SemNaaS: Semantic Web for Network as a Service

Morsey, M.; Zhu, H.; Canyameres, I.; Norbury, S.; Grosso, P.; Zivkovic, M.

DOI
10.5220/0005731200270036

Publication date
2016

Document Version
Final published version

Published in
CLOSER 2016: proceedings of the 6th International Conference on Cloud Computing and Services Science

SemNaaS: Semantic Web for Network as a Service

Mohamed Morsey[1], Hao Zhu[1], Isart Canyameres[2], Samuel Norbury[1],
Paola Grosso[1] and Miroslav Zivkovic[1]
[1]*Informatics Institute, University of Amsterdam, 1098 XH Amsterdam, The Netherlands*
[2]*i2CAT Foundation, 08034 Barcelona, Spain*

Keywords: Cloud Computing, Network as a Service, Network Markup Language, Semantic Web.

Abstract: Cloud Computing has several provisioning models, namely Infrastructure as a service (IaaS), Platform as a service (PaaS), and Software as a service (SaaS). However, cloud users (tenants) have limited or no control over the underlying network resources and services. Network as a Service (NaaS) is emerging as a novel model to bridge this gap. However, NaaS requires an approach capable of modeling the underlying network resources and capabilities in abstracted and vendor-independent form. In this paper we elaborate on SemNaaS, a Semantic Web based approach for supporting network management in NaaS systems. Our contribution is three-fold. First, we adopt and improve the Network Markup Language (NML) ontology for describing NaaS infrastructures. Second, based on that ontology, we develop a network modeling system that is integrated with the existing OpenNaaS framework. Third, we demonstrate the benefits that Semantic Web adds to the Network as a Service paradigm by applying SemNaaS operations to a specific NaaS use case.

1 INTRODUCTION

Cloud infrastructures and cloud services are becoming the de facto architecture to support operations of large web infrastructures, to process Big Data and to provide users with ubiquitous and scalable computing and storage services. This model of operation is proving effective thanks to its scalability and the associated economic advantages. Still there are less explored challenges for an optimal operation: one of them being the delivery of tailored network services.
New frameworks are emerging to define and create such dynamic network services; they effectively provide easy APIs to define services at the network level, while leveraging new capabilities such as Software Defined Networking methods and Network Virtualization techniques. These frameworks in essence support Network as a Service (NaaS) operations.
There are several definitions for NaaS:

- (Costa et al., 2012) defines it as a modern Cloud Computing paradigm in which the user has access to the network infrastructure. Via NaaS, the user can apply custom forwarding according to his/her application requirements.
- In (Minoves et al., 2012), the authors define NaaS as a strategy which enables the deployment of dynamic infrastructures which support various applications, with each application having its own network usage technique. The NaaS methodology offers virtual infrastructures to the user via software and/or web services.

The common element among the above definitions is that NaaS provides dynamicity and better matching between network performance and application requirements.

1.1 Benefits of Semantic Web for NaaS

The emerging NaaS software systems require powerful and rich vocabularies, such as the ones that can be provided by Semantic Web ontologies. Those vocabularies should hide the network implementation details from the user. Thus, NaaS users can fully express their needs in terms of services, without having to know the details of physical network devices. This is particularly significant when the requested network services span multiple domains that need to interoperate. OWL ontologies provide three main advantages as models for NaaS:

1. they are easy to extend;
2. they allow for automatic validation of both requests and provisioned services;
3. they enhance network resource discovery.

Morsey, M., Zhu, H., Canyameres, I., Norbury, S., Grosso, P. and Zivkovic, M.
SemNaaS: Semantic Web for Network as a Service.
In *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016) - Volume 1*, pages 27-36
ISBN: 978-989-758-182-3

The last advantage is the most distinguishing factor in adopting them as models for NaaS. Network users often request network resources providing different software programmability levels and different hardware configurations. An example would be a request for programmable network paths that use different links depending on the application data being passed between a sender and a receiver. This illustrates that discovering suitable resources that match high user/application dependent requirements is really a difficult task. However, the representation of this metadata in the ontology supports the keyword-based discovery of resources that often miss relatively capable resources. With semantic descriptions of resources, NaaS can not only discover the resource that exactly matches with that of the request, but also retrieves resources that exhibit subsumption relation when the exact match is not found.
Furthermore, RDF is a graph-based data model which promotes it for modeling computer networks, as a network forms a graph of interconnected nodes. This is quite advantageous for connectivity checking and path finding at early stage, i.e. prior to the actual network provisioning, which ultimately lowers the reservation cost (cf. Section 5).

