXService, and XenService) needed to build up and execute static and dynamic mashments. The Executor outputs are offered to the Dispatcher by means of JSON objects transported in HTTP responses.

The Dispatcher is a Web application, based on AJAX and Java Servlets, that sends for the Runtime Environments (see Subsection 4.1.4) the result of compositions conducted by the Executor. 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 mashments on smartphones, tablets, desktops, and laptops.

The Mashment Resource Repository, Mashment Repository, and NMSit Rule Repository were implemented in a unique database by using a MysqlServer. These repositories are hereinafter named MashmentDB. The metadata of every mashment and resource is stored into the MashmentDB for promoting their reuse and allowing the extension and improvement of the Reference Implementation. It is relevant to highlight that the metadata of every nmsit pattern is stored in the MashmentDB using JSON and not a specific rule language (*e.g.*, DRL) for facilitating the change of rules engine and, thus, of matching algorithm too.

4.1.3 Mashment maker prototype

Figure 4.1 depicts the GUI of the Mashment Maker Prototype that is formed by Visual Resources, Buttons (*New*, *Load*, *Save*, *Delete*, and *Run*), CHS (*Help*), and Designer. The Visual Resources, Buttons, and CHS are Web components developed by using Cascading Style Sheets (CSS) and JavaScript. Some Visual Resources implemented are *Beacon*, *Floodlight*, *POX*, *OF Monitor*, *Virtual Box Server*, *Xen Server*, *Monitoring Panel*, *Integrator*, *Open vSwitch*, and *Switch Traffic Grapher*. *Beacon*, *Floodlight*, and *POX* are the visual representation (*i.e.*, a high-level encapsulation of network technologies) of controllers Beacon, Floodlight, and POX, respectively.

OF Monitor is a collection of views and operations used to present, in a graphic way, information about networks (including flows, switches, links, and traffic) handled by one or more of the above mentioned OpenFlow controllers. Every operation (*e.g.*, traffic of an OpenFlow-enabled switch) of *OF Monitor* can easily be applied to controllers by drawing visual connections in the Designer.

Virtual Box Server and *Xen Server* are the visual representation (*i.e.*, a high-level encapsulation of system virtualization solutions) of Xen server and VirtualBox server, respectively. *Monitoring Panel* is a collection of views and operations used for graphically presenting information about Host Computer Systems and guests (*i.e.*, a common term used for referring to Hosted Virtual Computer Systems) running on them. Every operation of *Monitoring Panel* (*e.g.*, present statistics) can easily be applied to virtual nodes based on Xen and VirtualBox by drawing visual connections in the Designer. *Integrator* integrates Visual Resources (*i.e.*, aggregation in the GUI level) and can readily be applied, for instance, to attach the outputs of *OF Monitor* and *Monitoring Panel*.

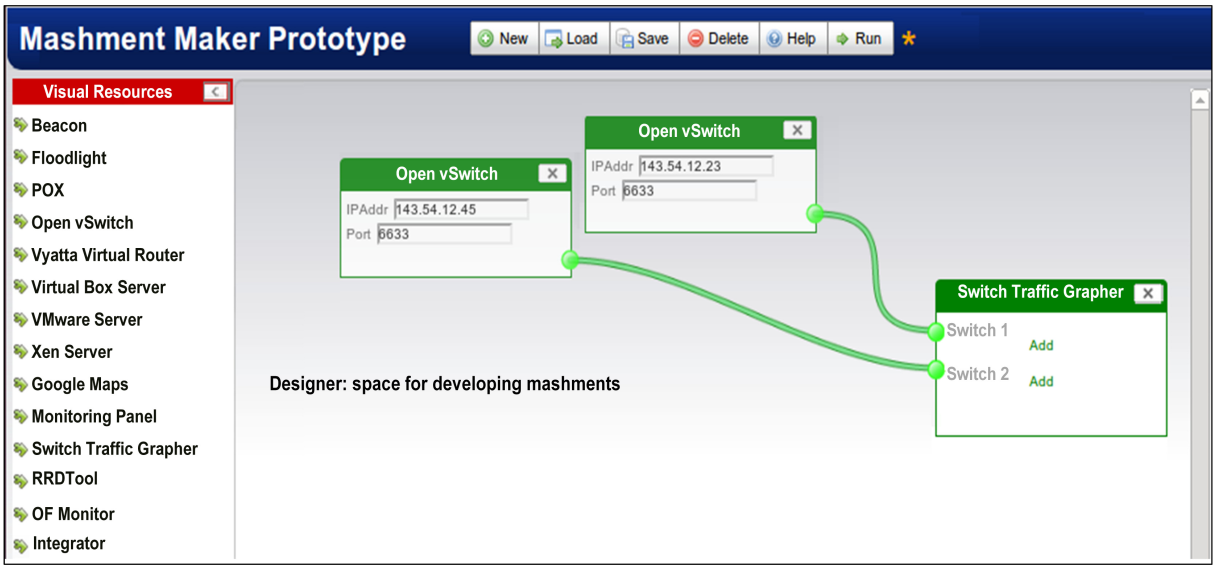


Figure 4.1 – User interface of the Mashment Maker

Open vSwitch is a visual resource that illustrates the high-level encapsulation of a specific network element. *Switch Traffic Grapher* is a collection of views and operations used for graphically presenting information about switches. Similarly to *OF Monitor*, the operations (*e.g.*, present percentages of transmitted and received packages) of *Switch Traffic Grapher* can readily be applied to switches by drawing visual connections in the Designer.

The Designer is a Web application that includes the Mashment Designer, Template Designer, and NMSit Designer. This application was built by using YUI 2.7 and WireIt 0.5. YUI is an open source framework based on CSS and JavaScript. YUI was used to implement the Drag-and-Drop Service. WireIt is a set of open source JavaScript libraries. WireIt was used to implement the Wire Service.

Finally, regarding the Mashment Maker Prototype is important to highlight that, first, all its functionalities are supported by the MashmentDB deployed in the Mashment System Server. Second, the prototype helps in its own extension by assisting the building up and improvement of mashments that are simply carried out by dragging-and-dropping and wiring Visual

Resources. In turn, such extension assists in the upgrade of workspaces used by Network Administrators.

4.1.4 Runtime environments

Every standardized Web browser that supports HTML version 5 and AJAX can be used as Runtime Environment to access and execute: (*i*) the Mashment Maker GUI that can be customized and extended during the development of mashments by Mashment Creators, (*ii*) the Mashments GUI that can be invoked by Network Administrators from the Mashment Maker or directly using URIs; and (*iii*) SMRS (including Wrappers) that can be requested by Resource Creators and Mashment Creators via URIs (*i.e.*, UWrappers), for instance, during the integration of a new OpenFlow controller or a novel system virtualization technology.

It is relevant to stand out that Runtime Environments can exchange JSON data with the Mashment Maker and the Mashment System Server by using a synchronous and/or asynchronous communication model. For instance, when storing mashments, the HTTP synchronous model is used to block the Mashment Maker GUI. Instead, during the performing of mashment operations (*e.g.*, to retrieve and show a Host Computer System list or a Open vSwitches list), the AJAX asynchronous model is used to avoid the blocking of Mashments GUI.

4.2 Case study on SDN

The case study on SDN is formed by a test environment (see Figure 4.2), a nmsit called *NMSit-SDN*, and experiments conducted to evaluate the addressing of such a nmsit with (*i.e., Performance Monitoring Mashment* - PMM) and without (*i.e., Situational Script*) the mashment-based approach. Regarding this case study, it is also important to mention that four metrics are measured in the experiments: complexity, time-consuming, time-response, and network traffic.

Test environment. Every OpenFlow controller (*i.e.*, Beacon, POX, and Floodlight) was executed on a machine with 2.33 Ghz core 2 duo processor, 2 GBytes RAM, and 160 GBytes hard disk. The Mashment System Server, Mashment Maker and MashmentDB 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 virtual Open vSwitches (handled by the above referred OpenFlow controllers) were deployed on a server with 8 GBytes RAM and 3.4 GHz core i7 processor. The user interfaces of the Mashment Maker, PMM, and *Situational Script* were executed on a Client with 2 GBytes RAM and 2.53 GHz core 2 duo processor.

NMSit-SDN. Let's suppose the following nmsit: A Network Administrator needs to investigate/identify which are the Open vSwitches that are causing sudden performance degradation in the OpenFlow-based networks of the test environment. In this way, he/she requires a situational solution that presents, in an integrated, visual, and intelligible way, information about Open vSwitches, links, and flows handled by Beacon, POX, and Floodlight. In order to get such a solution and deal with this nmsit, the Network Administrator tests two options: (*i*) without the proposed approach, creates, launches, and uses the *Situational Script*; and *(ii)* with the proposed approach, develops, launches, and uses PMM by the Mashment Maker. The following sections present the results and analysis of experiments conducted to evaluate such options.

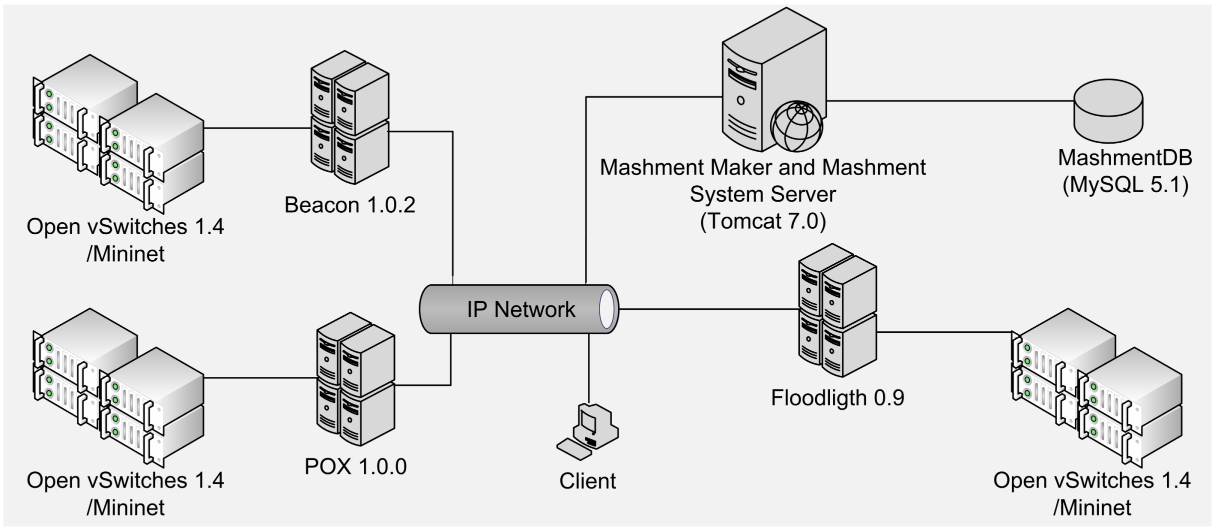


Figure 4.2 – Test environment on SDN

4.2.1 Complexity: results and analysis

Addressing without Maker. To start the evaluation, it is proceeded to measure the complexity of addressing the *NMSit-SDN* when the Network Administrator follows the proposed process to deal with nmsits but does not use the Maker. In a workspace without the Maker, the Network Administrator develops and executes the *Situational Script* that retrieves network traffic information from switches handled by Beacon, POX, and Floodlight. This *Situational Script* presents the retrieved information in a user interface formed by HTML tables and chart images. According the equation 3.1 and considering the no conducting of Tune ($i = j = k = o = 1$), $\zeta_{nomaker} = \zeta_{sel:nomaker} + \zeta_{con:nomaker} + \zeta_{com:nomaker} + \zeta_{lau:nomaker}$.

Select without Maker. The Network Administrator conducts the selection of controller tools (*i.e.*, Beacon Tool, POX Tool, and Floodlight Tool), specific commands of such tools, and visualization tool (*i.e.*, RRDTool) that allow to monitor Open vSwitches. An example of specific command is to retrieve statistics of Open vSwitches handled by Floodlight: $curl\ http : //IPController : port/wm/core/switch/switchId/statType/json$. This selection is no simple because it is not tool-assisted and guidelines about advanced commands of controller and visualization tools are scattered on the Internet. In this way, $\varsigma_m = 2$, $gF = 3$, and $cF = 1$. Using these values in the equation 3.2, $\zeta_{sel:nomaker} = \sum_{m=1}^{4} 2 + (nAvailableResources - 1) * 3 * 1$. Where, considering $nAvailableResources = 14$, $\zeta_{sel:nomaker} = 47$. Such value of available resources was used in order to facilitate the comparison with the Maker prototype.

Configure without Maker. The Network Administrator carries out the configuration of controller tools by providing their corresponding functioning parameters. According the equation 3.3, $\zeta_{con:nomaker} = \varsigma_{con:beacon} + \varsigma_{con:pox} + \varsigma_{con:floodlight} + \varsigma_{con:rrd}$. Since the Network Administrator obtains the configuration information of controller tools from documentation easy to find on the Internet and defines specific statistic commands after conducting additional search, $sourceParameter(statisticCommand) = 5$ and $sourceParameter(login) = sourceParameter(key) = sourceParameter(ip) = sourceParameter(port) = 2$. Furthermore, since the Network Administrator extrapolates the configuration information of RRDTool from documentation simple to find on the Internet, $\varsigma_{con:rrd} = 3$. Using the above values and considering $\varsigma_{con:beacon} = \varsigma_{con:pox} = \varsigma_{con:floodlight}$, $\zeta_{con:nomaker} = 42$.

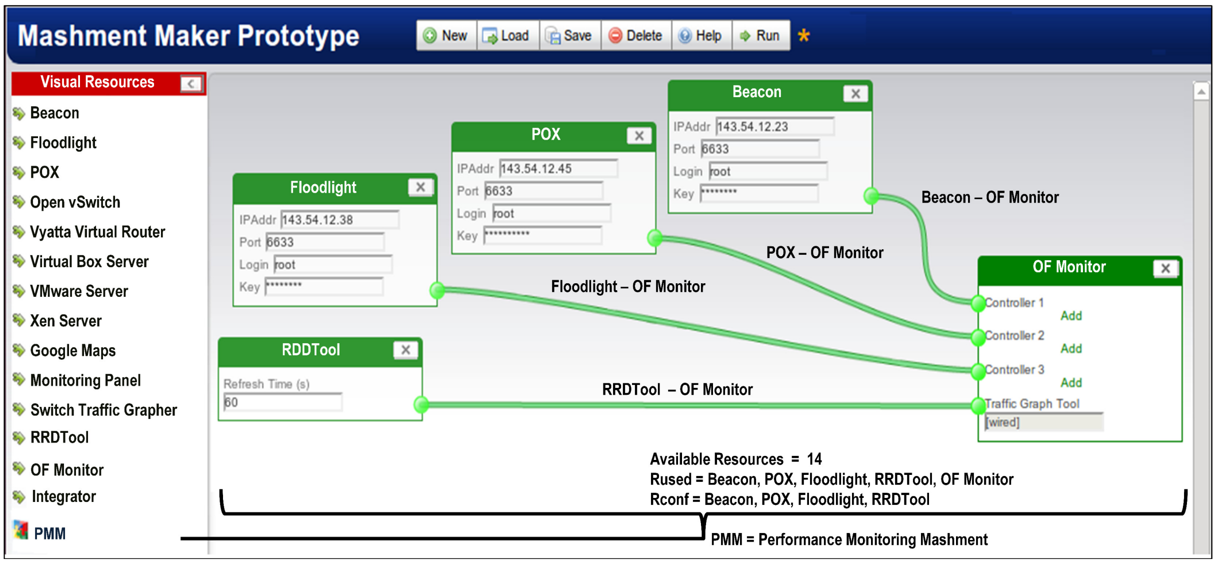
Combine without Maker. The Network Administrator manually creates (*i.e.*, writes programming code) one logical link among each controller tool and RRDTool. Regarding this creation of links, it is to point out that: (*i*) the Network Administrators is responsible for adapting the retrieved data because controller tools, involved in the *NMSit-SDN*, use different data types (*e.g.*, Beacon and POX employs data types based on Java and Python, respectively); and (*ii*) the Network Administrator neither has explicit nor centralized guidelines to develop these links. Therefore, $l = 4$, $\zeta_l = 3$, $goF = 3$, and $coF = 1$. Using these values in the equation 3.4, $\zeta_{com:nomaker} = 21$.

Launch without Maker. As the Network Administrator requests the execution of the *Situational Script* by typing a specific command in a Linux Command Line, $\zeta_{lau:nomaker} = 2$. After launching the *Situational Script*, the Network Administrator is able to find the Open vSwitches involved in the *NMSit-SDN* by analyzing RRDTool images and HTML tables.

Addressing with Maker. Once computed the complexity of facing the *NMSit-SDN* without the Maker, it is proceeded to evaluate the complexity of developing and launching PMM. In a broad sense, in the Maker, the Network Administrator builds and requests the execution of PMM (see Figure 4.3) by dragging-and-dropping, wiring, and clicking Visual Resources and Buttons. The Maker also assists such a process by providing contextual guidelines for the Network Administrator. According the equation 3.1 and considering the no carrying out of Tune, $\zeta_{pmm} = \zeta_{sel:maker} + \zeta_{con:maker} + \zeta_{com:maker} + \zeta_{lau:maker}$.

Select on Maker. The Network Administrator uses the Drag-and-Drop service (*i.e.*, a Maker-assisted way) to select the Visual Resources ($M = 5$) that form PMM. Thus, $R_{used} =$

{*Beacon, POX, Floodlight, RRDTool, OF Monitor*}. Furthermore, to facilitate such selection, the Maker via CHS provides contextual guidance about each available resource. Therefore, $\varsigma_m = 1$, $gF = 2$, $cF = 1$, and $nAvailableResources = 14$. Using these values in the equation 3.2, $\zeta_{sel:maker} = 31$.



Floodlight: IPAddr 143.54.12.38, Port 6633, Login root, Key ********

POX: IPAddr 143.54.12.45, Port 6633, Login root, Key *********

Beacon: IPAddr 143.54.12.23, Port 6633, Login root, Key ********

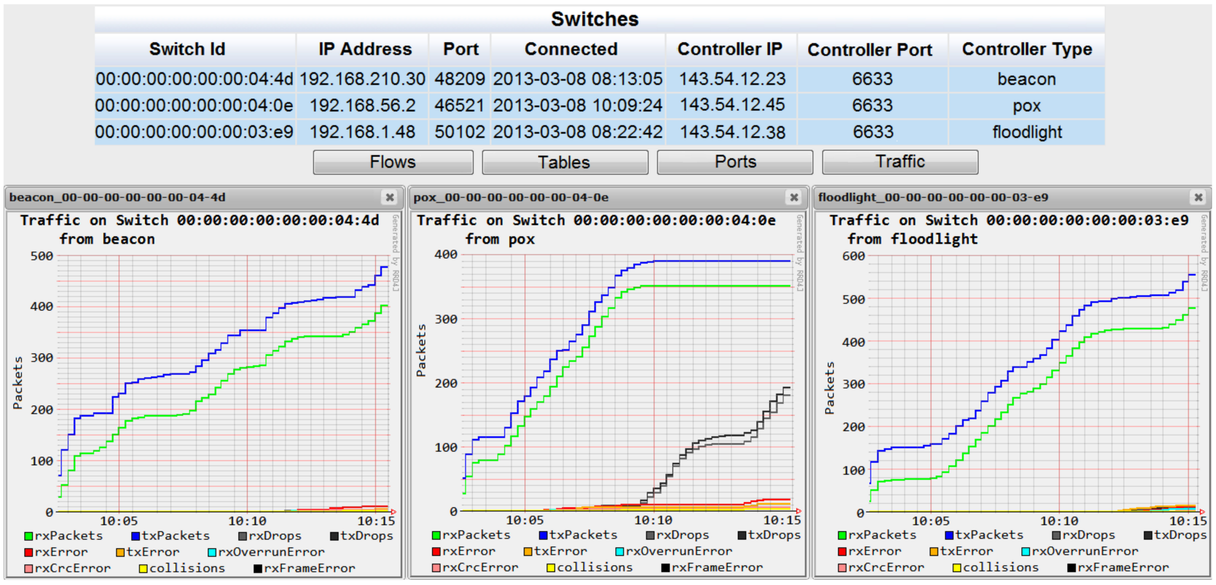
RDDTool: Refresh Time (s) 60

OF Monitor: Traffic Graph Tool [wired]

Available Resources = 14
Rused = Beacon, POX, Floodlight, RRDTool, OF Monitor
Rconf = Beacon, POX, Floodlight, RRDTool

PMM = Performance Monitoring Mashment

Figure 4.3 – PMM - development and launch



Switches

Switch Id	IP Address	Port	Connected	Controller IP	Controller Port	Controller Type
00:00:00:00:00:00:04:4d	192.168.210.30	48209	2013-03-08 08:13:05	143.54.12.23	6633	beacon
00:00:00:00:00:00:04:0e	192.168.56.2	46521	2013-03-08 10:09:24	143.54.12.45	6633	pox
00:00:00:00:00:00:03:e9	192.168.1.48	50102	2013-03-08 08:22:42	143.54.12.38	6633	floodlight

Flows
Tables
Ports
Traffic
beacon_00-00-00-00-00-00-04-4d
Traffic on Switch 00:00:00:00:00:00:04:4d from beacon
pox_00-00-00-00-00-00-04-0e
Traffic on Switch 00:00:00:00:00:00:04:0e from pox
floodlight_00-00-00-00-00-00-03-e9
Traffic on Switch 00:00:00:00:00:00:03:e9 from floodlight
Packets
10:05
10:10
10:15
rxPackets
txPackets
rxDrops
txDrops
rxError
txError
rxOverrunError
rxCrcError
collisions
rxFrameError

Figure 4.4 – PMM - user interface of traffic

Configure on Maker. The Network Administrator configures ($N = 4$) Visual Resources (*i.e.*, $R_{conf} = \{Beacon, POX, Floodlight, RRDTool\}$) by providing the corresponding functioning settings. According the equation 3.3, $\zeta_{con:maker} = \varsigma_{beacon} + \varsigma_{pox} + \varsigma_{floodlight} + \varsigma_{rrd}$. Where, $\varsigma_{beacon} = \varsigma_{pox} = \varsigma_{floodlight} = sourceParameter(login) + sourceParameter(key) + sourceParameter(ip) + sourceParameter(port)$ and $\varsigma_{rrd} = sourceParameter(refreshT)$.

As the Network Administrator obtains configuration guidelines about Visual Resources from the Maker via CHS, $\varsigma_{beacon} = 8$ and $\varsigma_{rrd} = 2$. Using these values, $\zeta_{con:maker} = 26$.

Combine on Maker. The Network Administrator uses the Wire Service (*i.e.*, a Maker-assisted way) to create ($L = 4$) links: *Beacon - OF Monitor*, *POX - OF Monitor*, *Floodlight - OF Monitor*, and *RRDTool - OF Monitor*. Regarding these links, it is to stand out that: (*i*) the Network Administrator does not need to adapt the data transferred because the Maker is responsible for hiding the data mapping (it is internally carried out by the Mediator Bus); and (*ii*) the Network Administrator obtains guidelines about links creation from the Maker via CHS. Therefore, $\zeta_l = 1$, $goF = 2$, and $coF = 1$. Using these values in the equation 3.4, $\zeta_{com:maker} = 12$.

Launch on Maker. As the Network Administrator requests the execution of PMM from the Maker by clicking the button Run, $\zeta_{lau:maker} = 1$. After launching, in the Traffic GUI of PMM (see Figure 4.4), the Network Administrator can identify the three Open vSwitches involved in the *NMSit-SDN* by analyzing, in an integrated way, RRDTool images and HTML tables.

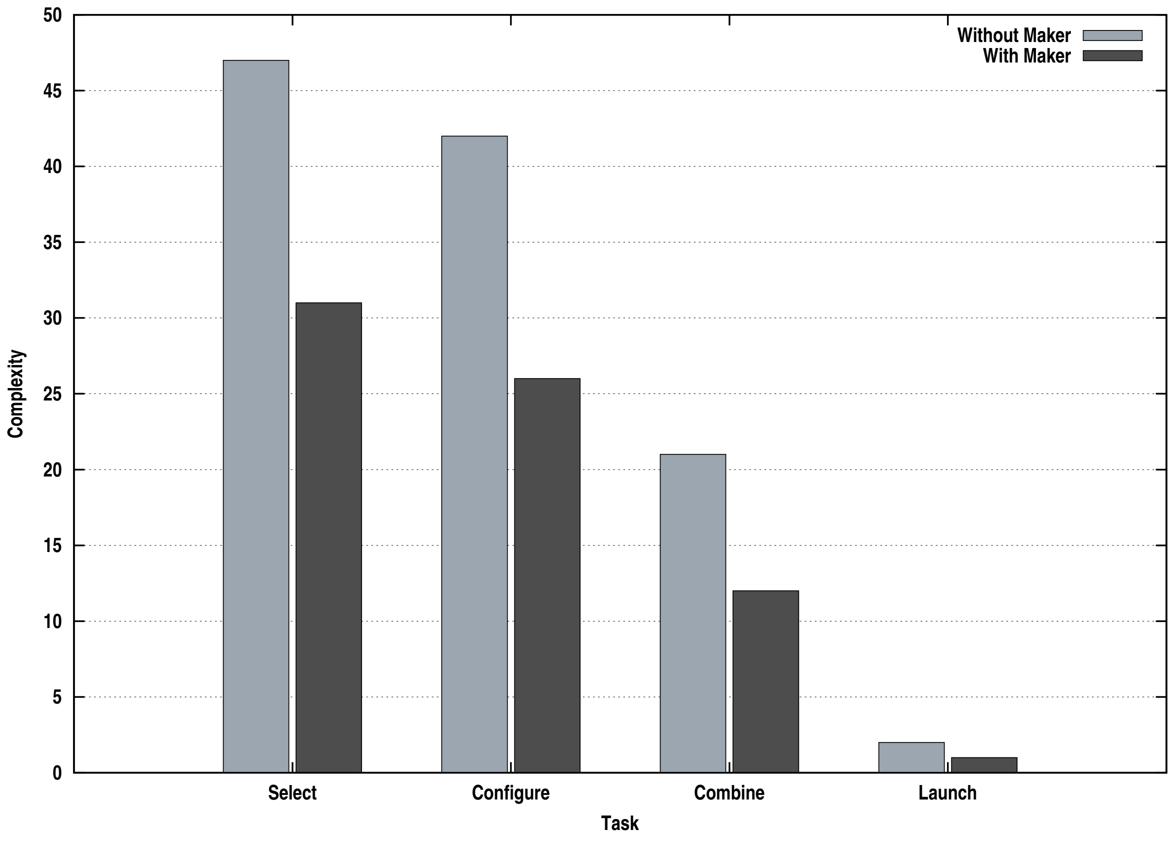


Figure 4.5 – Complexity on NMSit-SDN

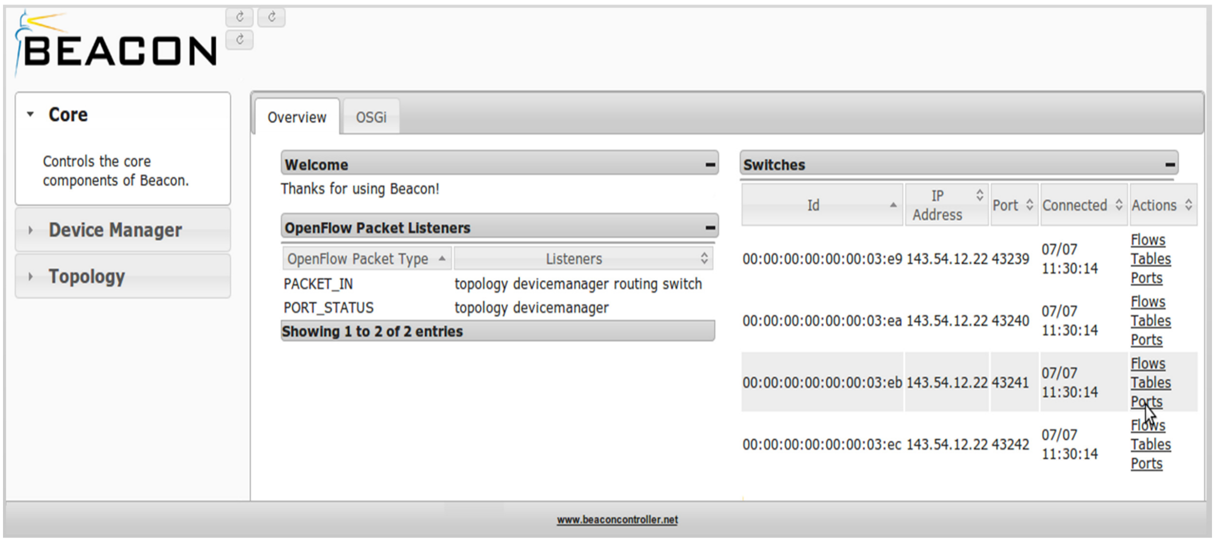
Figure 4.5 depicts the obtained results in the complexity assessment when the Network Administrator faces the *NMSit-SDN* with and without using the Mashment Maker. According these results: (*i*) the complexity of Select on Maker ($\zeta_{sel:maker} = 31$) is less than without Maker ($\zeta_{sel:nomaker} = 47$), attained by the services Drag-and-Drop and CHS, (*ii*) the complexity of

Configure on Maker ($\zeta_{con:maker} = 26$) is less than without Maker ($\zeta_{con:nomaker} = 42$), reached by CHS, (*iii*) the complexity of Combine on Maker ($\zeta_{com:maker} = 12$) is less than without Maker ($\zeta_{com:nomaker} = 21$), achieved by the Wire service and the Mediator Bus; and (*iv*) the complexity of Launch on Maker ($\zeta_{lau:maker} = 1$) is less than without Maker ($\zeta_{lau:nomaker} = 2$), obtained by the Designer.

Since in a Mashment Maker-based workspace the complexity of each task carried out to address the *NMSit-SDN* is less than the corresponding complexity when the Maker is not used, $\zeta_{pmm} = 70$ is also less (about 37.5%) than $\zeta_{nomaker} = 112$. This global result and the per task results demonstrate that, in terms of complexity, it is feasible to use the proposed approach for addressing nmsits like the raised *NMSit-SDN*.

4.2.2 Time-consuming: results and analysis

Addressing without Maker. To continue the evaluation, it is proceeded to measure the time-consuming of addressing the *NMSit-SDN* when the Network Administrator does not use the Maker. This time-consuming (*i.e.*, $T_{nomaker}$) was computed by using the time-average of KLM actions (see Table 3.1). The time-average for each KLM action is (KIERAS, 2001) (TIAN; WEBER; LUTTEROTH, 2011): (*i*) $k = 0.2s$, (*ii*) $p = 1.1s$, (*iii*) $b = 0.1s$, (*iv*) $h = 0.4s$, (*v*) $dnd = 1.3s$; and (*vi*) $wire = 4.1s$.



Welcome

Thanks for using Beacon!

OpenFlow Packet Listeners

OpenFlow Packet Type	Listeners
PACKET_IN	topology devicemanager routing switch
PORT_STATUS	topology devicemanager

Showing 1 to 2 of 2 entries

Switches

Id	IP Address	Port	Connected	Actions
00:00:00:00:00:00:03:e9	143.54.12.22	43239	07/07 11:30:14	Flows Tables Ports
00:00:00:00:00:00:03:ea	143.54.12.22	43240	07/07 11:30:14	Flows Tables Ports
00:00:00:00:00:00:03:eb	143.54.12.22	43241	07/07 11:30:14	Flows Tables Ports
00:00:00:00:00:00:03:ec	143.54.12.22	43242	07/07 11:30:14	Flows Tables Ports

Figure 4.6 – Beacon Web Tool

(ERICKSON, 2013)

Without the Maker, the Network Administrator can investigate the *NMSit-SDN* by using a monitoring Web-based tool per OpenFlow controller, such as Beacon Web Tool (ERICKSON, 2013), POX Web Tool (POX, 2013), and Floodlight Web Tool (FLOODLIGHT, 2013). As these tools operate similarly, their time-consuming is $T_{beaconWebTool} = T_{poxWebTool} = T_{floodlightWebTool}$.

The Network Administrator retrieves information in HTML tables about the packet traffic of a switch from the Beacon Web Tool (see Figure 4.6) by the following actions: (*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 the above actions, $T_{beaconWebTool} = h+4p+8b = 5.6s$ and the time of separately using the above mentioned tools is $T_{nonIntegrated} = 16.8s$.

In order to obtain in a unique GUI, the retrieved information by using the aforementioned Web-based tools, the Network Administrator writes the *Situational Script* that generates HTML tables and three RRD images in an integrated way. The time-consuming to develop and execute such *Situational Script* is expressed as follows: $T_{script} = T_{dev} + T_{lau}$. Where, $T_{dev} = T_{table} + T_{rrdImages}$. Considering only the time to type the code that generates the tables and images, $T_{dev} = (h + 290k) + (h + 1200k) = 298.8s$. In turn, $T_{lau} = h + p + 2b + 11k = 3.9s$. Therefore, $T_{script} = 302.7s$ and, so, the time-consuming without the Maker is $T_{nomaker} = T_{nonIntegrated} + T_{script} = 319.5s$.

Addressing with Maker. Once calculated $T_{nomaker}$, it is computed the time-consuming of dealing with the *NMSit-SDN* by developing, launching, and using PMM. Such time-consuming is $T_{maker} = T_{dev:maker} + T_{lau:maker} + T_{use:pmm}$. According the equation 3.5 and considering the no conducting of Tune ($i = j = k = o = 1$), $T_{dev:maker} = T_{sel:maker} + T_{con:maker} + T_{com:maker}$.

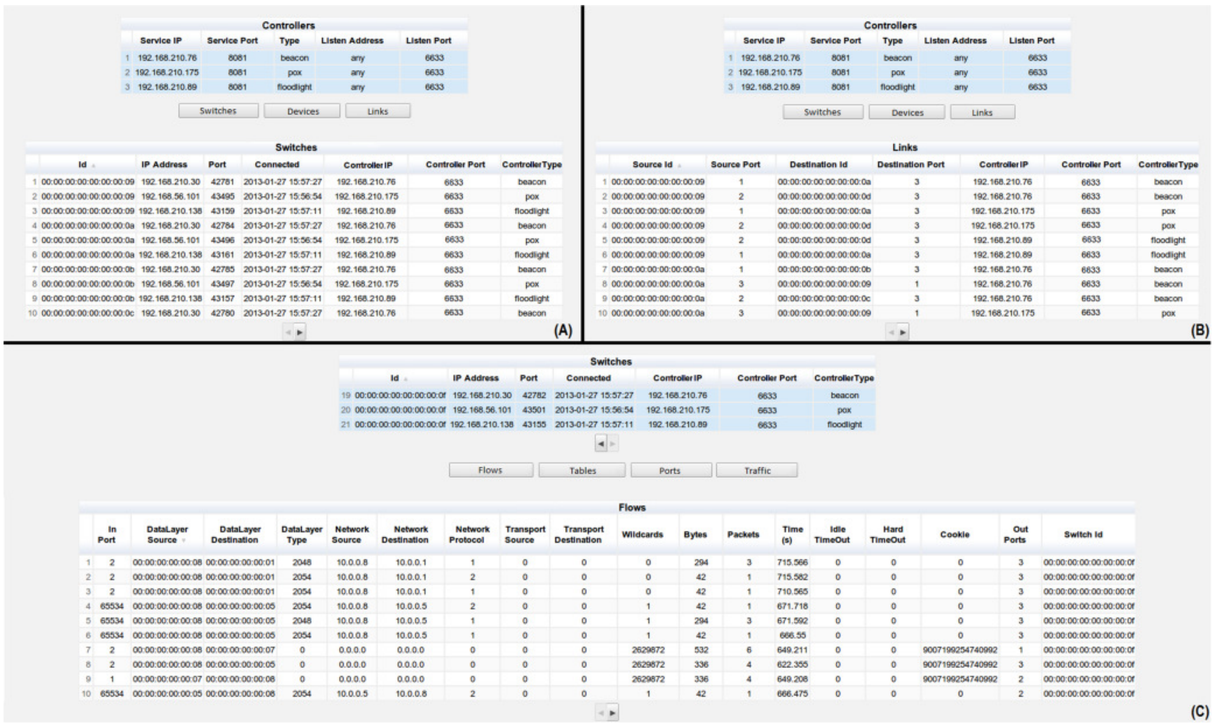
Select on Maker. The Network Administrator defines R_{used} by selecting the Visual Resources that form PMM as follows: (*i*) drag-and-drop *Beacon*, (*ii*) drag-and-drop *POX*, (*iii*) drag-and-drop *Floodlight*, (*iv*) drag-and-drop *RRDTool*; and (*v*) drag-and-drop *OF Monitor*. In this way, considering the equation 3.6, $T_{sel:maker} = \sum_1^5 dnd = 6.5s$.

Configure on Maker. The Network Administrator defines R_{conf} by providing the functioning parameters of *Beacon*, *POX*, *Floodlight*, and *RRDTool*. As the Network Administrator writes manually these parameters, $\tau_{beacon} = \tau_{pox} = \tau_{floodlight} = [4 * (p + h + 2b) + (16 + 8 + 8 + 5) * k] = 14.2s$ and $\tau_{rrd} = p + h + 2b + 3k = 2.3s$. Using these values in the equation 3.7, $T_{con:maker} = 44.9s$.

Combine on Maker. The Network Administrator creates the δ of PMM as follows: (*i*) wire *Beacon - OF Monitor*, (*ii*) wire *POX - OF Monitor*, (*iii*) wire *Floodlight - OF Monitor*; and (*iv*) wire *RRDTool - OF Monitor*. Therefore, according the equation 3.8, $T_{com:maker} = \sum_1^4 wire = 16.4s$.

Launch on Maker. Before requesting the execution of PMM, the Network Administrator conducts the following actions sequence to save 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, the Network Administrator launches it by clicking the button Run. Thus, $T_{lau:maker} = 3(h + p + 2b) + 3k = 5.7s$.

On runtime, PMM allows Network Administrators to retrieve information about the *NMSit-SDN*. Using PMM, the Network Administrator carries out the following actions sequence to obtain general information of switches (see Figure 4.7 (A)) or Links (see Figure 4.7 (B)) in three different controllers: (*i*) point the mouse to the Controllers list, (*ii*) press and release the mouse to select three distinct controllers, (*iii*) point the mouse to the button Switches or Links; and (*iv*) press and release the mouse to click the button Switches or Links. In addition, to retrieve information about flows (see Figure 4.7 (C)), tables, ports, or traffic (see Figure 4.4) of three switches, the Network Administrator: (*i*) press and release the mouse to select three switches each one in a different controller, (*ii*) point the mouse to the button Flows, Tables, Ports, or Traffic; and (*iii*) press and release the mouse to click the button Flows, Tables, Ports, or Traffic. Considering jointly the above sequences, $T_{use:pmm} = h + 3p + 16b = 5.3s$.



Controllers

	Service IP	Service Port	Type	Listen Address	Listen Port
1	192.168.210.76	8081	beacon	any	6633
2	192.168.210.175	8081	pox	any	6633
3	192.168.210.89	8081	floodlight	any	6633

Switches

	Id	IP Address	Port	Connected	Controller IP	Controller Port	ControllerType
1	00:00:00:00:00:00:00:09	192.168.210.30	42781	2013-01-27 15:57:27	192.168.210.76	6633	beacon
2	00:00:00:00:00:00:00:09	192.168.56.101	43495	2013-01-27 15:56:54	192.168.210.175	6633	pox
3	00:00:00:00:00:00:00:09	192.168.210.138	43159	2013-01-27 15:57:11	192.168.210.89	6633	floodlight
4	00:00:00:00:00:00:00:0a	192.168.210.30	42784	2013-01-27 15:57:27	192.168.210.76	6633	beacon
5	00:00:00:00:00:00:00:0a	192.168.56.101	43496	2013-01-27 15:56:54	192.168.210.175	6633	pox
6	00:00:00:00:00:00:00:0a	192.168.210.138	43161	2013-01-27 15:57:11	192.168.210.89	6633	floodlight
7	00:00:00:00:00:00:00:0b	192.168.210.30	42785	2013-01-27 15:57:27	192.168.210.76	6633	beacon
8	00:00:00:00:00:00:00:0b	192.168.56.101	43497	2013-01-27 15:56:54	192.168.210.175	6633	pox
9	00:00:00:00:00:00:00:0b	192.168.210.138	43157	2013-01-27 15:57:11	192.168.210.89	6633	floodlight
10	00:00:00:00:00:00:00:0c	192.168.210.30	42780	2013-01-27 15:57:27	192.168.210.76	6633	beacon

Controllers

	Service IP	Service Port	Type	Listen Address	Listen Port
1	192.168.210.76	8081	beacon	any	6633
2	192.168.210.175	8081	pox	any	6633
3	192.168.210.89	8081	floodlight	any	6633

Links

	Source Id	Source Port	Destination Id	Destination Port	Controller IP	Controller Port	ControllerType
1	00:00:00:00:00:00:00:09	1	00:00:00:00:00:00:00:0a	3	192.168.210.76	6633	beacon
2	00:00:00:00:00:00:00:09	2	00:00:00:00:00:00:00:0d	3	192.168.210.76	6633	beacon
3	00:00:00:00:00:00:00:09	1	00:00:00:00:00:00:00:0a	3	192.168.210.175	6633	pox
4	00:00:00:00:00:00:00:09	2	00:00:00:00:00:00:00:0d	3	192.168.210.175	6633	pox
5	00:00:00:00:00:00:00:09	2	00:00:00:00:00:00:00:0d	3	192.168.210.89	6633	floodlight
6	00:00:00:00:00:00:00:09	1	00:00:00:00:00:00:00:0a	3	192.168.210.89	6633	floodlight
7	00:00:00:00:00:00:00:0a	1	00:00:00:00:00:00:00:0b	3	192.168.210.76	6633	beacon
8	00:00:00:00:00:00:00:0a	3	00:00:00:00:00:00:00:09	1	192.168.210.76	6633	beacon
9	00:00:00:00:00:00:00:0a	2	00:00:00:00:00:00:00:0c	3	192.168.210.76	6633	beacon
10	00:00:00:00:00:00:00:0a	3	00:00:00:00:00:00:00:09	1	192.168.210.175	6633	pox

Switches

	Id	IP Address	Port	Connected	Controller IP	Controller Port	ControllerType
19	00:00:00:00:00:00:00:0f	192.168.210.30	42782	2013-01-27 15:57:27	192.168.210.76	6633	beacon
20	00:00:00:00:00:00:00:0f	192.168.56.101	43501	2013-01-27 15:56:54	192.168.210.175	6633	pox
21	00:00:00:00:00:00:00:0f	192.168.210.138	43155	2013-01-27 15:57:11	192.168.210.89	6633	floodlight

Flows

	In Port	DataLayer Source	DataLayer Destination	DataLayer Type	Network Source	Network Destination	Network Protocol	Transport Source	Transport Destination	Wildcards	Bytes	Packets	Time (s)	Idle TimeOut	Hard TimeOut	Cookie	Out Ports	Switch Id
1	2	00:00:00:00:00:08	00:00:00:00:00:01	2048	10.0.0.8	10.0.0.1	1	0	0	0	294	3	715.566	0	0	0	3	00:00:00:00:00:00:00:0f
2	2	00:00:00:00:00:08	00:00:00:00:00:01	2054	10.0.0.8	10.0.0.1	2	0	0	0	42	1	715.582	0	0	0	3	00:00:00:00:00:00:00:0f
3	2	00:00:00:00:00:08	00:00:00:00:00:01	2054	10.0.0.8	10.0.0.1	1	0	0	0	42	1	710.565	0	0	0	3	00:00:00:00:00:00:00:0f
4	65534	00:00:00:00:00:08	00:00:00:00:00:05	2054	10.0.0.8	10.0.0.5	2	0	0	1	42	1	671.718	0	0	0	3	00:00:00:00:00:00:00:0f
5	65534	00:00:00:00:00:08	00:00:00:00:00:05	2048	10.0.0.8	10.0.0.5	1	0	0	1	294	3	671.592	0	0	0	3	00:00:00:00:00:00:00:0f
6	65534	00:00:00:00:00:08	00:00:00:00:00:05	2054	10.0.0.8	10.0.0.5	1	0	0	1	42	1	666.55	0	0	0	3	00:00:00:00:00:00:00:0f
7	2	00:00:00:00:00:08	00:00:00:00:00:07	0	0.0.0.0	0.0.0.0	0	0	0	2629872	532	6	649.211	0	0	9007199254740992	1	00:00:00:00:00:00:00:0f
8	2	00:00:00:00:00:08	00:00:00:00:00:05	0	0.0.0.0	0.0.0.0	0	0	0	2629872	336	4	622.355	0	0	9007199254740992	3	00:00:00:00:00:00:00:0f
9	1	00:00:00:00:00:07	00:00:00:00:00:08	0	0.0.0.0	0.0.0.0	0	0	0	2629872	336	4	649.208	0	0	9007199254740992	2	00:00:00:00:00:00:00:0f
10	65534	00:00:00:00:00:05	00:00:00:00:00:08	2054	10.0.0.5	10.0.0.8	2	0	0	1	42	1	666.475	0	0	0	2	00:00:00:00:00:00:00:0f

Figure 4.7 – PMM - user interfaces of switches, links, and flows

Taking into account the values of $T_{dev:maker}$, $T_{lau:maker}$, and $T_{use:pmm}$, it is expected that the Network Admistrator spends $T_{maker} = 78.8s$ to deal with the *NMSit-SDN* by using the proposed approach. Once computed T_{maker}, it is proceeded to measure $T_{makerExperimental}$ by conducting a test with end-users. In such test participated 30 Network Administrators whose age ranged from 22 to 35. Although all participants frequently had used Web-based tools, none of them had used a mashup tool before. In this way, each participant was trained to use the Maker during 45 minutes. About this test, it is also important to mention that the time-consuming average in seconds was took with a 95% confidence level.

Figure 4.8 depicts the obtained results in the time-consuming assessment when the Network Administrator addresses the *NMSit-SDN* with and without the Maker. According these results: (*i*) the time-consuming of Develop on Maker ($T_{dev:maker} = 67.8s$ – $T_{dev:makerExperimental} = 71.1s$) is less than when the Maker is not used ($T_{dev:nomaker} = 298.8s$), attained by the services Drag-and-Drop and Wire, (*ii*) because every mashment must be saved in the Maker before being executed, the time-consuming of Launch on Maker ($T_{lau:maker} = 5.7s$ – $T_{lau:makerExperimental} = 6.6s$) is greater than without Maker ($T_{lau:nomaker} = 3.9s$); and (*iii*) the implementation of the Maker has a good behavior in front of KLM-based computations, it is because the evaluation with Network Administrators (*i.e.*, Experimental with Maker) corroborated the time-consuming computed with KLM (*i.e.*, With Maker).

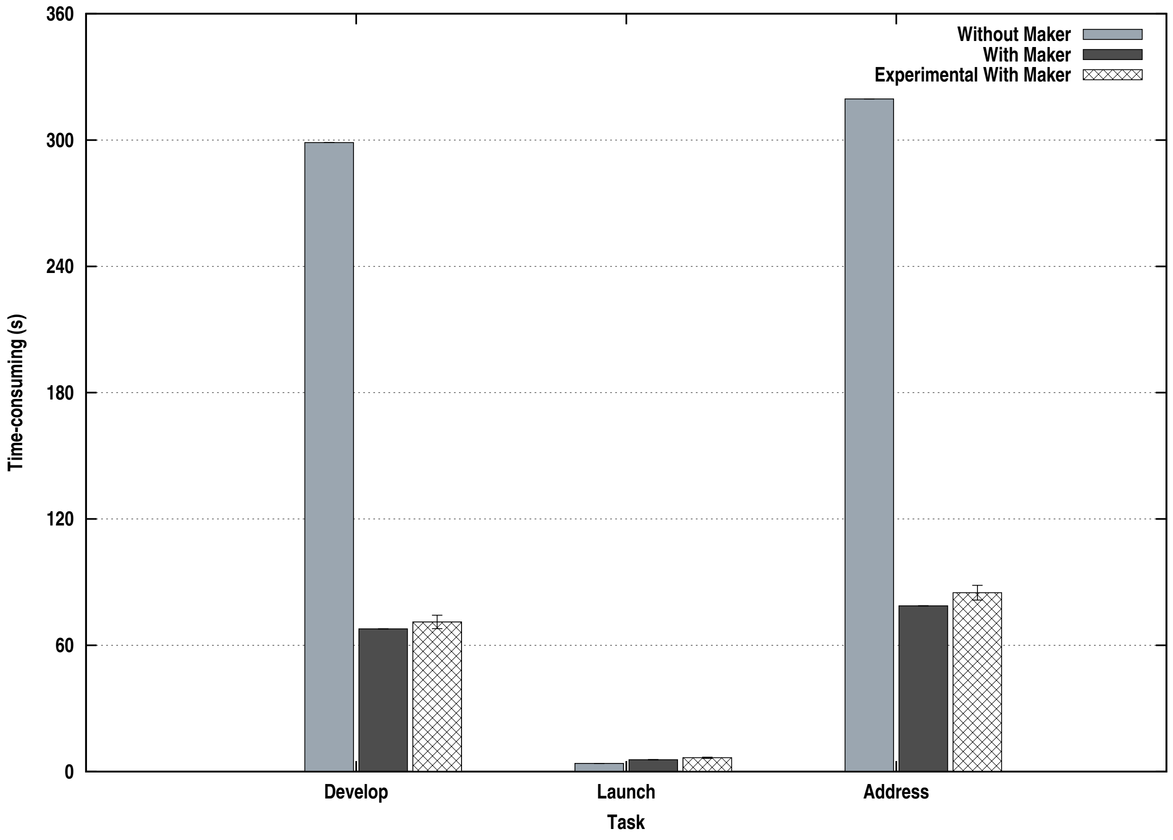


Figure 4.8 – Time-consuming on NMSit-SDN

Considering the above results, the time-consuming to address the *NMSit-SDN* with the Maker ($T_{maker} = 78.8s$ – $T_{makerExperimental} = 85.0s$) is less (about 75.3% - 73.4%) than without the Maker ($T_{nomaker} = 319.5s$). This global result and the per task results demonstrate that, in terms of time-consuming, it is feasible to use the proposed approach for handling nmsits like the raised *NMSit-SDN*.

4.2.3 Time-response: results and analysis

PMM and Beacon Web Tool. To continue the evaluation, it is proceeded to measure the time-response of PMM and *Beacon Web Tool* when conducting *SwitchesList*, *LinksList*, and *FlowsList*. In this and the next evaluations involving the average time-response in milliseconds (ms), 30 measurements were took with a 95% confidence level.

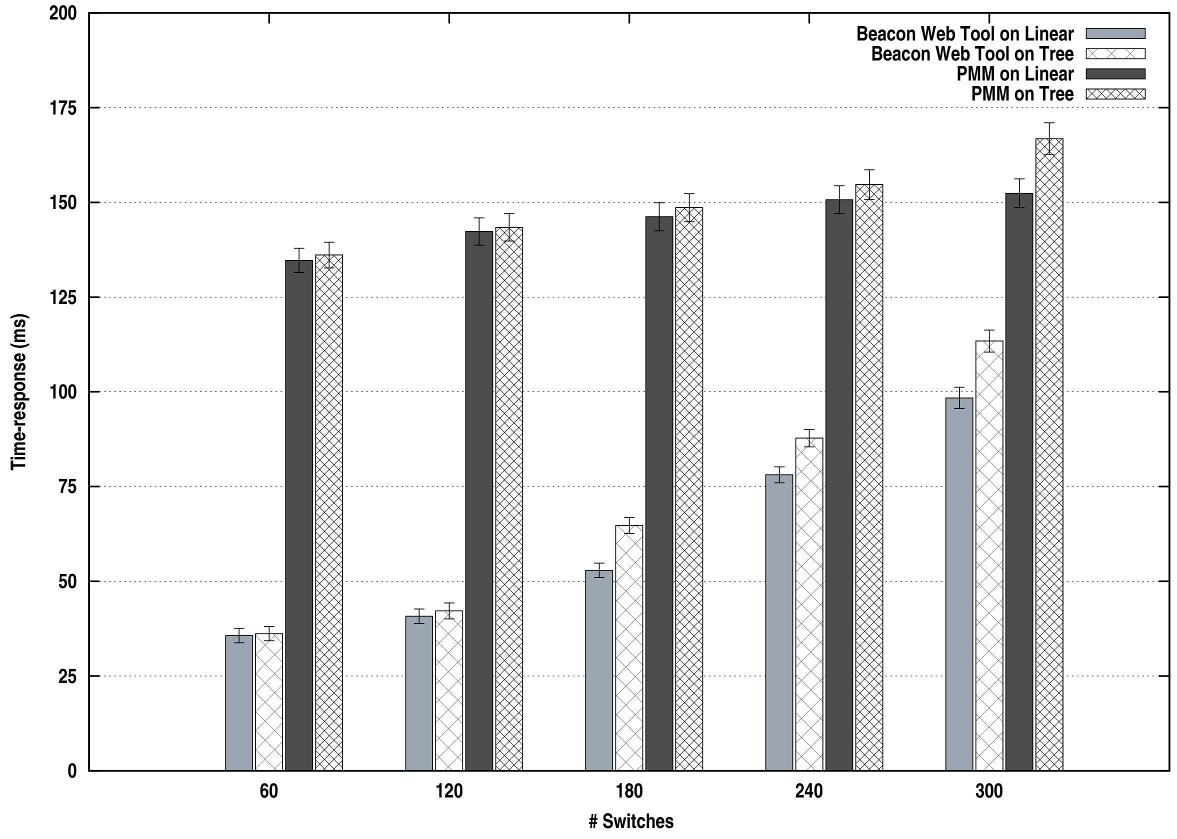


Figure 4.9 – Time-response on SwitchesList

In the time-response evaluation of the operation *SwitchesList*, the number of Open vSwitches was varied from 20 to 100 in each OpenFlow-based network. Thus, the total number of switches in each evaluation was 60, 120, 180, 240, and 300. Figure 4.9 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 ($r > 10000$) (JOINES; WILLENBORG; HYGH, 2002), the time-response results reveal: (*i*) *SwitchesList* of PMM 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 PMM; this result was expected because *Beacon Web Tool* works with one type of controller and PMM with three different types of controller.

