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

1550-445X/14 $31.00 © 2014 IEEE
DOI 10.1109/AINA.2014.142

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 runtime, 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).

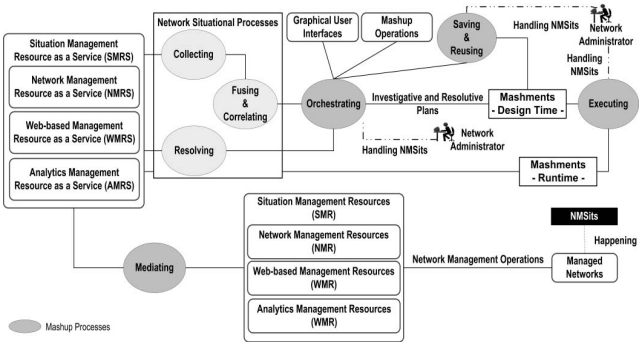

Figure 1. Addressing NMSits by Mashments: Conceptual Model

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 Procesess 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 $NMSit_{addr} \subseteq NMSit$.

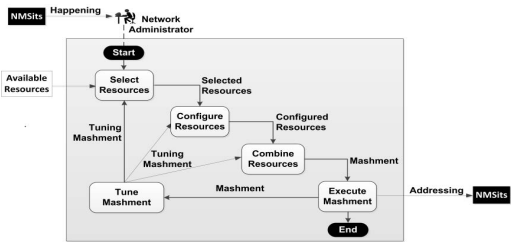

Figure 2. Process to Develop and Execute Mashments

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:

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

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:

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

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:

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

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

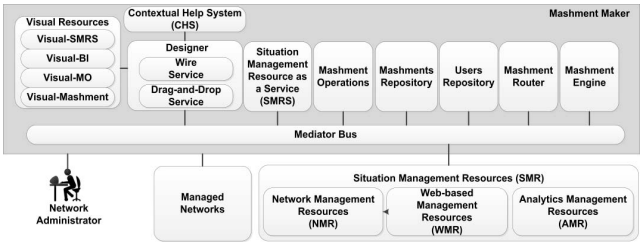

Figure 3. Mashment Maker Architecture

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.

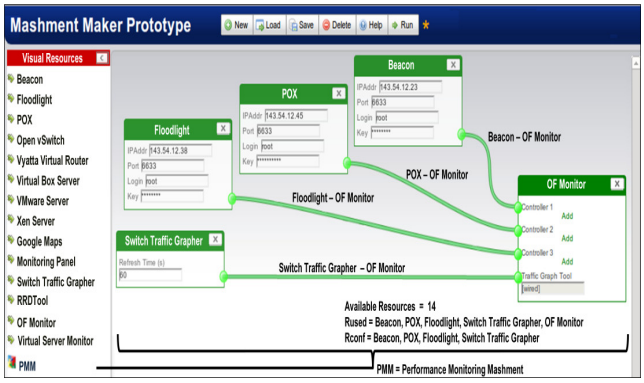

Figure 4. Mashment Maker Prototype and Performance Monitoring Mashment

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.

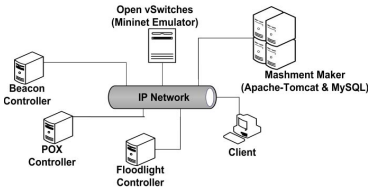

Figure 5. Test Environment

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, $\varsigma_{beacon} = \varsigma_{pox} = \varsigma_{floodlight} = sourceParameter(login) + sourceParameter(key) +$

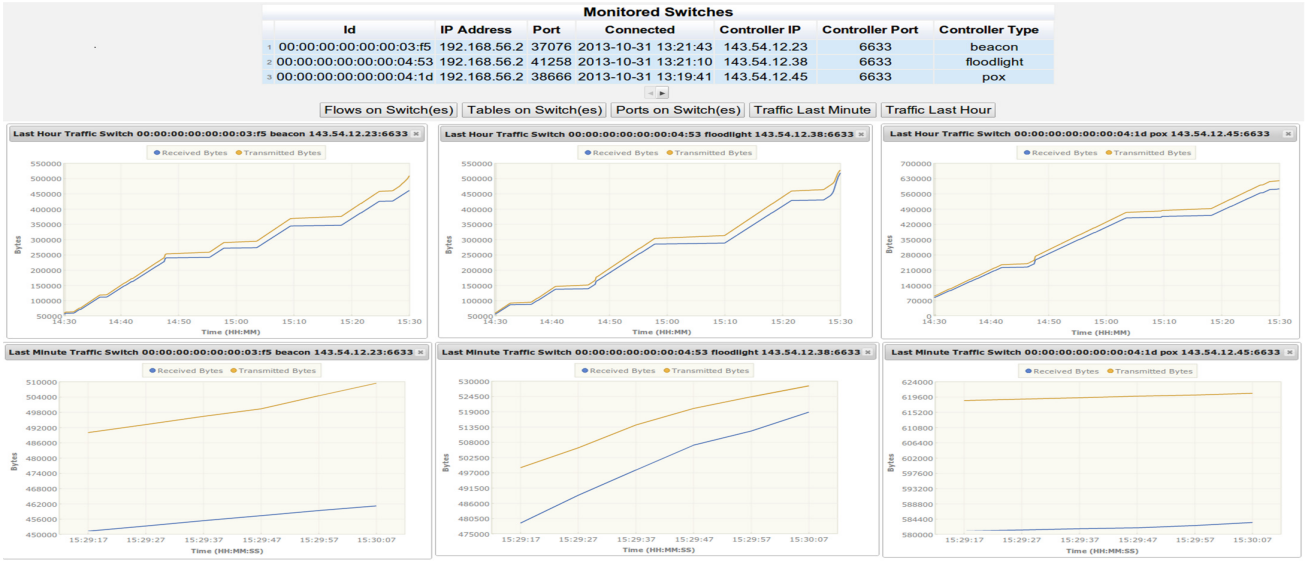

Figure 6. Performance Monitoring Mashment on Runtime

$sourceParameter(ip) + sourceParameter(port)$ 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.

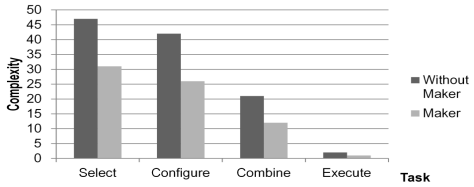

Figure 7. NMSit-SDN: Tasks Complexity

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.