1.2 The SemNaaS approach

We propose a novel approach to tackle the problem of augmenting NaaS for clouds with a semantic dimension. In the following we highlight:

1. our novel Semantic Web-based approach for NaaS, called SemNaaS;
2. our methods to empower the interconnection of distributed NaaS instances;
3. the application of our approach to OpenNaaS[1] as an example framework supporting NaaS;
4. a demonstration of a real world use case, which shows SemNaaS in operation.

The rest of the paper is structured as follows. First, we discuss related work in Section 2. Afterwards, we elaborate on the SemNaaS approach in 3. In 4, OpenNaaS is discussed in detail, which we use as the NaaS provider. In Section 5, we discuss the architecture of the SemNaaS system, and elaborate the SemNaaS features in Section 6. Finally, we evaluate our system using a use case showing the benefits of Semantic Web to NaaS in Section 7, before we conclude in Section 8.

[1]http://opennaas.org

2 RELATED WORK

Many disciplines leverage Semantic Web technology, including networks and Cloud Computing, in order to model their computing infrastructures. Peter Haase et al. (Haase et al., 2010) introduce a strategy for managing enterprise Cloud environments. They utilize Semantic Web to tackle the challenges involved in management of enterprise Cloud systems. They develop a product called eCloudManager, which depends on an ontology for modeling eCloudManager data. However, the system and its ontology only focus on the management aspect of Cloud systems. Furthermore, They do not address the problem of how various computing components are interconnected to each other.
Santana-Pérez et al. (Santana-Pérez and Perez-Hernández, 2012) propose a scheduling approach for federated hybrid Cloud systems, based on semantic technologies, capable of scheduling and allocating tasks to the most suitable resource(s). The approach reuses and modifies the UCI project ontologies[2], which comprise the information model. Although the ontologies cover a wide spectrum of details, they do not cover the interconnection between the various Cloud components.
Haak et al. (Haak and Grimm, 2011) present an ontology-based optimization framework that enables the Cloud service providers to figure out the best suiting resource combination that fulfills the user's request. The approach focuses only on the resource composition optimization without addressing the issue of using semantic technologies for administering the Cloud environments.
(Dastjerdi et al., 2010; Ngan and Kanagasabai, 2012) propose ontology-based methodologies for discovering and brokering Cloud services. Both works use semantics for user requirements and Cloud provider advertisements, and then apply a matchmaking algorithm to find the match between a requirement list and the list of advertised units. Both define multiple levels of matching, ranging from exact match to no match. Nevertheless, both concentrate only on Infrastructure as a Service (IaaS) provisioning.
Semantic Web Service (SWS) discovery (Junghans et al., 2010; Stollberg et al., 2007) tackles the problem of automated discovery of Web Services that fulfill the given requirements. The discovery process devotes a matchmaking algorithm for finding usable Web Services for solving the problem at hand. However, these approaches can not be applied on complex interconnected computing infrastructures.
Generally, all these approaches neither address the networking strategy, nor define a Network as a Service.

[2]http://code.google.com/p/unifiedcloud

This leads to the necessity of new methodology that gives the user more control over the networking and network resource reservation.

3 SemNaaS APPROACH

SemNaaS aims to improve OpenNaaS using Semantic Web technologies. The main advantage of SemNaaS is that it is a generic system in which OpenNaaS is a pluggable component. In other words, SemNaaS exposes its functionality as a set of APIs and RESTful web services, which OpenNaaS consumes. Hence, if OpenNaaS is replaced with any other Network as a Service platform, the latter can still benefit from the semantic capabilities SemNaaS offers. Furthermore, SemNaaS provides the capability to track and monitor the process of network resources provisioning at its various stages starting from resource(s) request to the release of those resource(s). SemNaaS is also open source[3] .
The main objective of SemNaaS is to add a semantic dimension to OpenNaaS, and subsequently to Network as a Service paradigm. That includes several potential improvement aspects for OpenNaaS:

- Support of network request validation and network connectivity check;
- System monitoring and failure condition detection;
- Capability of constructing distributed and interconnected OpenNaaS instances;
- Complex status report generation.