In the time-response evaluation of the operation *LinksList*, the number of links was varied from 50 to 250 in each OpenFlow-based network. Therefore, the total number of links in each evaluation was 150, 300, 450, 600, and 750. Figure 4.10 depicts the corresponding results that reveal: (*i*) *LinksList* of PMM 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 PMM; again, this result was expected because *Beacon Web Tool* works with one type of controller and PMM with three different types.

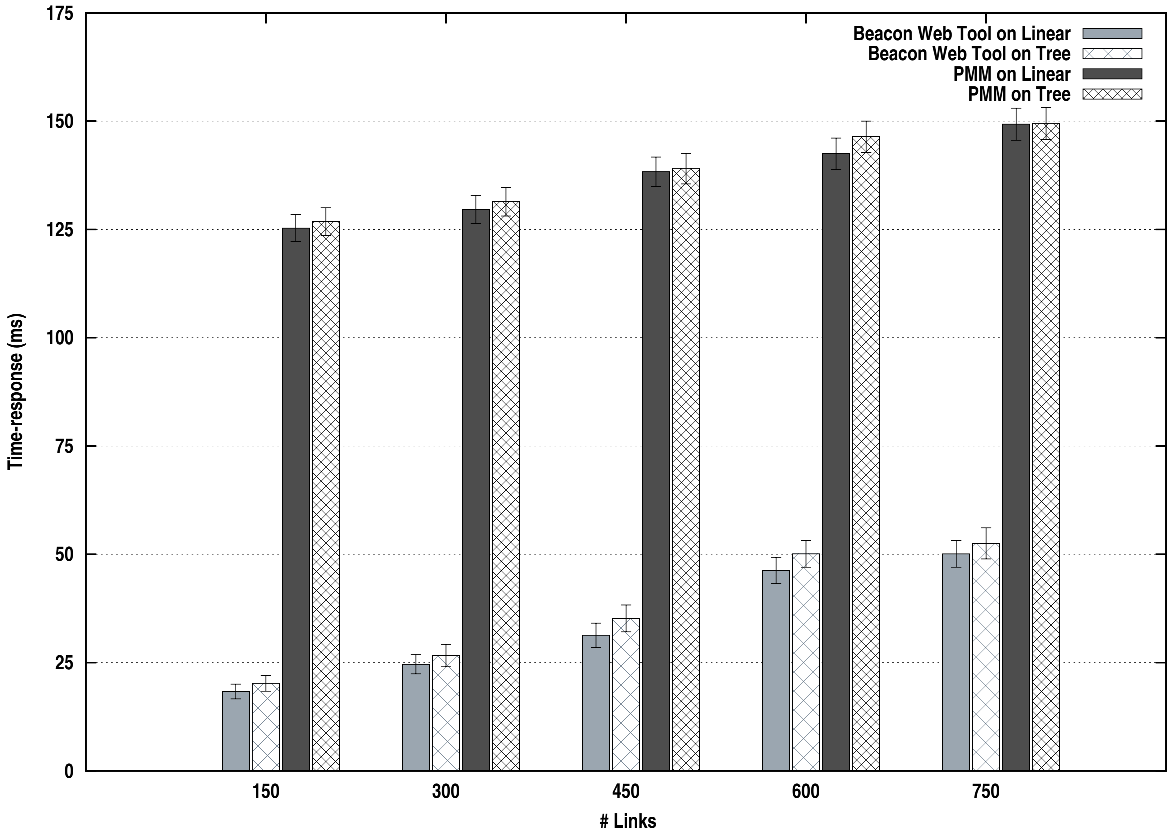


Figure 4.10 – Time-response on LinksList

In the time-response evaluation of the operation *FlowsList*, the number of flows was varied from 2000 to 10000. Figure 4.11 presents the corresponding results that reveal: (*i*) *FlowsList* of PMM has an admissible r that grows less than 1 ms per flow in tested topologies; and (*ii*) r of PMM and *Beacon Web Tool* is located at the same ranking for *FlowsList*. As in a network the number of flows may be large, in practice, PMM is constrained to retrieve 1000 flows per block, getting so a good r. Such constraint is not relevant because the use of a unique GUI to display all flows is not a good usability practice. Furthermore, using a mechanism of pagination, flows can be suitably retrieved and displayed for Network Administrators.

Although PMM 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, links, and flows. Such behavior is because the Mediator Bus hides the heterogeneity of controllers and the centralized-nature of controllers handles the number of network elements. In sum up, the time-response evaluation results demonstrate that, in terms of such metric, it is feasible to use the proposed approach for dealing with nmsits like the raised *NMSit-SDN*.

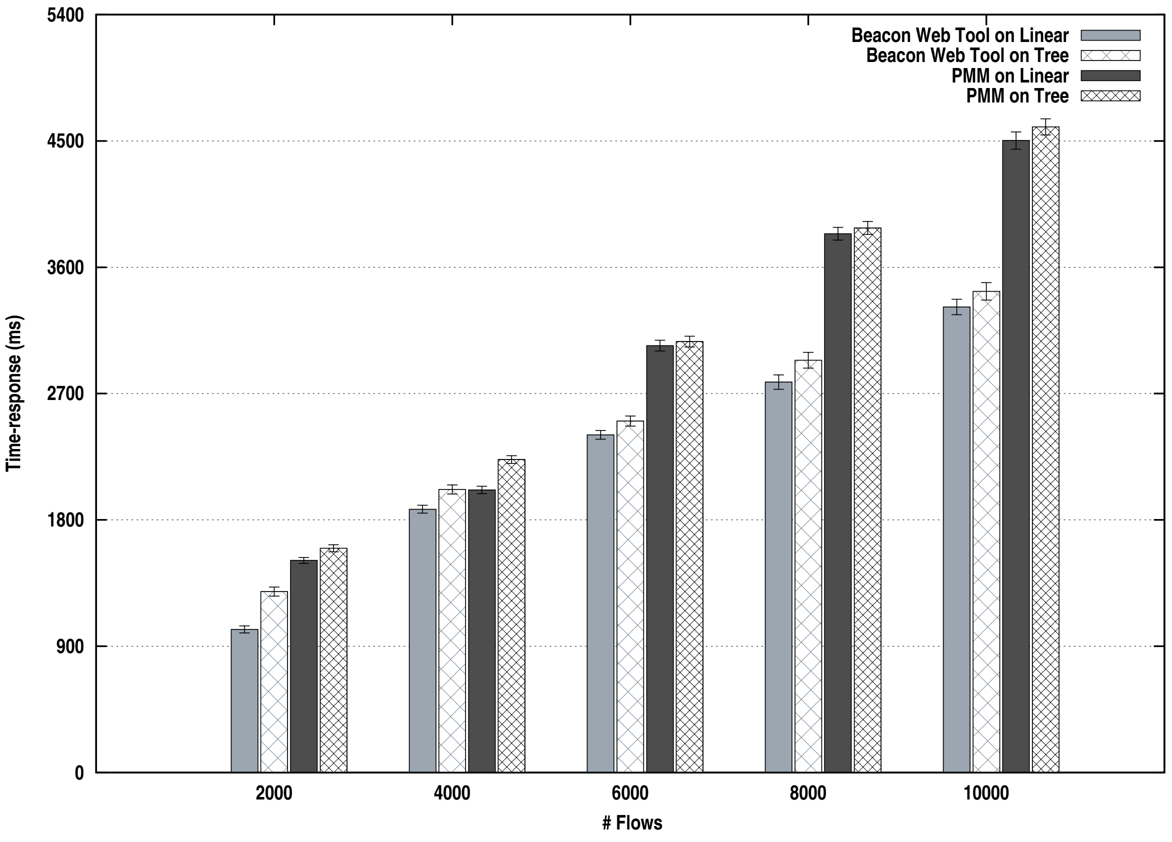


Figure 4.11 – Time-reponse on FlowsList

4.2.4 Traffic: results and analysis

PMM and Beacon Web Tool. To continue the evaluation, it is proceeded to measure the network traffic generated by PMM and *Beacon Web Tool* when carrying out *SwitchesList*, *LinksList*, and *FlowsList*. In this and the following evaluations, the traffic is expressed in $Bytes$ or $KBytes$.

In the traffic evaluation of the operation *SwitchesList*, the number of Open vSwitches was varied from 20 to 100 in each OpenFlow-based network. Thus, the total number of switches in each evaluation was 60, 120, 180, 240, and 300. Figure 4.12 presents the corresponding results in which there is not discrimination by topology because, the traffic generated by *SwitchesList* of PMM and *Beacon Web Tool* is independent of topologies (linear and tree) tested. In addition, these results reveal: (*i*) the traffic generated by *SwitchesList* of PMM grows negligibly (approx 112 $Bytes$ per switch) when the number of switches is increased, (*ii*) in relation to this operation, PMM generates more traffic than *Beacon Web Tool*; and (*iii*) the additional traffic generated by PMM is always less than 10%. Considering that *Beacon Web Tool* works with just one type of controller and PMM integrates data from three different types, the above facts corroborate that *SwitchesList* of PMM has a good behavior on network traffic.

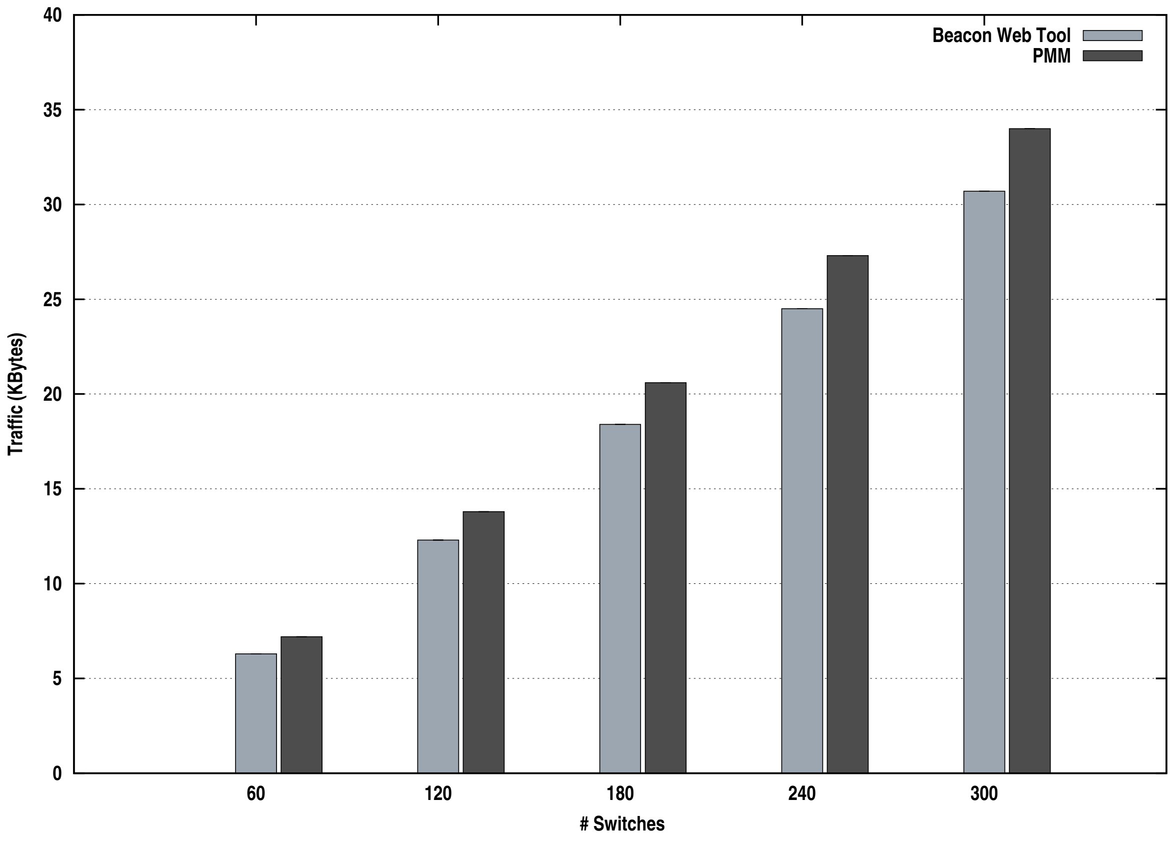


Figure 4.12 – Traffic on SwitchesList

In the traffic evaluation of the operation *LinksList*, the number of links was varied from 50 to 250 in each OpenFlow-based network. Therefore, the total number of links in each evaluation was 150, 300, 450, 600, and 750. Figure 4.13 depicts the corresponding results in which there is not discrimination by topology because the traffic generated by *LinksList* of PMM and *Beacon Web Tool* is independent of topologies tested. Furthermore, these results reveal: (*i*) the traffic generated by *LinksList* of PMM grows negligibly (approx 129 $Bytes$ per link) when the number of links is increased, (*ii*) regarding this operation, PMM generates more traffic than *Beacon Web Tool*; and (*iii*) the additional traffic generated by PMM is always less than 5%. Since the *Beacon Web Tool* works with just one type of controller and PMM integrates data from three different types, the above facts corroborate that *LinksList* of PMM has a good behavior on network traffic.

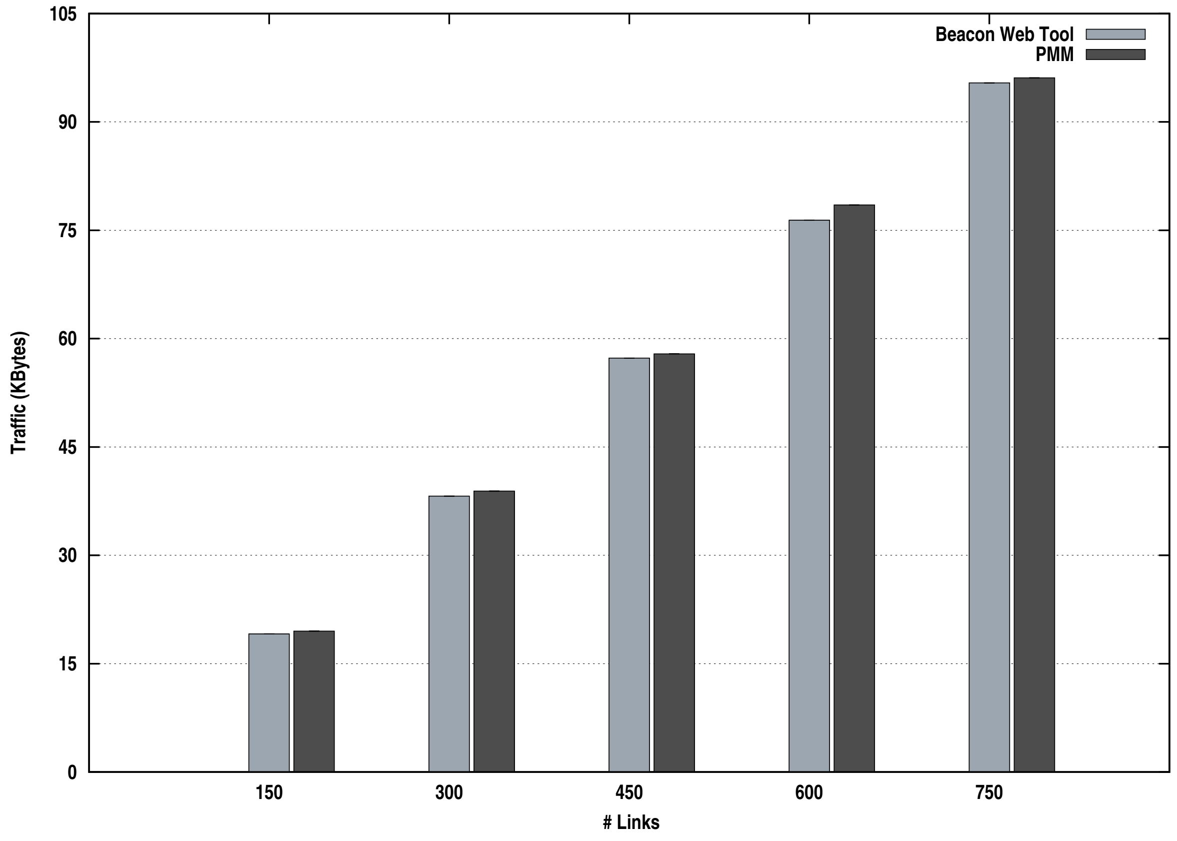


Figure 4.13 – Traffic on LinksList

In the traffic evaluation of the operation *FlowsList*, the number of flows was varied from 2000 to 10000. Figure 4.14 presents the corresponding results in which there is not discrimination by topology because the traffic generated by *FlowsList* of PMM and *Beacon Web Tool* is independent of topologies tested. Additionally, these results reveal: (*i*) the traffic generated by *FlowsList* of PMM grows negligibly (approx $328 Bytes$ per flow) when the number of flows is increased, (*ii*) *FlowsList* of PMM generates more traffic than the corresponding operation of *Beacon Web Tool*; and (*iii*) the additional traffic generated by PMM is always less than 10%. As the *Beacon Web Tool* works with one type of controller and PMM integrates data from three different types, the above facts corroborate that *FlowsList* of PMM has a good behavior on network traffic.

Regarding the results obtained in the network traffic evaluation of the operations *SwitchesList*, *LinksList*, and *FlowsList* of PMM, it is important to mention: (*i*) JSON was used to decrease the size of information exchanged between the layers (Adaptation, Composition and Presentation) of PMM because JSON is less verbose than XML (PAUTASSO; ZIMMERMANN; LEYMANN, 2008); and (*ii*) the size of Visual Resources is too small to impact the quantity of traffic generated by PMM.

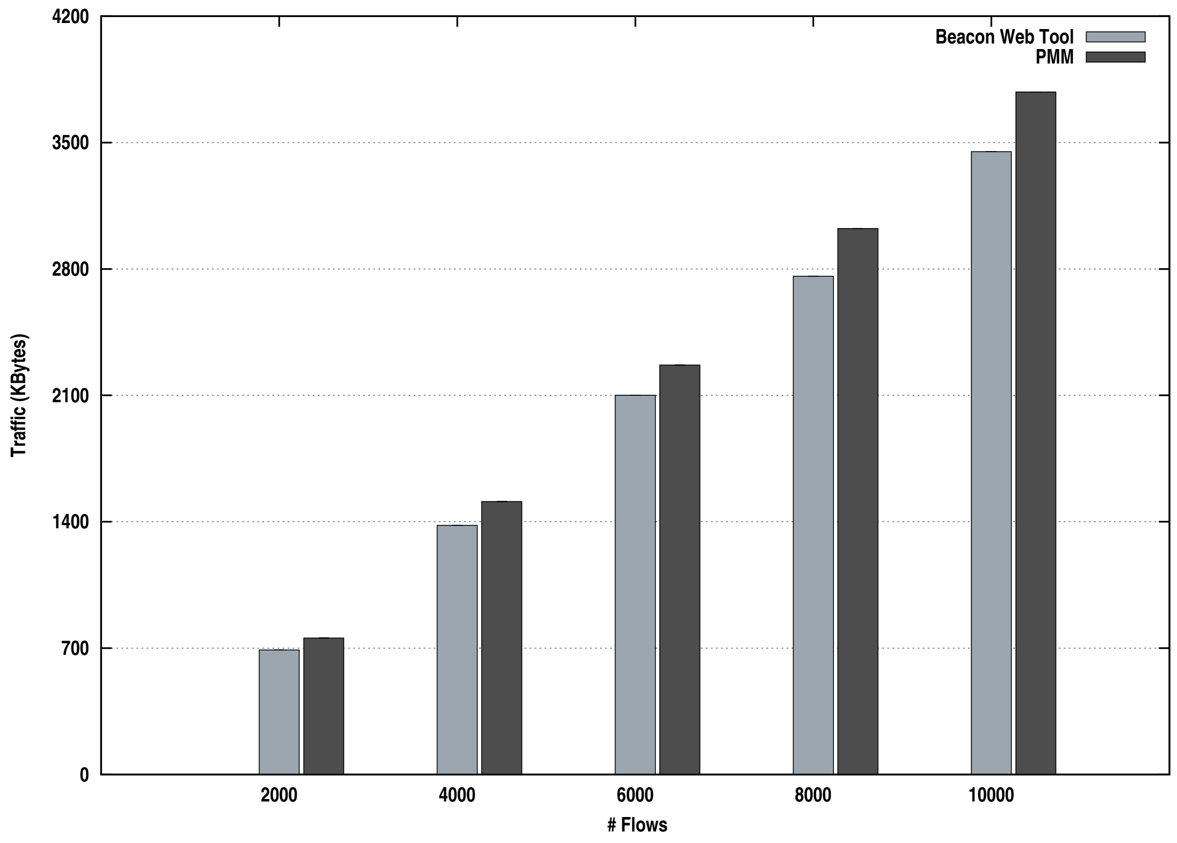


Figure 4.14 – Traffic on FlowsList

Although PMM integrates monitoring information from different controllers by using several additional software modules (*e.g.*, Mediator Bus), the traffic extra generated by its operations is always less than 10% (worst operation - *FlowsList*). Summarizing, the traffic evaluation results corroborate that, in terms of such metric, it is feasible to use the proposed approach for coping with nmsits like the raised *NMSit-SDN*.

4.3 Case study on virtual nodes

The case study on virtual nodes is formed by a test environment (see Figure 4.15), a nmsit called *NMSit-VN*, and experiments conducted to evaluate the addressing of such a nmsit with (*i.e., Virtual Node Monitoring Mashment* - $VNMM$) and without (*i.e., Monitoring Script*) the mashment-based approach. Regarding this case study, it is also important to mention that four metrics are measured in the experiments: complexity, time-consuming, time-response, and traffic.

Test environment. 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. The Mashment System Server, Mashment Maker, and MashmentDB 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. Virtual machines of Open vSwitch and Linux Ubuntu were deployed on Xen and VirtualBox. The virtualized Floodlight and the handled virtual Open vSwitches were deployed on a server with 8 GBytes RAM and 3.4 GHz core i7 processor. The user interfaces of the Mashment Maker, $VNMM$, and *Monitoring Script* were executed on a Client with 2 GBytes RAM and 2.53 GHz core 2 duo processor.

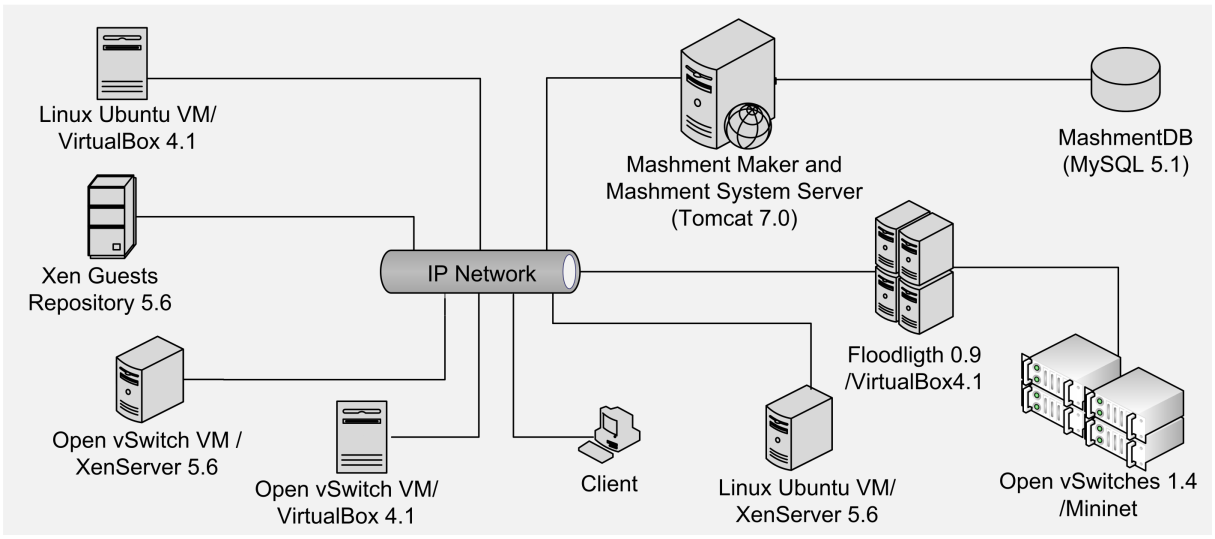


Figure 4.15 – Test environment on virtual nodes

NMSit-VN. Let's suppose the following nmsit: In the virtual infrastructure of the test environment, the Network Administrator needs to investigate/identify: (*i*) the Host Computer Systems and guests suffering unexpected overload in processor, memory, and/or network; and *(ii)* the Open vSwitches that are causing sudden performance degradation in an OpenFlow-based network handled by Floodlight. Therefore, he/she requires a situational solution that presents, in an integrated, visual, and intelligible way, information about Xen, VirtualBox, guests, and Open vSwitches (including links, flows, ports, and traffic). In order to get such a solution and cope with this nmsit, the Network Administrator tests two options: (*i*) without the proposed approach, creates, launches, and uses the *Monitoring Script*; and *(ii)* with the proposed approach, develops, launches, and uses $VNMM$ by the Mashment Maker. The following sections present the results and analysis of experiments conducted to evaluate such options.

4.3.1 Complexity: results and analysis

Addressing without Maker. To start the evaluation, it is proceeded to measure the complexity of addressing the *NMSit-VN* when the Network Administrator follows the proposed process to deal with nmsits but does not use the Maker. Without the Maker, the Network Administrator develops and executes the *Monitoring Script* that retrieves information about: (*i*) the use of memory, processor, and network from guests (virtual machines and/or virtual switches) hosted by XenServer and VirtualBox; and (*ii*) the network traffic, ports, and flows from switches handled by Floodlight. This *Monitoring Script* presents the retrieved information in a user interface formed by HTML tables and chart images. According the equation 3.1 and considering the no conducting of Tune, $\zeta_{nomaker} = \zeta_{sel:nomaker} + \zeta_{con:nomaker} + \zeta_{com:nomaker} + \zeta_{lau:nomaker}$.

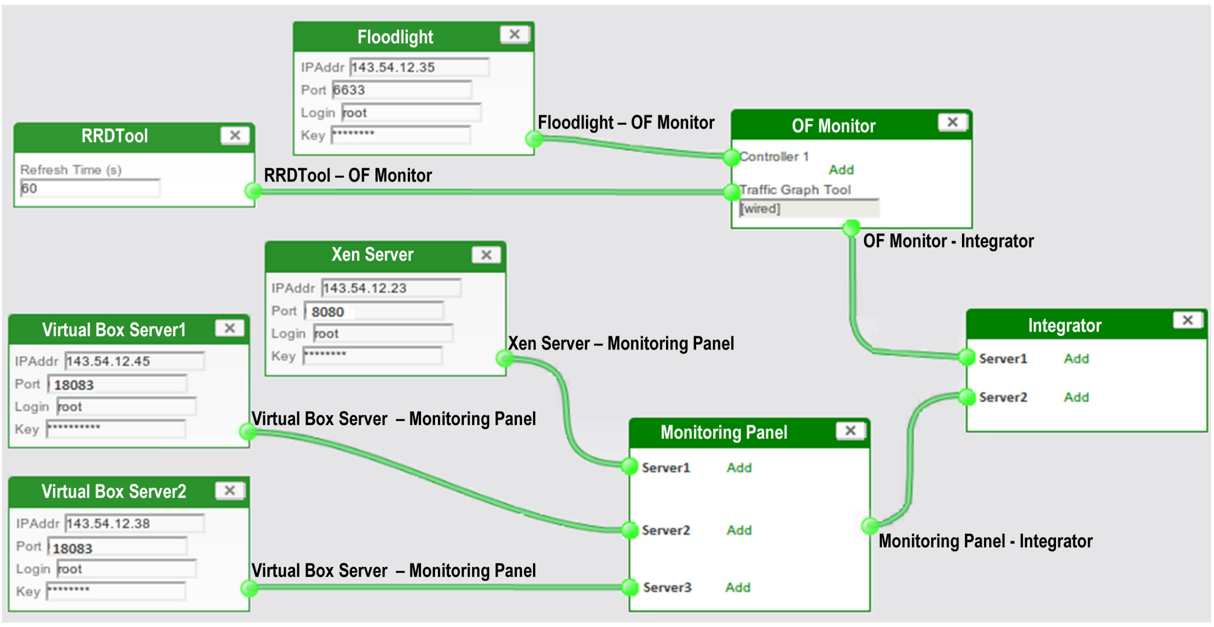
Select without Maker. The Network Administrator conducts the selection of controller tool (*i.e.*, Floodlight Tool), virtualization tool (*i.e.*, XenSDK and VirtualBox SDK), visualization tool (*i.e.*, RRDTool), and specific commands of such tools that allow to monitor heterogeneous virtual nodes. This selection is complex because it is not tool-assisted and guidelines about advanced commands of controller, virtualization, and visualization tools are scattered on the Internet. In this way, $\varsigma_m = 2$, $gF = 3$, and $cF = 1$. Using these values in the equation 3.2 and considering $nAvailableResources = 15$ (it grows up from 14 to 15 due to the creation of PMM carried out in the previous case study), $\zeta_{sel:nomaker} = 50$.

Configure without Maker. The Network Administrator carries out the configuration of selected tools by providing their corresponding functioning parameters. According the equation 3.3, $\zeta_{con:nomaker} = \varsigma_{con:xen} + \varsigma_{con:vbox} + \varsigma_{con:floodlight} + \varsigma_{con:rrd}$. Since the Network Administrator obtains general information about configuration of virtualization and controller tools from documentation easy to find on the Internet and defines specific statistic commands after additional search, $sourceParameter(statisticCommand) = 5$ and $sourceParameter(login) = sourceParameter(key) = sourceParameter(ip) = sourceParameter(port) = 2$. Furthermore, since the Network Administrator extrapolates the configuration information of RRDTool from documentation simple to find on the Internet, $\varsigma_{con:rrd} = 3$. Using these values, $\zeta_{con:nomaker} = 42$.

Combine without Maker. The Network Administrator manually creates one logical link among the tools of: (*i*) virtualization and visualization; and (*ii*) controller and visualization. This creation is complex because, first, the Network Administrator is responsible for adapting the retrieved data (tools involved in the *NMSit-VN* use different data types, for instance, Floodlight Tool and XenSDK use data types based on JSON and Java, respectively). Second, the Network Administrator neither has explicit nor centralized guidelines to develop these links. Therefore, $l = 4$, $\varsigma_l = 3$, $goF = 3$, and $coF = 1$. Using these values in the equation 3.4, $\zeta_{com:nomaker} = 21$.

Launch without Maker. As the Network Administrator requests the execution of the *Monitoring Script* by typing a specific command in a Linux Command Line, $\zeta_{lau:nomaker} = 2$. After launching the *Monitoring Script*, the Network Administrator is able to investigate the *NMSit-SDN* by analyzing RRD images and HTML tables that present information about the elements forming the virtual nodes.

Addressing with Maker. Once computed the complexity of facing the *NMSit-VN* without the Maker, it is proceeded to evaluate the complexity of developing and launching $VNMM$. In a general way, in the Maker, the Network Administrator builds and requests the execution of $VNMM$ (see Figure 4.16) by using the services Drag-and-Drop, Wire, CHS, and Launching. According the equation 3.1 and considering the no conducting of Tune, $\zeta_{vnmm} = \zeta_{sel:maker} + \zeta_{con:maker} + \zeta_{com:maker} + \zeta_{lau:maker}$.


Floodlight
IPAddr 143.54.12.35
Port 6633
Login root
Key *******

RRDTool
Refresh Time (s) 60

OF Monitor
Controller 1 Add
Traffic Graph Tool [wired]

Xen Server
IPAddr 143.54.12.23
Port 8080
Login root
Key *******

Virtual Box Server1
IPAddr 143.54.12.45
Port 18083
Login root
Key *********

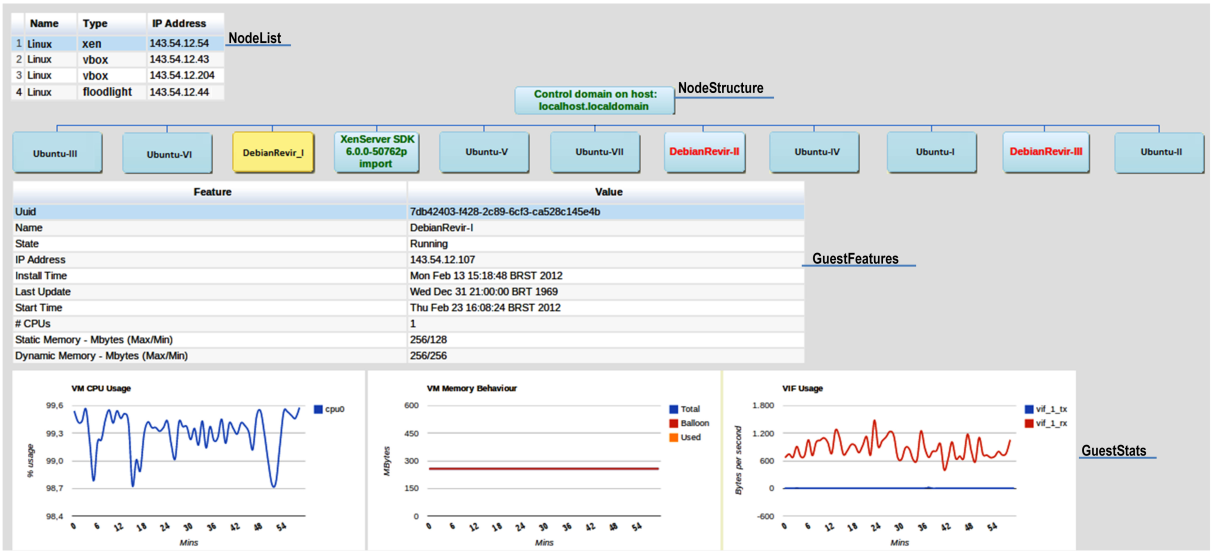
Virtual Box Server2
IPAddr 143.54.12.38
Port 18083
Login root
Key *******

Integrator
Server1 Add
Server2 Add

Monitoring Panel
Server1 Add
Server2 Add
Server3 Add

Floodlight – OF Monitor
RRDTool – OF Monitor
OF Monitor - Integrator
Xen Server – Monitoring Panel
Virtual Box Server – Monitoring Panel
Monitoring Panel - Integrator
Virtual Box Server – Monitoring Panel

Figure 4.16 – VNMM - development and launch


	Name	Type	IP Address
1	Linux	xen	143.54.12.54
2	Linux	vbox	143.54.12.43
3	Linux	vbox	143.54.12.204
4	Linux	floodlight	143.54.12.44

Feature	Value
Uuid	7db42403-f428-2c89-6cf3-ca528c145e4b
Name	DebianRevir-I
State	Running
IP Address	143.54.12.107
Install Time	Mon Feb 13 15:18:48 BRST 2012
Last Update	Wed Dec 31 21:00:00 BRT 1969
Start Time	Thu Feb 23 16:08:24 BRST 2012
# CPUs	1
Static Memory - Mbytes (Max/Min)	256/128
Dynamic Memory - Mbytes (Max/Min)	256/256

Figure 4.17 – VNMM - user interface of guest

Select on Maker. The Network Administrator uses the Drag-and-Drop service to select the Visual Resources ($M = 8$) that form $VNMM$. Thus, $R_{used} = \{$*Xen Server, Virtual Box Server1, Virtual Box Server2, Monitoring Panel, Floodlight, RRDTool, OF Monitor, Integrator*$\}$. As the definition of R_{used} is assisted by CHS, $\varsigma_m = 1$, $gF = 2$, and $cF = 1$. Furthermore, after creating PMM, the $nAvailableResources = 15$. Using these values in the equation 3.2, $\zeta_{sel:maker} = 36$.

Configure on Maker. The Network Administrator defines $R_{conf} = \{$*Xen Server, Virtual Box Server1, Virtual Box Server2, Floodlight, RRDTool*$\}$ by providing the corresponding operation settings. In this way, according the equation 3.3, $\zeta_{con:maker} = 2\varsigma_{vbox} + \varsigma_{xen} + \varsigma_{floodlight} + \varsigma_{rrd}$. Where, $\varsigma_{vbox} = \varsigma_{xen} = \varsigma_{floodlight} = sourceParameter(login) + sourceParameter(key) + sourceParameter(ip) + sourceParameter(port)$ and $\varsigma_{rrd} = sourceParameter(refreshT)$. As the Network Administrator obtains configuration guidelines about R_{conf} from CHS, $\varsigma_{vbox} = 8$ and $\varsigma_{rrd} = 2$. Using these values, $\zeta_{con:maker} = 34$.

Combine on Maker. The Network Administrator uses the Wire Service to create $(L = 7)$ links: *Xen Server - Monitoring Panel*, (2) *Virtual Box Server - Monitoring Panel*, *Floodlight - OF Monitor*, *RRDTool - OF Monitor*, *OF Monitor - Integrator*; and *Monitoring Panel - Integrator*. As the Network Administrator does not worry about the data mapping (because the Maker carries it out) and obtains guidelines about links creation from CHS, $\zeta_l = 1$, $goF = 2$, and $coF = 1$. Using these values in the equation 3.4, $\zeta_{com:maker} = 21$.

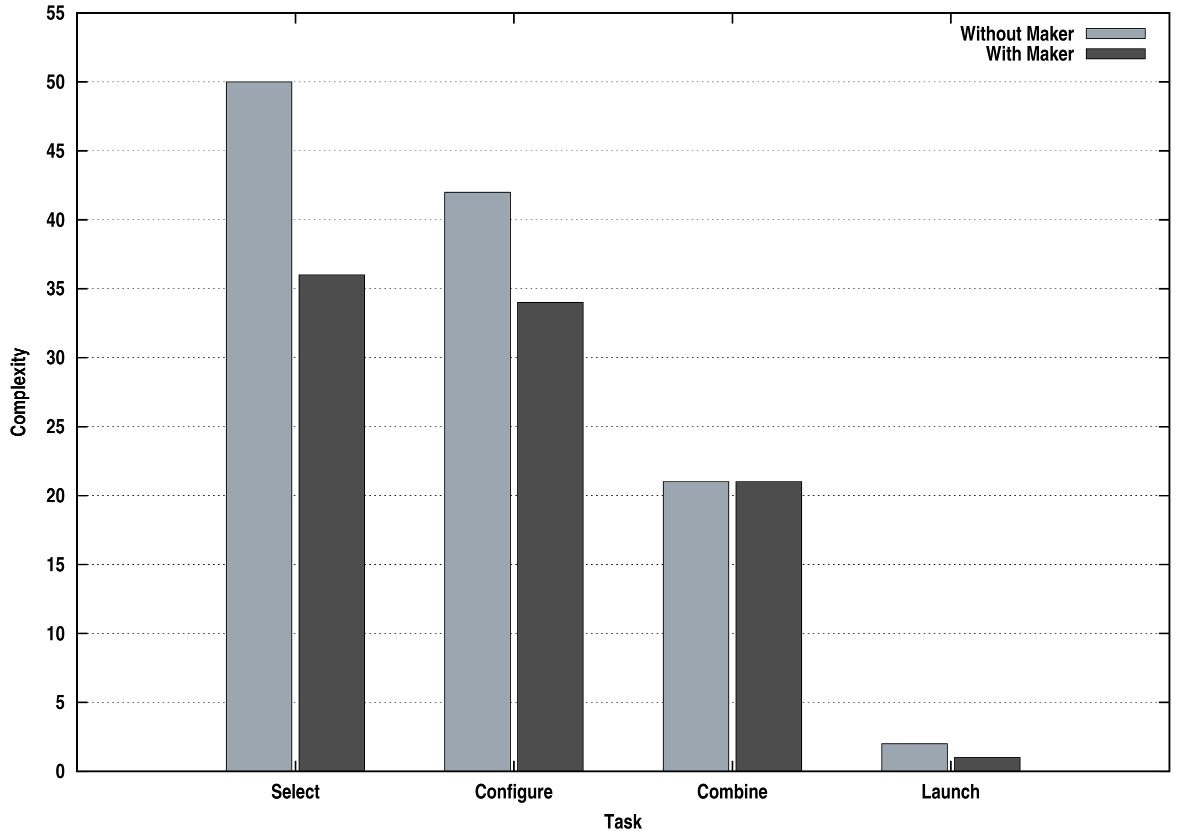


Figure 4.18 – Complexity on NMSit-VN

Launch on Maker. As the Network Administrator requests the execution of $VNMM$ from the Maker by clicking the button Run, $\zeta_{lau:maker} = 1$. After launching, in the GUI of $VNMM$, the Network Administrator can investigate the elements involved (Figure 4.17 presents details about guests in a virtual node) in the *NMSit-VN* by analyzing, in an integrated way, chart images and HTML tables.

Figure 4.18 depicts the obtained results in the complexity assessment when the Network Administrator faces the *NMSit-VN* with and without the Mashment Maker. According these results: (*i*) $\zeta_{sel:maker} = 36$ is less than $\zeta_{sel:nomaker} = 50$, attained by the services Drag-and-Drop and CHS, (*ii*) $\zeta_{con:maker} = 34$ is less than $\zeta_{con:nomaker} = 42$, reached by CHS, (*iii*) $\zeta_{com:maker} = 21$ is equal than $\zeta_{com:nomaker} = 21$, achieved by the Wire service and the Mediator Bus; and (*iv*) $\zeta_{lau:maker} = 1$ is less than $\zeta_{lau:nomaker} = 2$, obtained by the Designer.

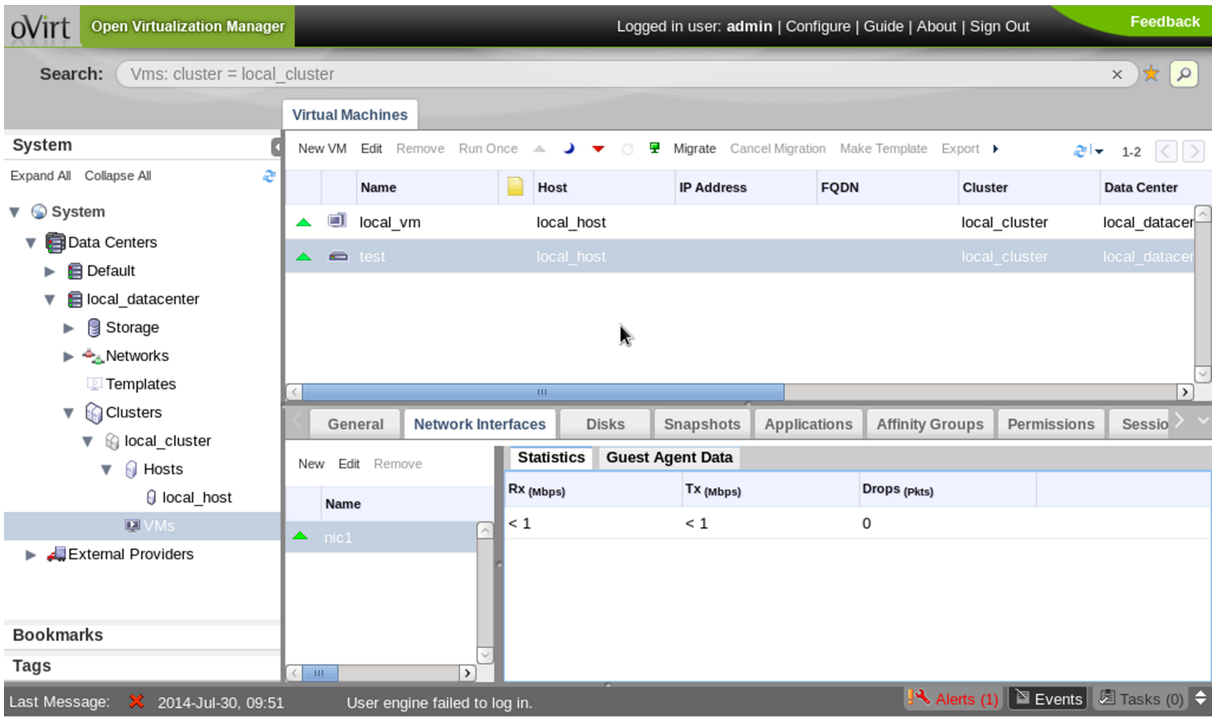
Since in a Mashment Maker-based workspace the complexity of each task carried out to address the *NMSit-VN* is less (or equal in Combine) than the corresponding complexity when the Maker is not used, $\zeta_{vnmm} = 92$ is also less (about 20%) than $\zeta_{nomaker} = 115$. This global result and the per task results demonstrate that, in terms of complexity, it is feasible to use the proposed approach for addressing nmsits like the raised *NMSit-VN*.

4.3.2 Time-consuming: results and analysis

Addressing without Maker. To continue the evaluation, it is proceeded to measure the time-consuming of addressing the *NMSit-VN* when the Network Administrator does not use the Maker. This time-consuming was also computed by using the time-average of KLM actions (see Table 3.1).

Without the Maker, the Network Administrator can investigate, in a non integrated way, the *NMSit-VN* by using oVirt (OVIRT, 2014) and the Floodlight Web Tool (FLOODLIGHT, 2013). Using oVirt (see Figure 4.19), the Network Administrator obtains information about the memory, processor, and network interfaces of a guest by conducting the following actions: (*i*) point the mouse to the drop down list System, (*ii*) press and release the mouse to select the System list, (*iii*) point the mouse to the drop down list Data Centers, (*iv*) press and release the mouse to select the Data Centers list, (*v*) point the mouse to the drop down list Local Data Center, (*vi*) press and release the mouse to select the Local Data Center list, (*vii*) point the mouse to the drop down list Clusters, (*viii*) press and release the mouse to select the Clusters list, (*ix*) point the mouse to the drop down list Local Cluster, (*x*) press and release the mouse to select the Local Cluster list, (*xi*) point the mouse to the drop down list Hosts, (*xii*) press and release the mouse to select the Hosts list, (*xiii*) point the mouse to a particular Host, (*xiv*) press and release the mouse to select such particular Host, (*xv*) point the mouse to a specific guest, (*xvi*) press and release the mouse to select such specific guest and, so, to obtain general information about it, (*xvii*) point the mouse to the tab Network Interfaces; and (*xviii*) press and release the mouse to select the Network Interfaces tab. Considering the above actions $T_{ovirt} = h + 9p + 18b = 12.1s$. Carrying out a similar analysis, the time-consuming to retrieve information in HTML tables about the packet traffic from the Floodlight Web Tool is $T_{floodlightWebTool} = h + 4p + 8b = 5.6s$. In this way, $T_{nonIntegrated} = 17.7s$.

In order to obtain in an integrated GUI, the retrieved information by using the aforementioned Web-based tools, the Network Administrator writes the *Monitoring Script* that generates HTML tables and RRD images. The time-consuming to develop and execute such *Monitoring Script* is expressed as follows: $T_{mscript} = T_{dev} + T_{lau}$. Where, $T_{dev} = T_{table} + T_{trafficImages} + T_{guestImages}$. Considering only the time to type the code that generates the tables and images, $T_{dev} = (h+290k)+(h+1200k)+(h+1200k) = 419.2s$. In turn, $T_{lau} = h+p+2b+11k = 3.9s$. Therefore, $T_{mscript} = 423.1$ and $T_{nomaker} = T_{nonIntegrated} + T_{mscript} = 440.8s$.


oVirt
Open Virtualization Manager
Logged in user: admin | Configure | Guide | About | Sign Out
Feedback
Search: Vms: cluster = local_cluster
Virtual Machines
System
Expand All Collapse All
System
Data Centers
Default
local_datacenter
Storage
Networks
Templates
Clusters
local_cluster
Hosts
local_host
VMs
External Providers
Bookmarks
Tags
New VM Edit Remove Run Once Migrate Cancel Migration Make Template Export
1-2
General Network Interfaces Disks Snapshots Applications Affinity Groups Permissions Sessio
New Edit Remove
Statistics Guest Agent Data
Alerts (1) Events Tasks (0)

Name	Host	IP Address	FQDN	Cluster	Data Center
local_vm	local_host			local_cluster	local_datacer
test	local_host			local_cluster	local_datacer

Name
nic1

Rx (Mbps)	Tx (Mbps)	Drops (Pkts)
< 1	< 1	0

Last Message: 2014-Jul-30, 09:51 User engine failed to log in.

Figure 4.19 – oVirt Web Tool

(OVIRT, 2014)

Addressing with Maker. Once calculated $T_{nomaker}$, it is computed the time-consuming of dealing with the *NMSit-VN* by developing, launching, and using $VNMM$. Such time-consuming is $T_{maker} = T_{dev:maker} + T_{lau:maker} + T_{use:vnmm}$. According the equation 3.5 and considering the no conducting of Tune, $T_{dev:maker} = T_{sel:maker} + T_{con:maker} + T_{com:maker}$.

Select on Maker. The Network Administrator defines R_{used} by selecting the Visual Resources that form $VNMM$: (*i*) drag-and-drop *Xen Server*, (*ii*) drag-and-drop *Virtual Box Server*, (*iii*) drag-and-drop *Virtual Box Server*, (*iv*) drag-and-drop *Monitoring Panel*, (*v*) drag-and-drop *Floodlight*, (*vi*) drag-and-drop *RRDTool*, (*vii*) drag-and-drop *OF Monitor*; and (*viii*) drag-and-drop *Integrator*. In this way, considering the equation 3.6, $T_{sel:maker} = \sum_1^8 dnd = 10.4s$.

Configure on Maker. The Network Administrator creates R_{conf} by providing the functioning parameters of (2) *Virtual Box Server*, *Xen Server*, *Floodlight*, and *RRDTool*. As the Network Administrator writes manually these parameters, $\tau_{vbox} = \tau_{xen} = \tau_{floodlight} = [4 * (p+h+2b) + (16+8+8+5) * k] = 14.2s$ and $\tau_{rrd} = p + h + 2b + 3k = 2.3s$. Using these values in the equation 3.7, $T_{con:maker} = 59.1s$.

Combine on Maker. The Network Administrator defines the δ of $VNMM$ as follows: (*i*) wire *Virtual Box Server1 - Monitoring Panel*, (*ii*) wire *Virtual Box Server2 - Monitoring Panel*, (*iii*) wire *Xen Server - Monitoring Panel*, (*iv*) wire *Floodlight - OF Monitor*, (*v*) wire *RRDTool - OF Monitor*, (*vi*) wire *OF Monitor - Integrator*; and (*vii*) wire *Monitoring Panel - Integrator*. Therefore, according the equation 3.8, $T_{com:maker} = \sum_1^7 wire = 28.7s$.

Launch on Maker. Before requesting the execution of $VNMM$, the Network Administrator conducts the following actions sequence to save 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 "$VNMM$". Once $VNMM$ has been saved, the Network Administrator launches it by clicking the button Run. Thus, $T_{lau:maker} = 3(h + p + 2b) + 4k = 5.9s$.

On runtime, $VNMM$ allows Network Administrators to retrieve information about the *NMSit-VN*. The Network Administrator conducts in $VNMM$ the following actions sequence to obtain detailed information about a guest (see Figure 4.17): (*i*) point the mouse to the Nodes list, (*ii*) press and release the mouse to select a node, (*iii*) point the mouse to a guest of the Node structure, (*iv*) press and release the mouse to retrieve general information about such guest, (*v*) point the mouse to the identifier of the guest selected; and (*vi*) press and release the mouse to obtain details about the memory, processor, and network of the selected guest. According this sequence, $T_{guest:vnmm} = h + 3p + 6b = 4.3s$. Carrying out a similar analysis, the time-consuming to obtain traffic information about a switch handled by Floodlight is $T_{traffic:vnmm} = h + 3p + 6b = 5.3s$. Because $T_{traffic:vnmm}$ is greater than $T_{guest:vnmm}$, $T_{use:vnmm} = 5.3s$.

Figure 4.20 depicts the obtained results in the time-consuming assessment when the Network Administrator addresses the *NMSit-VN* with and without the Maker. According these results: (*i*) $T_{dev:maker} = 98.2s$ is less than when the Maker is not used $T_{dev:nomaker} = 419.2s$, attained by the services Drag-and-Drop and Wire (*ii*) because every mashment must be saved in the Maker before being executed, $T_{lau:maker} = 5.9s$ is greater than without Maker ($T_{lau:nomaker} = 3.9s$); and (*iii*) $T_{maker} = 109.4s$ is less (about 75.1%) than $T_{nomaker} = 440.8s$. This global result and the per task results demonstrate that, in terms of time-consuming, it is feasible to use the proposed approach for handling nmsits like the raised *NMSit-VN*.

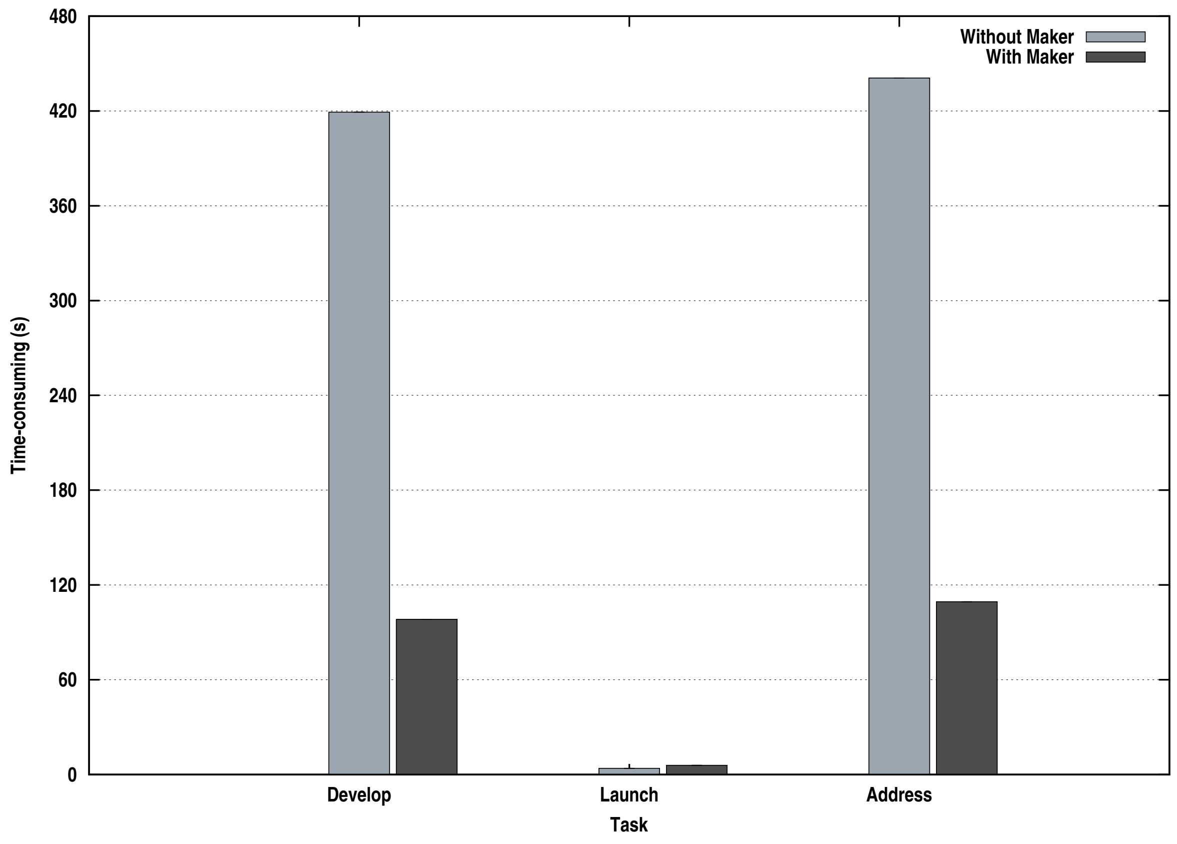


Figure 4.20 – Time-consuming on NMSit-VN

4.3.3 Time-response: results and analysis

VNMM and Monitoring Script. To continue the evaluation, it is proceeded to measure the time-response of $VNMM$ (see Figure 4.17) and *Monitoring Script* when carrying out the operations *NodeList*, *GuestFeatures*, *GuestStats*, and *NodeStructure*. It is important to note that the operations *SwitchesList*, *LinksList*, and *FlowsList* were already evaluated in the previous case study. In this time-response evaluation, measurements were took in virtual nodes with all virtual switches and virtual machines in active (*i.e.*,running on) state.

Table 4.1 presents the time-response evaluation results of the operations *NodeList*, *Guest-Features*, and *GuestStats* of $VNMM$ and *Monitoring Script* when used to monitor three heterogeneous virtual nodes. The first one, a $xenpool$ formed by two XenServers that support seven virtual machines of Linux Ubuntu O.S. and three Open vSwitch/Debian. Each one of others virtual nodes was composed of a $vbox$/Debian that supports two guests: one Open vSwitch/Debian and a virtual machine of Linux Ubuntu O.S. Considering again the ranking (optimal, good, admissible, and deficient) of time-response (r in ms), these results reveal about $VNMM$: (*i*) *NodeList* and *GuestStats* (*i.e.*, *GuestStatsXen* or *GuestStatsVB*) have a good r for virtual nodes based on XenServer and VirtualBox; and (*ii*) *GuestFeatures* (*i.e.*, *GuestFeaturesXen* and *Guest-*

FeaturesVB) has an optimal r for virtual nodes based on XenServer and VirtualBox. As expected, *NodeList* has the highest r because this operation integrates information of all virtual nodes rather than *GuestStats* and *GuestFeatures* that retrieve information from only one type of virtual node.

The evaluation results (see Table 4.1) also disclose that in relation to the operations *NodeList*, *GuestStats*, and *GuestFeatures*, *Monitoring Script* has better r (approx 10%) than $VNMM$; this result was expected because mashments use additional software layers to collect, aggregate, and present monitoring information from different virtual nodes. Notwithstanding this result, $VNMM$ has a good behavior in time-response because its r is still ranked as optimal or good.

Table 4.1 – Time-response of VNMM and Monitoring Script

	XenServer-based Node		VirtualBox-based Node	
Operation	VNMM (ms)	Monitoring Script (ms)	VNMM (ms)	Monitoring Script (ms)
NodeList	788 ± 30	725 ± 35	788 ± 30	725 ± 35
GuestFeatures	190 ± 3	174 ± 4	63 ± 3	61 ± 2
GuestStats	372 ± 4	340 ± 13	600 ± 15	550 ± 14

Once assessed the time-response of the operations *NodeList*, *GuestStats*, and *GuestFeatures* of $VNMM$ and *Monitoring Script*, it is proceeded to evaluate the time-response of the operation *NodeStructure* of $VNMM$. In this evaluation, in each virtual node, the number of guests was varied from 1 to 64. When the number of guests was ≤ 8, all guests were used in active state, otherwise, 8 guests were used in such a state and the others in inactive (*i.e.*, turned off) state.

Figure 4.21 depicts the time-response of the operation *NodeStructure* of $VNMM$. According these results, *NodeStructure* has a good r for virtual nodes based on XenServer and VirtualBox. This operation has better r for $xenpool$ than $vbox$/Debian when the number of guests is increased from 32 to 64. r of *NodeStructure* for $xenpool$ increases 3.1 ms per guest, and, for $vbox$/Debian, it grows 5.9 ms. In this way, if the number of guests is equal or greater than 48, r for $xenpool$ is better than for $vbox$/Debian. This behavior of r occurs because, when there is a large number of guests in the Managed Resources Layer, HTTP connections of the XenSDK are more efficient than SOAP/HTTP connections of the VirtualBox Web Service.

Although $VNMM$ uses additional software modules (*e.g.*, Mediator Bus) to retrieve, integrate, and present information from non homogeneous virtual nodes, its worst behavior in time-response is still ranked as good. Furthermore, it is important to highlight that such behavior is similar (extra time $< 10\%$) to that of *Monitoring Script*. Summing up, these time-response behaviors corroborate that, in terms of this metric, it is feasible to use the proposed approach for dealing with nmsits like the raised *NMSit-VN*.

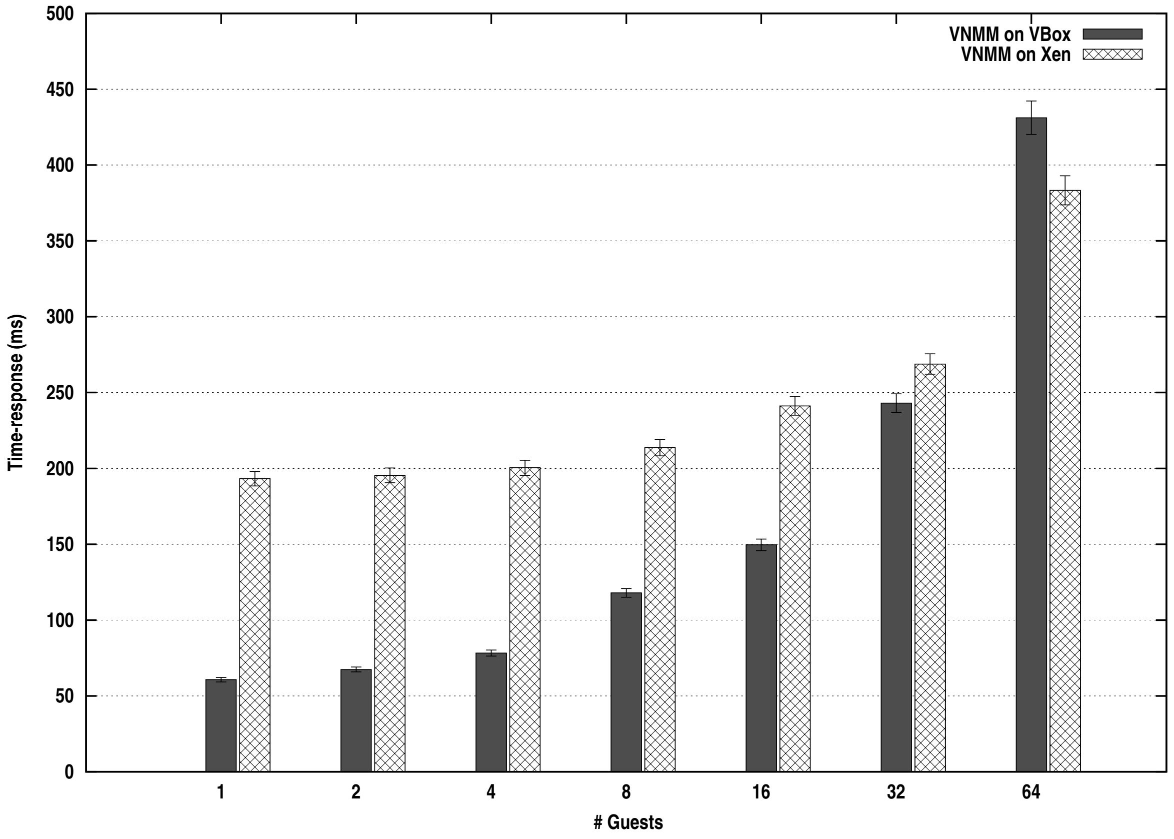


Figure 4.21 – Time-response on NodeStructure

4.3.4 Traffic: results and analysis

VNMM and Monitoring Script. To continue the evaluation, it is proceeded to measure the network traffic generated by $VNMM$ (see Figure 4.17) and *Monitoring Script* when used to monitor the virtual nodes before described. In this evaluation, the measurements in the virtual nodes were took with all virtual switches and virtual machines in active state.

Table 4.2 presents the traffic evaluation results of the operations *NodeList*, *GuestFeatures*, and *GuestStats* of $VNMM$ and *Monitoring Script*. These results reveal about $VNMM$: (*i*) *NodeList* and *GuestFeatures* generates low traffic for virtual nodes based on XenServer and VirtualBox; and (*ii*) *GuestStats* generates more traffic for XenServer than VirtualBox; it is because the XenSDK provides statistics by using a large XML document that contains information about all guests, instead, VirtualBox Web Service provides information in a XML document that contains only information about a specific guest.

As expected, the traffic evaluation results (see Table 4.2) also discloses that in relation to the operations *NodeList*, *GuestStats*, and *GuestFeatures*, $VNMM$ generates more network traffic than *Monitoring Script*. Although mashments use additional data to aggregate information from different virtual nodes, the difference of traffic generated by *Monitoring Scrip* and $VNMM$ is less than 10%. In this way, the evaluation results of the above operations confirm that $VNMM$ has a good behavior in traffic.

Table 4.2 – Network traffic of VNMM and Monitoring Script

	XenServer-based Node		VirtualBox-based Node	
Operation	VNMM (Bytes)	Monitoring Script (Bytes)	VNMM (Bytes)	Monitoring Script (Bytes)
NodeList	487	450	487	450
GuestFeatures	459	456	351	318
GuestStats	3061	2800	155	153

Once assessed the traffic generated by *NodeList*, *GuestStats*, and *GuestFeatures* of $VNMM$ and *Monitoring Script*, it is proceeded to evaluate the traffic generated by the operation *NodeStructure* of $VNMM$. In this evaluation, in each virtual node, the number of guests was varied from 1 to 64. If the number of guests was ≤ 8, all guests were used in active state, otherwise, 8 guests were used in such a state and the others in inactive state.

Figure 4.22 depicts the traffic results of the operation *NodeStructure* of $VNMM$. According these results, this operation generates more traffic for $vbox$/Debian than $xenpool$ when the number of guests is 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 by RESTful-based Web Services.

Although $VNMM$ uses data additional to aggregate information from different virtual nodes, the evaluation results reveal that it has a good behavior in network traffic. Furthermore, it is relevant to mention that such behavior is similar to that of *Monitoring Script*. Summarizing, the traffic evaluation results corroborate that, in terms of such metric, it is feasible to use the proposed approach for coping with nmsits like the raised *NMSit-VN*.

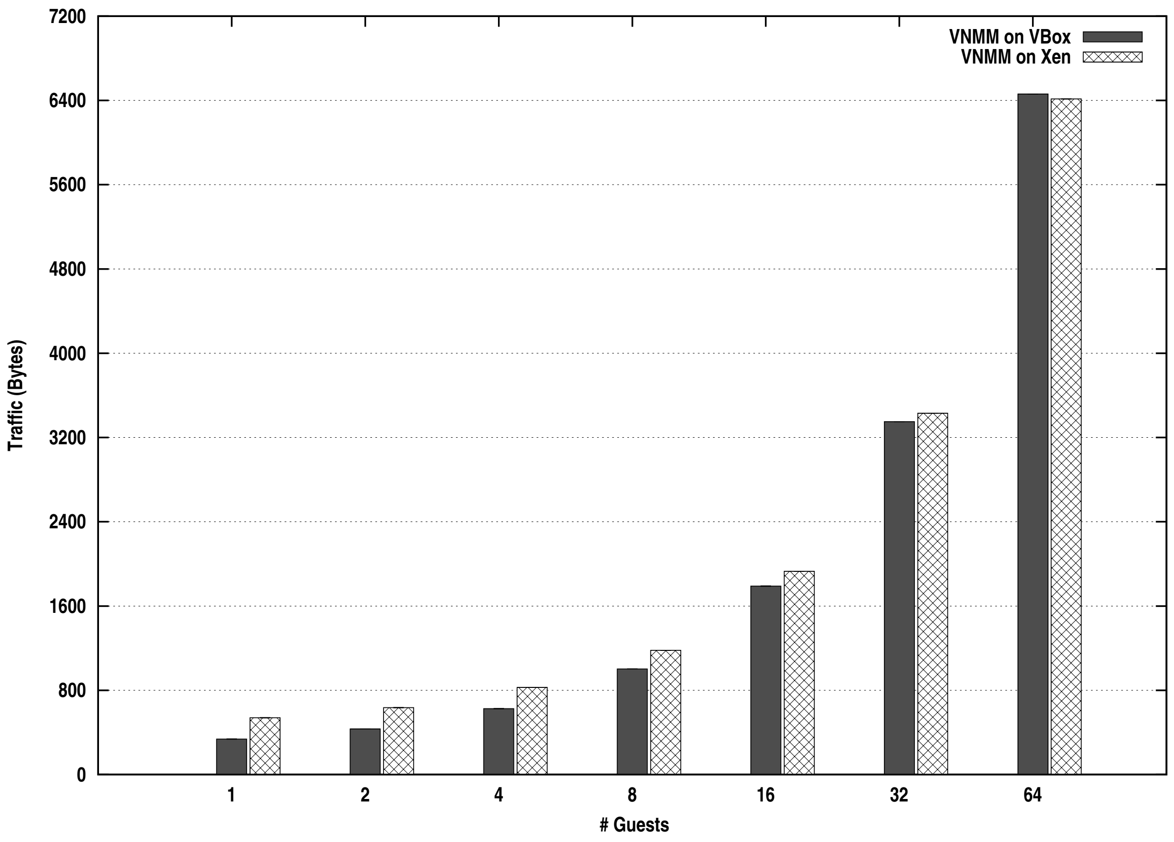


Figure 4.22 – Network traffic on NodeStructure

4.4 Case study for dynamic mashments

The case study for dynamic mashments is formed by a test environment (see Figure 4.23), several nmsit patterns (*i.e.*, instances of nmsit model), and experiments conducted to evaluate: (*i*) the time that the Mashment Maker takes for detecting nmsits (*i.e.*, time-recognition), (*ii*) the time that the Maker takes for customizing composition templates (*i.e.*, time-composition); and (*iii*) the impact, in terms of complexity and time-consuming, of mechanisms for automatic recognition of nmsits and dynamic composition of mashments in addressing *NMSit-SDN*.

Test environment. Every OpenFlow controller (*i.e.*, Beacon 1.0.2, POX 1.0.0, and Floodlight 0.9) was executed on a machine with 2.33 Ghz core 2 duo processor, 2 GBytes RAM, and 160 GBytes hard disk. The Mashment System Server and Mashment Maker were deployed on a Tomcat using a Web engine 7.0.26 and a Drools engine 6.1. The MashmentDB was deployed using a MySQL Server 5.5. The Mashment System Server, Maker, and MashmentDB were executed on a machine with Linux Ubuntu O.S., 2.53 GHz Intel Core i5 processor, 4 GBytes RAM, and 250 GBytes hard disk. The virtual Open vSwitches 1.4 (running on Mininet and handled by the above referred controllers) were deployed on a server with 8 GBytes RAM and 3.4 GHz core i7 processor. The test applications used in this case study were deployed on a

Client with 2 GBytes RAM and 2.53 GHz core 2 duo processor.

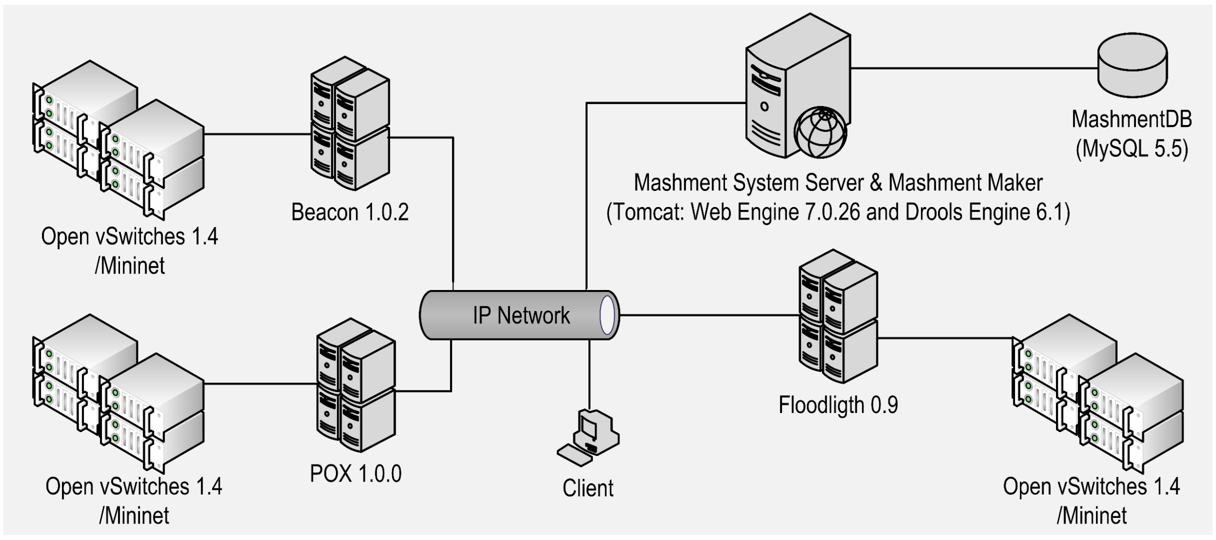
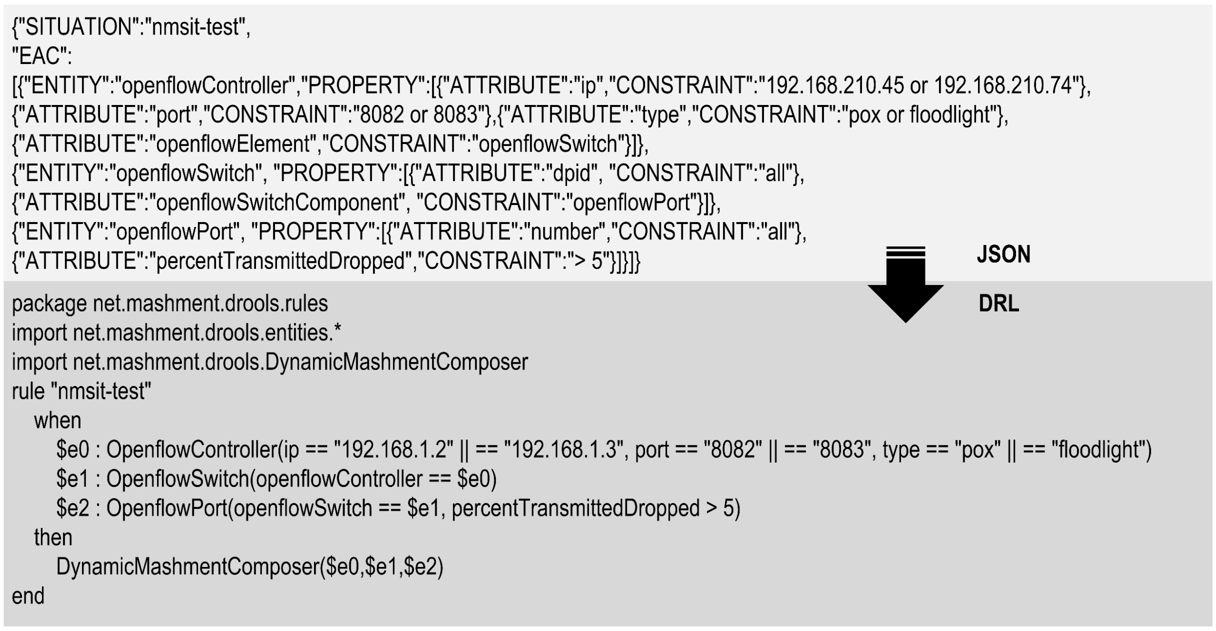

Open vSwitches 1.4 /Mininet
Beacon 1.0.2
Mashment System Server & Mashment Maker (Tomcat: Web Engine 7.0.26 and Drools Engine 6.1)
MashmentDB (MySQL 5.5)
IP Network
Floodligth 0.9
Open vSwitches 1.4 /Mininet
POX 1.0.0
Client
Open vSwitches 1.4 /Mininet

Figure 4.23 – Test environment - dynamic mashments

4.4.1 Time-recognition: results and analysis



```
{"SITUATION":"nmsit-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"}]}]}
```

```
package net.mashment.drools.rules
import net.mashment.drools.entities.*
import net.mashment.drools.DynamicMashmentComposer
rule "nmsit-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
```

Figure 4.24 – Example of nmsit for testing

The time-recognition is the time that the Mashment Maker takes for detecting nmsit patterns. In the time-recognition evaluation, two different controllers (*i.e.*, POX and Floodlight) were used; each one handling 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.

Summarizing, in total, 2 controllers, 518 switches, and 3626 ports were used in this evaluation.

In the time-recognition evaluation, by using the NMSit Designer, a nmsit pattern was defined for recognizing when any port of any switch handled by POX or Floodlight had more than 5% of dropped packets. Figure 4.24 depicts such pattern encoded on JSON and DRL. It is important to highlight that the translation from JSON to DRL is conducted by the Automatic NMSit Recognizer and hidden for Network Administrators and Mashment Creators.

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

Figure 4.25 presents the time-recognition results. These results reveal that the Mashment Maker is able to recognize nmsits in a short time; in the worst behavior approximately 30.3 ms for detecting 3500 nmsits having 400 loaded rules/patterns. Furthermore, the time-recognition is negligibly increased with the growth of the generated nmsits and the loaded rules. Consequently, the above results corroborate that, in terms of time-recognition, it is feasible to use the mashment-based approach for coping effectively with nmsits.

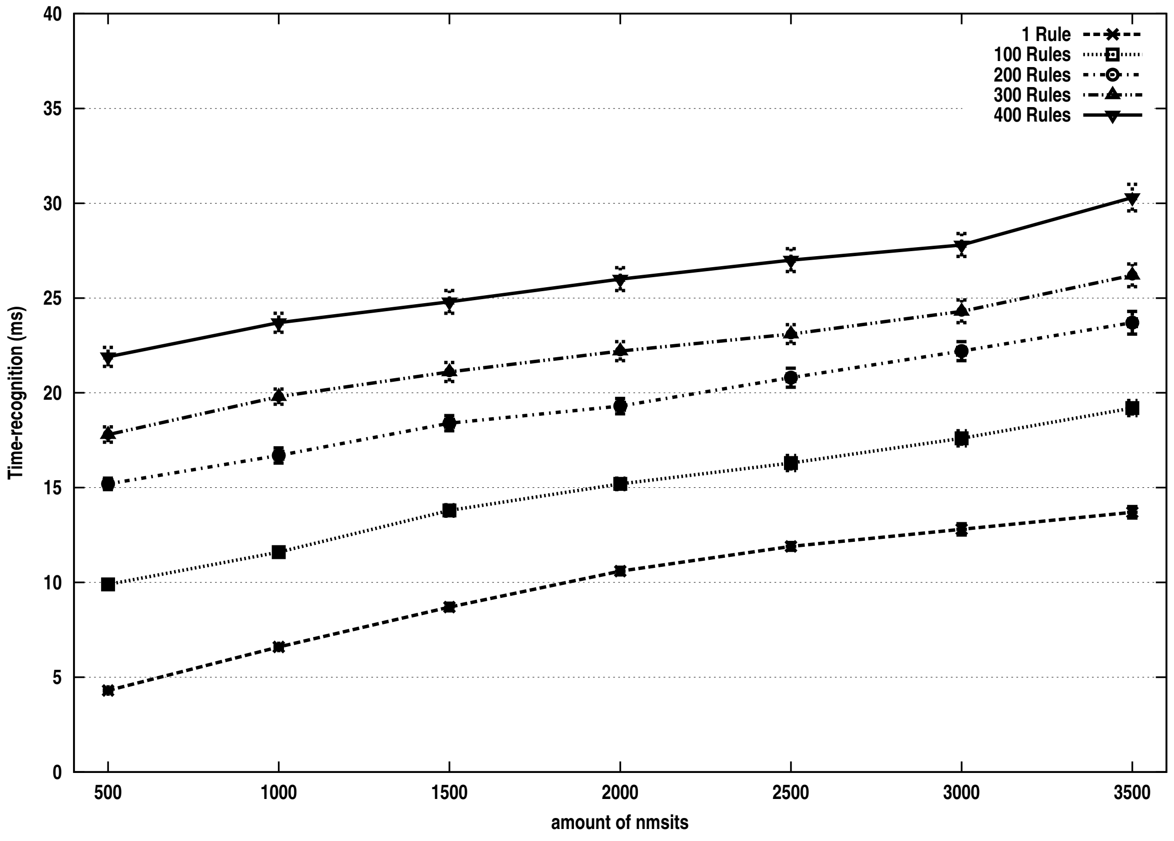
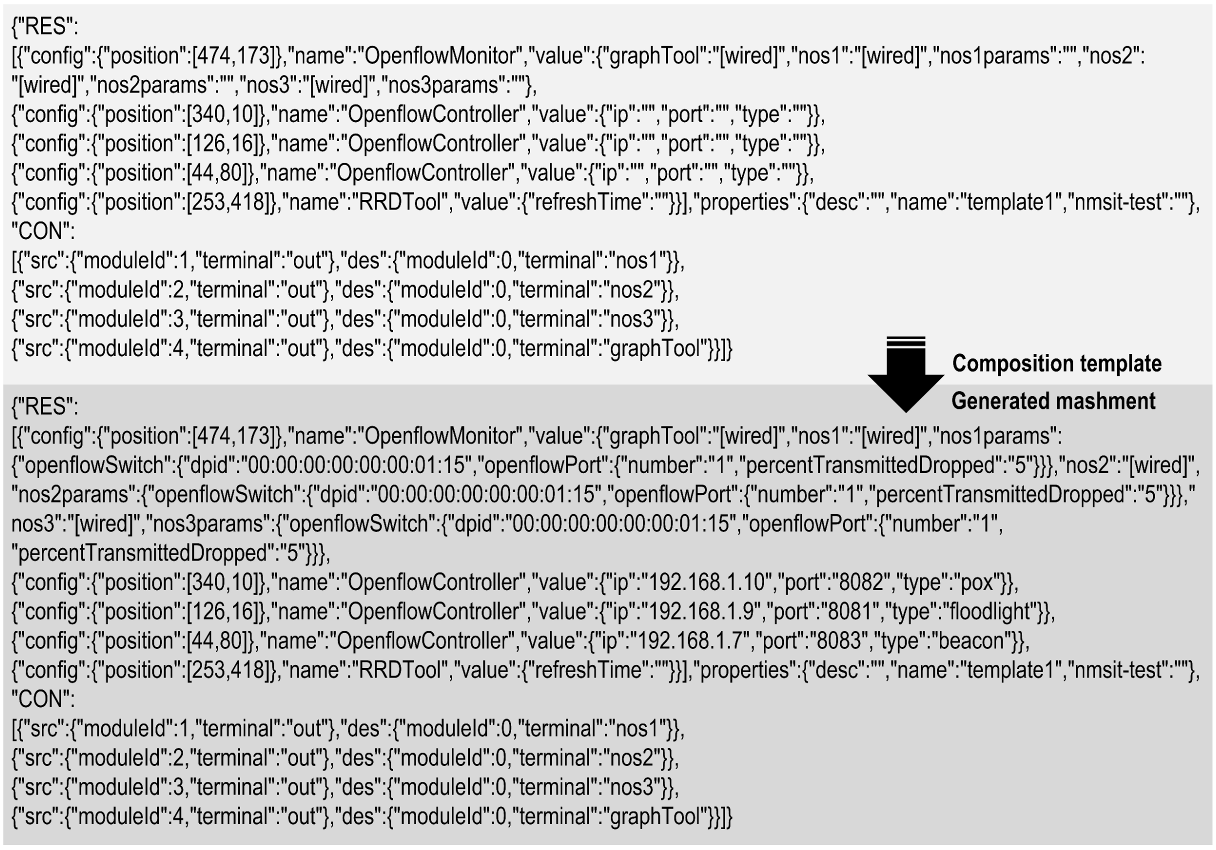


Figure 4.25 – Time-recognition behavior

4.4.2 Time-composition: results and analysis

The time-composition is the time that the Mashment Maker (*i.e.*, specifically, the Dynamic Mashment Composer) spends for customizing composition templates and, so, generating dynamic mashments. In the time-composition evaluation, three different controllers (*i.e.*, POX, Floodlight, and Beacon) were used; each one handling a datacenter network topology with 259 switches distributed in 4 levels of depth and 6 servers per rack. Summarizing, in total, 3 controllers, 777 switches, and 4662 ports were used in this evaluation.



```
{"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"}}]}
```

```
{"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"}}]}
```

Figure 4.26 – Example of composition template and generated mashment

In the time-composition evaluation, by using the Template Designer, composition templates were defined for monitoring when ports of switches handled by POX, Floodlight, and/or Beacon had more than 5% of dropped packages. Figure 4.26 depicts a snippet of a composition template and the corresponding dynamic mashment generated by the Mashment Maker.

In the time-composition evaluation, initially, the number of resources forming the composition templates was varied from 2 to 8; it is to noteworthy that more than 60% of mashups consist of $3-8$ modules/resources (HUANG et al., 2014). Subsequently, the number of templates was also modified from 10 to 50; it is important to highlight that the amount of templates defines the number of simultaneously generated dynamic mashments.

Figure 4.27 presents the time-composition results. These results reveal that the Mashment Maker is able to generate mashments by customizing composition templates in a short time; in the worst behavior, approximately $14500\ ms$ for composing 50 dynamic mashments formed by 8 resources. Furthermore, the time-composition is linearly increased with the growth of both the number of simultaneously generated mashments and the amount of resources per composition template.

Considering, first, the above results. Second, the mechanism for generating dynamic mashments had similar time-composition behavior to composition proposals introduced on other application domains (ORDONEZ et al., 2014). It can state that the time-composition evaluation corroborates, in terms of such metric, the feasibility of using the mashment-based approach for dealing effectively with nmsits.

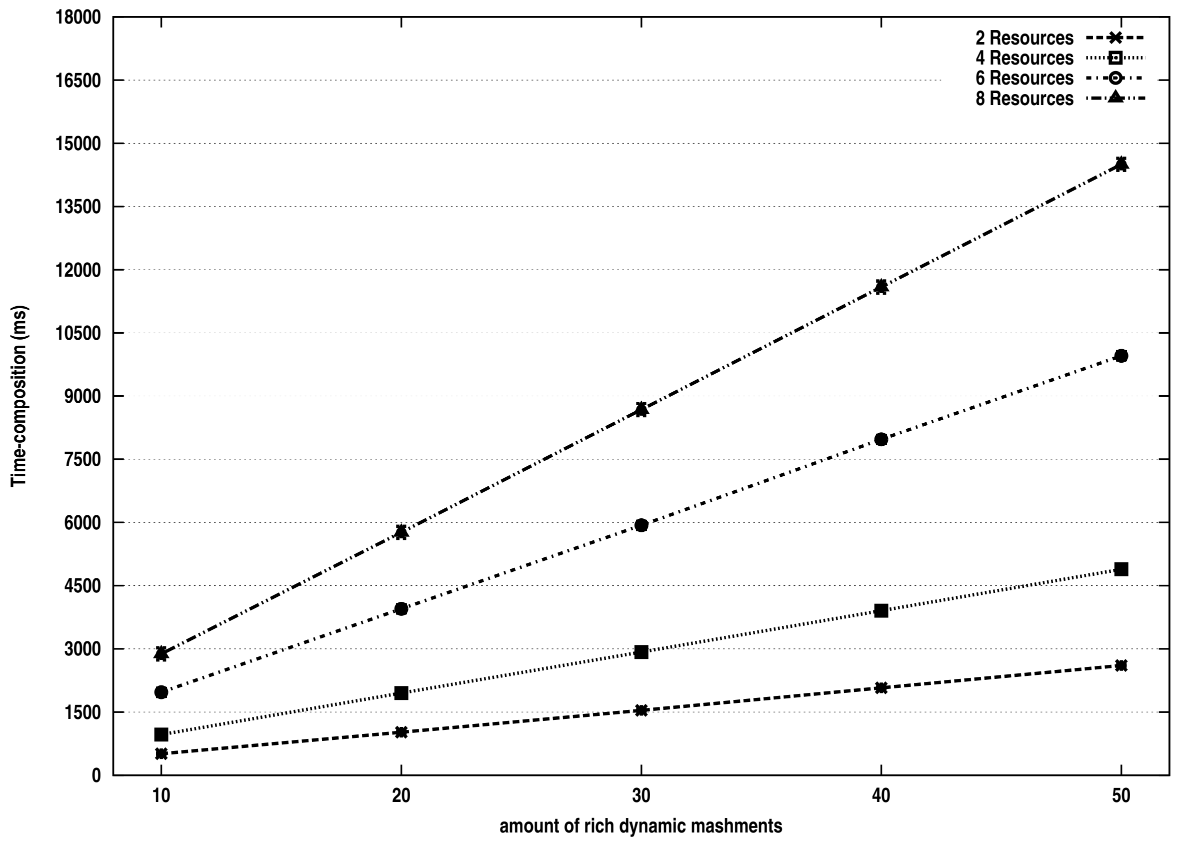


Figure 4.27 – Time-composition behavior

4.4.3 Complexity: results and analysis

The *NMSit-SDN* was once again raised in order to evaluate the complexity when the mashment-based approach uses the mechanisms for automatic recognition of nmsits and dynamic composition of mashments. To deal with the *NMSit-SDN*, the Network Administrator tests several options: (*i*) *Situational Script*, (*ii*) PMM; and (*iii*) the dynamic mashment of performance, hereinafter called $RDMP$. In brief, the *Situational Script* is an application programmed and executed by the Network Administrator in a low-abstraction level. PMM is a composite situational solution developed and launched by the Network Administrator in the Mashment Maker. $RDMP$ is a dynamic mashment generated by the Maker and launched by the Network Administrator. Regarding $RDMP$ is to noteworthy that, on runtime, it offers the same functionalities as PMM.

Addressing without Maker. The whole process of measuring the complexity on handling *NMSit-SDN* by *Situational Script* was detailed in the Subsection 4.2.1. In this way, in this evaluation, the corresponding complexity results are directly brought: (*i*) $\zeta_{sel:nomaker} = 47$, (*ii*) $\zeta_{con:nomaker} = 42$, (*iii*) $\zeta_{com:nomaker} = 21$; and (*iv*) $\zeta_{lau:nomaker} = 2$.

Addressing with PMM. The entire process of assessing the complexity on addressing *NMSit-SDN* by PMM was also presented in detail in the Subsection 4.2.1. Thus, in this evaluation, the respective complexity results are directly retrieved: (*i*) $\zeta_{sel:pmm} = 31$, (*ii*) $\zeta_{con:pmm} = 26$, (*iii*) $\zeta_{com:pmm} = 12$; and (*iv*) $\zeta_{lau:pmm} = 1$.

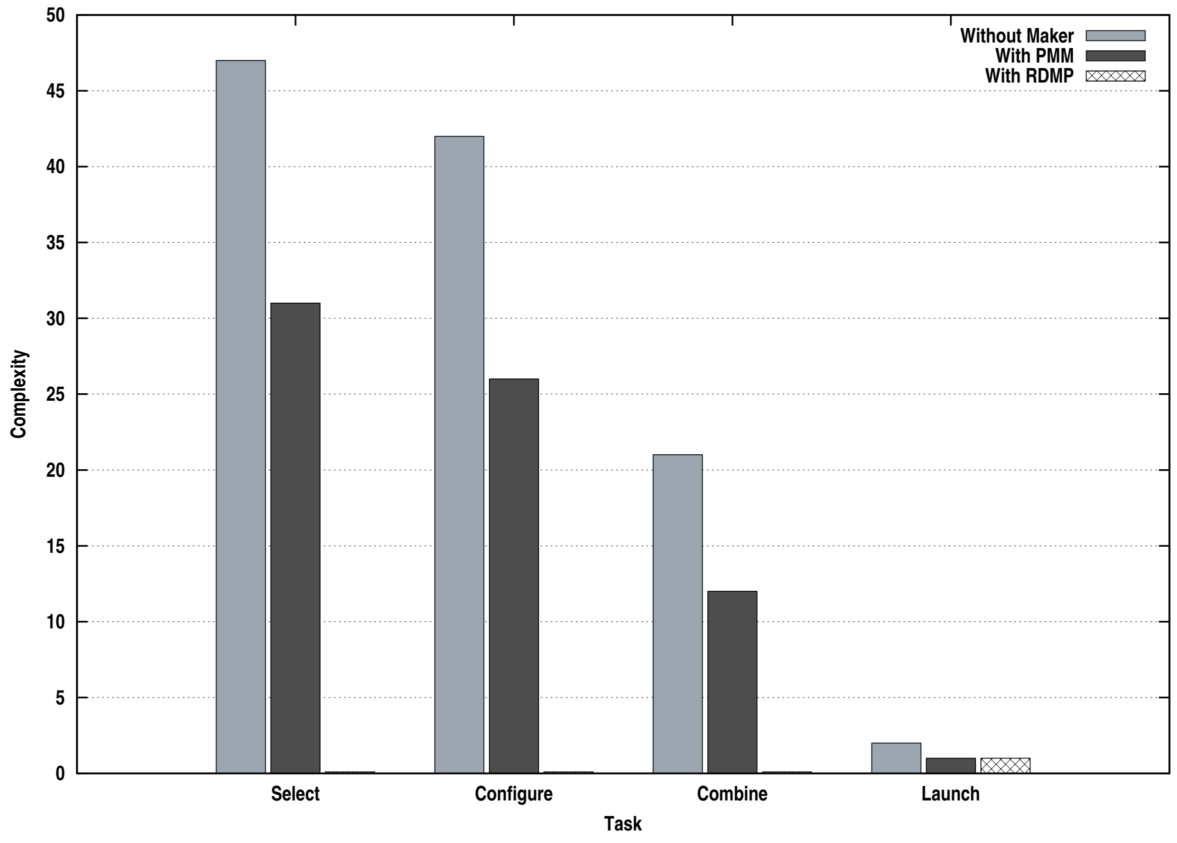


Figure 4.28 – Complexity - dynamic mashments

Addressing with RDMP. Once calculated the complexity on addressing *NMSit-SDN* without the Maker and with PMM, it is proceeded to compute the corresponding complexity with

$RDMP$. As $RDMP$ is generated by the Mashment Maker: (*i*) $\zeta_{sel:rdmp} = 0$, (*ii*) $\zeta_{con:rdmp} = 0$; and (*iii*) $\zeta_{com:rdmp} = 0$. Furthermore, since $RDMP$ is automatically stored by the Maker, the Network Administrator is able to launch this dynamic mashment simply by clicking the button Run and, so, $\zeta_{lau:rdmp} = 1$.

Figure 4.28 depicts the results of the complexity evaluation carried out in this case study. These results reveal: (*i*) the complexity that the Network Administrator perceives in developing (*i.e.*, Select, Configure, Combine) $RDMP$ is less than in PMM and the *Situational Script*, attained by mechanisms for automatic recognition of nmsits and dynamic composition of mashments; and (*ii*) as every generated mashment is ready to be launched by the Mashment Maker, the complexity perceived in launching $RDMP$ ($\zeta_{lau:rdmp} = 1$) is equal than in PMM ($\zeta_{lau:pmm} = 1$) and less than in *Situational Script* ($\zeta_{lau:nomaker} = 2$). Summarizing, the complexity for addressing *NMSit-SDN* with $RDMP$ ($\zeta_{rdmp} = 1$) is less than with PMM (about 99.98%) and with *Situational Script* (about 99.99%). This global result and the results per task corroborate that, in terms of complexity, the mechanisms for automatic recognition of nmsits and dynamic composition of mashments improve the effectivity of the mashment-based approach.

4.4.4 Time-consuming: results and analysis

The *NMSit-SDN* was once again raised in order to evaluate the time-consuming when the mashment-based approach uses the mechanisms for automatic recognition of nmsits and dynamic composition of mashments. To deal with the *NMSit-SDN*, the Network Administrator tests several options: (*i*) *Situational Script*, (*ii*) PMM; and (*iii*) $RDMP$.

Addressing without Maker. The whole process of measuring the time-consuming on addressing *NMSit-SDN* with the *Situational Script* was detailed in the Subsection 4.2.2. In this way, in this evaluation, the corresponding time-consuming results are directly brought: (*i*) $T_{dev:nomaker} = 298.8s$, (*ii*) $T_{lau:nomaker} = 3.9s$, (*iii*) $T_{use:nomaker} = 16.8s$; and (*iv*) $T_{nomaker} = 319.5s$.

Addressing with PMM. The entire process of assessing the time-consuming on coping with *NMSit-SDN* by PMM was also presented in detail in the Subsection 4.2.2. Thus, in this evaluation, the respective time-consuming results are directly retrieved: (*i*) $T_{dev:pmm} = 67.8s$, (*ii*) $T_{lau:pmm} = 5.7s$, (*iii*) $T_{use:pmm} = 5.3s$; and (*iv*) $T_{pmm} = 78.8s$.

Addressing with RDMP. Once calculated $T_{nomaker}$ and T_{pmm}, it is proceeded to compute the time-consuming of dealing with *NMSit-SDN* by $RDMP$. As $RDMP$ is generated by the Mashment Maker, $T_{rdmp} = T_{lau:rdmp} + T_{use:rdmp}$. Considering that $RDMP$ is automatically stored by the Maker, the Network Administrator is able to launch this mashment simply by clicking the button Run and, thus, $T_{lau:rdmp} = h+p+2b = 1.7s$. Furthermore, since on runtime $RDMP$ and PMM provide identical functionalities and show the same GUI (see Figures 4.4 and 4.7), $T_{use:rdmp} = 5.3s$. Taking into account $T_{lau:rdmp}$ and $T_{use:rdmp}$, $T_{rdmp} = 6s$.

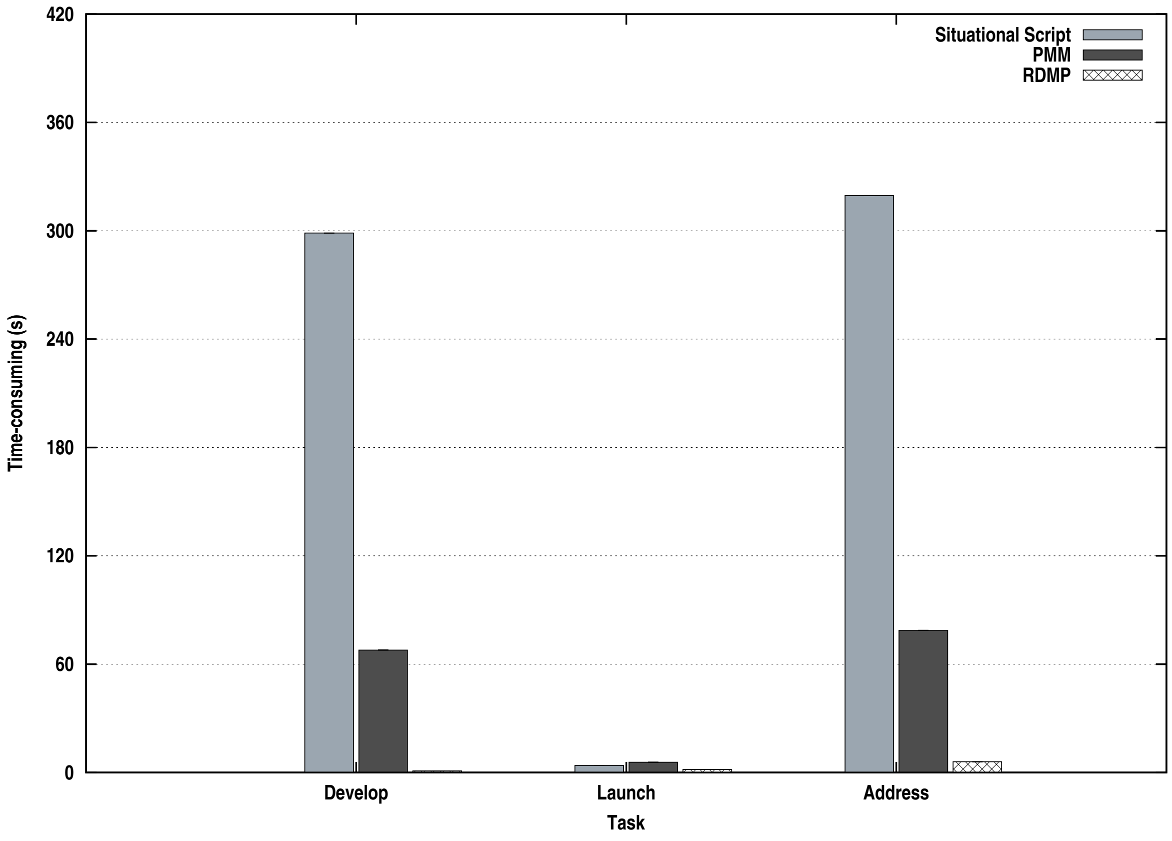


Figure 4.29 – Time-consuming - dynamic mashments

Figure 4.29 depicts the results of the time-consuming evaluation carried out in this case study. These results reveal that: (*i*) the time that the Network Administrator takes in developing $RDMP$ is less than in PMM and the *Situational Script*, attained by mechanisms for automatic recognition of nmsits and dynamic composition of mashments; and (*ii*) because every dynamic mashment is ready to be launched by the Mashment Maker, the time for launching $RDMP$ ($T_{lau:rdmp} = 1.7s$) is less than for the *Situational Script* ($T_{lau:nomaker} = 3.9s$) and PMM ($T_{lau:pmm} = 5.7s$). In sum up, the time-consuming for coping with *NMSit-SDN* by $RDMP$ ($T_{rdmp} = 6s$) is less than by PMM (about 92.3%) and the *Situational Script* (about 98.1%). This global result and the results per task corroborate that, in terms of time-consuming, the mechanisms for automatic recognition of nmsits and dynamic composition of mashments enhance the effectivity of the approach proposed in this thesis.

4.5 Final remarks

This chapter presented the evaluation and analysis of addressing, with and without the mashment-based approach, diverse nmsits raised in three case studies. Such evaluation and analysis was carried out in terms of following metrics: (*i*) complexity and time-consuming that are related to the process defined for facing nmsits, (*ii*) time-response and traffic that are associated with the behavior on runtime of solutions used to handle nmsits; and (*iii*) time-recognition and time-composition that are related to the mechanisms for automatic recognition of nmsits and dynamic composition of mashments, respectively.

The evaluation results revealed several facts. First, if Network Administrators coping with nmsits (*e.g.*, *NMSit-SDN* and *NMSit-VN*) by developing and launching static mashments, the complexity and time-consuming are decreased. Second, such decreasing is greater when mechanisms for automatic recognition of nmsits and dynamic composition of mashments are used (*e.g.*, $RDMP$). The above facts demonstrate that the mashment-based approach allows Network Administrators to address and overcome the complexity and time-consuming of nmsits. Third, although mashments (*e.g.*, PMM and $VNMM$) use extra software layers to face nmsits, such layers generate few additional time-response and traffic in relation to Web-based network management tools (*e.g.*, Beacon Web Tool) and proprietary scripts. This last fact demonstrates that the proposed approach has a good behavior in terms of time-response and network traffic.

In sum up, the evaluation results demonstrate that, in terms of complexity, time-consuming, time-response, and traffic, it is feasible to use the proposed approach for dealing effectively with nmsits. In this sense, such results confirming the relevance of: the concepts of mashment and nmsit, the Mashment Ecosystem, the process to develop and launch mashments, the Mashment System Architecture (including mechanisms for automatic recognition of nmsits and dynamic composition of mashments), and the Mashment Maker.

From a qualitative point of view, the main characteristics provided by the proposal introduced in this thesis are the flexibility and extensibility. The flexibility refers to that the mashment-based approach 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, $VNMM$, and $RDMP$) because the proposed approach provides a high-level abstraction of system and network virtualization technologies as well as of situational management operations; these technologies and operations are represented in a visual way as mashupable 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-to-use way for user-centric service composition (LIU et al., 2007).

Regarding the extensibility, with the proposal introduced in this thesis, 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 the mashment-based approach leverages the composition, abstraction, and reusing models from mashups as well as allows to implement the investigative and control aspects of SM. In this regard, it is important to higlight that, in a general way, building up a mashup from existing applications is easier than developing it from scratch (TATEMURA et al., 2007) (HASAN et al., 2008). About the extensibility is also relevant to mention that Mashment Creators are also able to extend the Mashment Ecosystem by aggregating resources, such as mashments, nmsit patterns, and templates, to the Mashment Maker. Such extension, in turn, leads to the improvement of workspaces of Network Administrators.

According the evaluation results and the qualitative characteristics of the proposed approach, 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 carry out its investigative and control aspects. The mashup technology is led to a novel application domain located in the intersection of SM and network management.

5 CONCLUSIONS

This chapter starts summarizing the research work carried out in this thesis. Then, answers are provided for the fundamental questions raised to guide the verification of the hypothesis defended in this thesis. Afterwards, the main contributions achieved when conducting such verification are presented. Finally, directions for future work are outlined.