3.1 Preliminaries on RDF and SPARQL 1.1

According to the abstract treatments of RDF (Arenas et al., 2012; Pérez et al., 2006), we assume I, $\mathcal{B}$, and $\mathcal{L}$ be three pairwise disjoint sets denoting the sets of IRIs, blank nodes, and literals respectively.

Definition 1 (RDF Triple)**.** *An RDF triple τ is a tuple $(s,p,o) \in (I \cup \mathcal{B}) \times I \times (I \cup \mathcal{B} \cup \mathcal{L})$ where s, p, o are called subject, predicate and object, respectively.*

Definition 2 (RDF Graph)**.** *An RDF graph is a finite set of RDF triples $\mathcal{G} = \{(s_1,p_1,o_1),(s_2,p_2,o_2),...,(s_n,p_n,o_n)\}$ where s_i is the subject, p_i is the predicate and o_i is the object of triple τ_i.*

Definition 3 (SPARQL Graph Pattern)**.** *A SPARQL triple pattern is a tuple of the form $(I \cup \mathcal{V}) \times (I \cup \mathcal{V}) \times (I \cup \mathcal{L} \cup \mathcal{V})$ where I is the set of IRIs, $\mathcal{L}$ is the set of literals, and $\mathcal{V}$ being the set of variables. SPARQL basic graph pattern (BGP) is a finite set of triple patterns.*

Definition 4 (SPARQL Property Path)**.** *According to (Harris and Seaborne, 2013) a property path can be defined as: (1) if $i \in I \longrightarrow i$ is a property path, and (2) if p and q are property paths, then ($\hat{}p$), (p/q), (p|q), (p^*), and (p^+) are property paths as well, where $\hat{}$ denotes inverse, / denotes sequence, | denotes alternative, * is Kleene star, and + is Kleene plus.*

3.2 Network Markup Language (NML)

The NML ontology is at the core of the SemNaaS system, as it constitutes the main concepts required to create, define and link various network nodes. We have reused and improved the Network Markup Language (NML)[4] ontology (van der Ham et al., 2013). NML is an ontology constituting the information model for describing and defining computer networks. The developers of NML kept it abstract with the possibility of extension in order to customize it for certain use cases.
In order to use NML for NaaS, we have enhanced the ontology in order to permit the creation of complex network topologies[5]. The enhancements include devising more classes and properties and improving the existing ones as well.

New Classes and Properties. A path between two network nodes is simply an ordered list of links, starting from the start node and ending at the end node. Hence, in order to represent a path between two `Node` instances, we added two classes, namely `ListItem` which is the parent class of `Link` and `LinkGroup`. We also added class `Path` which denotes an ordered list of `ListItem`s. Furthermore, we added properties `hasSourceNetworkObject` and `hasSinkNetworkObject` and declared them as the inverse of properties `isSource` and `isSink` respectively. Those properties assist in connectivity checking (cf. Section 5).

Improved Classes and Properties. Basically, the network provisioned by OpenNaaS might span multiple geographical locations, thus we declared `Location` as a subclass of `geo:SpatialThing`[6]. Property `locatedAt` is declared as subproperty of `geo:location`, and its domain is `NetworkObject`

[3] http://github.com/dana-i2cat/semnaas

[4] http://www.ogf.org/documents/GFD.206.pdf

[5] The ontology is available at http://bitbucket.org/uvasne/indl/src/master/nml.owl

[6] http://www.w3.org/2003/01/geo/wgs84˙pos#

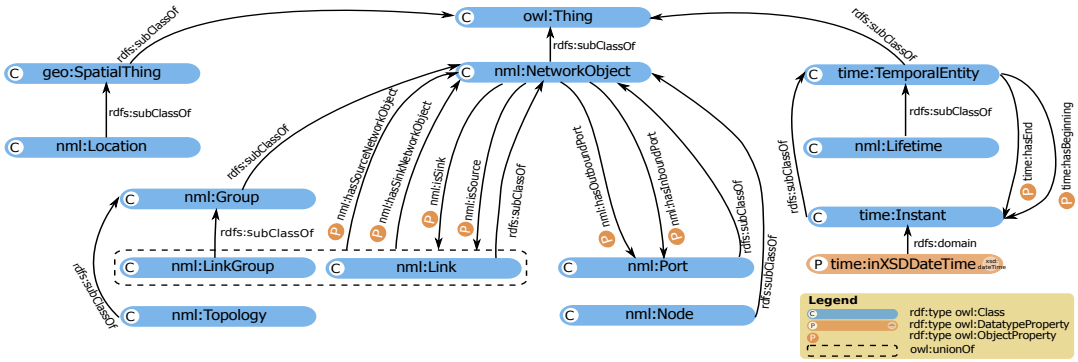


Figure 1: The key concepts and properties NML ontology.