This thesis presented the investigation carried out to verify the hypothesis: **"The employment of SM and mashups provides an effective approach for network management"**. Based on the hypothesis, an approach that uses the SM discipline and the mashup technology to enable the fulfillment of network management tasks was proposed. Such approach is formed by the concepts of nmsit and mashment, the Mashment Ecosystem, the process to develop and launch Mashments, the Mashment System Architecture (including mechanisms for automatic recognition of nmsits and dynamic composition of mashments), and the Mashment Maker.

This thesis also presented the Reference Implementation of the mashment-based approach as well as an extensive evaluation and analysis about effectively addressing diverse nmsits with and without it. In particular, nmsits were raised in different case studies conducted in realistic scenarios, based on SDN and virtual nodes, and in addition the effectivity was evaluated and analyzed in terms of the complexity, consuming of time, traffic, and time of response. The evaluation results demonstrated that the proposed approach is effective for network management because: (*i*) when mashments were developed/generated and launched for carrying out network management tasks, both the complexity and the consuming of time of the work performed by network administrators were decreased; and (*ii*) on runtime, mashments had good behavior on the time of response as well as on the network traffic.

5.1 Answer for the fundamental questions

At beginning of this thesis, three fundamental questions were defined in order to guide the investigation about the feasibility of using SM and mashups as an effective approach for network management. Such questions are revised and answered in the following paragraphs.

Fundamental question I. *What is the performance, in terms of the complexity and consuming of time, of solutions that use SM and mashups for network management?*

> **Answer:** The work carried out by network administrators to address unexpected, dynamic, and heterogeneous situations that happen in the network management domain is complex and consumes a lot of time. The proposed approach permitted network adminis-

trators to overcome such complexity and time, confirming the importance of the concepts of nmsit and mashment, the Mashment Ecosystem, the process to develop and launch Mashments, the Mashment System Architecture, and the Mashment Maker. Using per-task metrics (KIERAS, 2001) (DIAO; KELLER, 2006) for ITSM, it is demonstrated that in terms of complexity and consuming of time, the mashment-based approach has a good performance. In fact, the proposed approach is less complex and consuming of time than proprietary and incompatible CLIs and GUIs currently used for coping with nmsits like the raised in the case studies. In particular, with the mashment-based approach the complexity was decreased approximately 37% in the case study on SDN-based networks and 20% in the case study on virtual nodes. In turn, in both case studies, the consuming of time was diminished about 75%.

Fundamental question II. *What is the performance, in terms of traffic and time of response, of solutions based on SM and mashups for network management?*

Answer: Although the mashment-based approach employs additional software entities and layers to handle unexpected, dynamic, and heterogeneous situations that happen in the network management domain, it has a good behavior in terms of time of response. In fact, in the case studies on SDN-based networks and virtual nodes, the time of response of mashments was according the performance analysis for Java-based Web sites (JOINES; WILLENBORG; HYGH, 2002) ranked as optimal or good. Similarly, the proposed approach has a good behavior in terms of network traffic because its additional entities and layers generate few extra traffic in relation to the solutions currently (multiple and incompatible Web-based network management tools and proprietary scripts) used for coping with nmsits. Specifically, in both case studies, the additional traffic was less than 10%.

Fundamental question III. *Which mechanisms could be employed to improve the performance of solutions that use SM and mashups for network management?*

Answer: The mechanisms for automatic recognition of nmsits and dynamic composition of mashments automate the Select, Configure, and Combine tasks of the proposed process for developing and launching mashments. As these mechanisms are able to recognize nmsits and compose mashments in a short time, they help to address the complexity and time involved in the carrying out of the above mentioned tasks. In particular, when the mechanisms for automatic recognition of nmsits and dynamic composition of mashments were jointly used in a SDN-based test environment, the complexity was approximately 99% and the consuming of time was about 92% less than when such mechanisms were not used. Consequently, it can state than these mechanisns improve, in terms of complexity and consuming of time, the performance of solutions that use the SM discipline and the mashup technology for network management.

5.2 Contributions

This thesis investigated the feasibility of using the SM discipline and the mashup technology as an effective approach for network management. The carrying out of such investigation led to the following major contributions.

- **The nmsit concept.** This concept introduced how to characterize unexpected, dynamic, and heterogeneous situations in the network management domain by SM.
- **The mashment concept.** This concept presented how to use mashups for carrying out the investigative and control aspects of SM in the network management domain.
- **The mashment ecosystem.** This ecosystem defined the resources, stakeholders, software entities, activities, and interactions involved in addressing nmsits.
- **The process to develop and launch mashments.** This process encompassed a simple set of high-level tasks (Select, Configure, Combine, Launch, and Tune) for facing nmsits.
- **The model of complexity and time-consuming.** This model allowed to measure the complexity and time of addressing nmsits with and without the proposed approach.
- **The mashment system architecture.** This architecture supported the carrying out of the ecosystem and the process aforementioned and, therefore, the making of mashments.
- **The mechanism for automatic recognition of nmsits.** This mechanism introduced how to detect nmsits on the fly by using rules and matching algorithms.
- **The mechanism for dynamic composition of mashments.** This mechanism presented how to dynamically generate mashments by using composition templates.

5.3 Future work

During the carrying out of this thesis, interesting opportunities for further research were observed. These opportunities are outlined below.

- Distributed approach. The mashment-based approach was implemented and analyzed in a centralized setting. Therefore, there is an opportunity to extend it by adding support for detecting nmsits and composing mashments in a distributed environment, in which, for instance, wrappers will be located near to resources implicated in addressing nmsits.

- Deployment cost model. The mashment-based approach was analyzed in terms of the complexity, consuming of time, traffic, and time of response, but it was not analyzed from a cost point of view. Thus, there is a chance to propose its deployment cost model, which should consider among other issues the cost of all resources involved in nmsits.
- Reference implementation extension. The current reference implementation of the approach based on mashments provides features for conducting monitoring tasks. Therefore, there is a chance to enhance and improve such implememtation for supporting other management tasks, such as configuration and accounting.

In addition to the research opportunities presented in the above list, around the mashment-based approach there are other investigation chances to be explored. First, the evaluation of the performance of such approach in the Network Function Virtualization and the Cloud Networking. Second, the use of big data techniques, such as Neural Networks and Exploratory Data Analysis, for automatic recognition of nmsits. Third, the use of other mechanisms, such as Hierarchical Task Networks and, in general, Artificial Intelligence planners, for dynamic composition of mashments.

REFERENCES

ADAMS, J.; REYNOLDS, C. A Complex Situational Management Application Employing Expert Systems. In: **International Conference on Systems, Man, and Cybernetics**. Nashville, USA: IEEE, 2000. v. 3, p. 1959–1964.

ANGELL, R. C.; FREUND, G. E.; WILLETT, P. Automatic Spelling Correction Using a Trigram Similarity Measure. **Information Processing & Management**, Elsevier, New York, USA, v. 19, n. 4, p. 255–261, 1983. ISSN 0306-4573.

BADGER, M. **Zenoss Core Network and System Monitoring**. Birmingham, UK: Packt, 2008.

BARHAM, P. et al. Xen and the Art of Virtualization. **SIGOPS Operating System Review**, ACM, New York, USA, v. 37, n. 5, p. 164–177, October 2003. ISSN 0163-5980.

BARROS, A.; DUMAS, M. The Rise of Web Service Ecosystems. **IT Professional Magazine**, IEEE Computer Society, Los Alamitos, USA, v. 8, n. 5, p. 31–37, September 2006. ISSN 1520-9202.

BARTH, W. **Nagios: System and Network Monitoring**. 2nd. ed. San Francisco, USA: No Starch Press, 2008. ISBN 1593271794.

BEZERRA, R. et al. On the Feasibility of Web 2.0 Technologies for Network Management: A Mashup-based Approach. In: **Network Operations and Management Symposium (NOMS)**. Osaka, Japan: IEEE, 2010. p. 487–494. ISSN 1542-1201.

BOLTE, M. et al. Non-intrusive Virtualization Management using Libvirt. In: **Design, Automation & Test in Europe Conference & Exhibition (DATE)**. Dresden, Germany: IEEE, 2010. p. 574–579. ISSN 1530-1591.

BROWNE, P. **JBoss Drools Business Rules**. Birmingham, UK: Packt, 2009. ISBN 1847196063, 9781847196064.

BRUNS, R. et al. Using Complex Event Processing to support data fusion for ambulance coordination. In: **International Conference on Information Fusion (FUSION)**. Salamanca, Spain: IEEE, 2014. p. 1–7.

BURNETT, M. et al. Forms/3: A First-order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm. **Journal of Functional Programming**, Cambridge University Press, Cambridge, UK, v. 11, p. 155–206, 3 2001. ISSN 1469-7653.

BUTLER, M. Android: Changing the Mobile Landscape. **IEEE Pervasive Computing**, IEEE Computer Society, Los Alamitos, USA, v. 10, n. 1, p. 4–7, January 2011. ISSN 1536-1268.

CAPPIELLO, C. et al. Information Quality in Mashups. **IEEE Internet Computing Magazine**, IEEE Computer Society, Los Alamitos, USA, v. 14, n. 4, p. 14–22, July-August 2010. ISSN 1089-7801.

CAPPIELLO, C. et al. DashMash: A Mashup Environment for End User Development. In: AUER, S.; DíAZ, O.; PAPADOPOULOS, G. (Ed.). **Web Engineering**. New York, USA:

Springer Berlin Heidelberg, 2011, (Lecture Notes in Computer Science, v. 6757). p. 152–166. ISBN 978-3-642-22232-0.

CHEN, X. et al. Declarative Configuration Management for Complex and Dynamic Networks. In: **Conference on emerging Networking EXperiments and Technologies (Co-NEXT)**. New York, USA: ACM, 2010. p. 6:1–6:12. ISBN 978-1-4503-0448-1.

CHEUNG, E.; PURDY, K. An Application Router for SIP Servlet Application Composition. In: **IEEE International Conference on Communications (ICC)**. Beijin, China: IEEE, 2008. p. 1802–1806.

CHIANG, C.-Y. et al. Enabling Distributed Management for Dynamic Airborne Networks. In: **International Symposium on Policies for Distributed Systems and Networks (POLICY)**. London, UK: IEEE, 2009. p. 102–105.

CHOWDHURY, N. M. M. K.; BOUTABA, R. Network virtualization: State of the Art and Research Challenges. **IEEE Communications Magazine**, IEEE Communications Society, New York, USA, v. 47, n. 7, p. 20–26, July 2009.

CLAYMAN, S. et al. Monitoring, Aggregation and Filtering for Efficient Management of Virtual Networks. In: **International Conference on Network and Service Management (CNSM)**. Paris, France: IEEE, 2011. p. 1–7.

CLAYMAN, S.; GALIS, A.; MAMATAS, L. Monitoring Virtual Networks with Lattice. In: **Network Operations and Management Symposium (NOMS)**. Osaka, Japan: IEEE, 2010. p. 239 –246.

COGET, J.-F. The Apple Store Effect: Does Organizational Identification Trickle Down to Customers? **The Academy of Management Perspectives**, Academy of Management, New York, USA, v. 25, n. 1, p. 94–95, 2011.

DAVIES, M. et al. AutoMashUpper: Automatic Creation of Multi-Song Music Mashups. **IEEE/ACM Transactions on Audio, Speech, and Language Processing**, IEEE, v. 22, n. 12, p. 1726–1737, Dec 2014. ISSN 2329-9290.

DIAO, Y.; KELLER, A. Quantifying the Complexity of IT Service Management Processes. In: . Berlin, Heidelberg: Springer-Verlag, 2006. (International conference on Distributed Systems: operations and management (DSOM)), p. 61–73. ISBN 3-540-47659-8, 978-3-540-47659-7.

DOORENBOS, R. B. **Production Matching for Large Learning Systems**. Thesis (PhD), Pittsburgh, USA, 1995. UMI Order No. GAX95-22942.

DORIA, A. et al. **Forwarding and Control Element Separation (ForCES) Protocol Specification**. 2010. RFC 5810. Available from Internet: <http://datatracker.ietf.org/doc/rfc5810/>.

ENDRES-NIGGEMEYER, B. The Mashup Ecosystem. In: ENDRES-NIGGEMEYER, B. (Ed.). **Semantic Mashups**. New York, USA: Springer Berlin Heidelberg, 2013. p. 1–50. ISBN 978-3-642-36402-0.

ERICKSON, D. **Beacon Home**. 2013. [Accessed july 20, 2013]. Available from Internet: <https://openflow.stanford.edu/display/Beacon/Home>.

ESCOBEDO, J. et al. Testing Web Service Orchestrators in Context: A Symbolic Approach. In: **IEEE International Conference on Software Engineering and Formal Methods (SEFM)**. Pisa, Italy: IEEE, 2010. p. 257–267.

FIELDING, R. T.; TAYLOR, R. N. Principled Design of the Modern Web Architecture. **ACM Transactions on Internet Technology**, ACM, New York, USA, v. 2, n. 2, p. 115–150, May 2002. ISSN 1533-5399.

FLOODLIGHT. **Floodlight Home**. 2013. [Accessed july 20, 2013]. Available from Internet: <http://floodlight.openflowhub.org/>.

GEBHARDT, H. et al. From Mashups to Telco Mashups: A Survey. **IEEE Internet Computing Magazine**, IEEE Computer Society, Los Alamitos, USA, v. 16, n. 3, p. 70–76, May-June 2012. ISSN 1089-7801.

GEORGE, S. et al. DistressNet: a Wireless ad hoc and Sensor Network Architecture for Situation Management in Disaster Response. **IEEE Communications Magazine**, IEEE Communications Society, New York, USA, v. 48, n. 3, p. 128–136, 2010. ISSN 0163-6804.

GOPAL, R. Model based Framework for Implementing Situation Management Infrastructure. In: **Military Communications Conference (MILCOM)**. Orlando, USA: IEEE, 2007. p. 1–7.

GRAU, B. C. et al. OWL 2: The Next Step for OWL. **Web Semantics**, Elsevier, Amsterdam, The Netherlands, v. 6, n. 4, p. 309–322, November 2008. ISSN 1570-8268.

GRIGORI, D. et al. Ranking BPEL Processes for Service Discovery. **IEEE Transactions on Services Computing**, IEEE Computer Society, Los Alamitos, USA, v. 3, n. 3, p. 178–192, July 2010. ISSN 1939-1374.

HAN, Y. et al. Situational Data Integration with Data Services and Nested Table. **Service Oriented Computing and Applications**, Springer-Verlag, New York, USA, v. 7, n. 2, p. 129–150, 2013. ISSN 1863-2386.

HASAN, R. et al. Please Permit Me: Stateless Delegated Authorization in Mashups. In: **Annual Computer Security Applications Conference (ACSAC)**. Washington, USA: IEEE Computer Society, 2008. p. 173–182. ISBN 978-0-7695-3447-3.

HEIN, D. M. et al. Securing Mobile Agents for Crisis Management Support. In: **Workshop on Scalable Trusted Computing (STC)**. New York, USA: ACM, 2012. p. 85–90. ISBN 978-1-4503-1662-0.

HILL, E. F. **Jess in Action: Java Rule-Based Systems**. Greenwich, USA: Manning Publications Co., 2003. ISBN 1930110898.

HUANG, G. et al. Assisting Navigation and Complementary Composition of Complex Service Mashups. **IEEE Transactions on Services Computing**, IEEE, Los Alamitos, USA, PP, n. 99, p. 1–1, 2014. ISSN 1939-1374.

HUANG, K.; FAN, Y.; TAN, W. An Empirical Study of Programmable Web: A Network Analysis on a Service-Mashup System. In: **International Conference on Web Services (ICWS)**. Honolulu, HI: IEEE, 2012. p. 552–559.

IBSEN, C.; ANSTEY, J. **Camel in Action**. 1st. ed. Greenwich, USA: Manning Publications Co., 2010. ISBN 9781935182368.

JAKOBSON, G. On Conceptualization of Eventualities in Situation Management. In: **International Inter-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)**. San Diego, USA: IEEE, 2013. p. 75–82.

JAKOBSON, G. On Modeling Context in Situation Management. In: **International Inter-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)**. San Antonio, USA: IEEE, 2014. p. 1600–166.

JAKOBSON, G.; BUFORD, J.; LEWIS, L. Situation Management: Basic Concepts and Approaches. In: **Information Fusion and Geographic Information Systems**. New Yor, USA: Springer Berlin Heidelberg, 2007, (Lecture Notes in Geoinformation and Cartography). chp. 2, p. 18–33. ISBN 978-3-540-37628-6.

JAKOBSON, G. et al. Overview of Situation Management at SIMA 2005. In: **Military Communications Conference (MILCOM)**. Atlantic City, USA: IEEE, 2005. v. 3, p. 1630–1636.

JOINES, S.; WILLENBORG, R.; HYGH, K. **Performance Analysis for Java Websites**. Boston, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0201844540.

JONES, K. et al. Biology-inspired Architecture for Situation Management. In: **Military Communications Conference (MILCOM)**. Washington, USA: IEEE, 2006. p. 1–7.

KESHAV, S. **An Engineering Approach to Computer Networking**. Boston, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

KIERAS, D. Using the Keystroke-Level Model to Estimate Execution Times. **University of Michigan**, Ann Arbor, USA, 2001.

KIM, H.; FEAMSTER, N. Improving Network Management with Software Defined Networking. **IEEE Communications Magazine**, IEEE Communications Society, New York, USA, v. 51, n. 2, p. 114–119, 2013. ISSN 0163-6804.

KIM, N.; KIM, J. Building NetOpen Networking Services over OpenFlow-based Programmable Networks. In: **International Conference on Information Networking (ICOIN)**. Barcelona, Spain: IEEE, 2011. p. 525 –529. ISSN 1976-7684.

KOELLE, R.; TARTER, A. Towards a Distributed Situation Management Capability for SESAR and NextGen. In: **International Conference on Networking and Services (ICNS)**. Herndon, USA: IEEE, 2012. p. O6–1–O6–12. ISSN 2155-4943.

KOKAR, M. M.; MATHEUS, C. J.; BACLAWSKI, K. Ontology-based Situation Awareness. **Information Fusion**, Elsevier, Amsterdam, The Netherlands, The Netherlands, v. 10, n. 1, p. 83–98, January 2009. ISSN 1566-2535.

KROHNS-VALIMAKI, H.; STRANDEN, J.; SARSAMA, J. Improving Shared Situation Awareness in Disturbance Management. In: **International Conference on Electricity Distribution (CIRED)**. Stockholm, Sweden: IET, 2013. p. 1–4.

LAGA, N. et al. Widgets and Composition Mechanism for Service Creation by Ordinary Users. **IEEE Communications Magazine**, IEEE Communications Society, New York, USA, v. 50, n. 3, p. 52–60, 2012. ISSN 0163-6804.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In: **ACM SIGCOMM Workshop on Hot Topics in Networks**. New York, USA: ACM, 2010. p. 19:1–19:6. ISBN 978-1-4503-0409-2.

LARA, A.; KOLASANI, A.; RAMAMURTHY, B. Network Innovation using OpenFlow: A Survey. **IEEE Communications Surveys & Tutorials**, IEEE Communications Society, New York, USA, PP, n. 99, p. 1–20, 2013. ISSN 1553-877X.

LATIH, R. et al. Whip: A Framework for Mashup Development with Block-based Development Approach. In: **International Conference on Electronics Engineering and Informatics (ICEEI)**. Bandung, Indonesia: IEEE, 2011. p. 1–6. ISSN 2155-6822.

LIN, S.; GAO, Z.; XU, K. Web 2.0 Traffic Measurement: Analysis on Online Map Applications. In: **International Workshop on Network and Operating Systems Support for Digital Audio and Video**. Williamsburg, USA: ACM, 2009. p. 7–12. ISBN 978-1-60558-433-1.

LIU, X. et al. Towards Service Composition Based on Mashup. In: **Congress on Services**. Salt Lake City, USA: IEEE, 2007. p. 332–339.

MAGOUTAS, B.; MENTZAS, G.; APOSTOLOU, D. Proactive Situation Management in the Future Internet: The Case of the Smart Power Grid. In: **International Conference on Database and Expert Systems Applications (DEXA)**. Toulouse, France: IEEE, 2011. p. 267–271. ISSN 1529-4188.

MAJCHRZAK, A.; MORE, P. H. B. Emergency! Web 2.0 to the rescue! **Communications of the ACM**, ACM, New York, USA, v. 54, p. 125–132, April 2011. ISSN 0001-0782.

MASSIE, M. L.; CHUN, B. N.; CULLER, D. E. The Ganglia Distributed Monitoring System: Design, Implementation And Experience. **Parallel Computing**, Elsevier, New York, USA, v. 30, p. 2004, 2003.

MATSUYAMA, K. et al. A Path-based RDF Query Language for CC/PP and UAProf. In: **Pervasive Computing and Communications Workshops (PERCOMW)**. Orlando, USA: IEEE, 2004. p. 3–7.

MATTOS, D. et al. OMNI: OpenFlow MaNagement Infrastructure. In: **International Conference on Network of the Future (NOF)**. Paris, France: IEEE, 2011. p. 52–56.

MAXIMILIEN, E. M.; RANABAHU, A.; TAI, S. Swashup: Situational Web Applications Mashups. In: **Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)**. Montreal, Canada: ACM, 2007. p. 797–798. ISBN 978-1-59593-865-7.

MCKEOWN, N. et al. OpenFlow: Enabling Innovation in Campus Networks. **Computer Communications Review ACM**, ACM, New York, USA, v. 38, n. 2, p. 69–74, March 2008. ISSN 0146-4833.

MILLER, G. A. WordNet: A Lexical Database for English. **Communications of the ACM**, ACM, New York, USA, v. 38, n. 11, p. 39–41, November 1995. ISSN 0001-0782.

MONSANTO, C. et al. Composing Software-defined Networks. In: **Symposium on Networked Systems Design and Implementation (NSDI)**. Berkeley, USA: USENIX Association, 2013. p. 1–14.

NATASHA, G. et al. NOX: Towards an Operating System for Networks. **Computer Communications Review ACM**, ACM, New York, USA, v. 38, n. 3, p. 105–110, July 2008. ISSN 0146-4833.

NETWORKS, I. J. **Juniper Home**. 2014. [Accessed may 20, 2014]. Available from Internet: <http://www.juniper.net/us/en/products-services/network-edge-services/network-analytics/>.

NOX. **NOX Home**. 2013. [Accessed july 20, 2013]. Available from Internet: <http://www.noxrepo.org/nox/about-nox/>.

OETIKER, T. MRTG: The Multi Router Traffic Grapher. In: **Systems Administration Conference (LISA)**. Boston, USA: USENIX, 1998. p. 141–148. ISBN 1-880446-40-5.

OETIKER, T. Monitoring Your IT Gear: The MRTG Story. **IT Professional**, IEEE Educational Activities Department, Piscataway, USA, v. 3, n. 6, p. 44–48, november 2001. ISSN 1520-9202.

OGC. **Information Technology Infrastructure Library: Service Operation Version 3.0**. London, Inglaterra: Office of Government Commerce, 2007.

OGC. **Information Technology Infrastructure Library: Service Transition Version 3.0**. London, Inglaterra: Office of Government Commerce, 2007.

OGC. **Information Technology Infrastructure Library (ITIL)**. [S.l.]: Office of Government Commerce, 2011. Disponível em: <http://www.itil-officialsite.com/>. Acesso em: Jan. 2011.

ORDONEZ, A. et al. Automated context aware composition of Advanced Telecom Services for environmental early warnings. **Expert Systems with Applications**, Elsevier, New York, USA, v. 41, n. 13, p. 5907 – 5916, 2014. ISSN 0957-4174.

OVIRT. **OVIRT Home**. 2014. [Accessed july 20, 2014]. Available from Internet: <http://www.ovirt.org/Home>.

OZKAN, N.; ABIDIN, W. Investigation of Mashups for Managers. In: **International Symposium on Computer and Information Sciences (ISCIS)**. Guzelyurt, Turkey: IEEE, 2009. p. 622–627.

PATIL, A. A. et al. Meteor-s Web Service Annotation Framework. In: **International Conference on World Wide Web**. New York, USA: ACM, 2004. p. 553–562. ISBN 1-58113-844-X.

PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F. Restful Web Services vs. "Big"' Web Services: Making the Right Architectural Decision. In: **International Conference on World Wide Web**. New York, USA: ACM, 2008. p. 805–814. ISBN 978-1-60558-085-2.

PEREIRA, I.; COSTA, P.; ALMEIDA, J. A Rule-based Platform for Situation Management. In: **International Inter-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)**. San Diego, USA: IEEE, 2013. p. 83–90.

POX. **POX Home**. 2013. [Accessed july 20, 2013]. Available from Internet: <https://github.com/noxrepo/pox>.

RAYMUNDO, C. R. et al. An infrastructure for distributed rule-based situation management. In: **International Inter-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)**. San Antonio, USA: ACM, 2014. p. 202–208.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. **Unified Modeling Language Reference Manual, The (2Nd Edition)**. London, United Kingdom: Pearson Higher Education, 2004. ISBN 0321245628.

RUTHKOSKI, T. L. **Google Visualization API Essentials**. Birmingham, UK: Packt, 2013. ISBN 1849694362.

SALOMONI, P. et al. Profiling Learners with Special Needs for Custom e-learning Experiences, a Closed Case? In: **International Cross-disciplinary Conference on Web Accessibility (W4A)**. New York, USA: ACM, 2007. p. 84–92. ISBN 1-59593-590-8.

SALVADOR, E.; GRANVILLE, L. Using Visualization Techniques for SNMP Traffic Analysis. In: **International Symposium on Computers and Communications (ISCC)**. Marrakech, Morocco: IEEE, 2008. p. 806–811. ISSN 1530-1346.

SANTANNA, J. de; WICKBOLDT, J.; GRANVILLE, L. A BPM-based Solution for Inter-domain Circuit Management. In: **Network Operations and Management Symposium (NOMS)**. Maui, USA: IEEE, 2012. p. 385–392. ISSN 1542-1201.

SANTOS, C. dos et al. Botnet Master Detection Using a Mashup-based Approach. In: **International Conference on Network and Service Management (CNSM)**. Niagara Falls, Canada: IEEE, 2010. p. 390–393.

SANTOS, C. dos et al. On Using Mashups for Composing Network Management Applications. **IEEE Communications Magazine**, IEEE Communications Society, New York, USA, v. 48, n. 12, p. 112–122, December 2010. ISSN 0163-6804.

SANTOS, C. dos et al. A data confidentiality architecture for developing management mashups. In: **International Symposium on Integrated Network Management (IM)**. Dublin, Ireland: IEEE, 2011. p. 49–56.

SANTOS, C. dos et al. Performance management and quantitative modeling of IT service processes using mashup patterns. In: **International Conference on Network and Service Management (CNSM)**. Paris, France: IEEE, 2011. p. 1–9.

SANTOS, C. dos et al. Quality Improvement and Quantitative Modeling - Using Mashups for Human Error Prevention. In: **IFIP/IEEE International Symposium on Integrated Network Management (IM)**. Ghent, Belgium: IEEE, 2013. p. 143–150.

SIMMEN, D. E. et al. Damia: Data Mashups for Intranet Applications. In: **Special Interest Group on Management of Data (SIGMOD)**. Vancouver, Canada: ACM, 2008. p. 1171–1182. ISBN 978-1-60558-102-6.

SMITH, P. et al. Management patterns: SDN-enabled network resilience management. In: **Network Operations and Management Symposium (NOMS)**. Krakow, Poland: IEEE, 2014. p. 1–9.

SQUICCIARINI, A. C. et al. Situational Awareness Through Reasoning on Network Incidents. In: **Conference on Data and Application Security and Privacy**. Santo Antonio, USA: ACM, 2014. p. 111–122. ISBN 978-1-4503-2278-2.

STIRBU, V. et al. A Lightweight Platform for Web Mashups in Immersive Mirror Worlds. **IEEE Pervasive Computing**, IEEE Computer Society, Los Alamitos, USA, v. 12, n. 1, p. 34–41, 2013. ISSN 1536-1268.

TATEMURA, J. et al. Mashup Feeds: Continuous Queries over Web Services. In: **Special Interest Group on Management of Data (SIGMOD)**. New York, USA: ACM, 2007. p. 1128–1130. ISBN 978-1-59593-686-8.

TERESCO, J. D. Highway Data and Map Visualizations for Educational Use. In: **Technical Symposium on Computer Science Education**. Raleigh, USA: ACM, 2012. p. 553–558. ISBN 978-1-4503-1098-7.

TIAN, S.; WEBER, G.; LUTTEROTH, C. A Tuplespace Event Model for Mashups. In: **OZCHI**. New York, USA: ACM, 2011. p. 281–290. ISBN 978-1-4503-1090-1.

TOSTI, E.; SMARI, W. Sensors integration in a grid-based architecture for emergency management systems. In: **International Conference on Digital Ecosystems and Technologies for Complex Systems, Environment, and Service Engineering (DEST)**. Dubai, United Arab Emirates: IEEE, 2010. p. 435–442. ISSN 2150-4938.

ULC, S. I. **Sandvine Home**. 2014. [Accessed may 20, 2014]. Available from Internet: <https://www.sandvine.com/products/network-analytics/>.

VYATTA. **Vyatta Home**. 2013. [Accessed july 20, 2013]. Available from Internet: <http://www.vyatta.org/>.

WANG, J. et al. Application of Server Virtualization Technology based on CitriX XenServer in the Information Center of the Public Security Bureau and Fire Service Department. In: **International Symposium on Computer Science and Society (ISCCS)**. Kota Kinabalu, Malaysia: IEEE, 2011. p. 200–202.

WATSON, J. VirtualBox: Bits and Bytes Masquerading as Machines. **Linux Journal**, Belltown Media, Houston, TX, v. 2008, n. 166, February 2008. ISSN 1075-3583.

WEISS, M.; GANGADHARAN, G. R. Modeling the Mashup Ecosystem: structure and growth. **R & D Management**, Blackwell Publishing Ltd, Oxford, UK, v. 40, n. 1, p. 40–49, 2010. ISSN 1467-9310.

YU, J. et al. Understanding Mashup Development. **IEEE Internet Computing Magazine**, IEEE Computer Society, Los Alamitos, USA, v. 12, n. 5, p. 44–52, September-October 2008. ISSN 1089-7801.

ZAALOUK, A. et al. OrchSec: An orchestrator-based architecture for enhancing network-security using Network Monitoring and SDN Control functions. In: **Network Operations and Management Symposium (NOMS)**. Krakow, Poland: IEEE, 2014. p. 1–9.

APPENDIX A - SCIENTIFIC PRODUCTION

The research work presented in this thesis was reported to the scientific community through paper submissions to renowned conferences and journals. The process of doing research, submitting paper, gathering feedback, and improving the work helped to achieve the maturity hereby presented.

A.1 Papers: accepted and on reviewing

The list of accepted papers to date is as follows.

1. **OSCAR MAURICIO CAICEDO R.**, Carlos Raniery P. dos Santos, Arthur Selle Jacobs, Lisandro Z. Granville. **Monitoring Virtual Nodes Using Mashups**. Computer Networks (COMNET), v. 64, pp. 55-70, May 2014. ISSN 1389-1286.
   - Status: Published.
   - Qualis: A1.
   - Contribution: Mashment System Architecture.
   - 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.

2. **OSCAR MAURICIO CAICEDO R.**, Felipe Estrada-Solano, Lisandro Z. Granville. **An Approach to Overcome the Complexity of Network Management Situations by Mashments**. The 28th IEEE International Conference on Advanced Information Networking and Applications (AINA 2014), 13-16 May 2014, Victoria, Canada.
   - Status: Published.
   - Qualis: A2.
   - Contribution: Process to develop and lunch mashments, Mashment System Architecture, and complexity assessment model.
   - Abstract: The work performed by network administrators to address sudden, dynamic, heterogeneous, and time specific situations that happen in the network management domain is complex. In this paper, we introduce an approach that allows network administrators to overcome the complexity of handling these network management situations. The approach is made up of Mashments that are special mashups used to cope with nmsits, the process to develop and execute Mashments, and the Mashment Maker that supports such model and process. We use IT Service Management metrics to evaluate our approach, measuring the complexity of facing, with and without the Maker, a specific nmsit that occurs in several networks based on the Software Defined Networking paradigm. The evaluation results demonstrate that the complexity decreases when network administrators use our approach to handle nmsits.
3. **OSCAR MAURICIO CAICEDO R.**, Felipe Estrada-Solano, Lisandro Z. Granville. **A Mashup Ecosystem for Network Management Situations**. The IEEE Global Communications Conference (GLOBECOM 2013), 9-13 December 2013, Atlanta, United States.
   - Status: Published.
   - Qualis: A1.
   - Contribution: Concepts of mashment and nmsit, Mashment Ecosystem, and time-consuming assessment model.
   - Abstract: Current network management approaches and their implementations are not intended to address dynamic situations that need rapid delivery of good-enough and comprehensive solutions. In this paper, we introduce a novel mashup ecosystem, called Mashment Ecosystem, that allows Network Administrators to conduct on a Mashment Maker the activities and interactions necessary to provide Mashments. Mashments are mashups aimed to tackle network management situations. We evaluate the Mashment Ecoystem by estimating with the Keystroke-Level Model and measuring in a test scenario the time that Network Administrators take to perform the activities of creating, launching, and publishing Mashments. Similarly, we evaluate the time for retrieving information about a network management situation

by using or not Mashments. The evaluation results corroborated that Network Administrators, in our ecosystem, need short-time to deal with network management situations.

4. **OSCAR MAURICIO CAICEDO R.**, Felipe Estrada-Solano, Lisandro Z. Granville. **A Mashup-based Approach for Virtual SDN Management.** The 37th IEEE Annual International Computer Software & Applications Conference (COMPSAC 2013), 22-26 July 2013, Kyoto, Japan.
   - Status: Published.
   - Qualis: A2.
   - Contribution: Mashment System Architecture.
   - Abstract: The Software Defined Networks paradigm aided by the Network Virtualization is a key driver to cope the Internet ossification. There are different proposals to deploy this paradigm, but there is not an integrated or standardized way for the management of networks built with such proposals. In this sense, the network management becomes too complex because multiple solutions must be used by Network Administrators to perform their tasks. In this paper, we introduce a mashup-based approach that allows Network Administrators to customize and combine management solutions, in order to they build composite applications aiming the integrated management of Virtual Software Defined Networks in heterogeneous environments. We evaluate our approach by building a SDN Mashup for the management of a network slice that uses three distinct Network Operating Systems and by running performance tests, corroborating that the mashup built has small response time.

There is other paper that is still under reviewing.

1. **OSCAR MAURICIO CAICEDO R.**, Felipe Estrada-Solano, Vinicius Guimarães, Liane M. R. Tarouco, Lisandro Z. Granville. **Rych Dynamic Mashments: An Approach for Network Management Based on Mashups and Situation Management**. IEEE Transactions on Computers (TC). ISSN 0018-9340.
   - Status: Submitted.
   - Qualis: A1.
   - Contribution: Mechanisms for automatic recognition of nmsits and dynamic composition of mashments.
   - Abstract: In the network management domain, large research efforts have been made to automate and facilitate the daily tasks conducted by network administrators. However, so far, none of these efforts has carried out network management

using jointly the Situation Management discipline and the mashup technology. This paper introduces an approach called Rich Dynamic Mashments that aim to deal in an effective way with unexpected, dynamic, and heterogeneous situations (named nmsits - for instance, a sudden packet loss in a core router of a network backbone and an unforeseen slowness in data transmission over a link between two virtual routers) faced by network administrators in their everyday work. The proposed approach is formed by mechanisms for automatic recognition of nmsits and dynamic composition of mashments (tunable mashups that use Situation Management for conducting network management tasks) and an architecture supporting these mechanisms. We further implement a prototype of the proposed architecture and conduct an extensive analysis on networks based on the Software Defined Networking paradigm. The analysis results have provided directions and evidences that corroborate the effectivity, in terms of time-recognition, time-composition, and time-consuming, on using Rych Dynamic Mashments for network management.

A.2 Collaborations: accepted and on reviewing

There is other peer-reviewed publication that, although not directly related to this thesis, is linked to the design of network management solutions.

1. Wanderson Paim de Jesus, Ricardo L. dos Santos, **OSCAR MAURICIO CAICEDO R.**, Lisandro Zambenedetti Granville. **A Platform for Programmable Virtual Network Management**. The 31st Brazilian Symposium on Computer Networks and Distributed Systems (SBRC 2013), 6-10 May 2013, Brasilia, Brazil.
    - Status: Published.
    - Qualis: B2.
    - Abstract: With the evolvement of virtualization and network programming techniques, high-level applications can be used to define the behavior of network traffic while keeping isolation. However, to ensure a harmonious relationship between users, network applications and virtual networks, considerable management efforts are needed. In this paper we propose the ProViNet platform, a solution for managing the deployment of network applications in Programmable Virtual Networks. In addition to the management facilities, ProViNet contributes with an architecture that allows sharing the control plane of PVN in a scalable way. During the development of this work we identified the need for a standard representation of programmable virtual infrastructures, so it is also proposed an extension of the programmable virtual networks description language VXDL. In order to verify the feasibility of the

proposed platform, we implemented a prototype, which is analyzed and evaluated in this paper.

There are other two collaboration papers that are still under reviewing.

1. Jose A. Ordoñes, Vidal Alcazar, **OSCAR MAURICIO CAICEDO R**, Paolo Facarin, Juan C. Corrales, Lisandro Z. Granville. **Towards Automated Composition of Convergent Services: a Survey**. Computer Communications (COMCOM), ISSN 0140-3664.
   - Status: Submmitted.
   - Qualis: A2.
   - Abstract: A convergent service is defined as a service that exploits the convergence of communication networks and at the same time takes advantage of features of the Web. Nowadays, building up a convergent service is not trivial, because although there are significant approaches that aim to automate the service composition at different levels in the Web and Telecom domains, selecting the most appropriate approach for specific case studies is complex due to the big amount of involved information and the lack of technical considerations. Thus, in this paper, we identify the relevant phases for convergent service composition and explore the existing approaches and their associated technologies for automating each phase. For each technology, the maturity and results are analysed, as well as the elements that must be considered prior to their application in real scenarios. Furthermore, we provide research directions related to the convergent service composition phases.
2. Ricardo L. dos Santos, **OSCAR MAURICIO CAICEDO R.**, Juliano A. Wickboldt, Lisandro Z. Granville. **App2net: A Platform to Transfer and Configure Applications on Programmable Virtual Networks**. The IEEE International Conference on Communications (ICC 2015), 8-12 June 2015, London, United Kindong.
   - Status: Submitted.
   - Qualis: A2.
   - Abstract: In programmable virtual networks, simple tasks, like installing a software, can be extremely complex. This complexity occurs mainly because the code transference and initial functional settings in network execution environments are not automated. In addition, the same tasks have different requirements in each service lifecycle stage. In this sense, we propose the App2net platform that allows to transfer and configure network applications over programmable virtual networks with heterogeneous execution environments. We also propose a taxonomy for grouping code transfer techniques and, based on such techniques, we develop models for code

transfer. A prototype has been implemented and tested on realistic network topologies commonly found on the Internet. Results allow us to identify which models improve the code transfer and consume more resources, according to the requirements of service lifecycle stages and realistic network topologies.

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/© 2014 Elsevier B.V. All rights reserved.

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.

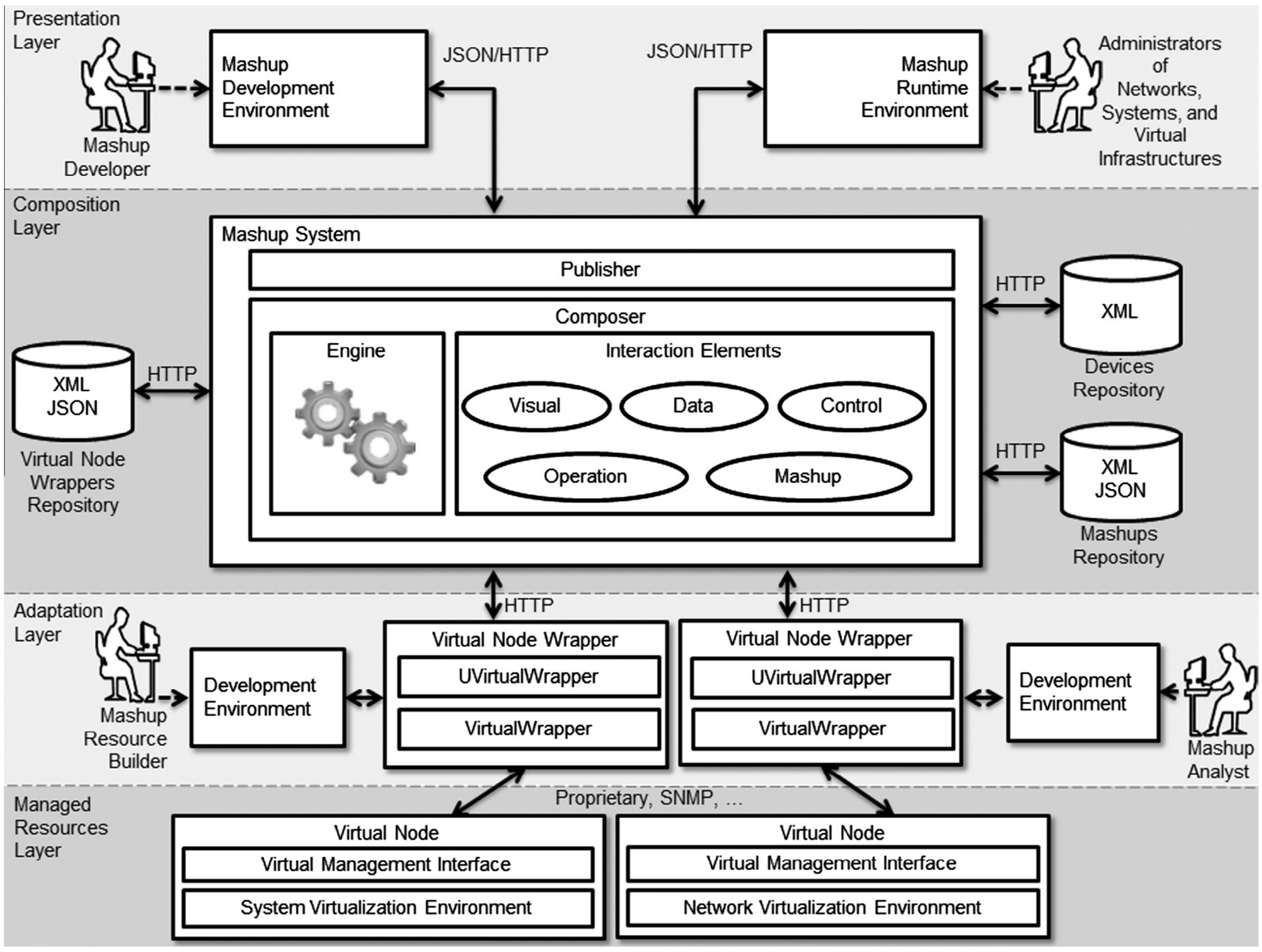


Fig. 1. Architecture for Virtual Nodes Monitoring based on mashups.

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

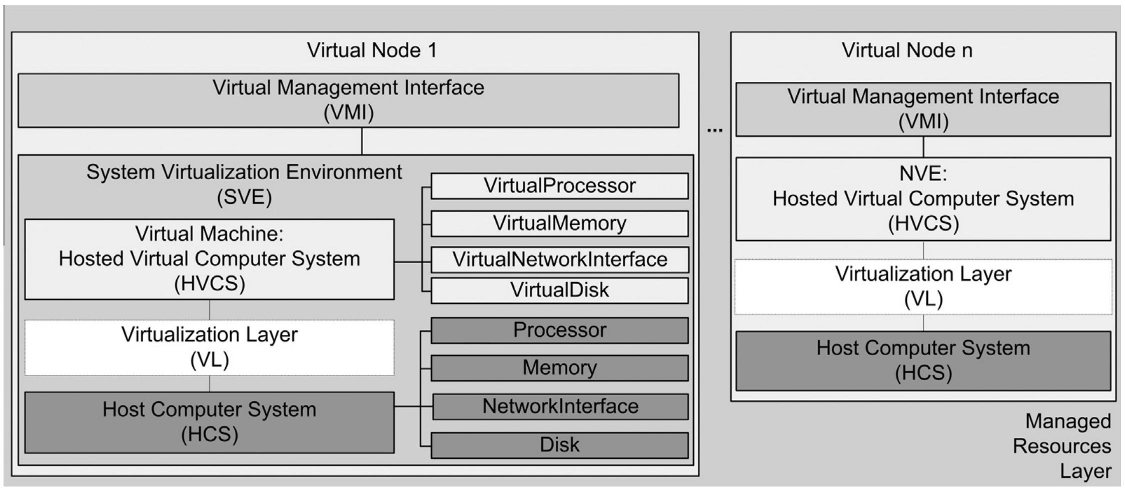

Virtual Node 1
Virtual Management Interface (VMI)
System Virtualization Environment (SVE)
Virtual Machine: Hosted Virtual Computer System (HVCS)
Virtualization Layer (VL)
Host Computer System (HCS)
VirtualProcessor
VirtualMemory
VirtualNetworkInterface
VirtualDisk
Processor
Memory
NetworkInterface
Disk
...
Virtual Node n
Virtual Management Interface (VMI)
NVE: Hosted Virtual Computer System (HVCS)
Virtualization Layer (VL)
Host Computer System (HCS)
Managed Resources Layer

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

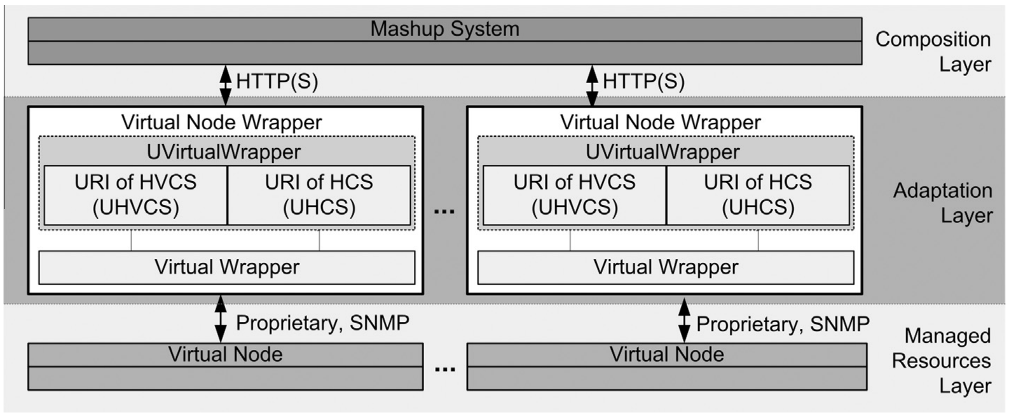


Fig. 3. Virtual Node Wrappers model.

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. $[\{IDRESOURCE : id, NAME : name, PAR : parameters, \quad TO : reSrc, FROM : reDes\}, ldots, \{IDRESOURCE : id, NAME : name, PAR : parameters, TO : resSrc, FROM : resDes\}]$. 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.

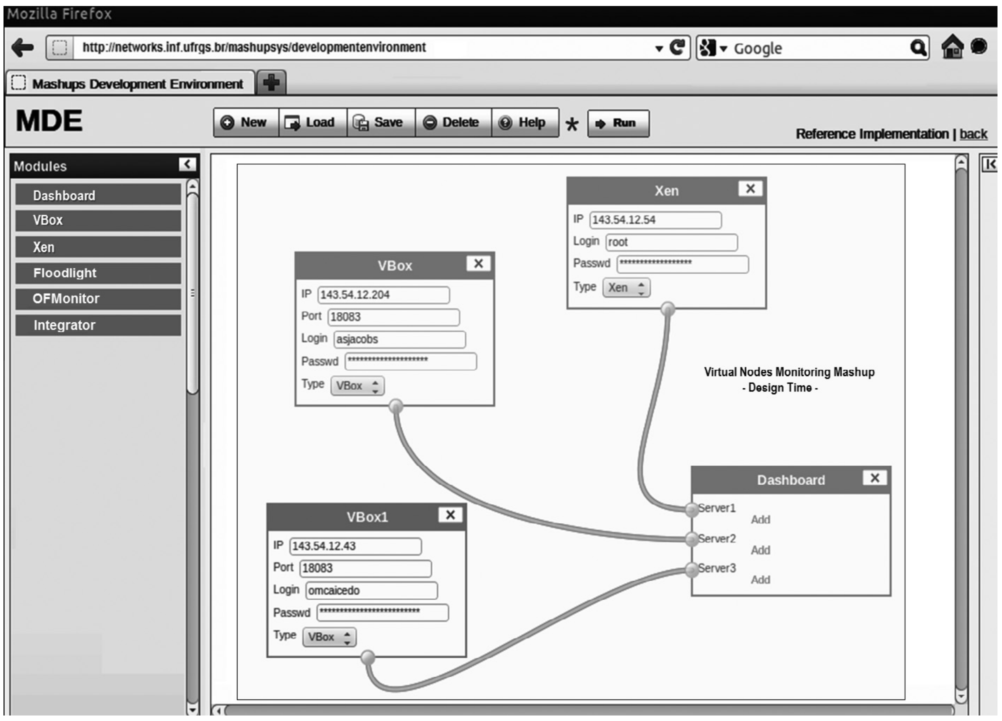
5.3. Mashup Development Environment

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,


Mozilla Firefox
http://networks.inf.ufrgs.br/mashupsys/developmentenvironment
Google
Mashups Development Environment
MDE
New
Load
Save
Delete
Help
Run
Reference Implementation | back
Modules
Dashboard
VBox
Xen
Floodlight
OFMonitor
Integrator
Virtual Nodes Monitoring Mashup
- Design Time -
Dashboard
Server1
Add
Server2
Add
Server3
Add

Xen
IP 143.54.12.54
Login root
Passwd ****************
Type Xen

VBox
IP 143.54.12.204
Port 18083
Login asjacobs
Passwd ******************
Type VBox

VBox1
IP 143.54.12.43
Port 18083
Login omcaicedo
Passwd **********************
Type VBox

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

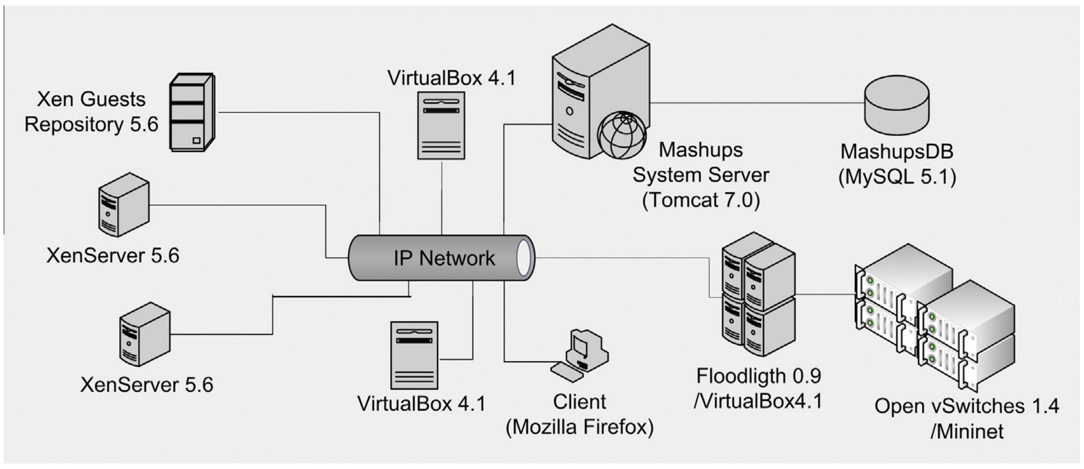

Xen Guests Repository 5.6
VirtualBox 4.1
Mashups System Server (Tomcat 7.0)
MashupsDB (MySQL 5.1)
XenServer 5.6
IP Network
XenServer 5.6
VirtualBox 4.1
Client (Mozilla Firefox)
Floodligth 0.9 /VirtualBox4.1
Open vSwitches 1.4 /Mininet

Fig. 5. Test environment.

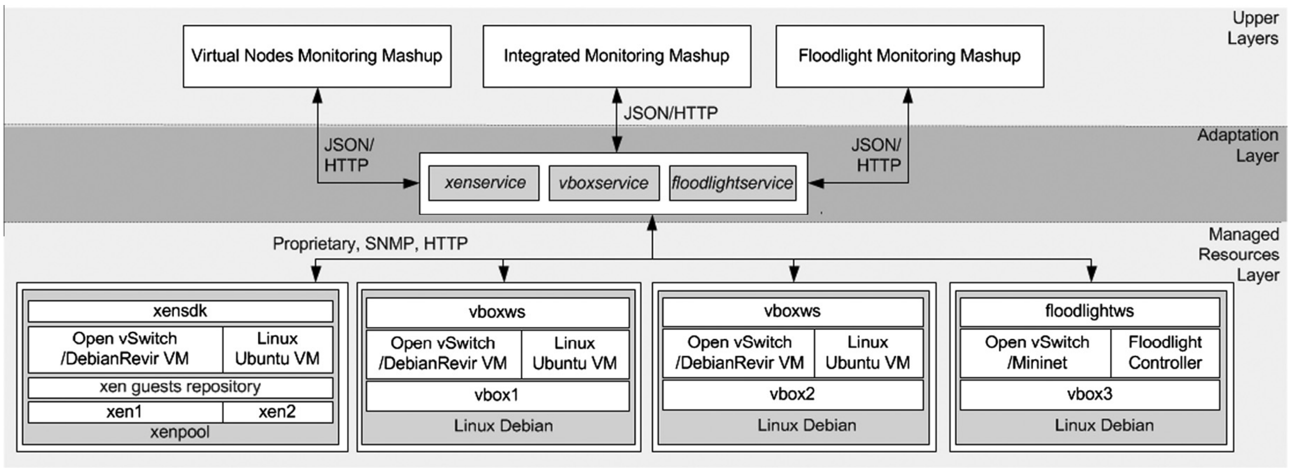
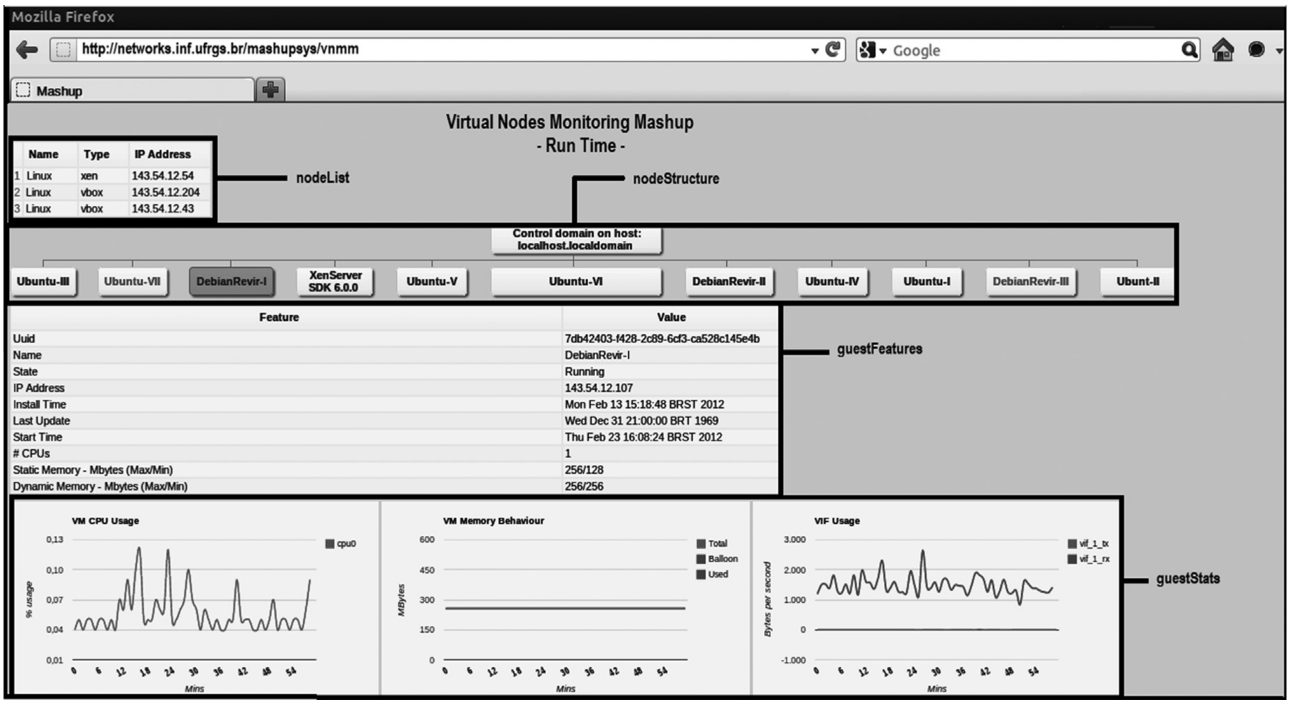


Fig. 6. VNMM, FMM, and IMM on the architecture.

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.

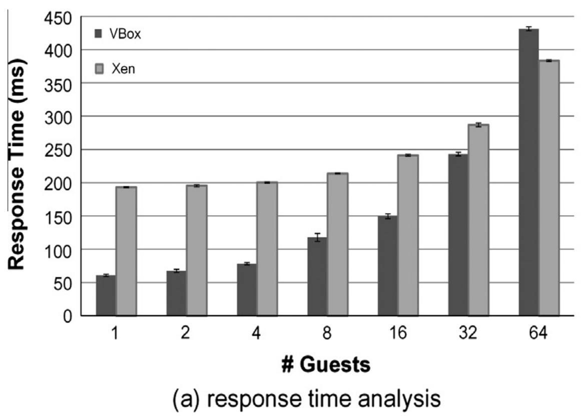
According the performance analysis of java Web sites [35], the response time (r in *seconds*) of this kind of system can be ranked as: optimal when $r \leq 0.1$, good for $0.1 < r \leq 1$, admissible for $1 < r \leq 10$, and deficient if $r > 10$. Considering these thresholds, Table 2 reveals that `nodeList`,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



	Name	Type	IP Address
1	Linux	xen	143.54.12.54
2	Linux	vbox	143.54.12.204
3	Linux	vbox	143.54.12.43

Feature	Value
Uuid	7db42403-f428-2c89-6cf3-ca528c145e4b
Name	DebianRevir-I
State	Running
IP Address	143.54.12.107
Install Time	Mon Feb 13 15:18:48 BRST 2012
Last Update	Wed Dec 31 21:00:00 BRT 1969
Start Time	Thu Feb 23 16:08:24 BRST 2012
# CPUs	1
Static Memory - Mbytes (Max/Min)	256/128
Dynamic Memory - Mbytes (Max/Min)	256/256

Fig. 7. Virtual Nodes Monitoring Mashup on runtime.



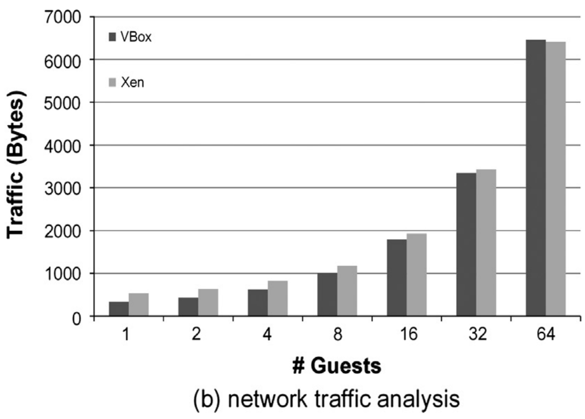


Fig. 8. VNMM – nodeStructure.

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

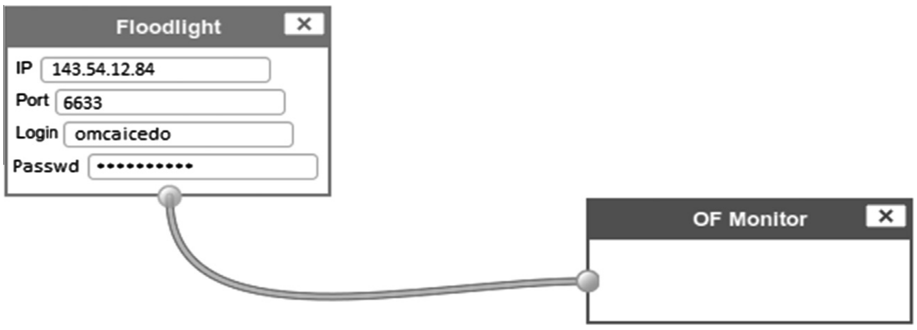
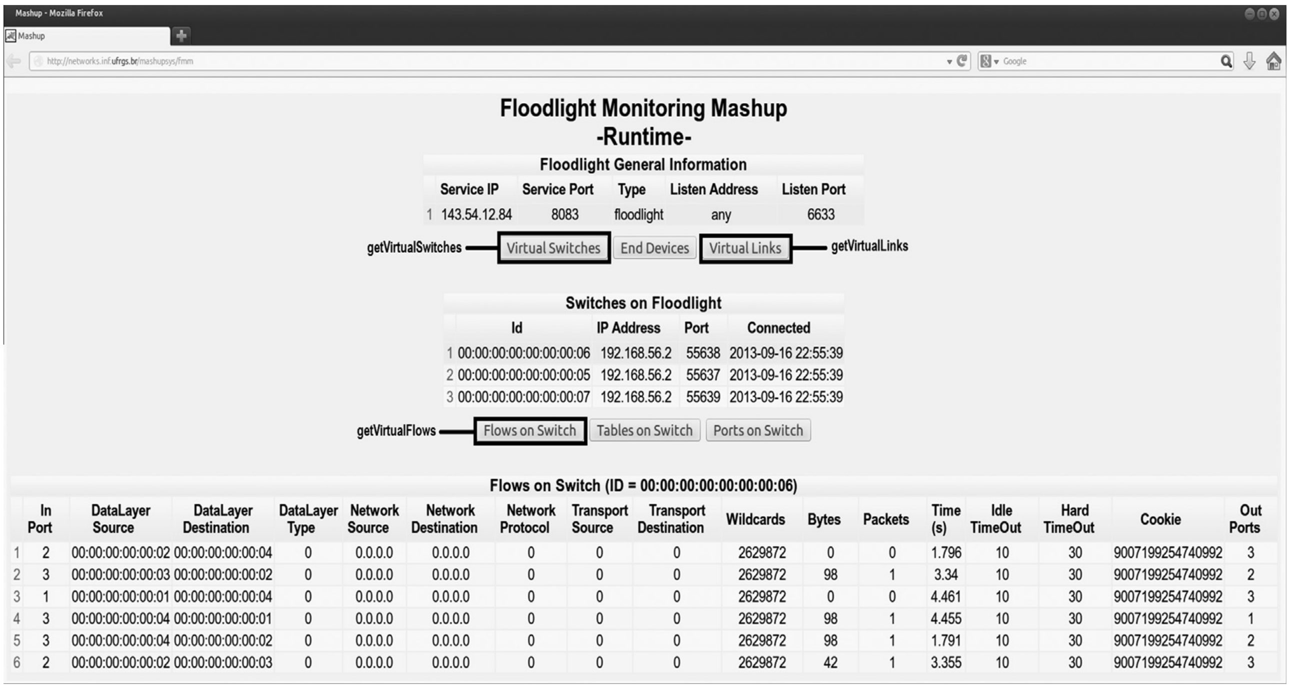


Fig. 9. FMM on design time.



Floodlight Monitoring Mashup
-Runtime-

Floodlight General Information

	Service IP	Service Port	Type	Listen Address	Listen Port
1	143.54.12.84	8083	floodlight	any	6633

Switches on Floodlight

	Id	IP Address	Port	Connected
1	00:00:00:00:00:00:00:06	192.168.56.2	55638	2013-09-16 22:55:39
2	00:00:00:00:00:00:00:05	192.168.56.2	55637	2013-09-16 22:55:39
3	00:00:00:00:00:00:00:07	192.168.56.2	55639	2013-09-16 22:55:39

Flows on Switch (ID = 00:00:00:00:00:00:00:06)

	In Port	DataLayer Source	DataLayer Destination	DataLayer Type	Network Source	Network Destination	Network Protocol	Transport Source	Transport Destination	Wildcards	Bytes	Packets	Time (s)	Idle TimeOut	Hard TimeOut	Cookie	Out Ports
1	2	00:00:00:00:00:02	00:00:00:00:00:04	0	0.0.0.0	0.0.0.0	0	0	0	2629872	0	0	1.796	10	30	9007199254740992	3
2	3	00:00:00:00:00:03	00:00:00:00:00:02	0	0.0.0.0	0.0.0.0	0	0	0	2629872	98	1	3.34	10	30	9007199254740992	2
3	1	00:00:00:00:00:01	00:00:00:00:00:04	0	0.0.0.0	0.0.0.0	0	0	0	2629872	0	0	4.461	10	30	9007199254740992	3
4	3	00:00:00:00:00:04	00:00:00:00:00:01	0	0.0.0.0	0.0.0.0	0	0	0	2629872	98	1	4.455	10	30	9007199254740992	1
5	3	00:00:00:00:00:04	00:00:00:00:00:02	0	0.0.0.0	0.0.0.0	0	0	0	2629872	98	1	1.791	10	30	9007199254740992	2
6	2	00:00:00:00:00:02	00:00:00:00:00:03	0	0.0.0.0	0.0.0.0	0	0	0	2629872	42	1	3.355	10	30	9007199254740992	3

Fig. 10. FMM on runtime.

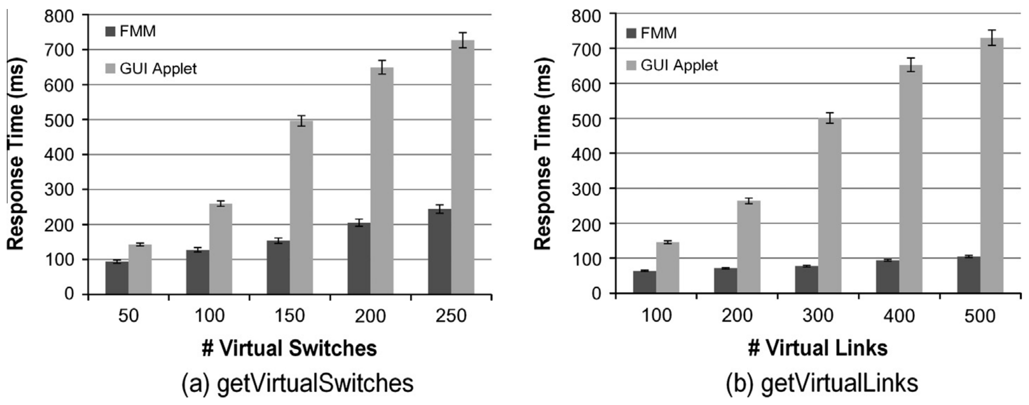


Fig. 11. FMM – response time.

(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`,

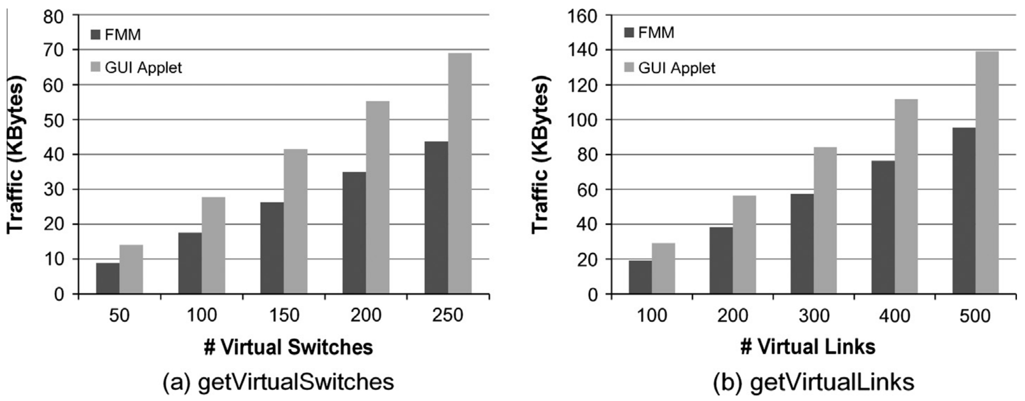


Fig. 12. FMM – network traffic.

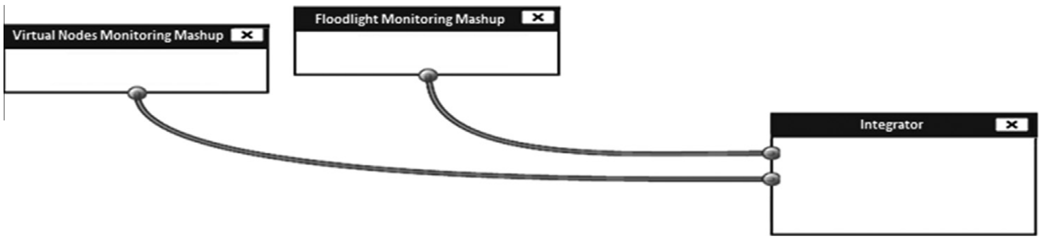


Fig. 13. IMM on design time.

`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.

2014 IEEE 28th International Conference on Advanced Information Networking and Applications

An Approach to Overcome the Complexity of Network Management Situations by Mashments

Oscar Mauricio Caicedo Rendon*†,Felipe Estrada-Solano†, and Lisandro Zambenedetti Granville*
*Computer Networks Group - Institute of Informatics - University Federal do Rio Grande do Sul
†Telematics Engineering Group - Telematics Department - University of Cauca
Email: omcrendon,granville@inf.ufrgs.br - omcaicedo,festradasolano@unicauca.edu.co

Abstract— **The work performed by network administrators to address sudden, dynamic, heterogeneous, and time specific situations that happen in the network management domain is complex. In this paper, we introduce an approach that allows network administrators to overcome the complexity of handling these network management situations (called NMSits). The approach is made up of Mashments that are special mashups used to cope with NMSits, the process to develop and execute Mashments, and the Mashment Maker that supports such model and process. We use IT Service Management metrics to evaluate our approach, measuring the complexity of facing, with and without the Maker, a specific NMSit that occurs in several networks based on the Software Defined Networking paradigm. The evaluation results demonstrate that the complexity decreases when network administrators use our approach to handle NMSits.**

Keywords-**Complexity; Mashment; Mashup; NMSit, Situation Management; Web-based Network Management;**

I. Introduction

The Situation Management (SM) discipline provides solutions that enable analyzing, correlating, and coordinating interactions among people, information, technologies, and actions intended to overcome situations happening or that might happen in dynamic systems [1] [2]. SM foundations are [3]: (*i*) a *Situation* that is 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 predict *Situations*.

SM has been used in domains such as disaster response [4], smart power grid networks [5], security crisis management [6], and public health [7]. However, to the best of our knowledge, there is no SM-based approach to address the complexity of sudden, dynamic, heterogeneous, and time specific *Situations* that network administrators face in their daily work. An example of network management *Situation* is the unexpected failures in the packet transmission of virtual routers belonging to a slice made up of SDN (Software Defined Networking) networks handled by different NOS (Network Operating Systems). Hereinafter, we will refer to this type of *Situations* in the network management domain as NMSits [8].

We argue that the work conducted by network administrators to address NMSits is complex. This complexity exists because, first, although large research efforts have been made to deal with the intricacy of network management [9] [10] [11] [12] [13] [14], they do not focus on handling such a complexity when *Situations* arise unexpectedly. Thus, they have a constrained response capacity to meet NMSits. Second, to face sudden *Situations*, network administrators must handle and rely on a vast amount of non-integrated tools (*e.g.*, traceroute, ZenOSS, OpenNMS, and so on), which hinders their work. Third, to cope with situational requirements, network administrators are usually forced to develop low-level scripts; developing these scripts itself is also complex because network administrators may not be experienced programmers.

Mashups are Web applications built up by end-users through the combination of Web resources available along the Internet [15] [16]. The mashup technology has been employed to manage *Situations* in many domains, such as project management [17], telco services [18], immersive mirror worlds [19], and data integration [20]. In our previous work, we analyzed mashups as a feasible mechanism to accomplish specific tasks for network management in both traditional [21] and SDN-based networks [22]. Nevertheless, we have not addressed how to overcome the complexity of tasks fulfilled by network administrators dealing with NMSits. We refer to mashups used to cope with one or more NMSits as Mashments [8].

In this paper, we take a step further, proposing a novel Mashment-based approach that assists network administrators to overcome the complexity of NMSits and, consequently, facilitates their work. The key contributions from this paper are: (*i*) a conceptual model that presents how to address NMSits by Mashments, (*ii*) a process to develop and execute Mashments, targeted to surpass the intrincacy of tasks carried out by network administrators to handle NMSits, (*iii*) a Mashment Maker prototype that supports the model and process abovementioned; and (*iv*) a Mashment that faces a NMSit on SDN, demonstrating the decrease of complexity when network administrators use our approach to deal with NMSits.

The remainder of this paper is organized as follows. In Section II, we review both the background and the related work. In Section III, we introduce the Mashment-based approach. In Section IV, we describe and discuss the case study raised to evaluate our approach. In Section V, we provide conclusions and implications for future work.

II. Background and Related Work

In this section, we describe research concerning SM, mashups, and mashups on network management. We also present the related work about handling the complexity of network management.

A. Situation Management

The goal of SM is to provide solutions aimed to investigate, control, and predict *Situations* that are composite entities whose components are other entities, their attributes, and relationships in a time interval [1] [2]. To accomplish such a goal, SM-based solutions offer a global vision of *Situations* by collecting, correlating, and merging information from multi-entities, seeking to maximize the user comprehension and, so, supporting the opportune and correct decision making.

SM has been used in diverse domains. In the disaster response [4], a situation-aware architecture and a set of protocols support the timely delivery of high volumes of accurate data that the disaster responders need to make correct decisions. In the smart grid power networks [5], an architecture, based on semantics, linked open data, and complex event processing, enables to respond intelligently to the active power demand of end-users. In the security crisis management [6], a mobile agents platform allows providing timely information to the on-site personnel, the tactical crisis command, and the off-site strategic command centre. In the public health [7], a platform permits the development of situation-aware applications targeted to monitor suspicious cases of tuberculosis. To the best of our knowledge, up to now, SM has been not used to handle the complexity of *Situations* in the network management domain.

B. Mashups

Mashups are Web applications formed by combining Web resources available on the Internet [15] [16]. This combination is mainly achieved by a simple composition model, which allows end-users the development of customized applications, in an easy and rapid way [23].

The mashup technology has been employed to manage *Situations* in several domains. In the project management [17], a mashup system allows managers to easily compose small solutions for displaying and filtering information about their projects. In the telco services [18], a reference architecture facilitates the provisioning of telco-mashups for end-users; a telco-mashup is a composite service that combines functionalities from telecom networks like streaming, quality of service, and billing. In the immersive mirror worlds [19], the Cloud City Scene platform enables end-users to create, in a mashup manner, realistic and immersive street-level representations of the physical world. In the data integration [20], Mashroom, a spreadsheet-like programming environment allows non-developers to create composite services, by aggregating data sources on the fly and interactively.

In our previous work, we analyzed the mashup technology as a feasible mechanism to carry out specific tasks in the network management domain. Initially, we used mashups to accomplish the botnet detection [24] and the traffic monitoring of the border gateway protocol among two autonomous systems [25]. Afterwards, we introduced a generic architecture to support the composing of network management applications [21]. Recently, we leveraged the features of the mashup technology to conduct the integrated monitoring of SDN-based networks [22] and identified a mashup ecosystem around NMSits [8]. However, we have not addressed how to overcome the NMSits complexity.

C. Complexity of Network Management

There is a lot of research works about addressing the complexity of network management. COOLAID [9] automates network configuration by queries performed on an abstract database containing network information. NetOpen [10] allows to build up SOA services for monitoring and configuring OpenFlow networks by networking primitives. OMNI [11] is a solution based on a multi-agent system that allows network administrators to control and monitor OpenFlow networks via a Web user interface. MEICAN [12] uses the business process management for permitting network administrators take part in the decision-making process of provisioning virtual inter-domain circuits. Pyretic [13] enables network programmers to build SDN applications using an abstract packet model, parallel and sequential composition operators, and topology abstraction. Procera [14] permits to manage SDN networks by expressing event-driven and reactive policies based on control domains. In the aforecited works, the network administrator is responsible for manually writing policies, queries, rules, or primitives in specific languages and/or controllers. Thus, his/her daily work to overcome sudden *Situations* of network management remains complex.

Unlike the above works, we consider concepts from SM and mashups, to propose an approach (section III) that acts in a more high-abstraction level and focuses on decrease the complexity of tasks fulfilled by network administrators when facing NMSits. Furthermore, as opposed to these works that have been evaluated using metrics, such as bandwidth, response time, and code lines, we concern about the complexity perceived by network administrators (section IV).

III. Mashments Complexity & NMSits

To better explain our approach, at start, we present a conceptual model about how to address NMSits by Mashments. Afterwards, we introduce the process and complexity to develop and execute Mashments. At last, we present the Mashment Maker.

A. Addressing NMSits by Mashments

A NMSit is a sudden, dynamic, heterogeneous, and time specific *Situation* happening or that might happen in the network management domain. Examples of NMSits, that may be faced by network administrators in their daily work, are: in Faults, to find the cause root of unexpected and multiple packet transmission failures in network slices formed by several OpenFlow networks. In Performance, (*i*) to control the abrupt performance degradation of one or more nodes (switches, routers, and so on) on networks that use diverse virtualization environments; and (*ii*) to monitor, nearly real time, sudden violations in service level agreements. NMSits as the aforementioned may be addressed by using mismatched tools but it overloads and becomes complex the work of network administrators. Also, network administrators may develop home-brewed situational scripts. However, such a development is daunting and complex for non-programmers.

The Figure 1 depicts the conceptual model for addressing NMSits by Mashments. If a NMSit happens, the network administrator: (*i*) orchestrates a Mashment (*i.e.*, combines services, processes, user interfaces, and mashup operations to define a plan and deal with such a NMSit), or (*ii*) reuses a Mashment (*i.e.*, takes advantage of existing plans to face the NMSit). Then, he/she executes the Mashment; on run-time, it performs Network Management Operations targeted to investigate/resolve the NMSit. These Operations are internally conducted via Network Situational Processes, Situation Management Resource as a Service (SMRS), Mediating, and Situation Management Resources (SMR).

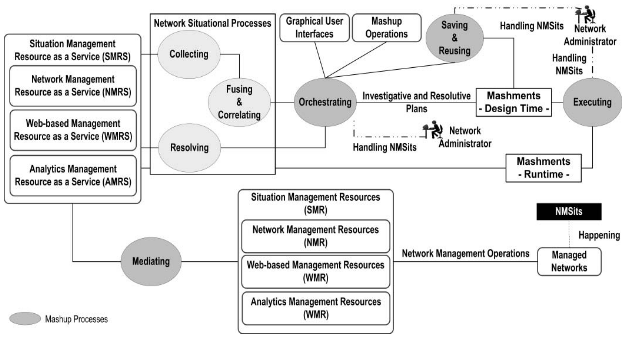

Network Situational Processes
Graphical User Interfaces
Mashup Operations
Saving & Reusing
Handling NMSits
Network Administrator
Handling NMSits
Situation Management Resource as a Service (SMRS)
Collecting
Network Management Resource as a Service (NMRS)
Fusing & Correlating
Orchestrating
Investigative and Resolutive Plans
Mashments - Design Time -
Executing
Web-based Management Resource as a Service (WMRS)
Resolving
Network Administrator
Handling NMSits
Analytics Management Resource as a Service (AMRS)
Mashments - Runtime -
Situation Management Resources (SMR)
NMSits
Network Management Resources (NMR)
Happening
Mediating
Web-based Management Resources (WMR)
Network Management Operations
Managed Networks
Analytics Management Resources (WMR)
Mashup Processes

Figure 1. Addressing NMSits by Mashments: Conceptual Model

A SMR is any solution that provides access and communication to and from network elements or entire networks involved in NMSits. There are three types of SMR: Network Management Resources (NMR) are solutions, like ZenOSS, Citrix Center, OMNI, and Nagios, intended to conduct network management operations. Web-based Network Management Resources (WMR) are tools available in the Internet, such as the Multi Router Traffic Grapher (MRTG) and RRDTool, useful to perform network management tasks. Analytics Management Resources (AMR) are solutions, like Junos Network Analytics Suite, Management Traffic Analyzer, and Sandvine Network Analytics, suitable to analyze network management information.

A SMRS is a software entity that offers the network management operations of a SMR to the Network Situational Processes, aiming to hide the complexity of NMR, WMR, and AMR. Specifically, a Network Management Resource as a Service (NMRS) is responsible for providing functionalities of NMR, a Web-based Management Resource as a Service (WMRS) is in charge of offering capabilities of WMR, and an Analytics Management Resource as a Service (AMRS) is responsible for supplying functionalities of AMR.

The Network Situational Processes help to automate and carry out the investigative/control aspects of SM by using NMRS, WMRS, and/or AMRS. There are three Network Situational Processes: (*i*) Collecting allows to retrieve information about NMSits through SMRS, (*ii*) Fusing&Correlating permits to merge and correlate the information retrieved by Collecting. Fusing&Correlating and Collecting aid in the creation of investigative plans that are useful to determine the cause of NMSits; and (*iii*) Resolving enables to perform, by using SMRS, network management operations aimed to control (change/preserve) NMSits. Consequently, Resolving supports the building up of resolutive plans.

The Mashup Processes provide the automation needed to orchestrate, save&reuse, and execute Mashments that are composite and customisable situational solutions which allow network administrators to deal with NMSits. In particular, Orchestrating includes selecting, configuring, and connecting the resources that form Mashments: SMRS, Network Situational Processes, GUI, Mashup Operations, and even Mashments.

The GUI are internal and external libraries helpful to generate advanced and integrated Mashment interfaces targeted to network administrators. An external GUI is an Application Program Interface (API), such as Yahoo Maps and Google Chart, provided by a third-party and that may be used to display composed information about networks and their devices. An internal GUI is, for instance, a specific user interface developed to show, correlatively, network traffic information.

The Mashup Operations are: (*i*) control patterns (*e.g.*, sequential, parallel, and conditional) that allow to define the process flow of Mashments and, consequently, investigative and resolutive plans, (*ii*) structures for configuring and invoking the resources that form Mashments; and (*iii*) structures for receiving, sorting, and filtering information from any SMRS.

Executing enables network administrators to run Mashments. On run time, all Mashments delegates their management operations to one or more SMRS via Network Situational processes. In turn, each SMRS carries out its operations through Mediating, which is a process always hidden for network administrators. To assist Executing and Orchestrating, Mediating offers SMRS to Network Situational Processes and delegates network management operations to SMR. This mediation is needed because there is not a common format neither a standardized interface/protocol to retrieve and/or bidirectionally interact with data, application logic, and user interfaces of SMR involved in NMSits. Saving&Reusing permits network administrators to store Mashments for their later reuse. Thus, Mashments can be extended and improved to create other ones or customized to handle analogous NMSits.

Leveraging the automation of Network Situational Processess and Mashup Processes, our approach enables network administrators to: (*i*) collect, correlate, and fuse information about NMSits, (*ii*) present information related to NMSits, in a visual and comprehensible way, (*iii*) perform network management operations to resolve (change or preserve) NMSits, (*iv*) build up composite situational solutions, in a Mashment manner, targeted to address NMSits; and (*v*) as a global result, to overcome the complexity of network management tasks in front of NMSits.

B. Process and Complexity in the Development and Execution of Mashments

Considering the above conceptual model, the set of Mashments is formally expressed as: $Mashment = \{mashment_x | mashment_x = (R_{used}, r_{root}, \delta, NMSit_{addr})\}$. Where, R_{used} is the set of resources (SMRS, GUIs, Mashup Operations, and Mashments) used in the $mashment_x$ creation, r_{root} is the root resource ($\in R_{used}$) that starts the $mashment_x$ execution, δ is the execution flow (*i.e.*, investigative and resolutive plans) of resources that make up the $mashment_x$, and $NMSit_{addr}$ is the set of one or more $nmsits$ addressed by the particular $mashment_x$. It is noteworthy to mention that $NMSit$ is the set of *Situations* happening in the network management domain and $NMSit_{addr} \subseteq NMSit$.

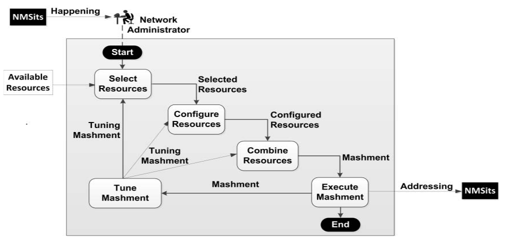


Figure 2. Process to Develop and Execute Mashments

In our approach, network administrators are able to tackle $nmsits$ by following the process (see Figure 2) to develop and execute $mashments$. Such a process is formed by the tasks: Select, Configure, Combine, Execute, and Tune.

Select Resources. The network administrator defines the R_{used} from Available Resources. This task is divided in two: ***(i)*** The network administrator selects the SMRS, GUIs, and Mashup Operations needed to create the $mashment_x$. ***(ii)*** If it is feasible (there is a $mashment_y$ that addresses similar $nmsits$), the network administrator chooses one or more elements of the set $Mashment$ (it is part of available resources) to reuse them. ***Configure Resources***. The network administrator provides the functioning settings of one or several elements belonging to R_{used}, defining the set of resources configured $R_{conf} \subseteq R_{used}$. ***Combine Resources***. The network administrator defines the δ of $mashment_x$ that is formed by combining (connecting/linking) the selected resources. It is important to highlight that the δ creation includes the definition of r_{root}. ***Execute Mashment***. The network administrator launches the $mashment_x$. ***Tune Mashment***. If it is needed, the network administrator tunes the δ of $mashment_x$ under construction, which may imply the selection, configuration, and combination of new resources or simply the rearrangement of R_{used}.

The complexity of $mashment_x$ (*i.e.*, ζ) is calculated by computing the individual complexity of tasks forming the process aforedescribed. In this way:

$$\zeta = \sum_{1}^{i} \zeta_{sel} + \sum_{1}^{j} \zeta_{con} + \sum_{1}^{k} \zeta_{com} + \sum_{1}^{e} \zeta_{exe} \qquad (1)$$

Where, ζ_{sel}, ζ_{con}, ζ_{com}, and ζ_{exe} represent the complexity of Select, Configure, Combine, and Execute, respectively. In turn, i, j, k, and e denote the number of times that such tasks are conducted, allowing to consider the complexity of Tune. In the next paragraphs, these complexities are expressed by using per-task metrics defined for IT Service Management processes [26].

The complexity of Select is expressed as:

$$\zeta_{sel} = \sum_{m=1}^{M} \varsigma_m + (nAvailableResources - 1) * gF * cF \quad (2)$$

Where, M is the total number of elements on R_{used} and $\varsigma_m = selType(m)$ is the complexity of selecting the m-resource. Here, $selType(m)$ can take one of three values depending on the automation of m-selection: 0 - if fully automated, 1 - if manual but tool-assisted, or 2 - if manual. $nAvailableResources$ is the number of resources available to build up the $mashment_x$ (*i.e.*, more available resources result in higher complexity of selection). gF is the grade of guidance provided to select the resources needed to form the $mashment_x$. gF can take one of three values: 1 - if correct recommendation about resources to be selected is offered, 2 - if general information about each available resource is supplied, or 3 - if information is not provided. cF represents the impact of wrong selection of resources and its value is: 0 - if negligible impact, 1 - if moderate impact, or 2 - if severe impact.

The complexity of Configure is defined as:

$$\zeta_{con} = \sum_{n=1}^{N} \varsigma_n . \qquad (3)$$

Where, N is the total number of resources on R_{conf} and ς_n is the complexity of configuring the n-resource. Note that as $R_{conf} \subseteq R_{used}$, so, $N \leq M$. $\varsigma_n = \sum_{p=1}^{P} sourceParameter(p)$. Here, P is the total number of parameters to be configured in the n-resource and $sourceParameter(p)$ can take one of seven values: 0 - if the p-parameter value is produced from automation, 1 - if the p-parameter value may be chosen freely (*e.g.*, a new password), 2 - if the p-parameter value is taken from task documentation (*e.g.*, set up port=8080 for a HTTP server), 3 - if the p-parameter value is extrapolated from task documentation (*e.g.*, define a range of IP addresses), 4 - if the p-parameter value is not trivial for unexperienced network administrators (*e.g.*, set up the URL=$http://IPAddressOfXenServer/rrdUpdates?host = true$ to retrieve statistics of virtual machines running on a determined XenServer), 5 - if the p-parameter is fixed by the environment to a specific value that is defined after additional research (*e.g.*, set up the SNMP OID=1.3.6.1.4.1.9.9.91.1.1.1.1.4 to obtain the temperature of Catalyst Cisco Switch), or 6 - if the p-parameter value is constrained by the environment to a limited set of possible choices where network administrators need to infer the right choice (*e.g.*, set up the type of server virtualization technology to be monitored: virtTech=VMware).

The complexity of Combine is expressed as:

$$\zeta_{com} = \sum_{l=1}^{L} linkType(l) + (M-1) * goF * coF \tag{4}$$

Where, L is the total number of links (logical connections) created to build up the $mashment_x$. $linkType(l)$ represents the complexity of creating the l-link that connects two elements of R_{used} and can take one of four values: 0 - if the l-link is automatically created, 1 - if the l-link is manually created by a support tool and data transferred among resources connected must not be adapted, 2 - if the l-link is manually created and data transferred among resources connected must not be adapted, and 3 - if the l-link is manually built and data transferred among resources connected must be adapted. M is the total number of R_{used} (*i.e.*, more selected resources result in higher complexity of combination). goF is the grade of guidance provided to link the selected resources and can take one of three values: 1 - if correct guidance to link the selected resources is supplied, 2 - if general information about the links that can be established is offered, or 3 - if information is not provided. coF represents the impact of wrong combination of resources, its value is: 0 - if negligible impact, 1 - if moderate impact, or 2 - if severe impact.

ζ_{exe} can take one of three values depending on the automation of $mashment_x$ execution: 0 - if entirely automated (*e.g.*, an autonomous mashment system that executes $mashments$ on demand), 1 - if manual but tool-assisted (*e.g.*, using an execution environment to start the $mashment_x$), or 2 - if manual (*e.g.*, programming/customizing a script every time the $mashment_x$ needs to be executed).

C. *Mashment Maker Architecture*

The Mashment Maker is defined to accomplish the following goal: to support the conceptual model of Mashments and, consequently, their developing and executing process. Therefore, the Maker is targeted to decrease ζ_{sel}, ζ_{con}, ζ_{com}, ζ_{exe}, and the intricacy of Tune (*i.e.*, re-performing the other tasks). The Figure 3 depicts the Mashment Maker Architecture that is formed by: SMRS, Mashment Operations, Mediator Bus, Visual Resources (*i.e.*, Visual-SMRS, Visual-BI, Visual-MO, and Visual-Mashment), Designer, Contextual Help System (CHS), Mashment Router, Mashment Engine, Mashment Repository, and Users Repository.

The Mediator Bus provides as a service the Mediating Process and enables the communication among all elements of the Maker Architecture. Mashment Operations are services that supply the functionalities of both Network Situational Processes and Mashup Operations. The functioning of SMRS (NMRS, WMRS, and AMRS), Network Situational Processes, Mashup Operations, and Mediating Process was already described in the subsection III-A. Here, it is important to point

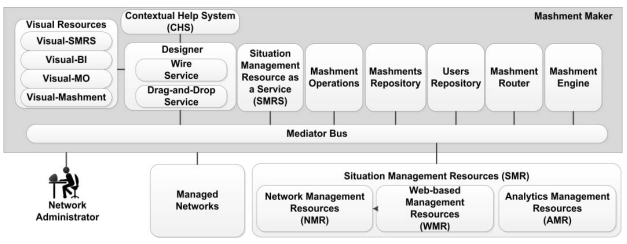


Figure 3. Mashment Maker Architecture

out that, first, the Bus, Mashment Operations, and SMRS are key to achieve the Maker goal because these architectural elements drive the intricacy of underlying technologies involved in the investigation and resolution of NMSits. Second, network administrators never have direct access to these three elements. This access is always conducted through Visual Resources.

The Visual Resources represent SMRS, GUI, Mashments Operations, and Mashments, in a high-level abstraction, in order to hide complexity for network administrators. Visual-SMRS includes: *(i)* Visual-NMRS (*e.g.*, a box offering management functionalities of Vyatta Virtual Router) represents NMRS, *(ii)* Visual-WMRS (*e.g.*, a box representing functional features of RRDTool) represents WRMS; and *(iii)* Visual-AMRS (*e.g.*, a box providing functions of the Management Traffic Analyzer) represents AMRS.

Visual-BI represents basic user interfaces that are useful to create the composite and advanced GUIs of Mashments. For instance, a Mashment GUI can be composed by inserting network traffic images (from MRTG) into a map (from Google Maps). Visual-MO represents Mashment Operations. A box offering a dashboard (it hides the collection, correlation, and fusion of network management information) to monitor heterogeneous OpenFlow Controllers is an example of Visual-MO. On design time, to facilitate the reuse, each existing Mashment is depicted as a Visual-Mashment.

The Designer allows network administrators to develop and execute Mashments. Accordingly, first, it provides services for the δ definition by means of Dragging-and-Dropping and Wiring of Visual Resources. Second, it offers capabilities for saving, deleting, loading, and launching Mashments. In this sense, on design time, Saving permits to write in the Mashment Repository the δ of Mashments. Deleting allows to remove a specific δ. Loading is responsible for reading δ and generating Visual-Mashments. Launching permits to request to the Engine the execution of a determined Mashment. Third, it uses CHS to offer guidance about Visual Resources, Mashments, and the Maker as a whole. All Designer functionalities are targeted to facilitate the creation, re-usage and execution of Mashments and, as a consequence, to reduce ζ_{sel}, ζ_{con}, ζ_{com}, and ζ_{exe}. Also, such functionalities are key to permit network administrators to customize their workspace when addressing NMSits.

The Mashment Repository stores the metadata of Mashments built in the Designer. The metadata of Mashments are objects containing the information/definition of δs. If a Mashment is formed by one or more Mashments, its metadata includes the metadata of these Mashments. This inclusion means that a δ can encompass other δs. The Users Repository stores the data of Network Administrators; this data is used to perform the access control to the Maker.

The Mashment Router is responsible for performing the δ of Mashments. Thus, on run time, the Router: *(i)* receives Mashments invocations from the Engine, which means that the Router is called by the Engine to select a Mashment to service an initial request, *(ii)* selects and links multiple resources (including Mashments into a Mashment) to attend invocations, by reading the needed information from repositories of Mashments and Users; and *(iii)* calls the Engine to request the instantiation of Mashments and their elements. It is to point out that the Router is required by the Engine to function, but the Router is a separate architectonic element.

The Mashment Engine is a lifecycle manager, responsible for creating, deleting, and caching instances of Mashments and their resources. The Engine is splitted in two: (*i*) the Server-Side is a Web engine that supports the execution of SMRS, Network Situational Processes, and Mashup Operations; and (*ii*) the Client-Side is a Web browser engine that supports Web 2.0 technologies to be able to run the integrated and advanced user interfaces of Mashments.

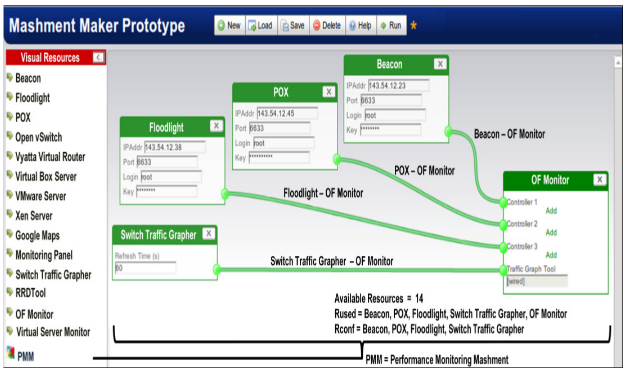
Concerning the Maker, it is important to highlight that: *(i)* the Drag-and-Drop Service assists Select, aiming to reduce $selType(m)$, *(ii)* CHS provides guidelines for supporting Select, Configure, and Combine, which is targeted to diminish, respectively, gF, $sourceParameter(f)$, and goF, *(iii)* the Wire Service, that does not require data mapping, bears Combine, aiming to cut down $linkType(l)$, *(iv)* the high-level launching mechanism, integrated in the Designer, expedites the running of every $mashment$, seeking to decrease ζ_{exe}; and *(v)* high-level Visual Resources allow the flexible construction of Mashments, in a designer-assisted way, which is directed to cut down ζ_{sel}, ζ_{con}, and ζ_{exe}.

IV. Case Study

To assess our approach, first, we performed a test environment made up of the Maker prototype, three SDN-based networks built using OpenFlow, and a NMSit that happens in these networks. Second, we conducted experiments to measure the complexity of addressing such a NMSit when the network administrator follows the proposed process with and without the Maker. Below, we describe the test conditions, present the experiments, and analyze the obtained results.

A. Test Environment

Mashment Maker Prototype. The Figure 4 depicts the Maker GUI that is formed by the Designer, the Buttons (New, Load, Save, Delete, Help, and Run), the Visual Resources (Beacon, Floodlight, POX, Open vSwitch, Vyatta Virtual Router, Virtual Box Server, VMware Server, Xen Server, Google Maps, Monitoring Panel, Switch Traffic Grapher, RRDTool, OF Monitor, Virtual Servers Monitor, and the *Performance Monitoring Mashment - after described*), and CHS. The Designer is a Web application built using YUI 2.7 and WireIt 0.5. YUI is an open source, CSS and javascript framework, used to implement the Drag-and-Drop Service. WireIt is a set of open source javascript libraries, used to create the Wire Service. The Buttons and Visual Resources (*e.g.*, the Switch Traffic Grapher) are javascript components, implemented on YUI. CHS is a GUI component developed using CSS and javascript.



Beacon: IPAddr 143.54.12.23; Port 6633; Login root; Key ********

POX: IPAddr 143.54.12.45; Port 6633; Login root; Key **********

Floodlight: IPAddr 143.54.12.38; Port 6633; Login root; Key ********

Switch Traffic Grapher: Refresh Time (s) 60

OF Monitor: Traffic Graph Tool [wired]

Available Resources = 14
Rused = Beacon, POX, Floodlight, Switch Traffic Grapher, OF Monitor
Rconf = Beacon, POX, Floodlight, Switch Traffic Grapher

PMM = Performance Monitoring Mashment

Figure 4. Mashment Maker Prototype and Performance Monitoring Mashment

The Mediator Bus, SMRS, and Mashment Operations are Web Services based on the Representational State Transfer (REST) architectural style. We use REST-based services because they are suitable to achieve integration and interoperability in heterogeneous environments. Each Web Service that interacts with Beacon, Floodlight, and POX was created using the Java Jersey API 2.3, the Floodlight REST API 1.0, and the Java Socket API 1.0, respectively. In turn, each Web Service that communicates with VirtualBox, Xen, and VMware was correspondingly implemented using the VirtualBox SDK API 4.1, the XenSDK API 6.0, and the VMware WebServices SDK 5.1. The Mashment Router was built with Java Servlets and the Asynchronous Javascript and XML (AJAX). We use Servlets and AJAX because they allow the interactive and asynchronous interaction among the Maker GUI and the REST Web Services.

The Maker prototype was unfolded (see Figure 5) in the Apache-Tomcat Server 7.0 and the MySQL Server 5.1. In the Apache-Tomcat were deployed the Designer, Buttons, Visual Resources, CHS, SMRS, Mediator Bus, and Mashment Operations. In the MySQL were installed the Users Repository and Mashments Repository. The browser Mozilla Firefox was the client used to run the Maker GUI.

OpenFlow Networks. Three OpenFlow networks (see Figure 5) were built using, in the control tier, Beacon 1.0, Floodlight 0.9, and POX 1.0. Each OpenFlow controller was deployed to handle 27 Open vSwitches located in the datapath tier. These switches were deployed, in a tree topology, on Mininet that is an emulation platform for OpenFlow networks. The communication among the controllers and their switches was made by the OpenFlow protocol 1.0.

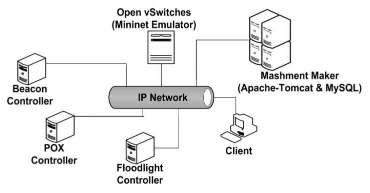


Figure 5. Test Environment

NMSit-SDN. Let's suppose the following *Situation*: the network administrator needs to identify which are the Open vSwitches that are causing sudden performance degradation on the OpenFlow networks described earlier. Thereby, he/she requires a situational solution that presents, in an integrated, visual, and intelligible way, network traffic information of Open vSwitches handled by Beacon, POX, and Floodlight. To get such a solution and deal with the *NMSit-SDN*, the network administrator has two options: *(i)* Without the Maker, to create and launch a *Situational Script*; or *(ii)* With the Maker, to develop and execute the *Performance Monitoring Mashment*. These options are evaluated and analyzed in the next subsection.

B. Complexity: Evaluation and Analysis

To evaluate our approach, initially, we measured the complexity of addressing the *NMSit-SDN* when the network administrator follows the proposed process to tackle NMSits but does not use the Maker. In a workspace without the Maker, he/she develops and executes a *Situational Script* that retrieves network traffic information from the switches handled by Beacon, POX, and Floodlight. Such a *Script* presents the information retrieved in a user interface formed by text-plane tables and chart images. According to the equation (1) and considering the no conducting of Tune ($i = j = k = e = 1$), $\zeta_{nomaker} = \zeta_{sel:nomaker} + \zeta_{con:nomaker} + \zeta_{com:nomaker} + \zeta_{exe:nomaker}$.

Select without Maker. The network administrator performs the selection of controller tools (BeaconTool, POXTool, and FloodlightTool) and their specific commands that allow to monitor Open vSwitches. An example of specific command is to retrieve the statistics of an Open vSwitch controlled by Floodlight: curl http://IPCtrller:8080/wm/core/switch/switchId/statType/json. This selection is complex because it is not tool-assisted and guidelines are scattered on the Internet. In this way, $\varsigma_m = 2$, $gF = 3$, and $cF = 1$. Using these values in the equation (2), $\zeta_{sel:nomaker} = \sum_{m=1}^{4} 2 + (nAvailableResources - 1) * 3 * 1$. Where, considering $nAvailableResources = 14$, we use this value to facilitate the comparison with the Maker prototype, $\zeta_{sel:nomaker} = 47$.

Configure without Maker. The network administrator configures BeaconTool, POXTool, FloodlightTool, and YUI Chart API by providing their corresponding functioning parameters. Thus, in accordance to the equation (3), $\zeta_{con:nomaker} = \varsigma_{beaconTool} + \varsigma_{poxTool} + \varsigma_{floodlightTool} + \varsigma_{yc}$. Where, $\varsigma_{beaconTool} = \varsigma_{poxTool} = \varsigma_{floodlightTool} = sourceParameter(login) + sourceParameter(key) + sourceParameter(ip) + sourceParameter(port) + sourceParameter(statisticCommand)$. As he/she takes the configuration information of controller tools from documentation easy to find on the Internet and defines the specific statistic commands after additional search, $sourceParameter(login) = sourceParameter(key) = sourceParameter(ip) = sourceParameter(port) = 2$ and $sourceParameter(statisticCommand) = 5$. Furthermore, since he/she extrapolates the YUI Chart configuration information from documentation simple to find on the Internet, $\varsigma_{yc} = 3$. Using these values, $\zeta_{con:nomaker} = 42$.

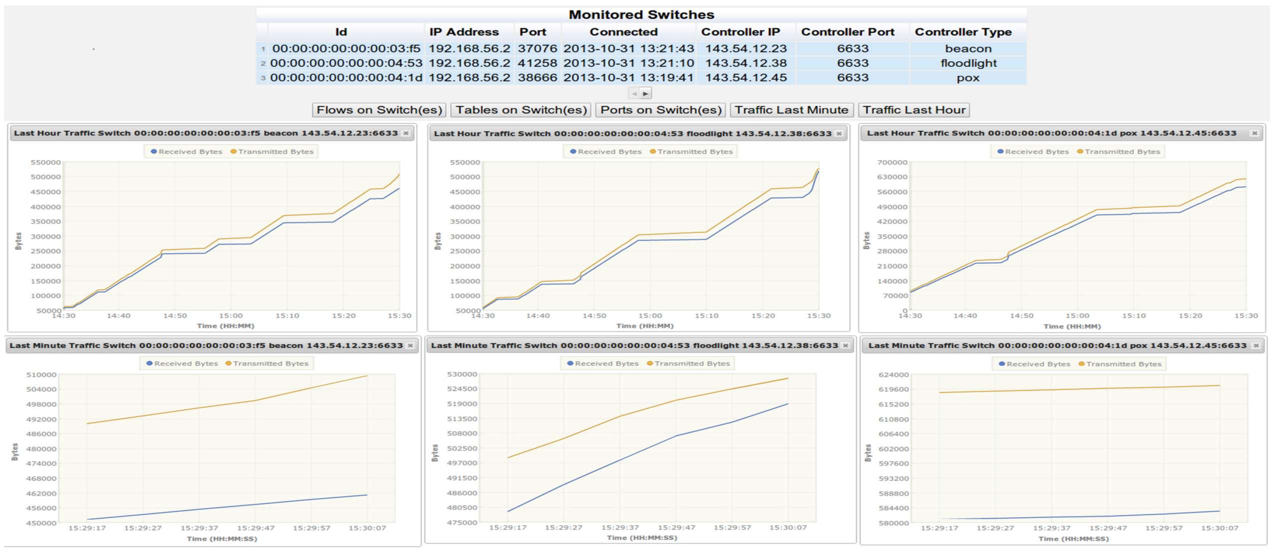
Combine without Maker. The network administrator manually develops (writes programming code) one logical link among each of controller tools and the YUI Chart API. Regarding these links, it is to point out that: *(i)* he/she adapts the data retrieved because controller tools, involved in the *NMSit-SDN*, use different data types (*e.g.*, Beacon employs data type in Java and Floodlight uses JSON); and *(ii)* he/she neither has explicit nor centralized guidelines to support the links development. Thus, $linkType(l) = 3$, $goF = 3$, and $coF = 1$. Using these values in the equation (4), $\zeta_{com:nomaker} = 21$.