and its range is `Location`. Furthermore, class `Lifetime` represents the time period during which a `NetworkObject` is active, and it is quite helpful in case of Network as a Service, as the network resources are reserved for a specific amount of time. `Lifetime` is declared as a subclass of `time:TemporalEntity` of the W3C Time ontology (Hobbs and Pan, 2006). We also utilized properties `time:hasBeginning` and `time:hasEnd` to denote start and end times of a reservation respectively.
1 illustrates the key classes and properties of NML.

4 OpenNaaS

OpenNaaS, i.e. Open Network as a Service, is an open source framework for network resources provisioning[7]. It enables the deployment, management, and configuration of dynamic network infrastructures. Furthermore, it defines vendor-independent interface to access and administer services provided by those network resources (Minoves et al., 2012; Aznar et al., 2013).
OpenNaaS was first conceived over the NaaS paradigm as a key enabler to offer virtual infrastructure services to third parties. It supports the creation of a virtual representation of a physical resource (e.g. router, switch), based on an abstract model of it which is often achieved by partitioning or aggregation.
It is built upon a set of core concepts, which are mainly resources, model, and capabilities. These concepts can be summarized as follows:

- Resources: The manageable units of OpenNaaS. They are the entities the user can own, manage and operate, e.g. a switch and a router.
- Model: The resource exposes a model that enables checking state and capabilities of this resource. Any change in resource state is reflected on its model.
- Capabilities: A capability is an interface to a resource functionality, e.g. Open Shortest Path First (OSPF) configuration capability for a router resource. A resource may contain several capabilities exposing its functionality.

In summary, resources represent physical and virtual devices, as well as networks. Resources use a model to represent their state and expose capabilities, which are vendor neutral interfaces to a given resource functionality. Each resource has a defined lifecycle, which defines its current state, e.g. active and inactive. Only when the resource is active, all its capabilities are initialized and available for its use. The same also applies to the resource model.

4.1 OpenNaaS Architecture

The OpenNaaS system consists of three layers, namely the platform layer, the NaaS layer, and the network intelligence layer, as illustrated in 2.

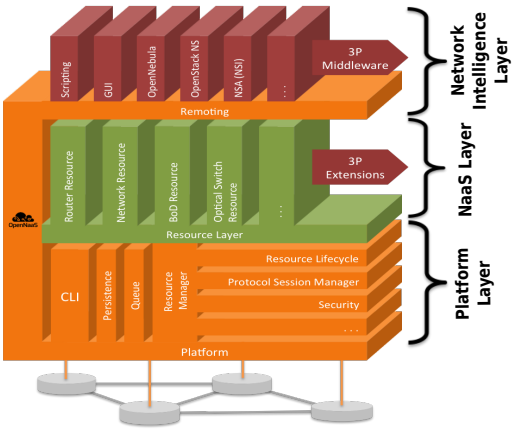


Figure 2: The OpenNaaS framework architecture.

[7]The source code of OpenNaaS is available at https://github.com/dana-i2cat/opennaas/

The Platform Layer. The platform layer, marked in orange in OpenNaaS architecture diagram, is the OpenNaaS technological framework. It is the foundation of the NaaS layer. The platform layer:

- is embeddable and interoperable. It can be a component of a bigger middleware (i.e. a cloud management infrastructure);
- provides reusable concepts across plugins, such as the definition for Resources, Capabilities, Actions, and their lifecycle. A command toolset is built around these concepts;
- handles lifecycle and persistence, it maintains protocol session lifecycle, with an eye on session reusability.

The NaaS Layer. The NaaS layer, highlighted in green, reuses platform defined components to expose user manageable resources. This layer offers support for concrete resource types, functionalities (called capabilities) and device types. Resources and capabilities are exposed to the user abstracted from implementation details and from hardware specifications. Hence, this layer acts as a Hardware Abstraction Layer. Capability implementation for specific devices (called drivers) are also part of this layer.