Execute without Maker. As the network administrator launches the *Situational Script* by typing a specific command in a Linux Command Line, $\zeta_{exe:nomaker} = 2$. After executing this *Script*, the network administrator is able to find the Open vSwitches involved in the *NMSit-SDN*, by analyzing YUI Chart images and text-plane tables.

Once computed the intricacy of facing the *NMSit-SDN* without the Maker, we proceed to evaluate the complexity of developing and executing the *Performance Monitoring Mashment*. In a broad sense, in the Maker, the network administrator builds and launches this Mashment (see Figure 4) by dragging-and-dropping, wiring, and clicking Visual Resources and Buttons. The Maker also assists such a process by providing contextual guidelines for the network administrator. In accordance to the equation (1) and considering the non performing of Tune, $\zeta_{pmm} = \zeta_{sel:maker} + \zeta_{con:maker} + \zeta_{com:maker} + \zeta_{exe:maker}$.

Select on Maker. The network administrator uses the Drag-and-Drop service (*i.e.*, a Maker-assisted way) to select the Visual Resources ($M = 5$) that form the *Performance Monitoring Mashment*. Thus, R_{used} ={*Beacon, POX, Floodlight, Switch Traffic Grapher, OFMonitor*}. Furthermore, to facilitate such a selection, the Maker via CHS provides him/her contextual guidance about each of $nAvailableResources = 14$. Therefore, $\varsigma_m = 1$, $gF = 2$, and $cF = 1$. Using these values in the equation (2), $\zeta_{sel:maker} = 31$.

Configure on Maker. The network administrator configures ($N = 4$) Visual Resources, R_{conf} ={*Beacon,POX,Floodlight,Switch Traffic Grapher*}, by providing their functioning settings. Then, in accordance to the equation (3), $\zeta_{con:maker} = \varsigma_{beacon} + \varsigma_{pox} + \varsigma_{floodlight} + \varsigma_{stg}$. Where, $\varsigma_{beacon} = \varsigma_{pox} = \varsigma_{floodlight} = sourceParameter(login) + sourceParameter(key) +$



Monitored Switches

	Id	IP Address	Port	Connected	Controller IP	Controller Port	Controller Type
1	00:00:00:00:00:00:03:f5	192.168.56.2	37076	2013-10-31 13:21:43	143.54.12.23	6633	beacon
2	00:00:00:00:00:00:04:53	192.168.56.2	41258	2013-10-31 13:21:10	143.54.12.38	6633	floodlight
3	00:00:00:00:00:00:04:1d	192.168.56.2	38666	2013-10-31 13:19:41	143.54.12.45	6633	pox

Flows on Switch(es)
Tables on Switch(es)
Ports on Switch(es)
Traffic Last Minute
Traffic Last Hour
Last Hour Traffic Switch 00:00:00:00:00:00:03:f5 beacon 143.54.12.23:6633
Last Hour Traffic Switch 00:00:00:00:00:00:04:53 floodlight 143.54.12.38:6633
Last Hour Traffic Switch 00:00:00:00:00:00:04:1d pox 143.54.12.45:6633
Last Minute Traffic Switch 00:00:00:00:00:00:03:f5 beacon 143.54.12.23:6633
Last Minute Traffic Switch 00:00:00:00:00:00:04:53 floodlight 143.54.12.38:6633
Last Minute Traffic Switch 00:00:00:00:00:00:04:1d pox 143.54.12.45:6633
Received Bytes
Transmitted Bytes
Bytes
Time (HH:MM)
Time (HH:MM:SS)

Figure 6. Performance Monitoring Mashment on Runtime

$sourceParameter(ip) + sourceParameter(port)$ and $\varsigma_{stg} = sourceParameter(refreshTime)$. Considering that the Maker via CHS offers him/her configuration guidelines about Visual Resources, $\varsigma_{beacon} = 8$ and $\varsigma_{stg} = 2$. Using these values, $\zeta_{con:maker} = 26$.

Combine on Maker. The network administrator uses the Wire Service ($linkType(l) = 1$) to create $L = 4$ links: Beacon - OF Monitor, POX - OF Monitor, Floodlight - OF Monitor, and Switch Traffic Grapher - OF Monitor. Regarding these links, it is to stand out that: *(i)* he/she does not need to adapt the data transferred because the Mediator Bus is responsible for hiding the data mapping; and *(ii)* he/she obtains guidelines about links creation from the Maker via CHS. Therefore, $goF = 2$ and $coF = 1$. Using these values in the equation (4), $\zeta_{com:maker} = 12$.

Execute on Maker. Since the network administrator can run the *Performance Monitoring Mashment* from the Designer by clicking the Run Button, $\zeta_{exe:maker} = 1$. After launching this *Mashment* (see Figure 6), the network administrator can identify the three Open vSwitches implicated in the *NMSit-SDN*, by analyzing, in an integrated GUI, Switch Traffic Grapher images and HTML tables.

The Figure 7 depicts the obtained results in the complexity assessment when the network administrator faces the *NMSit-SDN* with and without the Maker. In accordance to these results, the use of the Maker: *(i)* diminishes the complexity of Select in 34.04%, $\zeta_{sel:maker} = 31 < \zeta_{sel:nomaker} = 47$, attained by the services Drag-and-Drop and CHS, *(ii)* reduces the complexity of Configure in 38.09%, $\zeta_{con:maker} = 26 < \zeta_{con:nomaker} = 42$, reached by CHS, *(iii)* decreases the complexity of Combine in 42.85%, $\zeta_{com:maker} = 12 < \zeta_{com:nomaker} = 21$, obtained by the Wire Service and the Mediator Bus; and *(iv)* diminishes the complexity of Execute in 50%, $\zeta_{exe:maker} = 1 < \zeta_{exe:nomaker} = 2$, gotten by the Designer.

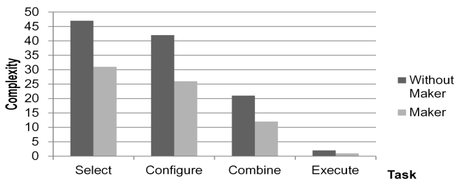


Figure 7. NMSit-SDN: Tasks Complexity

Since in a Maker-based workspace the complexity of each task is less than the corresponding complexity when the Maker is not used, $\zeta_{pmm} = 70$ is also less than $\zeta_{nomaker} = 112$ and the global reduction is 37.50%. Considering the above results, we demonstrated that if network administrators follow the process to develop and execute Mashments in the Maker, the complexity of handling NMSits is decreased. Consequently, we conclude that our approach can be used by network administrators to overcome the complexity of NMSits.

V. Conclusions and Future Work

In this paper, we introduced an approach that allows to overcome the complexity on the work performed by network administrators to face NMSits. The approach is formed by the Mashments conceptual model, the process to develop and execute Mashments, and the Maker that supports such model and process. Furthermore, we presented the complexity evaluation of process that the network administrator conducts to build up and run two solutions: *Situational Script* and *Performance Monitoring Mashment*. Both solutions were targeted to address the *NMSit-SDN*: identify switches that are suddenly causing performance degradation on OpenFlow networks handled by different controllers.

Our approach permitted the network administrator to address the complexity involved in overcoming the *NMSit-SDN*, confirming the importance of the Mashment conceptual model, the process to develop and execute Mashments, and the Maker. In this sense, using per-task metrics, we demonstrated that the complexity decreases when network administrators conduct the following situational tasks: Select, Configure, Combine, and Execute. Therefore, we can state that our approach cuts down the complexity on the work carried out by network administrators to cope with NMSits.

We consider the proposed approach as a step forward in the network management, the situation management, and the mashup technology. In this regard, we drive the first towards an environment focused on situations, composite situational solutions, and network administrators. We bring mashup foundations up to the second to perform its investigative and control aspects. We lead the third to a novel application domain located in the intersection of the situation management and the network management.

As future work, we plan to correlate time and complexity metrics, in order to evaluate the productivity of network administrators that face NMSits by Mashments. Furthermore, we are interested in propose the deployment costs model of our approach by considering the heterogeneity of resources to be integrated/combined. Finally, we also pretend to add more resources and services to improve the Maker implementation.

Acknowledgment

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

References

[1] G. Jakobson, J. Buford, and L. Lewis, "Situation Management: Basic Concepts and Approaches," in *Information Fusion and Geographic Information Systems*, ser. Lecture Notes in Geoinformation and Cartography, W. Cartwright, G. Gartner, L. Meng, M. . Peterson, V. . Popovich, M. Schrenk, and K. . Korolenko, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ch. 2, pp. 18–33.

[2] Kokar, Mieczyslaw M. and Matheus, Christopher J. and Baclawski, Kenneth, "Ontology-based Situation Awareness," *Information Fusion*, vol. 10, no. 1, pp. 83–98, january 2009.

[3] G. Jakobson, L. Lewis, C. Matheus, M. Kokar, and J. Buford, "Overview of Situation Management at SIMA," in *MILCOM*, 2005, pp. 1630 –1636 Vol. 3.

[4] S. George, W. Zhou, H. Chenji, M. Won, Y. O. Lee, A. Pazarloglou, R. Stoleru, and P. Barooah, "DistressNet: a Wireless ad hoc and Sensor Network Architecture for Situation Management in Disaster Response," *Communications Magazine*, vol. 48, no. 3, pp. 128–136, 2010.

[5] B. Magoutas, G. Mentzas, and D. Apostolou, "Proactive Situation Management in the Future Internet: The Case of the Smart Power Grid," in *DEXA*, september 2011, pp. 267 –271.

[6] D. M. Hein, R. Toegl, M. Pirker, E. Gatial, Z. Balogh, H. Brandl, and L. Hluchý, "Securing Mobile Agents for Crisis Management Support," in *STC*. New York, NY, USA: ACM, 2012, pp. 85–90.

[7] Pereira, I.S.A. and Costa, P.D. and Almeida, J.P.A., "A Rule-based Platform for Situation Management," in *CogSIMA*, 2013, pp. 83–90.

[8] O. Caicedo, F. Estrada, and Granville., "A Mashup Ecosystem for Network Management Situations," in *Globecom*, december 2013, pp. 2271–2277.

[9] X. Chen, Y. Mao, Z. M. Mao, and J. Van der Merwe, "Declarative Configuration Management for Complex and Dynamic Networks," in *Co-NEXT*. New York, NY, USA: ACM, 2010, pp. 6:1–6:12.

[10] N. Kim and J. Kim, "Building NetOpen Networking Services over OpenFlow-based Programmable Networks," in *ICOIN*, january 2011, pp. 525 –529.

[11] D. Mattos, N. Fernandes, V. da Costa, L. Cardoso, M. Campista, L. Costa, and O. Duarte, "OMNI: OpenFlow MaNagement Infrastructure," in *NOF*, november 2011, pp. 52 –56.

[12] J. de Santanna, J. Wickboldt, and L. Granville, "A BPM-based Solution for Inter-domain Circuit Management," in *NOMS*, 2012, pp. 385–392.

[13] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing Software-defined Networks," in *NSDI*. Berkeley, CA, USA: USENIX Association, 2013, pp. 1–14.

[14] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," *Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[15] D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, and A. Singh, "Damia: Data Mashups for Intranet Applications," in *ACM SIGMOD*. New York, NY, USA: ACM, 2008, pp. 1171–1182.

[16] N. Laga, E. Bertin, R. Glitho, and N. Crespi, "Widgets and Composition Mechanism for Service Creation by Ordinary Users," *Communications Magazine*, vol. 50, no. 3, pp. 52–60, 2012.

[17] N. Ozkan and W. Abidin, "Investigation of Mashups for Managers," in *ISCIS*, september 2009, pp. 622 –627.

[18] H. Gebhardt, M. Gaedke, F. Daniel, S. Soi, F. Casati, C. Iglesias, and S. Wilson, "From Mashups to Telco Mashups: A Survey," *Internet Computing*, vol. 16, no. 3, pp. 70–76, may-june 2012.

[19] V. Stirbu, Y. You, K. Roimela, and V. Mattila, "A Lightweight Platform for Web Mashups in Immersive Mirror Worlds," *Pervasive Computing*, vol. 12, no. 1, pp. 34–41, 2013.

[20] Y. Han, G. Wang, G. Ji, and P. Zhang, "Situational Data Integration with Data Services and Nested Table," *Service Oriented Computing and Applications*, vol. 7, no. 2, pp. 129–150, 2013.

[21] C. dos Santos, R. Bezerra, J. Ceron, L. Granville, and L. Rockenbach Tarouco, "On Using Mashups for Composing Network Management Applications," *Communications Magazine*, vol. 48, no. 12, pp. 112–122, december 2010.

[22] O. Caicedo, F. Estrada, and Granville., "A Mashup-based Approach for Virtual SDN Management," in *COMPSAC*, july 2013, pp. 143–152.

[23] J. Yu, B. Benatallah, F. Casati, and F. Daniel, "Understanding Mashup Development," *Internet Computing*, vol. 12, no. 5, pp. 44 –52, sept.-oct. 2008.

[24] C. dos Santos, R. Bezerra, J. Ceron, L. Granville, and L. Tarouco, "Botnet Master Detection Using a Mashup-based Approach," in *CNSM*, october 2010, pp. 390 –393.

[25] B. R.S., C. dos Santos, L. Bertholdo, L. Granville, and L. Tarouco, "On the Feasibility of Web 2.0 Technologies for Network Management: A Mashup-based Approach," in *NOMS*, april 2010, pp. 487 –494.

[26] Y. Diao and A. Keller, "Quantifying the Complexity of IT Service Management Processes," ser. DSOM'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 61–73.

A Mashup Ecosystem for Network Management Situations

Oscar Mauricio Caicedo Rendon
Computer Networks Group
Institute of Informatics
University Federal do Rio Grande do Sul
Email: omcrendon@inf.ufrgs.br

Felipe Estrada-Solano
Telematics Engineering Group
Telematics Department
University of Cauca
Email: festradasolano@unicauca.edu.co

Lisandro Zambenedetti Granville
Computer Networks Group
Institute of Informatics
University Federal do Rio Grande do Sul
Email: granville@inf.ufrgs.br

***Abstract*—Current network management approaches and their implementations are not intended to address dynamic situations that need rapid delivery of good-enough and comprehensive solutions. In this paper, we introduce a novel mashup ecosystem, called Mashment Ecosystem, that allows Network Administrators to conduct on a Mashment Maker the activities and interactions necessary to provide Mashments. Mashments are mashups aimed to tackle network management situations. We evaluate the Mashment Ecoystem by estimating with the Keystroke-Level Model and measuring in a test scenario the time that Network Administrators take to perform the activities of creating, launching, and publishing Mashments. Similarly, we evaluate the time for retrieving information about a network management situation by using or not Mashments. The evaluation results corroborated that Network Administrators, in our ecosystem, need short-time to deal with network management situations.**

I. Introduction

Nowadays, the use of computer networks has become vital to most enterprises. Up to now, in these networks, many management tasks require manual intervention by Network Administrators, mainly, to manage dynamic *Situations* that need rapid delivery of good-enough and comprehensive solutions [1]. In general, a *Situation* is a collection of entities (*i.e.*, things in a domain), their attributes, and relations in a time interval [2]. Hereinafter, we call a network management *Situation* as NMSit. Information technology departments do not provide situational solutions for NMSits because the requirements of these types of *Situations* are usually located in the long-tail of enterprise needs [3] [4]. As result, Network Administrators must create by themselves solutions for NMSits.

The Situation Management (SM) is an approach to provide solutions that enable to analyze, correlate, and coordinate the interaction between people, information, technologies, and actions intended to overcome *Situations* [2][5]. SM includes three aspects always linked to the time axis [2]: (*i*) the investigative aspect is related to retrospective cause analysis of *Situations* (*e.g.*, finding the root of a packet transmission failure in an OpenFlow-based network), (*ii*) the control aspect is directed to change or preserve *Situations* (*e.g.*, migrating a virtual switch from an overloaded server); and (*iii*) the predictive aspect is aimed to presage *Situations* (*e.g.*, projecting the appropriate time to migrate a critical router before a service disruption happen). In addition, as *Situations* are dynamic, SM emphasizes an adaptive management style.

Current network management solutions, such as Ganglia[6] and Nagios [7], are not intended to address the NMSits. Regarding these solutions, it is important to point out that they are not compatible, difficulting the information collection and information fusion that are necessaries to deal with a NMSit. Furthermore, the referred solutions are created without taking into account their rapid integration, extension, and improvement by Network Administrators, making hard the coping of NMSits. Thus, during a NMSit, the job of Network Administrators is hindered, since they are not able to enhance their workspace and must use numerous mismatched solutions from the Web and the network management.

Different research works have been developed about network management in traditional [8], virtual [9], Software Defined (SDN) [10], and cloud [11] networks. Such researches do not focus on face the NMSits and are developer centric. In the last years, mashups [12], that are end-user centric solutions formed by combining resources from different providers, have been applied in several domains as situational projects [13] and natural disasters [14] [15]. Nevertheless, mashups have not been used for tackling the NMSits. In this way, we raise the following question: how to tackle the NMSits by focusing on the Network Administrator?. In order to answer this question, we introduce a mashup ecosystem, named Mashment Ecosystem, which allows to carry out SM in network management. To the best of our knowledge, this work is the first to use an approach centric in the Network Administrator, SM, and mashup-based to deal with NMSits.

The Mashment Ecosystem, first, helps and encourages to Network Administrators to build up Mashments (*i.e.*, mashup used to deal with a NMSit) by themselves. Second, it allows Network Administrators to collect, correlate, and fuse information from heterogeneous resources offered by diverse providers. Third, it promotes the sharing and reuse of Mashments to avoid their wasting and to push the rise of innovative ones. Summarizing, the key contributions presented in this paper are to: (*i*) propose a novel Mashment Ecosystem for rapidly tackling NMSits by focusing in the Network Administrator; and (*ii*) demonstrate the short time that the Network Administrator needs to address a NMSit by building up and using a Mashment in our ecosystem.

The remainder of this paper is organized as follows. In Section II, we present SM and the mashup technology. In Section III, we introduce the Mashment Ecosystem. In Section IV, we expose and analyze the case study developed to evaluate our proposal. In Section V, we provide some conclusions and implications for future work.

II. Background

In this section, we present a SM background. Also, we describe mashups and research works about mashups–SM.

A. Situation Management

SM is an emerging approach to provide solutions that need the planning and implementing of actions aimed to overcome a determined Situation [2]. SM requires the use of novel techniques [5], first, to collect time/state information about *Situations*. Second, to correlate and fuse multi-source information for timely and correct decision making in *Situations*. Third, to analyze past and predict future *Situations*. Fourth, to present information aiming at the human comprehension maximization.

Solutions based on the SM concepts are found in several domains. For instance, in the domain of polyester film base manufacturing, a situational solution, based on an expert system, has been proposed for monitoring the non-steady state events and assisting human operators with the event tasks [16]. In the aviation domain, a scalable and distributed situational system has been introduced for the management of air security incidents such as terrorist attacks that need, to be overcomed, the coordination and sharing of information from different organizations [17]. An architecture, based on SM, the Service Oriented Architecture, and developer centric, has been outlined for supporting demand response aspects of the smart grid domain [18]. Althoug SM has been used in several domains, there is not a SM-based approach to deal with NMSits.

B. Mashups

Mashups are web applications centered in end-users and built up by combining several resources (*e.g.*, data, application logic, and user interfaces) from one or more providers [12]. Here, end-user centric means that mashups can be built by users without advanced programming skills. In addition, regarding the mashups is to noteworthy [19]. First, they encourage the sharing among end-users. Second, the providers that supply resources, the end-users/developers that create mashups, and the end-users that use mashups act as a single unit known as mashup ecosystem.

If a mashup is developed for rapidly coping an immediate need of one or a set of end-users, it can be considered as a situational solution [20]. Mashups have been useful to manage *Situations* in diverse domains. For instance, Mashups were used to help to overcome a fire emergency in San Diego (California, United States) by sharing weather and rescue information among civil organizations and the government [14]. In situational projects that involve a small number of users and have a short lifespan, a mashup environment has been introduced in order to support management tasks. In such environment, the project manager is able to quickly develop a mashup for visualizing and filtering the information of his/her project [13]. An architecture, based on the Web 2.0 and wireless sensor networks, has been proposed in order to estimate the speed and timing of possible floods. A mashup prototype that collects, correlates, and presents data from multiple wireless sensors was developed to test the architecture [15]. Despite the use of mashups in several domains, there is not a mashup-based approach for tackling the NMSits.

III. Mashment Ecosystem

In order to better explain our proposal, we present an overview of the Mashment Ecosystem. Subsequently, we describe its Resources, Stakeholders, Activities&Interactions, and Software Entities.

A. Overview

Current network management approaches do not focus on dealing with NMSits. In the same way, although the mashup technology provides good basis for developing composite situational solutions by end-users, it has not been used for tackling the NMSits. Therefore, there is a gap in the mashup and network management related research and, consequently, there is a chance for innovation. Hereinafter, we present how a Mashment Ecosystem, based on the abstraction of resources, the mashups composition model, and a Network Administrator centric approach, can be targeted to address the NMSits. In particular for coping the NMSits, the Mashment Ecosystem faces three issues: (*i*) the complexity and heterogeneity to collect, correlate, and fuse information from multiple resources of the Web and the network management, (*ii*) the demand by functionalities that allow Network Administrators to rapidly create adaptable solutions for NMSits; and (*iii*) the need by visualization functionalities that enable Network Administrators to get NMSit information, in a very understandable way.

Before detailing the Mashment Ecosystem, we introduce the Mashment concept and a motivating scenario. A Mashment is a tunable situational mashup that allows Network Administrators (their end-users) to tackle a NMSit by combining diverse types of resources from multiple providers. Tunable means that Mashments are adaptable and easily customizable. Since Mashments are a special type of mashup, they can be created by Network Administrators. Regarding a Mashment is also relevant to point out. First, it hides the heterogeneity, complexity, and stiffness of resources used to deal with a NMSit. Second, it bears the easy collection, correlation, and fusion of information about a NMSit. Third, it presents NMSit information, in a visual and clear way. Fourth, it can be rapidly created to cope a determined NMSit.

Motivating Scenario. Let's suppose the following NMSit: in a virtual network formed by OpenFlow-based heterogeneous Slices from different providers (NP_a, NP_b, and NP_c), a packet failure transmission occurs because of sudden and unidentified errors. To tackle this NMSit, the Network Administrator needs to found errors from Slices in NP_a, NP_b, and NP_c. As every NP uses a different OpenFlow Controller, aiming at overcoming this NMSit, the Network Administrator has three options. The first one is to collect, correlate, and visualize network monitoring information by using disparate solutions, such as command line interfaces to execute specific commands on each Controller, distinct web user interfaces to monitor virtual switches, and external web tools to display non-integrated information about packet traffic. A drawback of first option is that the use of several mismatched solutions consumes more time than use an integrated solution. The second option is to develop a low-level script to integrate the aforementioned commands, user interfaces, and web tools. This option also consume a lot of time because, the Network Administrator usually does not have advanced knowledge in programming. The third option is to participate in the Mashment Ecosystem. A Network Administrator in our ecosystem is able to quickly build up, in a high-level abstraction, by him/herself a Mashment to face the described NMSit. This Mashment hides the resources heterogeneity from NP_a, NP_b, and NP_c. Furthermore, the Mashment presents the network management information of virtual network in an integrated and intelligible way.

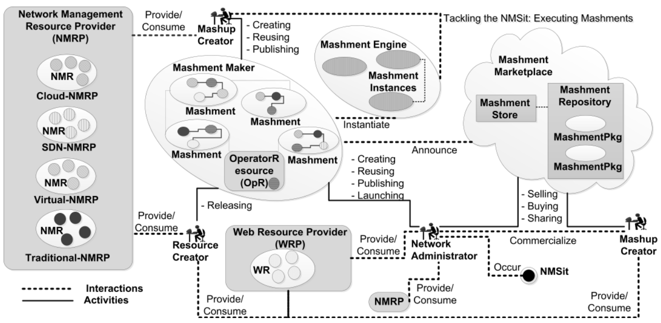


Fig. 1: Mashment Ecosystem

The Mashment Ecosystem (see Figure 1) is formed by: resources (Network Management Resources, Web Resources, and Operator Resources), Mashments, stakeholders (Network Administrator, Mashup Creator, Resource Creator, Web Resource Providers, Network Management Resource Providers, and Software Entity Providers), software entities (Mashment Maker, Mashment Engine, Mashment Repository, and Mashment Store), activities performed by stakeholders, interactions between stakeholders, and interactions between software entities. In this ecosystem, Network Administrators and Mashup Creators build up Mashments by using the Mashment Maker. Mashments are made up of resources from different providers and are executed in the Mashment Engine. The resources are released by the Resource Creator. In the Martketplace, the Mashments are shared, sold, and purchased by Mashup Creators and Network Administrators.

If a NMSit occurs, a Network Administrator can address it as follows: (*i*) buying or getting free of charge a Mashment(s), rapidly creating one or more Mashments, or quickly reusing a Mashment(s) previously built; and (*ii*) executing the purchased or created Mashment(s). It is important to highlight that the Mashment Ecosystem evolves over time because of the emerging and perishing of resources, the sharing and commercialization of Mashments, and the dynamic interactions between stakeholders.

The Mashment Ecosystem can be expressed as $MEco = \{M, St, A, I, Se\}$. Where: St, A, I, and Se are sets of stakeholders, activities, interactions, and software entities, respectively. In turn, the set of Mashments $M = \{m_i | m_i = (R_{used}, r_{root}, \delta, nmsit) : R_{used} \subset R, r_{root} \in R, nmsit \in NMSit\}$. Here, R_{used} is the set of resources on m_i, r_{root} is the root resource that starts the m_i execution, δ is the execution flow of resources on m_i, and $nmsit$ is the specific NMSit tackled by m_i. The other sets forming our ecosystem are described in next subsections.

B. Resources

A resource is a clearly identifiable entity in a time interval, which is conceived or can be adapted to tackle a NMSit. The set of resources is $R = \{r_i | r_i \in NMR \cup WR \cup OpR\}$. Where, Network Management Resources (NMR) are entities intended for the network management. NMR examples are Ganglia to manage traditional networks, Citrix Center for monitoring virtual resources, NetOpen to control OpenFlow-based networks, network monitoring systems based on the Simple Network Management Protocol, and all Application Programming Interfaces (API) that provide interaction with network elements.

Web Resources (WR) are Internet entities conceived or useful (via adaptation) for the network management. WR examples are the Google Maps API to show the geographic location of several network devices, the Multi Router Traffic Grapher (MRTG) to generate web pages with images presenting the traffic of network links, and the RRDTool to display over time the performance data of routers.

Operator Resources (OpR) are entities for combining resources (*i.e.*, NMR, WR, and even Mashments). There are two classes of OpR. Configuration_OpR to set up parameters for both access and communication to resources. An example of Configuration_OpR is a service to configure the security credentials required to monitor a virtual router. Control_OpR are composition patterns, such as $Split$, $Merge$, $Aggregate$, $Invoke$, $Trigger$, and $Receive$, useful, for instance, to collect, correlate, and fuse resources.

C. Stakeholders

A stakeholder affects and is affected by the activities and interactions performed by other one. The set of stakeholders is $St = \{st_i | st_i \in NMRP \cup WRP \cup SEP \cup ResourceCreators \cup MashupCreators \cup NetworkAdministrators\}$. Where, The Network Management Resource Provider (NMRP) is in charge of supplying NMR. Citrix Systems and Cisco Systems, providing solutions and programming interfaces to manage virtual servers and network devices, are examples of NMRP. The Web Resource Provider (WRP) is responsible for supplying WR. An example of WRP is a big player as Yahoo Inc. that provides visualization libraries and map services useful to present network management information. Another example is Oetieker&Partners Inc. that supplies web solutions intended for network monitoring, such as RDDTool and SmokePing.

The Software Entity Provider (SEP) is in charge of offering one or more software entities. In general, the Mashment Maker (containing visual representations of NMR, WR, OpR, and Mashments) and the Mashment Engine are provided, in an unified way, by the same SEP. In turn, the Mashment Repository and Mashment Store are usually offered, in a distributed way, by different SEP. In the Mashment Ecoystem can exist several Makers, Engines, Repositories, and Stores.

Before participating in the building of a Mashment, many WR and NMR need of adaption, in data format and/or communication protocol. The Resource Creator is responsible for this adaptation that we called releasing. Since such a releasing requires strong programming skills, Resource Creators are usually software companies and professional developers. The Open Software community, that provides APIs to interact via standardized protocols with network devices and servers containing virtual routers, is an example of Resource Creator.

The Mashup Creator creates, publishes, and launches Mashments by means of the Mashment Maker and the Mashment Engine. Also, he/she is able to share, sell, and buy Mashments in the Marketplace. A Mashup Creator can get profits by commercializing Mashments. Software companies, professional developers, and end-users are examples of this class of stakeholder.

The Network Administrator is responsible for tackling the NMSits in traditional, virtual, SDN, and cloud networks. The Mashment Ecosystem allows Network Administrators to: (*i*) create and execute Mashments, (*ii*) reuse existing Mashments, NMR, OpR, and WR, (*iii*) improve their workspace as result of (*i*) and (*ii*); and (*iv*) get profits through publishing and selling Mashments.

D. Activities and Interactions

The activities are actions conducted by stakeholders in the software entities. The set of activities is $A = \{Releasing, Creating, Reusing, Publishing, Launching, Selling, Buying, Sharing\}$. Where, $Releasing$ is carried out by Resource Creators to enable the combination of heterogeneous resources through adapting NMR and WR. After $Releasing$, the adapted resources can be used to build up Mashments.

Mashup Creators and Network Administrators perform $Creating$, $Reusing$, $Publishing$, $Sharing$, $Selling$, and $Buying$. $Creating$ allows to build up a Mashment (*i.e.*, define δ or execution flow), which involves: (*i*) discover the available resources (NMR, WR, OpR, and Mashments), (*ii*) select the suitable resources to address a NMSit, (*iii*) orchestrate a static or dynamic plan for tackling a NMSit, by combining the previously selected resources, (*iv*) monitor if the WR, NMR, and Mashments used on the orchestrated plan are available and flawless; and (*v*) reconfigure the orchestrated plan if any resource is in flaw or unavailable.

$Reusing$ enables to take advantage of existing Mashments, aiming at the creation of more complex and innovative Mashments. $Publishing$ allows to package and put Mashments in the Mashment Repository, aiming at their sharing, selling, and buying. $Sharing$ enables to offer and get Mashments free of charge. $Selling$ and $Buying$ allow the commercialization of Mashments. $Sharing$, $Selling$, and $Buying$ promote the reuse of Mashments and encourage the evolution of the Mashment Ecosystem. Mashments (built and purchased) are sent to execute through performing $Launching$, which, in turn, is conducted by Network Administrators. Every Mashment launched is called Mashup Instance.

The interactions take place in the relationships: stakeholder/stakeholder and software entity/software entity. The set of interactions can be expressed as $I = \{Provides, Consume, Tackling, Commercialize, Occur, Instantiate, Announce\}$. Where, $Provide$ and $Consume$ occur from the need of supplying and consumption of NMR and WR, during: the building up of the Mashment Maker, the resources releasing, and the Mashments creation that enhances and improves the Mashment Maker. $Provide$ and $Consume$ take place among: Resource Providers, Resource Creator, Mashup Creators, and Network Administrators.

$Instantiate$ is conducted among the Mashment Maker and the Mashment Engine when a Network Administrator launches a Mashment. $Announce$ is performed among the Mashment Maker and the Mashment Marketplace, aiming at publishing of Mashments that can be later shared, purchased, and sold in $Commercialize$. Mashup Creators and Network Administrators carry out $Commercialize$. $Occur$ and $Tackling$ are special interactions used to represent the emerging of a NMSit and the corresponding responses offered by Mashment Instances. During $Tackling$, Mashment Instances interact with NMR, WR, and OpR in order to face a NMSit.

E. Software Entities

The software entities are responsible for supporting and automating the activities and interactions aforedescribed. The set of software entities is $Se = \{Maker, Engine, Marketplace\}$. Where, the Mashment Maker allows Network Administrators and Mashment Creators to build up Mashments, in an easy and rapid way. The Maker is a development environment that provides a composition approach, high-level programming tools, and a lightweight development process. The composition approach consists of four phases related to $Creating$ and $Reusing$ activities: (*i*) discover and select, (*ii*) orchestrate, (*iii*) monitor and reconfigure; and (*iv*) reuse. The above phases enable, first, to use the last information retrieved from available resources. Second, to avoid the use of redundant, incomplete, or irrelevant information provided by resources in failure. Third, to avoid the lack of information because of unavailable resources.

The high-level programming tools are visual facilities, based on drag-and-drop and wire mechanisms, that allow to perform the composition approach, $Publishing$, and $Launching$. Such tools are responsible for hiding the data mapping among WR, NMR, and OpR. The data mapping is a problem particularly daunting for Network Administrators, because, generally, they are not expert developers. As the Mashment Maker is devoted to Network Administrators, we define a simple process to tackle whatever NMSit: (*i*) *conduct the approach of composition* above described and so build up Mashments for the NMSit or *buy the Mashment* for the NMSit, (*ii*) *use the Mashments* to deal the NMSit; and (*iii*) *maintenance of Mashments* to avoid their malfunctioning.

The Mashment Marketplace allows to establish a new value chain in which revenues are shared not only by WRP and NMRP but all stakeholders. Marketplaces that involve end-users (as Network Administrators) and professional developers (as Mashup Creators) have proved valuable to promote the evolution over time of Service Ecosystems (as the Mashment Ecosystem). The Android Market and the Apple Store are succesful examples of solutions marketplaces. The Mashment Marketplace is make up of the Mashment Store(s) and the Mashment Repository(s). In the Mashment Store are performed

$Selling$, $Sharing$, and $Buying$. As result of $Announce$, in the Mashment Respositoy are stored MashmentPkgs (a packaged Mashment) to be sold or shared. The reuse of existing Mashments, by $Selling$ and $Sharing$, is a key aspect for the evolution of proposed ecosystem.

The Mashment Engine is responsible for the lifecycle (*i.e.*, $Instantiate$) of Mashment Instances that are Mashments on run time. A Mashment Instance allows to tackle (*i.e.*, $Tackling$) a NMSit. As the Mashments are make up of WR, NMR, OpR, and even Mashments, that function in back-end or front-end, the Mashment Engine is an enabler to create, destroy, and cache such resources into both web servers and web/mobile clients.

IV. Case Study

To assess our proposal, we perform a test environment for the Mashment Ecosystem. This section describes the test conditions and analyzes the obtained results.

A. Test Environment

The Figure 2 depicts the test environment for the raised case study. To set up such environment, first, we created a heterogeneous virtual network. Second, we developed the Mashment Maker, the Mashment Engine, and the Marketplace Repository (as a Database). Third, we released into the Mashment Maker the resources to create, launch, and publish a Mashment, hereinafter called MVN. When suddenly and unidentified transmission errors occur (*i.e.*, the NMSit) in the virtual network built, MVN allows the Network Administrator to visually look for them (*i.e.*, tackling the NMSit) by presenting, in a visual and integrated way, the collected, correlated, and fused information about packet traffic from switches.

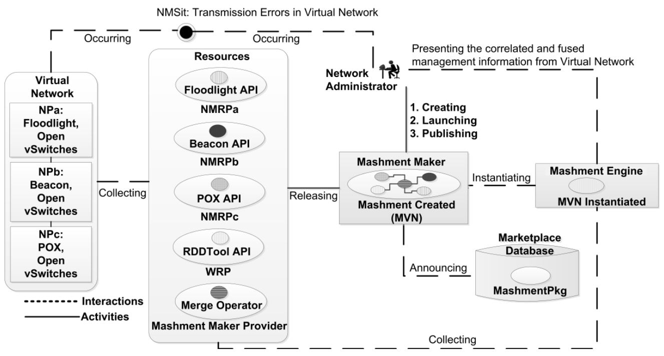


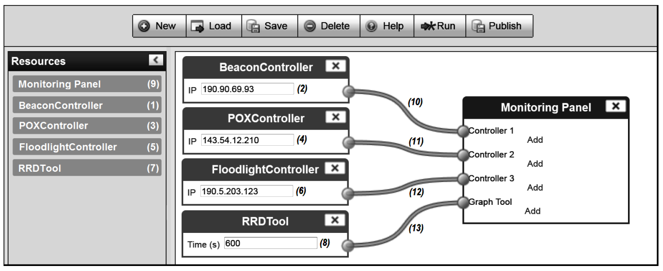
Fig. 2: Test Environment

The virtual network was created by using Open vSwitch and one different OpenFlow Controller in each network provider (NP_a, NP_b, and NP_c), namely, Floodlight, Beacon, and POX. Beacon and Floodlight are controllers based on the Java programming language. In turn, POX is based on Python. The controllers and switches were deployed on the Mininet that is a software for emulating OpenFlow networks. The Mininet was executed on Oracle VM VirtualBox. The Mashment Maker was developed by using Asynchronous Javascript and XML (AJAX), web services based on the Representational State Transfer (REST), and APIs for Floodlight, Beacon, POX, and RRDTool. The Mashment Maker was deployed on the Apache Tomcat Server. This Apache Tomcat was used as the Mashment Engine in the server-side. In the client-side, the Mashment Engine used was Firefox. The Marketplace Database was implemented on a MySQL Server. Network Administrators created the MVN by using the Mashment Maker.

B. Evaluation and Analysis

To evaluate the Mashment Ecosystem, we estimated and experimentally measured the time that Network Administrators take to perform $Creating$, $Launching$, and $Publishing$ MVN. MVN (see Figure 3) is formed by five visual components (*i.e.*, R_{used}): BeaconController, FloodlightController, and POXController representing the OpenFlow controllers (*i.e.*, NMR) used in the virtual network, RRDTool (*i.e.*, WR) representing the web tool used to generate images that present packet traffic information, and Monitoring Panel that is a merge operator (*i.e.*, OpR) and the root resource (*i.e.*, r_{root}). Subsequently, we also estimated and experimentally measured the time that Network Administrators take to retrieve, by using MVN, information about the NMSit above presented.

The time estimation was made by using the Keystroke-Level Model (KLM). In KLM, each activity is modeled as a sequence of actions. The time average for KLM actions is [21]: (*i*) Press and release a key $\rightarrow K = 0.2s$, (*ii*) Type a string $\rightarrow T_n = n * K$, (*iii*) Hold or release the mouse $\rightarrow B = 0.1s$, (*iv*) Point the mouse $\rightarrow P = 1.1s$, (*v*) Move the hand from mouse to keyboard or viceversa $\rightarrow H = 0.4s$; and (*vi*) Mental preparation $\rightarrow M = 1.35s$. In addition to these actions, we used, drag-and-drop a visual element $\rightarrow T_{dnd}$ and wire two visual elements $\rightarrow T_{wire}$. T_{dnd} and T_{wire} are given by [4]: $T_{dnd} = P + 2B = 1.3s$ and $T_{wire} = 3P + 8B = 4.1s$.



IP 190.90.69.93 (2)

IP 143.54.12.210 (4)

IP 190.5.203.123 (6)

Time (s) 600 (8)

Fig. 3: Mashment Maker - Creating MVN

To tackle the NMSit, the Network Administrator creates MVN, the corresponding actions sequence (*i.e.*, δ) is as follows (see Figure 3): ***(1)*** Drag-and-drop Beacon $\rightarrow T_{dnd}$, ***(2)*** Configure Beacon (IP Address=190.90.69.93) $\rightarrow T_{con12}$, ***(3)*** Drag-and-drop POX $\rightarrow T_{dnd}$, ***(4)*** Configure POX (IP Address=143.54.12.210) $\rightarrow T_{con13}$, ***(5)*** Drag-and-drop Floodlight $\rightarrow T_{dnd}$, ***(6)*** Configure Floodlight (IP Address=190.5.203.123) $\rightarrow T_{con13}$, ***(7)*** Drag-and-drop RRDTool $\rightarrow T_{dnd}$, ***(8)*** Configure RRDTool (Time in seconds = 600) $\rightarrow T_{rrd}$, ***(9)*** Drag-and-drop Monitoring Panel $\rightarrow T_{dnd}$, ***(10)*** Wire Beacon to Monitoring Panel $\rightarrow T_{wire}$, ***(11)*** Wire POX to Monitoring Panel $\rightarrow T_{wire}$, ***(12)*** Wire Floodlight to Monitoring Panel $\rightarrow T_{wire}$; and ***(13)*** Wire RRDTool to Monitoring Panel $\rightarrow T_{wire}$. Where, $T_{con38} = P + 2H + T_{n=38} = 8.8s$ and $T_{rrd} = P + 2H + T_{n=3} = 2.5s$.

According the previous sequence, the estimated time for *Creating* MVN is $\mathbf{C_{est} = 13M + T_{con38} + T_{rrd} + 5T_{dnd} + 4T_{wire}}$. Then, it is expected that, by using the Mashment Maker, Network Administrators take $56.25s$ to build up MVN. We consider this $\mathbf{C_{est}}$ is good because, for instance, just typing the example script (10 lines with 40 characters each one) to generate a single RRD image takes $T_{script} = 10M + T_{n=400} = 93.5s$. In this way, we can state that Network Administrators, participating in the proposed ecosystem, can rapidly create Mashments aimed to tackle a NMSit. Generalizing, the estimated time to create any Mashment can be expressed as: $\delta(t) = T_{sel} + T_{conn} + T_{conf} + T_{ment}$. Where, $T_{sel} = \sum_1^i T_{dnd}$, $T_{conn} = \sum_1^j T_{wire}$, $T_{conf} = P + H + (nt * K)$, and $T_{ment} = M * (i + j + o)$. Here, i is the number of elements in the set R_{used} (drag-and-dropped), j is the number total of wires linking R_{used}, nt is the number total of characters to configure R_{used}, and o is the number of R_{used} to be configurated. In $\delta(t)$ a dynamic composition model could be used to decrease both $Tsel$ and T_{conn}. This dynamic composition for Mashments is a future work.

The Network Administrator performs the following actions sequence for *Launching* MVN. This sequence is the same for every Mashment: ***(1)*** Drag-and-drop MVN (when a Mashment is created, it is represented as a resource in the Mashment Maker) $\rightarrow T_{dnd}$, ***(2)*** Point the mouse to Run button $\rightarrow P$; and ***(3)*** Press and release Run button $\rightarrow 2B$. Therefore, the estimated time for *Launching* is given by $\mathbf{L_{est} = 3M + P + T_{dnd} + 2B}$. As result, it is expected that Network Administrators take $6.65s$ to start MVN by means of the Mashment Maker.

The Network Administrator performs the following actions sequence for *Publishing* MVN. This sequence is the same for every Mashment. ***(1)*** Point the mouse to Save button $\rightarrow P$, ***(2)*** Point the mouse to dialog that asks the Mashment name $\rightarrow P$, ***(3)*** Type the string MVN $\rightarrow T_{n=3}$, ***(4)*** Mouse press and release to store MVN in the Mashment Maker $\rightarrow 2B$, ***(5)*** Point the mouse to Publish button $\rightarrow P$, ***(6)*** Point the mouse to dialog that asks the Marketplace location $\rightarrow P$, ***(7)*** Type a repository string, for instance, http://www.mashments.mplace.com/repos $\rightarrow T_{n=37}$; and ***(8)*** Mouse press and release to store MVN in the Marketplace Database $\rightarrow 2B$. Therefore, the time estimated for the *Publishing* activity is given by $\mathbf{P_{est} = 8M + 4(P + B) + T_{n=3} + T_{n=37}}$. As result, it is expected that the Network Administrator takes $23.60s$ to publish any Mashment. Afterwards, MVN and, in general, any Mashments can be shared, sold, and purchased in the Marketplace Store that will be presented in a future work.

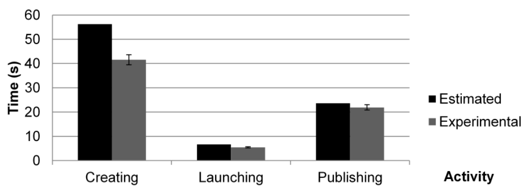
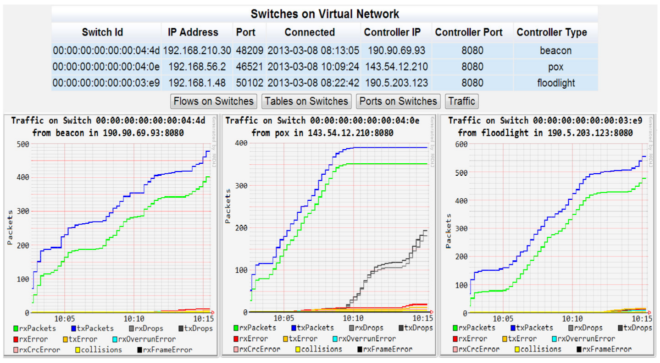


Fig. 4: Activities Time: Estimated vs Experimental

We also conducted an experimental study to measure the time that Network Administrators take to perform *Creating*, *Launching*, and *Publishing*. In the study participated 30 Network Administrators whose age ranged from 22 to 35. Although all participants frequently had used web tools none of them had used a mashup maker before. Thus, each participant was trained to use the Mashment Maker by 45 minutes. We took the experimental average time in seconds with a 95% confidence level. The results of estimated and experimental times (see Figure 4) corroborate the short time that Network Administrators need to tackle a NMSit by performing activities in the proposed ecosystem. Furthermore, as the experimental times of *Creating* ($41.55s$), *Launching* ($5.46s$), and *Publishing* ($21.92s$) were always less than the corresponding estimated times, we can state that the implementation of proposed ecosystem had a good behavior in front of KLM estimations.


Switches on Virtual Network

Switch Id	IP Address	Port	Connected	Controller IP	Controller Port	Controller Type
00:00:00:00:00:00:04:4d	192.168.210.30	48209	2013-03-08 08:13:05	190.90.69.93	8080	beacon
00:00:00:00:00:00:04:0e	192.168.56.2	46521	2013-03-08 10:09:24	143.54.12.210	8080	pox
00:00:00:00:00:00:03:e9	192.168.1.48	50102	2013-03-08 08:22:42	190.5.203.123	8080	floodlight

Flows on Switches
Tables on Switches
Ports on Switches
Traffic
Traffic on Switch 00:00:00:00:00:00:04:4d from beacon in 190.90.69.93:8080
Traffic on Switch 00:00:00:00:00:00:04:0e from pox in 143.54.12.210:8080
Traffic on Switch 00:00:00:00:00:00:03:e9 from floodlight in 190.5.203.123:8080
Packets
10:05
10:10
10:15
rxPackets
txPackets
rxDrops
txDrops
rxError
txError
rxOverrunError
rxCrcError
collisions
rxFrameError

Fig. 5: MVN on runtime

On runtime, MVN (see Figure 5) allows Network Administrators to tackle the raised NMSit. MVN during its execution presents, simultaneously in an integrated user interface, the information of flows, links, and packect traffic of Open vSwitches, regardless of controllers from network providers. For instance, into MVN, the actions sequence to retrieve packet traffic information of three switches, each one in a different controller, is as follows: ***(1)*** Point the mouse to controllers list $\rightarrow P$, ***(2)*** Mouse press and release to select three controllers $\rightarrow 6B$; ***(3)*** Point the mouse to Switches button $\rightarrow P$, ***(4)*** Mouse press and release the Switches button $\rightarrow 2B$, ***(5)*** Mouse press and release to select three switches $\rightarrow 6B$, ***(6)*** Point the mouse to Traffic button $\rightarrow P$, ***(7)*** Mouse press and release the Traffic button to open the RRDTool images that contain the packet traffic information $\rightarrow 2B$. The estimated time to the above sequence is given by $\mathbf{R_{est} = 7M + 3P + 16B}$. Thus, it is expected that, by using MVN, Network Admistrators take $14.35s$ to deal the raised NMSit, by analyzing, in an integrated user interface, three RRDTool images that present information about packets received, transmitted, dropped, and with error. This result was corroborated by the experimental study in which $\mathbf{R_{exp} = 9.01s < R_{est}}$.

If the Network Administrator does not participate in the Mashment Ecosystem, he/she performs the following actions sequence to retrieve the information about the packet traffic on one switch from a specific controller web tool: ***(1)*** Point the mouse to Switches tab $\rightarrow P$, ***(2)*** Mouse press and release to select the Switches tab $\rightarrow 2B$, ***(3)*** Point the mouse to select a switch $\rightarrow P$, ***(4)*** Mouse press and release to select a switch $\rightarrow 2B$, ***(5)*** Point the mouse to Ports button $\rightarrow P$; and ***(6)***

Mouse press and release to select ports of switch $\rightarrow 2B$. These actions must be repeated three times, one by each controller web tool. Therefore, without MVN the estimated time to retrieve non-integrated information about the packet traffic on three switches is: $\mathbf{R_{3s} = 3(6M + 6B + 3P) = 36s}$. Therefore, $\mathbf{R_{exp} < R_{est} < R_{3s}}$. In this sense, it is important to highlight that the retrieving time for MVN is significantly smaller (a 60% taking into account the estimated time) than for the non-MVN case. According this result, we can state that a Network Administrator in the Mashment Ecosystem can tackle a NMSit faster than one out of it.

V. Conclusions and future work

In this paper, we introduced a mashup ecosystem (Mashment Ecosystem) that allows to tackle network management situations (NMSit). The Mashment Ecosystem and its implementation are based on the high-level abstraction of NMR, WR, and OpR, the composition model of mashups, and an approach centered in the Network Administrator for building up of composite solutions. Our ecosystem empowers the Network Administrator with the important ability to rapidly create, launch, and publish Mashments that are mashups devised to collect, correlate, fuse, and present integrated information about a NMSit. We also presented experimental measured and KLM estimation of time that the Network Administrator take to: (*i*) create, launch, and publish MVN that is a Mashment aimed to address a specific NMSit: transmission errors in a heterogeneous virtual network; and (*ii*) retrieve, by using MVN, the integrated information about the referred NMSit.

The aforementioned NMSit has a particular challenge: it needs the fast development of a solution (MVN) able to retrieve, merge, and rapidly present, in an integrated way, network management information from different OpenFlow controllers and their underlying virtual network elements. The Mashment Ecosystem allowed the Network Administrator to overcome such a challenge, corroborating its significance and the relevance of Mashment concept. Through an experimental and KLM evaluation, we have confirmed, first, the short time that a Network Administrator takes to MVN: create (estimated=$56.25s$, experimental=67.24), launch (estimated=$6.65s$, experimental=$5.46s$), and publish (estimated=$23.60s$, experimental=21.92). Second, the short time that a Network Administrator, that is using MVN, takes to retrieve (estimated=$14.35s$, experimental=$9.01s$) the integrated information about the raised NMSit. The experimental evaluation confirmed KLM predictions and, consequently, the feasibility of using our ecosystem to tackle any NMSit. Furthermore, it is important to highlight that the time to retrieve non-integrated information about this NMSit, by using mistmached solutions, is $36s$. Thus, it is expected that a Network Administrator in the Mashment Ecosystem can tackle a NMSit 60% faster than one out of it.

As future work, we plan to propose and implement a Mashment dynamic composition model in order to tackle the NMSits more rapidly. Furthermore, we are interested in evaluating the productivity of Network Administrators participating in the Mashment Ecosystem. We also plan to implement the Mashment Marketplace to evaluate its feasibility. The acceptance of Mashments by Network Administrators is a topic to explore too.

Acknowledgment

The research of PhD(c) Caicedo is funded by two scholarships one PECPG of the CAPES (Brazil) and another one of the University of Cauca (Colombia).

References

[1] Z. Zhao, S. Bhattarai, J. Liu, and N. Crespi, "Mashup services to daily activities: end-user perspective in designing a consumer mashups," in *iiWAS '11*. New York, NY, USA: ACM, 2011, pp. 222–229.

[2] G. Jakobson, J. Buford, and L. Lewis, "Situation Management: Basic Concepts and Approaches," in *Information Fusion and Geographic Information Systems*, ser. Lecture Notes in Geoinformation and Cartography, W. Cartwright, G. Gartner, L. Meng, M. . Peterson, V. . Popovich, M. Schrenk, and K. . Korolenko, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ch. 2, pp. 18–33.

[3] G. Bader, W. He, A. Anjomshoaa, and A. Tjoa, "Proposing a context-aware enterprise mashup readiness assessment framework," *Information Technology and Management*, vol. 13, pp. 377–387, 2012.

[4] S. Tian, G. Weber, and C. Lutteroth, "A tuplespace event model for mashups," in *OzCHI '11*. New York, NY, USA: ACM, 2011, pp. 281–290.

[5] G. Jakobson, L. Lewis, C. Matheus, M. Kokar, and J. Buford, "Overview of situation management at sima 2005," in *MILCOM '05. IEEE*, 2005, pp. 1630 –1636 Vol. 3.

[6] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: Design, implementation and experience," *Parallel Computing*, vol. 30, p. 2004, 2003.

[7] W. Barth, *Nagios: System and Network Monitoring*, 2nd ed. San Francisco, CA, USA: No Starch Press, 2008.

[8] G. Pavlou, "On the evolution of management approaches, frameworks and protocols: A historical perspective," *J. Netw. Syst. Manage.*, vol. 15, no. 4, pp. 425–445, Dec. 2007.

[9] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, M. Rabbani, Q. Zhang, and M. Zhani, "Data center network virtualization: A survey," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1–20.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[11] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, apr 2010.

[12] E. M. Maximilien, A. Ranabahu, and S. Tai, "Swashup: situational web applications mashups," in *OOPSLA '07*. New York, NY, USA: ACM, 2007, pp. 797–798.

[13] N. Ozkan and W. Abidin, "Investigation of mashups for managers," in *ISCIS 2009*, sept. 2009, pp. 622 –627.

[14] A. Majchrzak and P. H. B. More, "Emergency! web 2.0 to the rescue!" *Commun. ACM*, vol. 54, pp. 125–132, April 2011.

[15] E. Tosti and W. Smari, "Sensors integration in a grid-based architecture for emergency management systems," in *DEST '10*, April, pp. 435–442.

[16] J. Adams and C. Reynolds, "A complex situational management application employing expert systems," in *Systems, Man, and Cybernetics, 2000. IEEE*, vol. 3, 2000, pp. 1959 –1964 vol.3.

[17] R. Koelle and A. Tarter, "Towards a distributed situation management capability for sesar and nextgen," in *ICNS '12*, april 2012, pp. O6–1 –O6–12.

[18] B. Magoutas, G. Mentzas, and D. Apostolou, "Proactive situation management in the future internet: The case of the smart power grid," in *DEXA '11*, 29 2011-sept. 2 2011, pp. 267 –271.

[19] K. Huang, Y. Fan, and W. Tan, "An empirical study of programmable web: A network analysis on a service-mashup system," in *ICWS '12*, june 2012, pp. 552 –559.

[20] R. Latih, A. Patel, A. Zin, T. Yiqi, and S. Muhammad, "Whip: A framework for mashup development with block-based development approach," in *ICEEI '11*, july 2011, pp. 1 –6.

[21] D. Kieras, "Using the keystroke-level model to estimate execution times," in *University of Michigan*, 2001.

2013 IEEE 37th Annual Computer Software and Applications Conference

A Mashup-based Approach for Virtual SDN Management

Oscar Mauricio Caicedo Rendon
Computer Networks Group
Institute of Informatics
University Federal do Rio Grande do Sul
Email: omcrendon@inf.ufrgs.br

Felipe Estrada-Solano
Telematics Engineering Group
Telematics Department
University of Cauca
Email: festradasolano@unicauca.edu.co

Lisandro Zambenedetti Granville
Computer Networks Group
Institute of Informatics
University Federal do Rio Grande do Sul
Email: granville@inf.ufrgs.br

***Abstract*—The Software Defined Networks paradigm aided by the Network Virtualization is a key driver to cope the Internet ossification. There are different proposals to deploy this paradigm, but there is not an integrated or standardized way for the management of networks built with such proposals. In this sense, the network management becomes too complex because multiple solutions must be used by Network Administrators to perform their tasks. In this paper, we introduce a mashup-based approach that allows Network Administrators to customize and combine management solutions, in order to they build composite applications (called SDN Mashups) aiming the integrated management of Virtual Software Defined Networks in heterogeneous environments. We evaluate our approach by building a SDN Mashup for the management of a network slice that uses three distinct Network Operating Systems and by running performance tests, corroborating that the mashup built has small response time.**

***Keywords*-OpenFlow; SDN; SDN Mashup; Virtual SDN; Web-based Management;**

I. Introduction

The Internet constantly evolves to support a lot of new technologies and protocols in Application and Link Layers. However, at the Internet core (Transport and Internet Layers), the evolution has come to a standstill that is known as Internet ossification. The Software Defined Networks (SDN) and the Network Virtualization are key drivers to overcome such ossification [1]. The SDN paradigm proposes to separate data (packet forwarding) and control (decision policies) planes in order to simplify the network operation [2]. The Network Virtualization allows to share a network physical infrastructure among several virtual networks. This type of virtualization may help to deploy the SDN-based networks, because it facilitates the control plane migration from network devices (*e.g.*, routers, switches) to servers and allows to perform network experiments in an isolated way [3]. Hereinafter, we will call a SDN aided by the Network Virtualization as Virtual SDN.

There are different proposals for deploying the SDN paradigm, such as OpenFlow [4] and the Forwarding and Control Element Separation (ForCES) framework [5]. In these proposals common components are: the Network Operating System (NOS) at the control plane and the Network Services running on it. However, there is not an integrated or standardized way for managing these components, which certainly is not suitable for the whole management of SDN-based networks on heterogeneous and virtual environments. For instance, in a future scenario, if a Network Administrator needs to manage several Virtual SDN Slices from different providers, which are using distinct NOS to operate and provide Virtual SDN Resources, the SDN management will become too complex because multiple tools must be used to perform control and monitoring tasks.

Although large research efforts have been made about the SDN deployment [2] [6] [7], few investigations are found in the literature concerning the control and monitoring of non-homogeneous SDNs. In this paper, we take a step further, proposing a novel mashup-based approach that provides a suitable model of composition and abstraction to cope the heterogeneity of virtual resources on SDN. In the approach, we introduce the SDN Mashup concept that lets Network Administrators create SDN Management solutions (called SDN Mashups) to meet their own requirements. The SDN Mashups stimulate Network Administrators to customize and combine, in a high-level abstraction, their SDN management tools, aiming to facilitate the enforcement of management tasks.

In summary, the key contributions presented in this paper are: (*i*) propose a mashup-based approach aimed to manage Virtual SDNs on heterogeneous environments and allow Network Administrators to build up SDN Management solutions, (*ii*) present a SDN Mashup prototype based on the representation of Virtual SDN Resources as Services; and (*iii*) demonstrate a monitoring scenario of a Virtual SDN that uses three different NOS, confirming the small response time of the mashup built.

The remainder of this paper is organized as follows. In Section II, we present both the background and the related work. In Section III, we introduce the SDN Mashup concept. In Section IV, we present the SDN Mashup System. In Section V, we expose and analyze the case study developed to evaluate our approach. The paper concludes in Section VI.

0730-3157/13 $26.00 © 2013 IEEE
DOI 10.1109/COMPSAC.2013.22

II. Background and Related Work

In this section, first, we present a mashups background. Second, we describe the main SDN concepts. Third, we discuss the related work about the SDN management.

A. Mashups

Mashups are Web applications created through the integration of different resources (*e.g.*, data, application logic, and user interfaces) available on the Internet [8]. The Mashup technology has been considered a fundamental piece in Web 2.0 [9], allowing end-users, without advanced programming skills, to create their own and customized applications. Furthermore, mashups encourage both cooperation and reuse among end-users [10].

In accordance to the ProgrammableWeb site, the main mashup service directory, about 60% of current mashups are related to mapping services [11]. The Mashup technology has been also used in many other areas, for instance, helping to overcome an emergency situation [12] by sharing weather and rescue information among civil organizations and government entities. In the telecommunications area, the telco-mashup concept [13] was defined to provide composite services for end-users, by combining features like streaming, Quality of Service, and billing. Mashups, based on the REpresentational State Transfer (REST) architectural model and the semantic Web, were proposed to facilitate the composition of small applications by end-users [14]. In this paper, we introduce the SDN Mashup concept to extend the use areas of mashups and cover the SDN management.

B. Software Defined Networks

The SDN paradigm has emerged as an important trend that defines how future networks are architected. A SDN is formed by three architectural components [2] [6]: the packet forwarding datapath (*e.g.*, switches and routers passing packets), the NOS that controls such datapath through a vendor-independent protocol, and the Network Services (or Network Features) running on the top of NOS. The possibility to add these Network Services, in an easier way, is the key advantage of SDN to facilitate the innovation in the Internet.

The SDN deployment proposals define aforementioned components in a different way. For instance, in the ForCES framework [5], the ForCES protocol is used to communicate Control Elements (*i.e.*, the NOS) and Forwarding Elements (*i.e.*, the datapath). In such framework, Network Services can be developed as distributed features in Control Elements. In an OpenFlow-based SDN [4], a Controller (*i.e.*, the NOS), such as POX [15], Beacon [16], and FloodLight [17], uses the OpenFlow protocol to control OpenFlow-capable network devices (*i.e.*, the datapath). The Controller is also used for deploying new-centralized Network Services (*e.g.*, a new routing protocol) that are known as Network Applications.

C. Management of Software Defined Networks

Although, in previous researches, the problems about the management of heterogeneous SDNs by using high-level tools have not been directly addressed. Below, we review some of the most important OpenFlow management solutions found in the literature.

The Stanford University introduced a graphical tool, called OpenRoads [18], to facilitate the management of IP addresses in OpenFlow networks and to show monitoring information of switches on the datapath. The OpenFlow MaNagement Infrastructure (OMNI) [19] is a solution aimed to control and monitor OpenFlow networks. This solution is based on a multi-agent system that can be accessed by Network Administrators from a Web user interface. The NetOpen [20] uses a Service Oriented Architecture (SOA) to support the creation of Network Services by combinig basic SOA services that are named networking primitives. The NetOpen considered, among others, the following Network Services: to retrieve information of switches, link states, and flow tables, and to configure the network device capabilities.

It is worth noting that the described solutions were not devised to be extended and enhanced by Network Administrators themselves. Such solutions can be solely improved by network programmers in a low-level abstraction. Moreover, up to now, OpenRoad and OMNI were just tested in network slices controlled by NOX that is an OpenFlow-based NOS implemented in the C language. In turn, NetOpen can be considered as a specialization of NOX. Consequently, OpenRoad, OMNI, and NetOpen cannot manage a Virtual SDN that uses more than one type of NOS. Thus, regarding the NOS, these solutions are constrained to homogeneous environments.

III. SDN Mashups

In order to better explain our approach, first, we present the global vision of SDN Mashups. Second, we describe a network management scenario in which is necessary to use such type of mashup.

A. Global Vision

Before defining what is a SDN Mashup, we present the main concepts used in our approach. A Virtual Network Provider (VNP) is a company in charge of operating Virtual SDN Resources and providing them to distinct Virtual Network Operators (VNOs). A VNO is a company responsible for supplying the Virtual SDN Slices requested by customers and/or applications [1]. A Virtual SDN is a subset of the underlying physical network and, usually, can be formed by several Virtual SDN Resources [3]. One or more Virtual SDN form a Virtual SDN Slice.

Every Virtual Network Element (VNE), Network APplication (NAP), and NOS is a Virtual SDN Resource. A VNE is located at the bottom of the SDN architecture. Virtual network devices (*e.g.*, the Vyatta Router and the

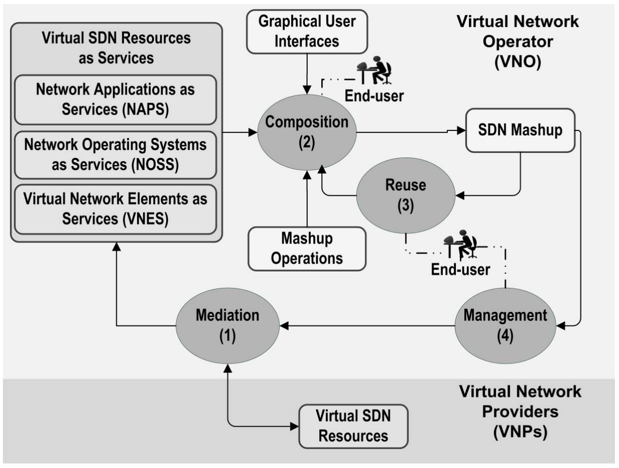
Open vSwitch), virtual nodes and hosts (using, for instance, VMWare, Xen, or VirtualBox), links, and flows are types of VNE. A NAP is a program that handles the control software of VNE, through interfaces and protocols provided by NOS. Rendezvouz services and applications to path selection are examples of NAP. A NOS is in charge of monitoring and handling the resources and the entire state of Virtual SDN.

The SDN Mashups are composite Web applications aimed to manage any SDN that has been deployed using Network Virtualization. In our approach, Network Administrators are able to create SDN Mashups by using wiring and drag-and-drop mechanisms. Thus, Network Administrators do not require intimate knowledge about the Application Program Interfaces (APIs) of NAP, NOS, and VNE, or concerning the data mapping among these APIs. It is important to highlight that an end-user programming approach, as the used by SDN Mashups, provides flexibility for Network Administrators to build their solutions by themselves, and promotes the innovation in SDN management solutions.

A SDN Mashup poses some features that existing mashups do not support. First, it combines information, on the fly, from multiple resources, such as NAP, NOS, and VNE. Second, it hides the heterogeneity and complexity of Virtual SDN Resources in order to facilitate the carrying out of management tasks. Third, it blends local and external visualization APIs to generate integrated Graphical User Interfaces (GUIs). Fourth, it provides access to multiple end-users to enable communication and collaboration among them by sharing and reusing SDN Mashups. It is worth noting that, by the SDN Mashup concept and the aforementioned features, we lead the Network Management towards an end-user centric environment, where millions of Network Administrators are able to participate and collaborate in order to cope their own needs, and even obtain profits.

In a general way, a SDN Mashup is formed by combining Virtual SDN Resources represented as Services, Mashup Operations, and GUIs. The representation of Resources as a service consists on defining and providing a common data-format to interchange information of resources, a well-known interface to each resource, and a common protocol to communicate with every interface. Specifically, we define the following Virtual SDN Resources as a service (see details in the section IV): Network Operating System as a service (NOSS), Network APplication as a service (NAPS), and Virtual Network Element as a service (VNES). The Mashup Operations are structures of composition, such as *Sequential*, *Split*, and *Merge*. A Mashup Operation can be used, for instance, to sort, filter, and aggregate the information retrieved from one or more NOSS. The GUIs represent visualization and presentation libraries used to generate the integrated user interfaces of SDN Mashups.

The Figure 1 presents the global vision of SDN Mashups, in which we mainly propose the creation of SDN management solutions by end-users: *(1)* The mediation process



Virtual SDN Resources as Services
Network Applications as Services (NAPS)
Network Operating Systems as Services (NOSS)
Virtual Network Elements as Services (VNES)
Graphical User Interfaces
End-user
Composition (2)
SDN Mashup
Reuse (3)
Mashup Operations
End-user
Mediation (1)
Management (4)
Virtual Network Operator (VNO)
Virtual Network Providers (VNPs)
Virtual SDN Resources

Figure 1. Global Vision of SDN Mashups

is responsible for offering the Virtual SDN Resources as Services. This mediation is necessary because there is not a standardized interface/protocol to access the data, the application logic, and the user interfaces provided by different types of resources. *(2)* The end-user (*e.g.*, a SDN Administrator: the Network Administrator of a VNO), in the composition process, defines the Mashup Operations that act on NOSS, NAPS, and/or VNES. The results of these Operations are shown by Web 2.0 GUIs, that are also defined and customized by the end-user in the composition process. *(3)* In the reuse process, a SDN Mashup can be used to create another one. Different end-users may use the same or similar candidate resources/services/mashups and glue them to compose a new complex SDN Mashup. *(4)* The end-user, by executing SDN Mashups, is able to manage one or several Virtual SDNs that are formed by Virtual SDN Resources belonging to VNPs. A SDN Mashup carries out their management tasks through the mediation process that is always hidden for the end-user.

B. Motivating Scenario

Management of Virtual SDN. Let's suppose that a Network Administrator, here called SDN Administrator, requires to purchase new Virtual SDN Slices to satisfy the demand of its customers, for instance, Internet Service Providers and small companies. Usually, VNP_a is the choosen option to meet such requirement. However, at this time, the SDN Administrator decides to buy required Slices from VNP_b because of economic profits. As a result, the SDN Administrator will need to control and monitor the Slices provided by VNP_a and VNP_b.

Considering that VNP_a uses a different NOS than VNP_b, the SDN Administrator will have to manage each type of Virtual SDN Slice by using disparate management solutions, such as proprietary command line interfaces to execute specific commands on each NOS or dissimilar Web user interfaces to administrate virtual routers. Instead, if the SDN Administrator uses our approach, he/she will be able to build by him/herself a SDN Mashup devoted to manage the Virtual SDN Slices, in an integrated way. This SDN Mashup will hide the NOS heterogeneity from VNP_a and VNP_b. Thus, the complexity of SDN management tasks carried out by the SDN Administrator will be also mitigated.

IV. SDN Mashup System

Usually generic mashup systems (in the literature, they are also known as mashup makers) provide good basis for developing small composite applications, named mashups. However, these systems do not address, in a native way, special concerns of the SDN management. In particular, the complexity, heterogeneity, and high-level interaction of SDN Resources must be driven to enable the control and monitoring of SDN Slices in virtual environments. Therefore, there is a gap in the mashup-and-SDN related research and, consequently, there is a chance for innovation. In next paragraphs, we describe how a system based on the abstraction and composition models of the mashup technology, called SDN Mashup System, can be targeted to resolve the shortcomings of the SDN management in non-homogeneous and virtual surroundings.

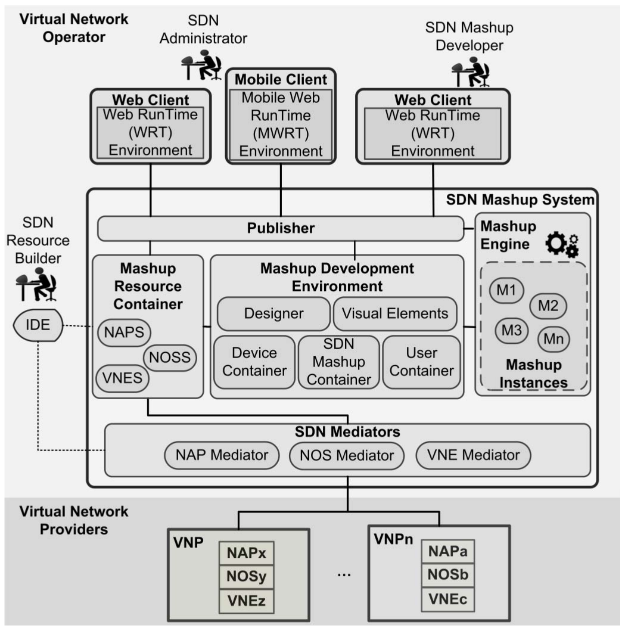


Figure 2. SDN Mashup System

The Figure 2 depicts the SDN Mashup System that enables to carry out the SDN Mashup concept, the System users, and the Virtual SDN Resources (*i.e.*, NAP, NOS, and VNE) to be managed by SDN Mashups. The SDN Mashup System is made up by the SDN Mediators, the Mashup Resource Container, the Mashup Development Environment, the Publisher, and the Mashup Engine. The users that interact with our System by using a Web Client, a Mobile Client, and/or an Integrated Development Environment (IDE) are the SDN Administrator, the SDN Mashup Developer, and the SDN Resource Builder.

The Virtual SDN Resources, provided by VNPs, are heterogeneous. Therefore, in the SDN Mashup System, these resources are accessed and handled through SDN Mediators. A Mediator hides the complexity of one or more resources in two ways: (*i*) accessing and retrieving the information from Virtual SDN Resources, and presenting it to the SDN Mashup System in a standardized data-format (*e.g*, XML and JSON); and (*ii*) providing a two-way communication between the Virtual SDN Resources and the SDN Mashup System via gateways (*e.g*, SNMP/HTTP and Proprietary/HTTP). This communication allows complete interaction among any VNO and its VNPs.

In the SDN Mashup System, SDN Mediators were defined for NAP, NOS, and VNE. A new Mediator must be developed every time a new kind of Virtual SDN Resource arises. In our approach, we propose that Mediators must be developed and extended by the SDN Resource Builder through the use of a conventional IDE. For example, if the NOX is integrated into a VNP, the SDN Resource Builder will be in charge of developing the corresponding Mediator (*e.g.*, NOX Mediator) to adapt such NOS into the SDN Mashup System.

The Mashup Resource Container stores services that represent the Virtual SDN Resources in the SDN Mashup System. We define three types of services: (*i*) NAPS that offers the functionalities provided by Network Applications (*e.g.*, a Video Multicasting solution) running on the top of a specific NOS, (*ii*) NOSS, in turn, provides the management facilities (*e.g.*, slice topology discovery) supplied by a determined NOS; and (*iii*) VNES that offers the information about one or a set of virtual network elements, for instance, quantity of sent/lost packets in a Vyatta virtual router. The communication between the Mashup Resource Container and every SDN Mediator is made via a standardized protocol (*e.g.*, HTTP and SOAP). NAPS, NOSS, and VNES interact with corresponding Virtual SDN Resources through SDN Mediators. Similarly to Mediators, services in the Resource Container must be implemented by the SDN Resource Builder.

In very general terms, SDN Administrators and SDN Mashup Developers use the Mashup Development Environment to compose and execute SDN Mashups (SDN management solutions). The Mashup Development Environment provides flexibility to the SDN Mashup System through a high-level abstraction of Virtual SDN Resources, GUIs, and Mashup Operations used in the composition process. In this sense, it is important to point out that we propose a Mashup Development Environment in which, during the building of SDN Mashups, it is not necessary to work with data mapping. The data mapping is one of the least intuitive tasks in current mashup makers because non-programmers (as the SDN Administrators) are usually not able to specify it correctly.

The Mashup Development Environment is formed by the Visual Elements, the Designer, the SDN Mashup Container, the Device Container, and the User Container. The Visual Elements are graphical representations of the Mashup Operations, the services stored into the Mashup Resource Container, and the SDN Mashups. In addition to SDN Mashups, we define four types of Visual Elements: Visual_NAP, Visual_NOS, Visual_VNE, and Visual_MashupOperation. An instance of Visual_NAP is a box symbolizing a new tunneling algorithm to be executed on the top of a NOS. An example of Visual_NOS is a box representing a particular NOS as Beacon, NOX, FloodLight, or POX. A type of Visual_VNE is a box symbolizing a virtual switch as the Open vSwitch. A visual filter to be applied to the information collected from NOSS invocations is an example of Visual_MashupOperation.

The Designer is an user interface based on drag-and-drop and wiring mechanisms. Using these mechanisms, SDN Mashup Developers and SDN Administrators can blend, in an easy way, different Visual Elements to create SDN Mashups. By considering, the Visual Elements, the Mediators, and the Designer, the Mashup Development Environment becomes technology-agnostic. Here, technology-agnostic means that the Mashup Development Environment allows to combine Resources/Services regardless the corresponding underlying protocols (*e.g.*, OpenFlow/ForCES), controller libraries (*e.g.*, Beacon/POX API), and so on.

In the Designer, the SDN Mashups can be used to develop new and complex ones, which promotes: (*i*) the reuse of SDN Mashups, (*ii*) the extension and improvement of SDN Mashups and the Mashup Development Environment; and *(iii)* the fast development of SDN Mashups. In brief, the SDN Administrator and the SDN Mashup Developer can use the Designer to extend and enhance their SDN Management solutions and the own Designer. Likewise, the SDN Resource Builder can also improve the Mashup Development Environment (including the Designer) by adding new Visual Elements using an IDE.

The SDN Mashup Container stores the metadata of all SDN Mashups built in the Mashup Development Environment. This metadata is used on design time to present each SDN Mashup as a Visual Element. Thus, the SDN Mashup Container is also a key module that enables the reuse in our approach. On runtime the metadata is read to execute every SDN Mashup. The User Container stores the user profiles metadata, that is used to control the access to SDN Mashups. The Device Container hosts the information related to device capabilities. This information is processed to identify what type of Client device is able to run the SDN Mashups and the Mashup Development Environment.

The Publisher module is responsible for adapting the GUI of each SDN Mashup to different Client devices (*i.e.*, the Web Client and the Mobile Client). Moreover, this module controls the access to available elements in the containers of the SDN Mashup System. After that a SDN Mashup is launched, for example from the Mashup Development Environment, the Mashup Engine acts as the SDN Mashup life cycle manager in charge of creating, deleting, and caching Mashups Instances. As a result, this Engine interacts with all modules of the SDN Mashup System.

The Web Client and the Mobile Client are software entities in charge of running and showing SDN Mashups, anywhere and anytime. The former uses a Web RunTime environment and the latter a Mobile Web RunTime environment to execute client-side mashup functionalities. The SDN Mashups can be executed on both types of Clients. Therefore, browsers, running on personal computers, notebooks, and smartphones, are enabled to be used as front-end of SDN management solutions based on mashups. The Mashup Development Environment only can be executed on Web Clients, which means that SDN Mashups are programmed on Web and not on Mobile environments. Since SDN Mashups and the Mashup Development Environment are Web 2.0 solutions, Client devices must support Javascript, Asynchronous Javascript And XML (AJAX), Cascading Style Sheets (CSS), and HyperText Markup Language (HTML) version 5, among other Web 2.0 technologies.

Regarding the users of the SDN Mashup System, we consider: first, the SDN Resource Builder is an information technology developer responsible for programming and providing SDN Resources as Services, Visual Elements, and Mediators. The Resource Builder interacts with our Mashup System by means of a conventional IDE. Second, the SDN Mashup Developer is in charge of combining Visual Elements and even existing SDN Mashups in order to develop new mashups. The Mashup Developer interacts with our Mashup System via the Mashup Development Environment running on a Web Client. Third, the SDN Administrator is responsible for the management of virtual and heterogeneous SDNs through mashups running on the Web Client and/or the Mobile Client. Using the Mashup Development Environment, the SDN Administrator is also able to perform two actions: (*i*) create, customize, and improve composite applications addressed to manage Virtual SDNs; and (*ii*) as a result of developing mashups, extend, and enhance his/her workspace.

V. Case Study

In order to evaluate our approach, first, we created an infrastructure of OpenFlow-based Virtual SDNs. Second, we developed a SDN Mashup for monitoring such infrastructure. Third, we conducted experiments to measure the response time of the SDN Mashup built.

A. OpenFlow Virtual SDN

The Figure 3 presents the test environment of our case study. VNP_a has an OpenFlow-based network where virtual switches (Open vSwitches), links and flows are monitored via Beacon version 1.0.2. The Beacon is an OpenFlow Controller developed in the Java language. VNP_b has an OpenFlow-based network that is monitored by POX version 1.0.0. The POX is a Controller implemented in the Phyton language. VNP_c has an OpenFlow-based network that is monitored by FloodLight version 0.9.0. The FloodLight is a Controller developed in the Java language. All OpenFlow-based Virtual SDNs were deployed on Mininet version 1.0.0 [6] that, in turn, was executed on Oracle VM VirtualBox version 4.2.6. The Mininet is a software for emulating OpenFlow networks. Here, we use OpenFlow because of its commercial and research significance [4].

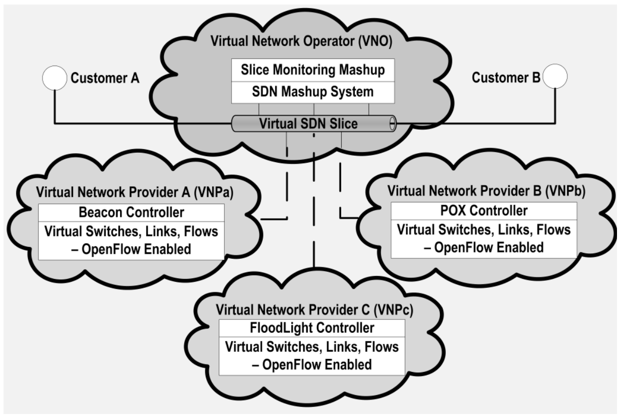

Virtual Network Operator (VNO)
Slice Monitoring Mashup
SDN Mashup System
Virtual SDN Slice
Customer A
Customer B
Virtual Network Provider A (VNPa)
Beacon Controller
Virtual Switches, Links, Flows – OpenFlow Enabled
Virtual Network Provider B (VNPb)
POX Controller
Virtual Switches, Links, Flows – OpenFlow Enabled
Virtual Network Provider C (VNPc)
FloodLight Controller
Virtual Switches, Links, Flows – OpenFlow Enabled

Figure 3. Test Environment

VNO provides network services to Customers A and B by means a Virtual SDN Slice made up of OpenFlow Controllers, virtual switches, links and flows from VNP_a, VNP_b, and VNP_c (see Figure 3). In this sense, the SDN Administrator of VNO requires to monitor Virtual SDN Resources, in an integrated way, regardless of controllers, network topologies, and implementation technologies. The previous requirement is met by the *Slice Monitoring Mashup* that is an instantiation of our SDN Mashup concept.

B. Slice Monitoring Mashup

The Slice Monitoring Mashup was composed in the Designer of the SDN Mashup System by connecting the VisualBeacon (*i.e.*, a Visual_NOS), the VisualFloodLight, and the VisualPOX to the MonitorSDN (*i.e.*, a Visual_MashupOperation). The VisualBeacon, the VisualPOX, and the VisualFloodLight are boxes built to represent, respectively, Beacon, POX, and FloodLight. In turn, the MonitorSDN is a visual box created to encapsulate the next monitoring operations: `SwitchesList`, `LinksList`, and `FlowsList`. These operations are applied to the VisualBeacon, the VisualPOX, and/or the VisualFloodLight to retrieve the list of virtual switches, links, and flows. The Slice Monitoring Information results from executing one or more of the above mentioned operations.

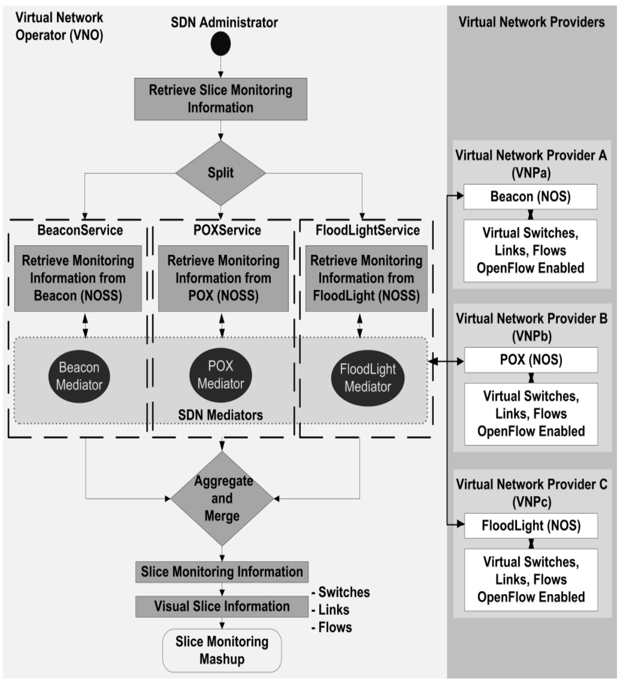
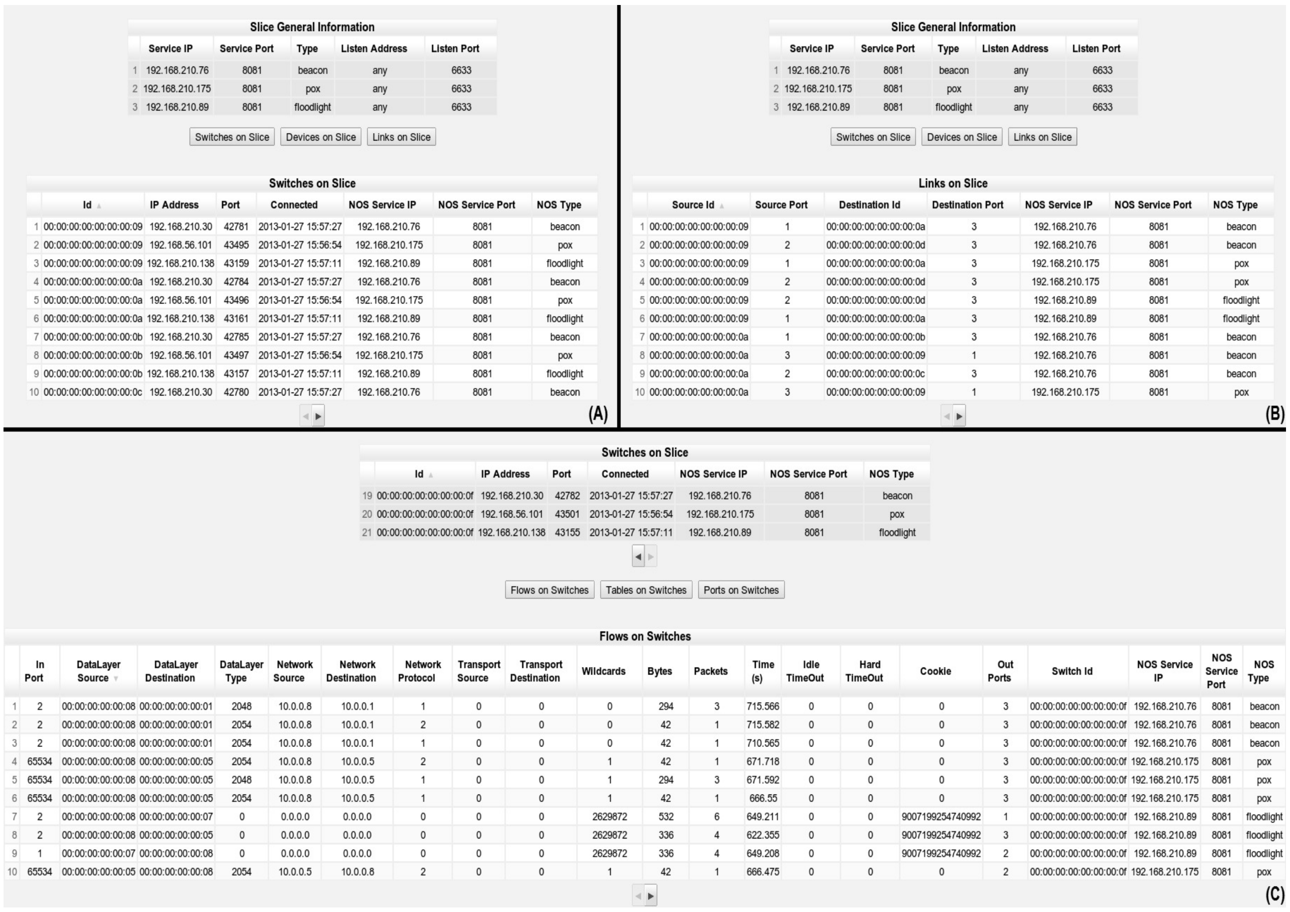

Virtual Network Operator (VNO)
SDN Administrator
Virtual Network Providers
Retrieve Slice Monitoring Information
Split
BeaconService
POXService
FloodLightService
Retrieve Monitoring Information from Beacon (NOSS)
Retrieve Monitoring Information from POX (NOSS)
Retrieve Monitoring Information from FloodLight (NOSS)
Beacon Mediator
POX Mediator
FloodLight Mediator
SDN Mediators
Aggregate and Merge
Slice Monitoring Information
Visual Slice Information
- Switches
- Links
- Flows
Slice Monitoring Mashup
Virtual Network Provider A (VNPa)
Beacon (NOS)
Virtual Switches, Links, Flows OpenFlow Enabled
Virtual Network Provider B (VNPb)
POX (NOS)
Virtual Switches, Links, Flows OpenFlow Enabled
Virtual Network Provider C (VNPc)
FloodLight (NOS)
Virtual Switches, Links, Flows OpenFlow Enabled

Figure 4. Internal Operation of Slice Monitoring Mashup

In a general way, the *Slice Monitoring Mashup* is in charge of splitting, aggregating, and merging the information collected from different Virtual SDN Resources in VNP_a, VNP_b, and VNP_c. The Figure 4 depicts the internal operation of the mashup developed to the raised case study: (*i*) in the Mashup Development Environment running on a Firefox Web Client, the SDN Administrator sends a request to execute the *Slice Monitoring Mashup*, (*ii*) this request is *Splitted* in three solicitations, the first solicitation targeted to the BeaconService, the second to the POXService, and the third to the FloodLightService, (*iii*) each Service invokes NOSS operations (`SwitchesList`, `LinksList`, and `FlowsList`) to collect information from a particular Controller, (*iv*) each Service carries out the corresponding mediation process to interact with its specific OpenFlow-based Virtual SDN, (*v*) the information retrieved by FloodLight/POX/Beacon Services is *Aggregated* and *Merged* to generate the Slice Monitoring Information; and (*vi*) finally, such information is shown, in an integrated way, in the Web GUI (see Figure 5) of the *Slice Monitoring Mashup*.

As result of composing and executing the *Slice Monitoring Mashup*, the SDN Administrator is able to observe, in an unique GUI, the detailed information of resources



(A)

Slice General Information

	Service IP	Service Port	Type	Listen Address	Listen Port
1	192.168.210.76	8081	beacon	any	6633
2	192.168.210.175	8081	pox	any	6633
3	192.168.210.89	8081	floodlight	any	6633

Switches on Slice

	Id	IP Address	Port	Connected	NOS Service IP	NOS Service Port	NOS Type
1	00:00:00:00:00:00:00:09	192.168.210.30	42781	2013-01-27 15:57:27	192.168.210.76	8081	beacon
2	00:00:00:00:00:00:00:09	192.168.56.101	43495	2013-01-27 15:56:54	192.168.210.175	8081	pox
3	00:00:00:00:00:00:00:09	192.168.210.138	43159	2013-01-27 15:57:11	192.168.210.89	8081	floodlight
4	00:00:00:00:00:00:00:0a	192.168.210.30	42784	2013-01-27 15:57:27	192.168.210.76	8081	beacon
5	00:00:00:00:00:00:00:0a	192.168.56.101	43496	2013-01-27 15:56:54	192.168.210.175	8081	pox
6	00:00:00:00:00:00:00:0a	192.168.210.138	43161	2013-01-27 15:57:11	192.168.210.89	8081	floodlight
7	00:00:00:00:00:00:00:0b	192.168.210.30	42785	2013-01-27 15:57:27	192.168.210.76	8081	beacon
8	00:00:00:00:00:00:00:0b	192.168.56.101	43497	2013-01-27 15:56:54	192.168.210.175	8081	pox
9	00:00:00:00:00:00:00:0b	192.168.210.138	43157	2013-01-27 15:57:11	192.168.210.89	8081	floodlight
10	00:00:00:00:00:00:00:0c	192.168.210.30	42780	2013-01-27 15:57:27	192.168.210.76	8081	beacon

(B)

Slice General Information

	Service IP	Service Port	Type	Listen Address	Listen Port
1	192.168.210.76	8081	beacon	any	6633
2	192.168.210.175	8081	pox	any	6633
3	192.168.210.89	8081	floodlight	any	6633

Links on Slice

	Source Id	Source Port	Destination Id	Destination Port	NOS Service IP	NOS Service Port	NOS Type
1	00:00:00:00:00:00:00:09	1	00:00:00:00:00:00:00:0a	3	192.168.210.76	8081	beacon
2	00:00:00:00:00:00:00:09	2	00:00:00:00:00:00:00:0d	3	192.168.210.76	8081	beacon
3	00:00:00:00:00:00:00:09	1	00:00:00:00:00:00:00:0a	3	192.168.210.175	8081	pox
4	00:00:00:00:00:00:00:09	2	00:00:00:00:00:00:00:0d	3	192.168.210.175	8081	pox
5	00:00:00:00:00:00:00:09	2	00:00:00:00:00:00:00:0d	3	192.168.210.89	8081	floodlight
6	00:00:00:00:00:00:00:09	1	00:00:00:00:00:00:00:0a	3	192.168.210.89	8081	floodlight
7	00:00:00:00:00:00:00:0a	1	00:00:00:00:00:00:00:0b	3	192.168.210.76	8081	beacon
8	00:00:00:00:00:00:00:0a	3	00:00:00:00:00:00:00:09	1	192.168.210.76	8081	beacon
9	00:00:00:00:00:00:00:0a	2	00:00:00:00:00:00:00:0c	3	192.168.210.76	8081	beacon
10	00:00:00:00:00:00:00:0a	3	00:00:00:00:00:00:00:09	1	192.168.210.175	8081	pox

(C)

Switches on Slice

	Id	IP Address	Port	Connected	NOS Service IP	NOS Service Port	NOS Type
19	00:00:00:00:00:00:00:0f	192.168.210.30	42782	2013-01-27 15:57:27	192.168.210.76	8081	beacon
20	00:00:00:00:00:00:00:0f	192.168.56.101	43501	2013-01-27 15:56:54	192.168.210.175	8081	pox
21	00:00:00:00:00:00:00:0f	192.168.210.138	43155	2013-01-27 15:57:11	192.168.210.89	8081	floodlight

Flows on Switches

	In Port	DataLayer Source	DataLayer Destination	DataLayer Type	Network Source	Network Destination	Network Protocol	Transport Source	Transport Destination	Wildcards	Bytes	Packets	Time (s)	Idle TimeOut	Hard TimeOut	Cookie	Out Ports	Switch Id	NOS Service IP	NOS Service Port	NOS Type
1	2	00:00:00:00:00:08	00:00:00:00:00:01	2048	10.0.0.8	10.0.0.1	1	0	0	0	294	3	715.566	0	0	0	3	00:00:00:00:00:00:00:0f	192.168.210.76	8081	beacon
2	2	00:00:00:00:00:08	00:00:00:00:00:01	2054	10.0.0.8	10.0.0.1	2	0	0	0	42	1	715.582	0	0	0	3	00:00:00:00:00:00:00:0f	192.168.210.76	8081	beacon
3	2	00:00:00:00:00:08	00:00:00:00:00:01	2054	10.0.0.8	10.0.0.1	1	0	0	0	42	1	710.565	0	0	0	3	00:00:00:00:00:00:00:0f	192.168.210.76	8081	beacon
4	65534	00:00:00:00:00:08	00:00:00:00:00:05	2054	10.0.0.8	10.0.0.5	2	0	0	1	42	1	671.718	0	0	0	3	00:00:00:00:00:00:00:0f	192.168.210.175	8081	pox
5	65534	00:00:00:00:00:08	00:00:00:00:00:05	2048	10.0.0.8	10.0.0.5	1	0	0	1	294	3	671.592	0	0	0	3	00:00:00:00:00:00:00:0f	192.168.210.175	8081	pox
6	65534	00:00:00:00:00:08	00:00:00:00:00:05	2054	10.0.0.8	10.0.0.5	1	0	0	1	42	1	666.55	0	0	0	3	00:00:00:00:00:00:00:0f	192.168.210.175	8081	pox
7	2	00:00:00:00:00:08	00:00:00:00:00:07	0	0.0.0.0	0.0.0.0	0	0	0	2629872	532	6	649.211	0	0	9007199254740992	1	00:00:00:00:00:00:00:0f	192.168.210.89	8081	floodlight
8	2	00:00:00:00:00:08	00:00:00:00:00:05	0	0.0.0.0	0.0.0.0	0	0	0	2629872	336	4	622.355	0	0	9007199254740992	3	00:00:00:00:00:00:00:0f	192.168.210.89	8081	floodlight
9	1	00:00:00:00:00:07	00:00:00:00:00:08	0	0.0.0.0	0.0.0.0	0	0	0	2629872	336	4	649.208	0	0	9007199254740992	2	00:00:00:00:00:00:00:0f	192.168.210.89	8081	floodlight
10	65534	00:00:00:00:00:05	00:00:00:00:00:08	2054	10.0.0.5	10.0.0.8	2	0	0	1	42	1	666.475	0	0	0	2	00:00:00:00:00:00:00:0f	192.168.210.175	8081	pox

Figure 5. Slice Monitoring Mashup

forming the Virtual SDN Slice. For instance, (*i*) the Figure 5 (A) depicts the information about several virtual switches, monitored by either Beacon, POX, or FloodLight, (*ii*) the Figure 5 (B) presents details of links located in three OpenFlow-based SDNs; (*iii*) and the Figure 5 (C) illustrates flows in three virtual switches, each one monitored by a different OpenFlow Controller.

C. Implementation Highlights

Regarding the communication details, it is important to highlight that, first, the interaction between each Controller (Beacon, POX, and FloodLight) and its corresponding virtual switches (Open vSwitches) is made by the OpenFlow protocol. Second, the interaction between mediation processes and Controllers is based on Remote Procedure Calls (Beacon and POX do not expose their monitoring operations through interfaces of services) and HTTP (FloodLight exposes its monitoring operations as Web services). In this sense, the interactions Beacon-Controller/BeaconService, POXController/POXService, and FloodLightController/FloodLightService were implemented by using the Java Remote Method Invocation API, the PYthon Remote Objects Library, and the FloodLight REST API, respectively.

The POXService, the BeaconService, and the FloodLight-Service are based on the REST architectural model [21]. We have used REST because it is becoming in the de-facto model for developing mashups. Furthermore, REST-based solutions are suitable for heterogeneous environments (as in our case study) because their HTTP interaction is independent of programming languages. Specifically, the POXService was developed by using the Phyton Flask API. In turn, the BeaconService and the FloodLightService were implemented by using the Java Jersey API.

The GUIs of the *Slice Monitoring Mashup* and the Mashup Development Environment were built using the Yahoo User Interface (YUI) API, that provides a lot of high-level widgets, and the Google Chart API, that allows to create advanced graphics for websites. It is to point out that both APIs are based on AJAX. AJAX granted us two aspects: dynamic and interactive SDN Mashups, and asynchronous interaction of GUIs with POX/Beacon/FloodLightServices.

D. Evaluation and Analysis

The test environment (see Figure 3) was deployed on servers and personal computers running the Linux Ubuntu O.S. 11.10 (64 bits). The SDN Mashup System was executed on a server with 4 GBytes RAM and 2.53 GHz core 2 duo processor. Each virtual OpenFlow-based network, in a tree or linear topology, was deployed on a server with 8 GBytes RAM and 3.4 GHz core i7 processor. The *Slice Monitoring Mashup* was ran on a personal computer with 2 GBytes RAM and 2.53 GHz core 2 duo processor.

To test our approach, we evaluate the operations (`SwitchesList`, `LinksList`, and `FlowsList`) of the *Slice Monitoring Mashup* when it is used to monitor, in an integrated way, the Virtual SDN Slice composed by Virtual SDN Resources in VNP_a, VNP_b, and VNP_c. In all evaluation cases, we took 30 measurements with a 95% confidence level for the average response time in milliseconds.

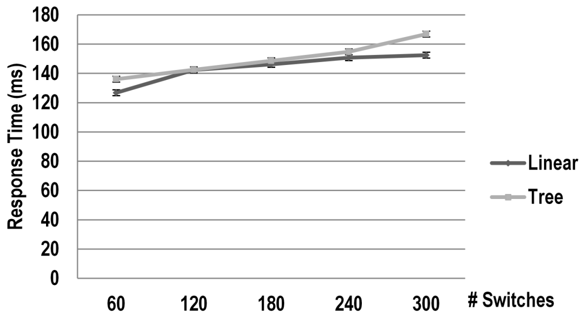


Figure 6. Slice Monitoring Mashup - SwitchesList

The Figure 6 depicts the response time results of the `SwitchesList` operation when the number of switches was increased from 20 to 100 in each Virtual SDN of VNP_a, VNP_b, and VNP_c. Thus, pertest, the total number of switches was 60, 120, 180, 240, and 300. Since the response time (r in milliseconds - ms) of Web systems can be ranked as optimal ($r \leq 100$), good ($100 < r \leq 1000$), admissible ($1000 < r \leq 10000$), and deficient ($r > 10000$) [22], we can state that the *Slice Monitoring Mashup* has a good r when executing the `SwitchesList` operation. Moreover, r has a similar behavior in tested topologies and its growth is less than 1 ms per switch.

The Figure 7 presents the response time results of the `LinksList` operation when the number of links was increased from 50 to 250 in each Virtual SDN of our case study. Thus, pertest, the total number of links was 150, 300, 450, 600, and 750. According to obtained results, we can assert that the `LinksList` operation of the *Slice Monitoring Mashup* has a good r that grows negligibly (less than 1 ms per link) when the number of links is raised in linear and tree topologies.

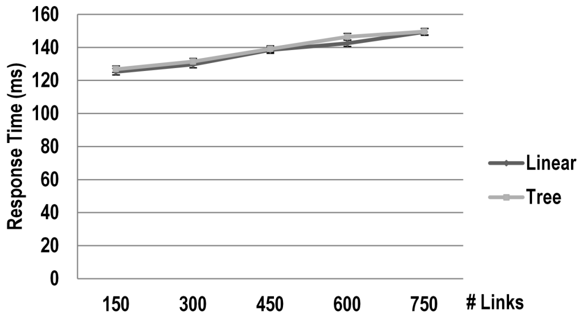


Figure 7. Slice Monitoring Mashup - LinksList

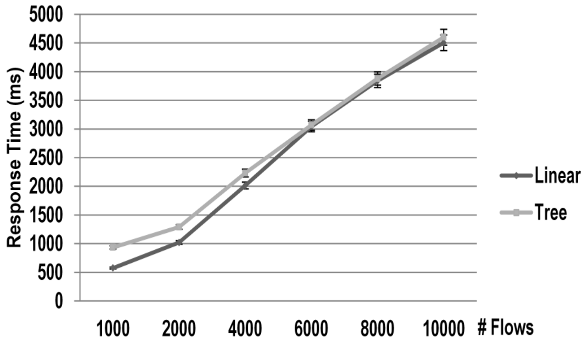


Figure 8. Slice Monitoring Mashup - FLowsList

The Figure 8 presents the response time results of the `FlowsList` operation when the number of flows is increased from 1000 to 10000 in the Virtual SDN Slice. For this operation, the *Slice Monitoring Mashup* has an admissible r that grows less than 3 ms per flow, regardless network topology. As in a network the number of flows may be large, in practice, `FlowsList` retrieves 1000 flows per block, getting so a good r. Such constraint is not relevant because the use of an unique GUI to display all flows is not a good practice of usability. Furthermore, using a mechanism of pagination, flows can be suitably retrieved and displayed to the SDN Administrator.

Summarizing, although the *Slice Monitoring Mashup* uses 11 additional software modules to integrate the monitoring information of the Virtual SDN Slice, the response time of all mashup operations is good on heterogeneous environments. In this way, we can state that the *Slice Monitoring Mashup* can be used to monitor a Virtual SDN Slice regardless of NOS, the network topology, and the number of virtual switches, links, and flows. The NOS heterogeneity is hidden by NOSS and SDN Mediators. The topologies and the number of virtual resources are handled by own centralized-nature of each NOS.

On the other hand, since the SDN Mashup System is based on visual mechanisms as dragging-and-dropping and wiring, we can state that the SDN Administrator and the SDN Mashup Developer do not need programming skills to develop similar SDN Mashups to the *Slice Monitoring Mashup*. In the mashup composition process, the Administrator and the Developer only need to link Visual Elements and provide configuration information, such as IP address and type of NOS. Thus, if a SDN Administrator or a SDN Mashup Developer have to build SDN Mashups, he/she does not need to worry about the data mapping between Visual Elements.

VI. Conclusions and Future Work

In this paper, we introduced a mashup-based approach formed by the SDN Mashup concept and the SDN Mashup System that allows to carry it out. The concept and its instantiation are based on the abstraction and representation of any SDN Resource as a Service, and on the end-user centric development of composite applications. Thus, our approach empowers the SDN Administrator with the important ability to build, extend, and customize SDN management systems. We also presented a realistic Virtual SDN management scenario where multiple information sources and services are aggregated/merged to create a new application, namely the *Slice Monitoring Mashup* that is a SDN Mashup aimed to meet a specific purpose: integrated monitoring of a Slice formed by Virtual SDNs that use different NOS.

The aforementioned scenario has a particular challenge: the monitoring of heterogeneous Virtual SDNs. Our approach was able to overcome such challenge, corroborating the significance of the SDN Mashup concept and the SDN Mashup System. In this sense, through a quantitative evaluation, we have confirmed, first, a good response time ($r \leq 1000$) of the *Slice Monitoring Mashup*, regardless of network topologies and Virtual SDN Resources (NOS, virtual switches, links, flows). Second, the negligible growth of this response time as the number of Virtual SDN Resources increases. The evaluation of the *Slice Monitoring Mashup* confirms the feasibility of using our approach to cope the complexity and heterogeneity of the Virtual SDN management.

From a qualitative point of view, the use of Visual Elements, drag-and-drop, and wiring facilities, provides an easy-to-use Mashup Development Environment with little compromise on usability, particularly during the SDN Mashup composition process. The Visual Elements and the Mashup Designer allow to create and reuse SDN Mashups to manage, in an integrated manner, Virtual SDNs. Additionally, the SDN Mashup System, as a whole, hides implementation details about types of NOS, network topologies, virtual switches, flows, and links. Thus, our approach overcomes the stiffness of current SDN management solutions. In this sense, we consider SDN Mashups a step forward, in both the mashup technology and the network management area. We lead the former towards a new application domain and the latter to an environment centric in the Network Administrator.

As future researches, we plan to extend the SDN Mashup System, adding new features to perform other management tasks and appending more powerful GUIs to automatically compose SDN Mashups. Also, we are interested on evaluating the decrease on the carrying out time of SDN management tasks by using our mashup-based approach. The acceptance by Network Administrators of mashups as network management solutions is a topic that we are going to explore too.

Acknowledgment

The research of PhD(c) Caicedo is supported in part by a scholarship PECPG (Agreement Program Students Graduate) of the CAPES (Brazil). The University of Cauca (Colombia) also funds the research of PhD(c) Caicedo.

References

[1] N. Chowdhury and R. Boutaba, “Network Virtualization: State of the Art and Research Challenges,” *Communications Magazine, IEEE*, vol. 47, no. 7, pp. 20–26, july 2009.

[2] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “NOX: Towards an Operating System for Networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008. [Online]. Available: http://doi.acm.org/10.1145/1384609.1384625

[3] A. Khan, A. Zugenmaier, D. Jurca, and W. Kellerer, “Network virtualization: a Hypervisor for the Internet?” *Communications Magazine, IEEE*, vol. 50, no. 1, pp. 136–143, january 2012.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, march 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[5] A. Doria, J. Hadi Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern, “Forwarding and Control Element Separation (ForCES) Protocol Specification,” RFC 5810, march 2010. [Online]. Available: http://datatracker.ietf.org/doc/rfc5810/

[6] B. Lantz, B. Heller, and 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*. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466

[7] A. Tootoonchian and Y. Ganjali, “HyperFlow: a Distributed Control Plane for OpenFlow,” in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, ser. INM/WREN’10. Berkeley, CA, USA: USENIX Association, 2010, pp. 3–3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863133.1863136

[8] D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, and A. Singh, “Damia: Data Mashups for Intranet Applications,” in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2008, pp. 1171–1182. [Online]. Available: http://doi.acm.org/10.1145/1376616.1376734

[9] C. Cappiello, F. Daniel, M. Matera, and C. Pautasso, “Information Quality in Mashups,” *Internet Computing, IEEE*, vol. 14, no. 4, pp. 14–22, july-august 2010.

[10] J. Yu, B. Benatallah, F. Casati, and F. Daniel, “Understanding Mashup Development,” *Internet Computing, IEEE*, vol. 12, no. 5, pp. 44–52, september-october 2008.

[11] J. J. Jung, “Collaborative browsing system based on semantic mashup with open apis,” *Expert Syst. Appl.*, vol. 39, no. 8, pp. 6897–6902, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2012.01.006

[12] A. Majchrzak and P. H. B. More, “Emergency! Web 2.0 to the Rescue!” *Commun. ACM*, vol. 54, pp. 125–132, April 2011. [Online]. Available: http://doi.acm.org/10.1145/1924421.1924449

[13] H. Gebhardt, M. Gaedke, F. Daniel, S. Soi, F. Casati, C. Iglesias, and S. Wilson, “From Mashups to Telco Mashups: A Survey,” *Internet Computing, IEEE*, vol. 16, no. 3, pp. 70 –76, may-june 2012.

[14] A. P. Sheth, K. Gomadam, and J. Lathem, “SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups,” *IEEE Internet Computing*, vol. 11, pp. 91–94, 2007.

[15] P. Community. (2012) POX Home. [Accessed july 20, 2012]. [Online]. Available: https://github.com/noxrepo/pox

[16] D. Erickson. (2012) Beacon Home. [Accessed july 20, 2012]. [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home

[17] F. Community. (2011) Floodlight Home. [Accessed july 20, 2012]. [Online]. Available: http://floodlight.openflowhub.org/

[18] K.-K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, and N. McKeown, “The Stanford OpenRoads Deployment,” in *Proceedings of the 4th ACM international workshop on Experimental evaluation and characterization*. New York, NY, USA: ACM, 2009, pp. 59–66. [Online]. Available: http://doi.acm.org/10.1145/1614293.1614304

[19] D. Mattos, N. Fernandes, V. da Costa, L. Cardoso, M. Campista, L. Costa, and O. Duarte, “OMNI: OpenFlow MaNagement Infrastructure,” in *Network of the Future (NOF), 2011 International Conference on the*, november 2011, pp. 52 –56.

[20] N. Kim and J. Kim, “Building NetOpen Networking Services over OpenFlow-based Programmable Networks,” in *Information Networking (ICOIN), International Conference on*, jan. 2011, pp. 525 –529.

[21] R. T. Fielding and R. N. Taylor, “Principled Design of the Modern Web Architecture,” *ACM Transactions on Internet Technology*, vol. 2, no. 2, pp. 115–150, may 2002. [Online]. Available: http://doi.acm.org/10.1145/514183.514185

[22] S. Joines, R. Willenborg, and K. Hygh, *Performance Analysis for Java Websites*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

ProViNet: Uma Plataforma para Gerenciamento de Redes Virtuais Programáveis

Wanderson Paim de Jesus[1], **Ricardo Luis dos Santos**[1], **Oscar Maurício Caicedo Rendón**[1]
Lisandro Zambenedetti Granville[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{wpjesus, rlsantos, omrendom, granville}@inf.ufrgs.br

Abstract. *With the evolvement of virtualization and network programming techniques, high-level applications can be used to define the behavior of network traffic while keeping isolation. However, to ensure a harmonious relationship between users, network applications and virtual networks, considerable management efforts are needed. In this paper we propose the ProViNet platform, a solution for managing the deployment of network applications in Programmable Virtual Networks (PVN). In addition to the management facilities, ProViNet contributes with an architecture that allows sharing the control plane of PVN in a scalable way. During the development of this work we identified the need for a standard representation of programmable virtual infrastructures, so it is also proposed an extension of the programmable virtual networks description language VXDL. In order to verify the feasibility of the proposed platform, we implemented a prototype, which is analyzed and evaluated in this paper.*

Resumo. *Ao passo que evoluem as técnicas de virtualização e programação de redes, aplicativos de alto nível podem ser utilizados para definir o comportamento de tráfegos de rede isoladamente. Entretanto, garantir um relacionamento harmônico entre usuários, aplicativos de rede e redes virtuais exige grandes esforços de gerenciamento. Neste trabalho propomos a plataforma ProViNet, uma solução para o gerenciamento da implantação de aplicativos de rede em Redes Virtuais Programáveis (RVP). Além de agregar facilidades no gerenciamento, ProViNet contribui com uma arquitetura que permite o compartilhamento do plano de controle de RVP de forma escalável. Durante o desenvolvimento do trabalho foi identificada a necessidade de uma representação padrão da infraestrutura virtual programável, para tanto propõe-se também, uma extensão da linguagem de definição de redes virtuais VXDL. A fim de verificar a viabilidade da plataforma proposta, foi implementado um protótipo, o qual é analisado e avaliado neste trabalho.*

1. Introdução

Historicamente, o núcleo das redes de computadores, quando comparado com os servidores, *desktops* e dispositivos móveis da borda das redes, é um ambiente hostil à inovação. No contexto específico da Internet, esse fato é geralmente referenciado como ossificação [Hausheer *et al.* 2011]. Para exemplificar, soluções propostas a mais de dez anos, como IPv6 e IPSec, ainda não estão amplamente em uso. São apontadas como possíveis causas: *(i)* a necessidade de modificações globais, ocasionalmente exigindo a substituição de equipamentos; *(ii)* a lentidão no processo de padronização que trata da interoperabilidade com serviços legados; *(iii)* e a abordagem adotada pelas fabricantes de equipamentos, de implementar e implantar soluções baseadas em seu retorno financeiro.

Na intersecção entre o conceito de Virtualização de Redes [Chowdhury e Boutaba 2010] e o de Programabilidade de Redes [Kanaumi *et al.* 2010], emergem as Redes Virtuais Programáveis (RVP), as quais promissoramente prometem reverter o cenário de lentidão testemunhado nas redes de computadores. Uma das abordagens adotadas pelas RVP segue o formato das Redes Definidas por *Software* (SDN - *Software-Defined Networks*) [Lantz *et al.* 2010], que define o desacoplamento dos planos de controle e de dados. Algumas implementações do plano de controle se baseiam na Arquitetura Orientada a Serviços (SOA - *Service Oriented Architecture*) para prover a comunicação com aplicativos de rede. Dessa forma, utilizando uma interface padronizada de definição de serviços, os aplicativos de rede podem ser programados em linguagens distintas, se tornam menos dependentes da tecnologia utilizada no plano de controle.

Por um lado, o desacoplamento entre aplicativos de rede, plano de controle e infraestrutura programável torna a arquitetura SDN flexível e escalável [Gutz *et al.* 2012]. Por outro lado, induz uma grande complexidade no gerenciamento. Modelos de gerenciamento utilizados nas redes comuns não são adequados às redes programáveis, uma vez que não tratam da implantação dinâmica de novos serviços. Além disso, dependendo das políticas de acesso às infraestruturas programáveis, um grande número de usuários poderão propor e implantar seus próprios aplicativos de rede. Nesse cenário, harmonizar os aplicativos, usuários e redes virtuais, mantendo a confiabilidade e escalabilidade dos serviços implantados é um problema em aberto.

As propostas para o gerenciamento de RVP variam de acordo com o ambiente de implantação e com os requisitos dos usuários. Nos ambientes de *testbeds*, as propostas focam em prover aos experimentadores e cientistas soluções para o controle do provimento de *Slices*. Entretanto, o gerenciamento da programabilidade que os usuários possuem sobre os *Slices* ainda é incipiente. São exemplos desse tipo de proposta o ProtoGENI [ProtoGENI 2012] e o OFELIA *Control Framework* [Kopsel 2011]. Nos ambientes de *Cloud*, mais direcionados ao mercado, implementam-se soluções para gerenciar os serviços que os provedores de *Cloud* oferecerão aos seus clientes. Ou seja, o foco maior está em prover controle aos gerentes e administradores da *Cloud*, deixando o usuário final sem chance de propor novos aplicativos de rede. As propostas da CITRIX, XenServer *Distributed vSwitch Controller* e da CISCO, OnePK, são exemplos.

O problema de pesquisa deste trabalho está em como prover acesso de múltiplos usuários finais a uma infraestrutura de RVP, agregando facilidades no gerenciamento e implantação de aplicativos de forma a encorajar o desenvolvimento de novas soluções de rede. Propõe-se para isso a plataforma de gerenciamento ProViNet (***Pro****grammable* ***Vi****rtual* ***Net****work Managemet Platform*). ProViNet contribui para o estado-da-arte em quatro pontos: *(i)* na elaboração de uma arquitetura para gerenciamento de RVP que provê escalabilidade e alta disponibilidade de serviços; *(ii)* em uma abordagem para implantação dinâmica de novos serviços no plano de controle; *(iii)* na extensão da linguagem de descrição de rede virtual programável, *Virtual Resources and Interconnection Networks Description Language* (VXDL) [Koslovski *et al.* 2008]; *(iv)* e por fim, no desenvolvimento, como parte da plataforma, de um sistema com interface de acesso Web que facilita a compreensão e interação dos usuários finais com ambientes de RVP.

O restante do artigo está organizado conforme segue. Na Seção 2, são descritos os principais trabalhos que envolvem o gerenciamento de RVP. Na Seção 3 é apresentada a plataforma ProViNet, discutindo a arquitetura conceitual e conceitos empregados. Em seguida, na Seção 4 é detalhado o protótipo utilizado como base para a avaliação e análise apresentada na Seção 5. Por fim, a Seção 6 conclui o artigo com as considerações finais e perspectivas para trabalhos futuros.

2. Trabalhos Relacionados e Contextualização

Os conceitos de Virtualização de Redes [Chowdhury e Boutaba 2009] [Chowdhury e Boutaba 2010] e a Programabilidade de Redes [Campbell *et al.* 1999] [Lin *et al.* 2011] são bem discutidos na literatura. A junção desses conceitos formam as Redes Virtuais Programáveis (RVP), as quais são mais comumente aplicadas em dois ambientes, nos projetos de plataformas de testes, também conhecidos como *testbeds* e em Nuvens privadas (*Private Clouds*). Os trabalhos relacionados apresentados neste capítulo são organizados conforme esses dois ambientes, buscando evidenciar em cada um deles o nível de abstração da infraestrutura virtual, a abordagem utilizada e a natureza da licença.

Dentre as propostas no contexto dos *testbeds*, as quais focam em auxiliar os pesquisadores em seus experimentos, destaca-se o *framework* de controle OFELIA (OCF) [Kopsel 2011], o qual é uma derivação da plataforma Expedient proposta pela Universidade de Stanford. Esse *framework* auxilia os pesquisadores na criação de *Slices* utilizando recursos de várias federações. Além disso, lhes oferece também a capacidade de associar esses *Slices* a controladores previamente configurados. Apesar dessa funcionalidade de associação através da interface gráfica, o OCF não provê o gerenciamento da implantação de aplicativos de rede, considerada uma das maiores preocupações da virtualização de redes [Chowdhury e Boutaba 2010].

Outra proposta, ainda no contexto dos *testbeds*, foi criada no projeto GENI [GENI 2011] e é chamada ProtoGENI [ProtoGENI 2012]. Tal proposta também implementa uma interface de acesso via Web que assiste aos pesquisadores na criação de *Slices* com recursos oriundos de diversas federações. Ao interagir com a interface do ProtoGENI, pesquisadores podem instanciar nós virtuais e conecta-los dinamicamente. Assim como a maioria das propostas desenvolvidas nesse mesmo contexto, o foco está no provimento do *Slice* e não na programabilidade do mesmo. Em geral, essas soluções são livres de licença, entretanto existe um conjunto burocrático de regras para utilização e acesso aos recursos geridos pelas ferramentas citadas.

As soluções no contexto de *Cloud Computing* se diferem das propostas de *testbeds*, principalmente pelo foco comercial. Os ambientes de *Cloud* em geral demandam uma grade quantidade de recursos de rede, pois comercializam serviços com alta taxa de disponibilidade e qualidade de serviço. Portanto, a criação de serviços de rede customizados para atender a demandas especiais é visto como um grande atrativo para os provedores de *Cloud*. Para atender esses provedores, surgiram soluções que aumentam o poder de customização dos serviços providos nas redes virtuais de seus *datacenters*. Algumas dessas soluções são comercializadas, tais como Nexus 1000v e OnePK da CISCO e DVSC (*Distributed Virtual Switch Controller*) da CITRIX. Outras são de código aberto, como os *plugins open-vSwitch*, Ryu Plugin e o *restproxy*, para a plataforma OpenStack [OpenStack 2011].

Tanto em *Cloud* quanto em *testbeds*, é crescente o número de propostas que empregam os conceitos da arquitetura SDN em suas soluções de programabilidade em redes virtuais. Rubio *et al.* [Rubio-Loyola *et al.* 2011], por exemplo, propôs um plano de orquestração, o qual é complementar aos planos originalmente definidos na arquitetura SDN (controle e dados). O propósito desse novo plano é controlar dinamicamente o comportamento das redes virtuais em resposta às constantes variações, gerando assim um sistema de gerenciamento autonômico. Embora esse sistema tenha vantagens, como resposta rápida às falhas geradas por variações no comportamento da rede, requer soluções padronizadas e bem testadas. Em consequência, o usuário final continua distante da criação e implantação de aplicativos na rede virtual programável. Tal fato, contribui para baixa taxa de inovação nas redes, pois mantem a natureza restritiva e a pequena quantidade de pessoas aptas a desenvolver e implantar novas soluções.

Em resumo, apesar de existirem propostas recentes envolvendo redes virtuais programáveis, nenhuma das pesquisadas promove o gerenciamento da programabilidade em uma infraestrutura de RVP, considerando o acesso de múltiplos usuários finais. Além disso, maior parte das propostas analisadas se limitam ao provisionamento da rede virtual (controle de máquinas virtuais e configuração da rede virtual), não oferecendo funcionalidades para o gerenciamento dos aplicativos de rede que podem ser instalados dinamicamente nas redes virtuais programáveis.

3. ProViNet

Uma das abordagens para as Redes Virtuais Programáveis é a utilização da arquitetura de Redes Definidas por *Software*. Tal arquitetura define que os planos de controle e de dados sejam desacoplados. Sendo assim, eles necessitam de um protocolo para comunicação. Recentemente tem se empregado o termo *Southbound API* (SBAPI) para se referir aos protocolos que provejam essa comunicação entre o plano de controle e de dados. Conforme ilustrado na Figura 1 a SBAPI **(2)** é utilizada para comunicação entre o *Pool* de Controle e os *Slices*, que serão descritos mais diante.

Ao passo que surgiram diversas soluções para a camada de controle, e cada implementação adota um padrão de linguagem, os aplicativos de rede criados seguiam tal heterogeneidade, ficando assim dependentes das tecnologias dos controladores e isoladas entre si. Atualmente a tendência é que as diferentes implementações do plano de controle ofereçam uma interface padrão de comunicação com aplicativos de rede externos. É comum se referir a esse tipo de interface como *Northbound API* (NBAPI) **(1)**. Em geral, elas independem de linguagem, baseando-se na arquitetura *Representational State Transfer* (REST), por exemplo. Desse modo, novos aplicativos que venham a interagir com a camada de controle necessitam apenas das especificações de serviços providas por cada implementação de controlador.

Conforme ilustrado na Figura 1, a plataforma ProViNet auxilia os usuários finais no gerenciamento e implantação de aplicativos de rede em *Slices* de Redes Virtuais Programáveis. Para isso, se apoia no conceito de separação de planos, sejam eles: o plano de dados, representado pelos elementos que formam a rede virtual nos *Slices* e utilizam a SBAPI para se comunicarem com o plano de controle; o plano de controle, formado

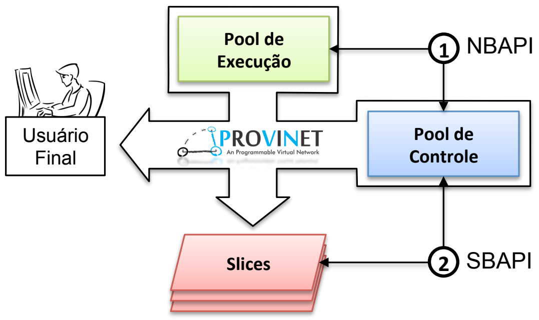


Figura 1. Módulos e relacionamentos

pelos controladores, que são agrupados no *Pool* de Controle para prover alta disponibilidade, conforme será apresentado na Subseção 3.1; e pelo plano de aplicativos, presente no *Pool* de Execução, o qual organiza, armazena e executa os aplicativos de rede que se aproveitam do conceito de NBAPI para comunicação com o plano de controle.

A plataforma ProViNet se aplica a qualquer ambiente que utilize infraestrutura compatível com o conceito de RVP. Em geral, a aplicação se dá em dois níveis, um no nível de rede virtual, no qual o plano de dados seriam representado pelos *vSwitches*, e outro no nível físico, utilizando os *switches* físicos compatíveis como plano de dados. Ou até mesmo em um modelo híbrido, com controle em ambos os níveis.

3.1. Arquitetura Conceitual

A arquitetura apresentada na Figura 2 ilustra os componentes da plataforma ProViNet assim como suas relações em alto nível. O usuário interage com a plataforma através de uma interface Web que expõe graficamente as funcionalidades providas por seus módulos. Conforme ilustrado, essas funcionalidades se subdividem em quatro interações. Na interação **(a)**, o módulo Controle de Usuários atende a requisições de autenticação e cadastro. O processo de gerenciamento de implantação e execução de aplicativos, executado na interação **(b)**, é tratado pelo módulo Controle de Aplicativos. As solicitações de infraestrutura virtual são enviadas na interação **(c)** e tratadas pelo módulo de Controle de *Slices* e Referências. Por fim, as configurações inerentes aos três módulos citados são apresentadas em uma interface de administração **(d)** para o Administrador do ambiente de implantação do ProViNet.

ProViNet fornece escalabilidade utilizando uma arquitetura baseada no conceito chamado de *Resource Pool*. Cada *Pool* representa um conjunto de servidores de virtualização (*hypervisors*) interligados e controlados por uma plataforma de gerenciamento única. O *Pool* de Controle é utilizado para execução de máquinas virtuais com sistemas idênticos, que executam uma implementação de controlador pré-definida e compatível com o conceito de NBAPI. Os aplicativos escritos pelos usuários são executados em máquinas virtuais individuais alocadas no *Pool* de Execução. A comunicação entre os controladores do *Pool* de Controle e as VMs do *Pool* de Execução é provida pela NBAPI. Já a comunicação entre os controladores e os elementos do plano de dados nos *Slices* é provida pela SBAPI. As requisições realizadas pelos módulos do ProViNet aos *Pools* utilizam as interfaces de comunicação providas pela plataforma de virtualização adotada. Finalmente, a requisição ao Provedor de Infraestrutura Virtual ocorre por meio de uma requisição HTTP, enviando um documento de descrição de infraestrutura virtual, a ser comentado mais adiante.

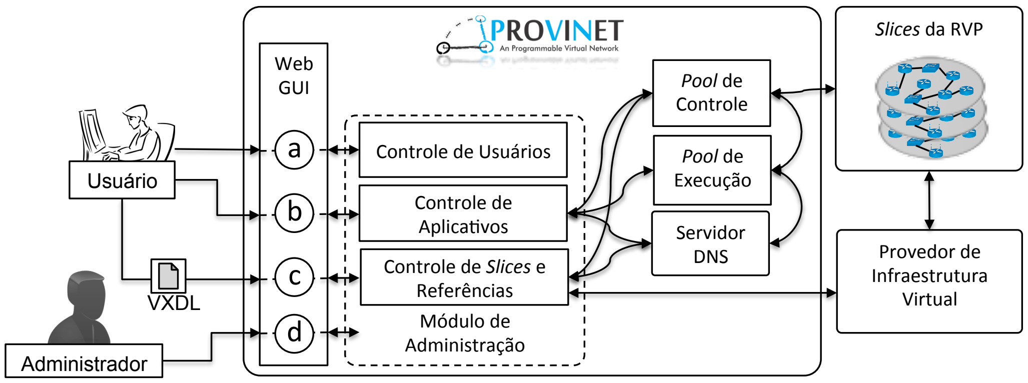


Figura 2. Arquitetura conceitual

Analisando novamente a Figura 2, percebe-se a esquerda o Usuário Final e a direita a Rede Virtual Programável. Entre esses elementos está a plataforma ProViNet, provendo a ligação entre eles. Nessa posição, o ProViNet oferece o gerenciamento da implantação de aplicativos, o controle de acesso para ambientes com múltiplos usuários e os requisitos não funcionais de disponibilidade, confiabilidade e escalabilidade. Com exceção do módulo de Controle de Usuários e de Administração, devido a simplicidade, os outros são detalhados nas subseções seguintes.

3.2. Controle de *Slices* e Referências

Uma Rede Virtual Programável (RVP) dispõe de recursos computacionais e de rede necessários para a criação de redes isoladas e programáveis. Para isso, se apoiam em tecnologias de virtualização (tais como XenServer, VMWare e Virtualbox) e de programabilidade (como o OpenFlow, empregado neste trabalho). Seja qual for a tecnologia e a abordagem de provimento da RVP, o resultado é que o usuário terá uma rede virtual programável que interliga exclusivamente suas máquinas virtuais. Por compartilharem os recursos do provedor (ou *datacenter*), é conveniente e usual se referir ao subconjunto de recursos de rede, computacionais e de armazenamento pertencentes a um usuário, de *Slice*.

Não é função da plataforma ProViNet o provimento do *Slice*, ou seja, inicializar máquinas virtuais e configurar a rede virtual que interliga as mesmas. Portanto, o módulo Controle de *Slices* e Referências tem, entre outras, a função de receber e encaminhar um documento de descrição de infraestrutura virtual especificando a topologia e os recursos a serem alocados pelo Provedor de Infraestrutura Virtual (PIV). Existem diversas propostas para esse tipo de documento, tais como Rspec (GENI), NDL-OWL(RENCI), NMC (OGF) e *Virtual Resources and Interconnection Networks Description Language* (VXDL) do projeto INRIA [Koslovski *et al.* 2008]. Neste trabalho propomos uma extensão para a linguagem VXDL, tornando-a compatível com arquiteturas de programabilidade de redes em que haja separação de planos (de controle e de dados).

Originalmente, a linguagem VXDL, apresentada no trabalho de Koslovski *et.al.* [Koslovski *et al.* 2008], define que os recursos possuem um nome e podem ter uma lista de funções, parâmetros, softwares e uma localização conforme se nota no trecho destacado de sua definição e apresentado abaixo:

```
<resource> ::= "resource" "(" <name> ")" "{"
               ["function" <elementary-functions>]
               ["parameters" <resource-parameters>]
               ["software" <software-list>]
               ["anchor" <location>] "}"
```

O atributo *function* pode assumir diversos valores, tais como *endpoints*, *aquisition* e *router*. Propõe-se a adição de um novo atributo inerente ao valor *router*, o qual nomeia-se *<controller-list>*. Esse valor representa uma lista de controladores. Foi adotado uma lista pois alguns roteadores ou *switches* compatíveis com SDN aceitam redundância de controladores. O *switch* virtual Open vSwitch por exemplo aceita a configuração de um *master* e diversos *slaves*. Cada elemento *<controller>* é composto por atributos que definem o tipo de conexão (ftp, ptcp e ssl são exemplos), o endereço IP e a porta em que o controlador remoto está configurado.

```
<elementary-functions> ::= <function> ("," <function> )*
<function> ::= "endpoint" | "aquisition" | "storage"
    | "computing" | "visualization" | "network_sensor"
    | "router" "(" "ports" <ports> [<controller-list>] ")"
<ports> ::= <number>
<controller-list> ::= <controller> ("," <controller> )*
<controller> ::= "(" <connection-type> <ip-address> <port> ")"
```

O usuário inicialmente deve elaborar este arquivo sem as informações sobre o controlador, pois estas serão adicionadas automaticamente pelo ProViNet durante o processo de requisição de infraestrutura virtual. Isso se justifica pelo fato do usuário não ter informações sobre os endereços de IP dos controladores, até mesmo porque eles são dinâmicos, conforme será apresentado adiante.

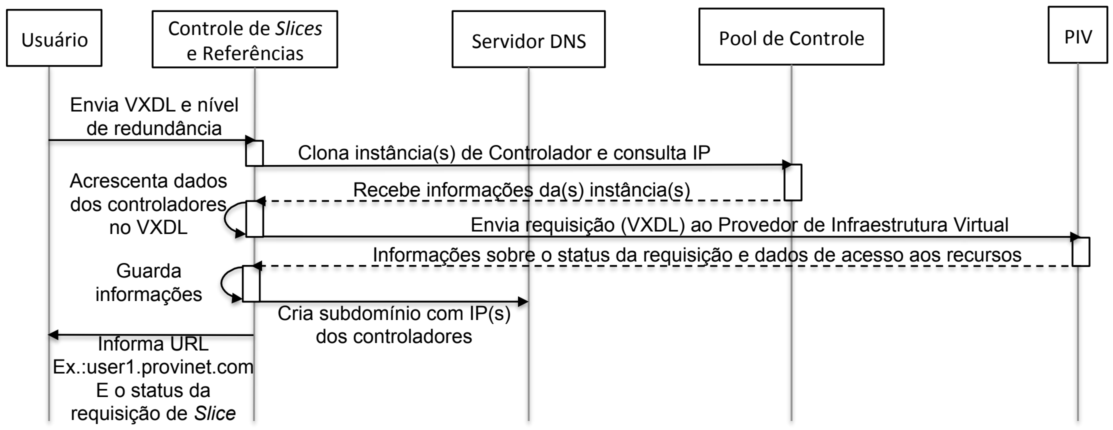

Usuário
Controle de *Slices* e Referências
Servidor DNS
Pool de Controle
PIV
Envia VXDL e nível de redundância
Clona instância(s) de Controlador e consulta IP
Recebe informações da(s) instância(s)
Acrescenta dados dos controladores no VXDL
Envia requisição (VXDL) ao Provedor de Infraestrutura Virtual
Informações sobre o status da requisição e dados de acesso aos recursos
Guarda informações
Cria subdomínio com IP(s) dos controladores
Informa URL Ex.:user1.provinet.com E o status da requisição de *Slice*

Figura 3. Diagrama de sequência da abordagem de criação de *Slices*

De acordo com o diagrama da Figura 3, quando o usuário já elaborou o documento VXDL, ele o envia ao módulo Controle de *Slices* e Referências via formulário de *upload*. No mesmo formulário, o usuário deve definir um nível de redundância de controladores no *Pool* de Controle que pretende ser associado ao *Slice* requerido. Após a instanciação dos controladores, o *Pool* de Controle reponde a requisição informando o IP dos mesmos. Esses IPs são adicionados ao VXDL juntamente com o tipo de conexão e porta. Em seguida, o documento é enviado ao PIV por uma requisição HTTP, gerada automaticamente pela plataforma.

Ao receber a resposta sobre o status da requisição e com informações sobre o acesso aos recursos do *Slice*, a plataforma registra no Servidor DNS os IPs dos controladores (os mesmos adicionados no documento VXDL). Esse registro é a adição do nome do usuário como subdomínio do domínio *www.provinet.local*. Ou seja, cada usuário tem um subdomínio no qual são associados os IPs dos controladores por ele requeridos. Caso o usuário tenha definido grau e redundância maior que um, o Servidor DNS aplicará o algoritmo Round Robin entre as referências. Abaixo segue um exemplo de configuração de uma domínio em que o "user1" possui dois controladores e o "user2" tem um.

```
...
                NS      servidor.provinet.local.
servidor        A       xxx.xxx.xxx.xxx
user1           A       <IP-controlador-1>
user1           A       <IP-controlador-2>
user2           A       <IP-controlador-3>
...
```

Por fim, o módulo apresenta ao usuário a URL a ser utilizada em seus aplicativos para fazer chamadas aos serviços (Ex.: http://user1.provinet.local). São fornecidos, também, os dados de acesso aos recursos virtuais recebidos do PIV. Dessa forma, o aplicativo do usuário poderá enviar requisições aos serviços providos pelos controladores instanciados no *Pool* de Controle.

3.3. Controle de Aplicativos

Uma vez que a estrutura de programabilidade já está estabelecida, ou seja, o usuário possui um *Slice* e um plano de controle devidamente configurado, resta ao usuário desenvolver e executar os aplicativos de rede. Para que o desenvolvimento seja possível, o usuário precisa saber quais são os serviços a sua disposição. Por isso, é disponibilizado através da interface Web do ProViNet, uma documentação completa sobre tais serviços, os quais dependem da tecnologia utilizada no *Pool* de Controle.

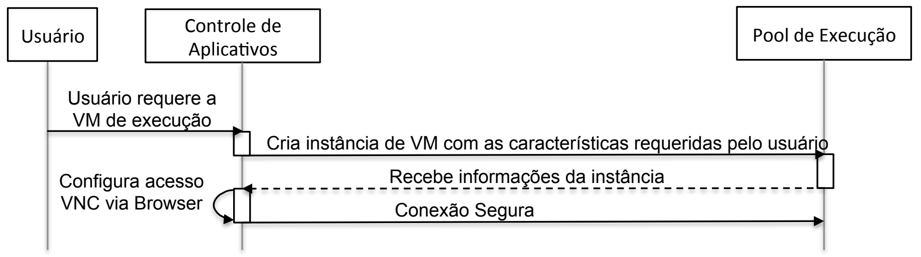

Usuário
Controle de Aplicativos
Pool de Execução
Usuário requere a VM de execução
Cria instância de VM com as características requeridas pelo usuário
Recebe informações da instância
Configura acesso VNC via Browser
Conexão Segura

Figura 4. Abordagem de referências para provimento de escalabilidade e disponibilidade

De acordo com o diagrama apesentado na Figura 4, no primeiro passo o usuário acessa a plataforma e requer a VM que rodará seus aplicativos. Nesta requisição são apresentados perfis de VM, variando o Sistema Operacional e quantidade de recursos, tais como memória, processamento e armazenamento. Após a escolha de um perfil, a plataforma, por meio de uma interface de comunicação com o *Pool* de Execução requer uma VM com tais características. Para evitar qualquer tipo de bloqueio por *firewall*, ao receber os dados da VM criada, o módulo Controle de Aplicativos configura uma interface de acesso remoto via *Browser*. Ou seja, o usuário é capaz de visualizar e interagir com a VM criada por meio de um terminal apresentado em seu *Browser*.

Como ilustrado na Figura 5, o aplicativo do "user1" rodando no *Pool* de Execução poderia fazer a chamada *http://user1.provinet.local/getTopology* para consultar a topologia da rede virtual disponibilizada em seu *Slice*. Se o usuário tem mais de um controlador, as requisições dos aplicativos serão direcionadas para os controladores conforme o algoritmo Round Robin, implementado pelo Servidor DNS. Assim, existirá um balanceamento de carga entre os controladores. Além disso, em caso de falha de um controlador, terá outro para atender as requisições.

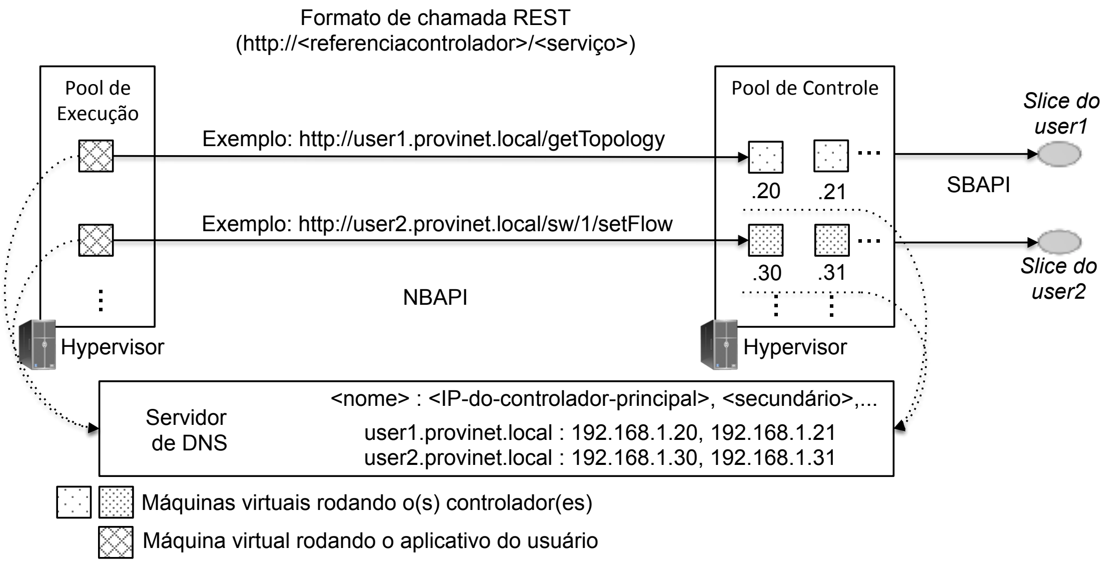

Formato de chamada REST
(http://<referenciacontrolador>/<serviço>)
Pool de Execução
Pool de Controle
Exemplo: http://user1.provinet.local/getTopology
Exemplo: http://user2.provinet.local/sw/1/setFlow
.20 .21
.30 .31
SBAPI
NBAPI
Slice do user1
Slice do user2
Hypervisor
Hypervisor
Servidor de DNS
<nome> : <IP-do-controlador-principal>, <secundário>,...
user1.provinet.local : 192.168.1.20, 192.168.1.21
user2.provinet.local : 192.168.1.30, 192.168.1.31
Máquinas virtuais rodando o(s) controlador(es)
Máquina virtual rodando o aplicativo do usuário

Figura 5. Abordagem de referências para provimento de escalabilidade e confiabilidade

O conjunto de serviços disponíveis deve variar de acordo com a tecnologia utilizada no plano de controle. Todavia, diante da necessidade de um novo serviço, o mesmo pode ser desenvolvido e instalado. Em geral, os controladores seguem uma arquitetura que permite o acoplamento de módulos com a implementação de novos serviços. Baseado nas informações de quais controladores pertencem a quais usuários, a plataforma ProViNet auxilia o usuário na instalação desses módulos.

Para implementar o módulo a ser instalado, o usuário deve seguir os tutoriais[1] disponibilizados pela fabricante do controlador adotado no plano de controle. O processo de instalação de um novo módulo se inicia pelo envio do módulo compactado para a plataforma. O módulo responsável então o encaminha a todos os controladores do usuário que estão associados a um determinado *Slice*. Em cada controlador, um *deamon* configurado para fazer a implantação de módulos executa uma rotina de instalação. Uma vez que os módulos estão instalados, os aplicativos dos usuários podem enviar requisições a ele (Ex.: *http://user1.provinet.local/modules/newModule/service*).

[1]Tomando como exemplo o controlador Floodlight, estão disponíveis no endereço *http://www.openflowhub.org/display/floodlightcontroller/How+to+Write+a+Module*, um conjunto de instruções para o desenvolvimento de novos módulos.

4. Protótipo

Com o objetivo de demonstrar a viabilidade da arquitetura conceitual detalhada na Figura 2 do Capítulo 3, foi desenvolvido um protótipo. Sua implementação baseia-se no desenvolvimento dos cinco principais módulos, a Interface Web, o Controle de Usuários, o Controle de Aplicativos, o Controle de *Slices* e Referências e a Administração do ProViNet. Tais módulos foram implementados utilizando o *framework* Django 1.4.3, a linguagem Python 2.7.3 e o sistema gerenciador de banco de dados PostgreSQL 9.1.6. A fim de fornecer maior compatibilidade do sistema, foi utilizado o servidor Web Apache 2.2.23.

O módulo Controle de Usuários fornece na interface do ProViNet, formulários de registro e login. O Controle de Aplicativos apresenta na interface uma área especial para apresentação dos serviços disponibilizados no plano de controle e uma área para requisição, controle e interação com as VMs no Pool de Execução. É disponibilizado pelo módulo de Controle de *Slices* e Referências um formulário para *upload* do documento VXDL e definição do nível de redundância. Finalmente, a área de Administração apresenta um conjunto de funcionalidades de configuração, que incluem o endereço de IP e dados de acesso dos *Pools*, do Servidor DNS, do Provedor de Infraestrutura Virtual e ainda a definição dos perfis de máquinas virtuais que serão disponibilizadas aos usuários.

Os *Pools* de Execução e de Controle são, na prática, servidores com alguma plataforma de virtualização instalada. No protótipo desenvolvido foi utilizado o *hypervisor* XenServer da CITRIX. Sendo assim, a comunicação entre os módulos da plataforma ProViNet e os Pools se dão pela utilização do XenServer SDK. Com o SDK é possível controlar e monitorar o *hypervisor* através de chamadas XML-RPC. O último módulo da plataforma é o Servidor DNS, o qual foi instalado em uma máquina com os sistemas Bind9 e Apache2. Para o controle dinâmico de configuração do DNS, um serviço web foi implementado para que, a partir de chamadas HTTP a uma URL específica, seja feita a adição e remoção de entradas no arquivo de configuração do bind9.

A implementação de controlador utilizada foi o Floodlight, por prover uma API RESTfull que possibilita o consumo dos serviços disponibilizados no controlador por meio de chamadas HTTP. Outras implementações poderiam ser utilizadas, desde que seja possível a instalação de módulos para provimento de serviços customizados e a disponibilização de serviços utilizando a arquitetura REST.

5. Caso de Estudo e Análise de Resultados

Para avaliar a solução apresentada foi elaborado um caso de estudo que aborda cada um dos diagramas de sequência apresentados no Capítulo 3. O cenário de execução é composto por servidores Intel Xeon CPU E3-1220 3.1GHz, 4GB RAM, com o *hypervisor* XenServer 6.1 representando o *Pool* de Controle e de Execução. O controlador OpenFlow utilizado foi o Floodlight v0.90, e é iniciado, por *script* durante a inicialização da VM (Ubuntu 12.04 com 1 vCPU e 384MB RAM) no *hypervisor*. Para executar o *framework* Django com o ProViNet foi utilizado um *laptop* Intel Core i7 2.8GHz e 4GB RAM. Por fim, o Servidor DNS foi instalado e configurado em um terceiro PC (Intel Core 2 Duo 2.33GHz e 4GB RAM) na mesma rede local que os outros PCs. Vale ressaltar que os tempos apresentados neste trabalho foram obtidos após 30 execuções, e representam a média uma vez que o coeficiente de variação foi muito próximo de zero.

Instanciar dois controladores	Adicionar informações no VXDL	Provimento da infraestrutura virtual (PIV)	Configuração de subdomínios	Total
12,824s	0,003s	57,81s	0,04s	75,677s

Tabela 1. Tempos para requisição de *Slice* e configuração de subdomínios

Atraídas pelo baixo custo de manutenção, segurança e armazenamento, é cada vez mais comum que empresas migrem parte de sua infraestrutura de computação para ambientes de *Cloud*, seja ela pública, privada ou híbrida. Para representar esse caso de estudo, consideramos uma rede composta por 7 *switches* e 4 *hosts*, organizados em uma topologia de árvore. Consideramos também que o usuário responsável pela migração já está registrado na plataforma ProViNet. Inicialmente, o usuário faz a requisição da infraestrutura virtual enviando um arquivo VXDL com a descrição da topologia (*switches*, *hosts* e *links*). A avaliação desse processo é apresentada na Tabela 1 e pode ser acompanhada pelo diagrama da Figura 3. Os tempos apresentados consideram nível de redundância igual a dois, ou seja, o *Slice* terá dois controladores associados.

Vale ressaltar que o tempo gasto para instanciar a infraestrutura virtual deve variar de acordo com a abordagem de mapeamento utilizada pelo PIV. A utilização de máquinas virtuais ao invés de *bridges* para representar os *switches* virtuais, por exemplo, certamente acarreta em maior custo de tempo. Os tempos apresentados consideram um sistema desenvolvido em um trabalho anterior chamado HyFS [Wickboldt *et al.* 2012] como provedora da infraestrutura virtual programável. O HyFS é atualmente a única plataforma de *private Cloud* capaz de receber requisições de rede virtual programável com topologias variadas. A topologia virtual é criada utilizando o modelo *overlay*, no qual *switches* e *hosts* são implantados em máquinas virtuais. Uma vez instanciados, os *software switches* são configurados para receberem informações de controle de controladores externos, estabelecidos pela plataforma ProViNet.

Após o provimento da infraestrutura virtual, o usuário responsável pela migração deve avaliar os serviços presentes no cenário anterior a migração, tais como *Firewall*, Balanceadores de carga e outros serviços necessários. Uma vez que a rede virtual a ser criada é programável, esses serviços podem ser implementados em forma de aplicativos de rede. Utilizando uma notação simplificada, um exemplo de aplicativo para tratamento de ataques de negação de serviço (DDoS) é apresentado no Algoritmo 1. Esse aplicativo roda no plano de execução, em uma VM que o usuário acessa através do *Browser*.

O algoritmo apresentado analisa o tráfego recebido por um *switch* a cada 10 segundos, se esse tráfego for superior a um limite pré definido o sistema toma alguma ação. Essa ação depende do número de ocorrências em que se detectou tráfego superior ao limite definido. Na segunda ocorrência o sistema bloqueia o tráfego ICMP, na terceira, o tráfego Web (porta 80) é bloqueado, e por fim, na quarta ocorrência, todo tráfego é bloqueado e um gerente é contactado. Certamente essa não é a melhor solução, mas é um exemplo de um possível aplicativo de rede que pode ser desenvolvido pelo usuário e implantado no *Pool* de Controle para proteger seu *Slice* desse tipo de ataque.

Para medir desempenho das requisições do aplicativo do usuário, foram realizados experimentos com varições no nível de redundância no plano de controle. Espera-se uma variação no desempenho das chamadas, pois ao utilizar uma maior quantidade de controladores, o balanceamento de carga realizado pelo Servidor DNS torna o processo

Algorithm 1 Exemplo de Aplicativo: Solução ProViNet para DDoS

```
Require: limite - Limite de tráfego considerado normal
Require: swdpid - DPID do switch de entrada de tráfego rede
 1: $limite \leftarrow 2000MB$
 2:
 3: $anterior \leftarrow httpRequest(user1.provinet.local/readstatus/swdpid)$
 4: $ocorrencias \leftarrow 1$
 5: while True do
 6:     $atual \leftarrow httpRequest(user1.provinet.local/readstatus/swdpid)$
 7:     if $(atual - anterior) \geq limite$ then
 8:         if $ocorrencias == 2$ then
 9:             $httpRequest(user1.provinet.local/addFlow/swdpid/blockICMP)$
10:         else if $ocorrencias == 3$ then
11:             $httpRequest(user1.provinet.local/addFlow/swdpid/blockWeb)$
12:         else if $ocorrencias \geq 3$ then
13:             $httpRequest(user1.provinet.local/addFlow/swdpid/blockAll)$
14:             $CallManager()$
15:         end if
16:         $ocorrencias + +$
17:     end if
18:     $anterior \leftarrow atual$
19:     $sleep(10)$
20: end while
```

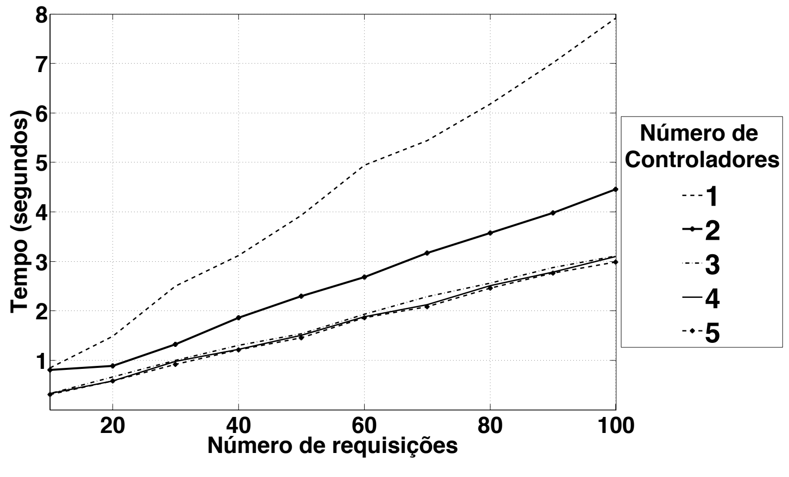


Figura 6. Performance do plano de controle com balanceamento de carga

mais rápido. Tal fato pode ser acompanhado no gráfico da Figura 6, que mostra o tempo gasto para concluir X requisições, sendo X, valores entre 10 e 100 com intervalos de 10 unidades. A requisição executada para obtenção dos valores apresentados foi *provinet.local/wm/core/controller/switches/json*, a qual retorna a lista de *switches* presentes no *Slice*.

Analisando os valores apresentados no gráfico 6, percebe-se que o balanceamento de carga provido pela abordagem proposta e gerenciado pelo ProViNet é efetivo e implica em uma redução do tempo gasto para execução das requisições. Entretanto, o ganho se torna menos significativo para um número de controladores maior que 3. Ou seja, utilizando apenas 1 controlador, foram gastos em média 7,91 segundos para concluir 100 requisições, ao passo que com 2 controladores esse valor caiu para 4,45. O ganho ao aumentar o número de controladores de 4 para 5, não é tão expressivo quanto de 1 para 2, saindo de 3,09 para 2,98 segundos nesse caso. Discussões mais aprofundadas nesse contexto foram apresentadas por Heller *et.al.* [Heller *et al.* 2012].

Para avaliar a funcionalidade de implantação de módulos sob demanda, desinstalou-se do Floodlight um módulo que originalmente já vem instalado, o módulo de *firewall*. Compactou-se tal módulo em um arquivo e, através da interface do ProViNet, foi feita a requisição de instalação. Uma vez que o módulo Controle de Aplicativos possui cadastrado o endereço IP de todos os controladores e seus respectivos usuários, após receber o arquivo por *upload*, um *script* de envio é disparado. O papel desse *script* é acessar via ssh a VM de cada controlador, fazer a cópia do novo módulo para uma pasta específica na VM e ativar um segundo *script* na VM que faz a instalação do mesmo. Esse segundo *script* segue as informações disponibilizadas no site do Floodlight para instalação de módulos. O tempo médio gasto nesse processo foi de 23,435 segundos.

6. Conclusões e Trabalhos Futuros

A diversidade de ambientes computacionais requerem distintos serviços de comunicação em redes. As soluções desenvolvidas no passado, e implementadas de acordo com as vontades das fabricantes de equipamentos de redes, podem não ser mais suficientes. Entretanto, com o surgimento de propostas abertas de virtualização e programabilidade, como as Redes Definidas por *Software*, a criação de novas soluções se torna, de certa forma, mais democrática. Ou seja, depende menos dos anseios financeiros das grandes fabricantes.

Todavia, a complexidade inerente ao gerenciamento de ambientes de Rede Virtual Programável (RVP) ainda representa um grande desafio. Neste trabalho, propomos a plataforma ProViNet para o gerenciamento da implantação de aplicativos de rede em ambiente de RVP. A plataforma ProViNet contribui com uma arquitetura escalável, utilizando o conceito de *Resource Pool*, com uma abordagem para a instalação dinâmica de novos módulos no plano de controle, com a extensão da linguagem de definição de infraestrutura virtual de rede virtual programável, chamada VXDL, e por fim, com o desenvolvimento de um sistema com interface de acesso Web, facilitando a compreensão e interação com usuários finais.

Como trabalhos futuros pretende-se investigar mais precisamente, como os ambientes de *Cloud* poderiam prover pratilheiras de serviços de rede dinâmicas. As quais seriam ocupadas por soluções desenvolvidas e consumidas pelos próprios usuários de *Cloud*.

Referências

Campbell, A. T., Meer, H. G. D., Kounavis, M. E., Miki, K., Vicente, J. B., e Villela, D. (1999). A survey of programmable networks. *Computer Communication Review*, 29:7–23.

Chowdhury, N. e Boutaba, R. (2009). Network virtualization: state of the art and research challenges. *Communications Magazine, IEEE*, 47(7):20 –26.

Chowdhury, N. M. K. e Boutaba, R. (2010). A survey of network virtualization. *Computer Network*, 54(5):862–876.

GENI (2011). Global Environment for Network Innovations. Disponível em: http://www.geni.net/. Acessado em: Julho 2012.

Gutz, S., Story, A., Schlesinger, C., e Foster, N. (2012). Splendid isolation: a slice abstraction for software-defined networks. Em *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, páginas 79–84, New York, NY, USA. ACM.

Hausheer, D., Parekh, A., Walrand, J., e Schwartz, G. (2011). Towards a compelling new internet platform. Em *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, páginas 1224 –1227.

Heller, B., Sherwood, R., e McKeown, N. (2012). The controller placement problem. Em *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, páginas 7–12, New York, NY, USA. ACM.

Kanaumi, Y., Saito, S., e Kawai, E. (2010). Deployment of a programmable network for a nation wide randd network. Em *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, páginas 233 –238.

Kopsel, W. (2011). Ofelia - pan-european test facility for openflow experimentation.

Koslovski, G. P., Primet, P. V.-B., e Charão, A. S. (2008). Vxdl: Virtual resources and interconnection networks description language. volume 2 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, páginas 138–154. Springer.

Lantz, B., Heller, B., e McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. Em *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, páginas 19:1–19:6, New York, NY, USA. ACM.

Lin, P., Bi, J., Hu, H., Feng, T., e Jiang, X. (2011). A quick survey on selected approaches for preparing programmable networks. Em *Proceedings of the 7th Asian Internet Engineering Conference*, AINTEC '11, páginas 160–163, New York, NY, USA. ACM.

OpenStack (2011). Open source software for building private and public clouds. Disponível em: http://www.openstack.org/. Acessado em: Julho 2012.

ProtoGENI (2012). Control Framework for GENI Cluster C. Disponível em: http://www.protogeni.net/trac/protogeni. Acessado em: Dezembro 2012.

Rubio-Loyola, J., Galis, A., Astorga, A., Serrat, J., Lefevre, L., Fischer, A., Paler, A., e Meer, H. (2011). Scalable service deployment on software-defined networks. *Communications Magazine, IEEE*, 49(12):84 –93.

Wickboldt, J. A., Granville, L. Z., Schneider, F., Dudkowski, D., e Brunner, M. (2012). A new approach to the design of flexible cloud management platforms. Em *8th International Conference on Network and Service Management (CNSM)*, páginas 155–158, Las Vegas, USA.