The Network Intelligence Layer. On top of the abstraction provided by the NaaS layer, applications willing to manage the network may be built. These applications compose the Network Intelligence layer, illustrated in red. It receives this name as it is where managing policies may be defined and actions based on them applied to the network using OpenNaaS. The OpenNaaS architecture is discussed in detail in (man, 2013). From the OpenNaaS perspective, SemNaaS system introduced in this paper is an application belonging to this layer.

5 SemNaaS ARCHITECTURE

The SemNaaS system consists of four components:

1. Request validation and connectivity checking component;
2. Monitoring component;
3. Report generator component;
4. OpenNaaS component, which is a pluggable component, that supports the network resources provisioning.

SemNaaS architecture is illustrated in 3. The OpenNaaS component was discussed in Section 4, the remaining components are discussed in the following subsections.

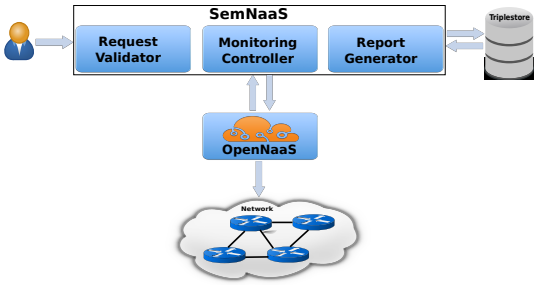


Figure 3: The SemNaaS general system architecture.

Request Generator and Validation. A crucial NaaS function is to provide users with a network on demand with a specified topology. The users can independently use the network to do experiments or execute jobs. In order to acquire a network topology, the user should formulate a request, which can be generated via the SemNaaS user interface. That request contains information about the various network components, e.g. ports and links, that should be created by OpenNaaS and their interconnection together. The NML ontology is used to formulate that request properly (cf. Section 3.2). Example network topology is shown in 4, which is described in NML (in Turtle format) as indicated in 3. SemNaaS enables the user to graphically create her topology, and validate it to detect any errors and/or warnings. 5 indicates the generated request, and the output of its validation. On other hand, SemNaaS also enables the tenant to create his/her required topology graphically, using drag & drop. Via this interface, he/she can set the namespace and component IDs, which will contribute to the generated URIs of various components.
SemNaaS performs two levels of validation, namely

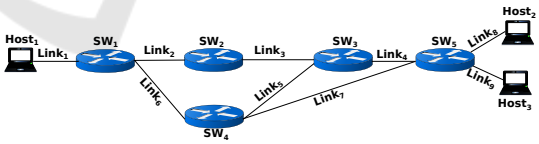


Figure 4: A network topology example.

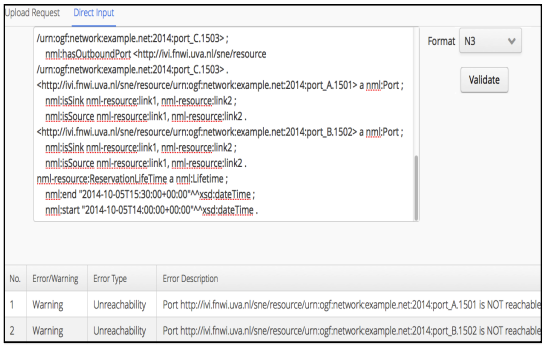


Figure 5: SemNaaS system validating a topology request.

request validation, and connectivity check. The first level ensures that the request is syntactically correct, i.e. no syntactic error exists in the request. Further, SemNaaS uses the Pellet reasoner (Sirin et al., 2007) to validate the request with respect to the NML ontology. In other words, it checks that the request adheres to the NML vocabulary, and no violation exists, e.g. no component is declared as a port and as link at the same time, as those concepts are disjoint.
The second level of validation, is the connectivity validation, in which SemNaaS validates the request and detects if a network node is unreachable from another node. That is how to detect the inaccessibility of some components at early stage, which helps in avoiding that problem when the resources are allocated. We utilize SPARQL (Harris and Seaborne, 2013) queries to detect any unreachability problems among the various nodes of the requested network.

Definition 5 (Connectivity Check)**.** $\forall n_i, n_j \in \mathbb{N}, \exists p^+(n_i, n_j)$ *where* n_i, n_j *are the start node and end nodes respectively,* $\mathbb{N}$ *is the set of all nodes and* p^+ *represents the property path between* n_i *and* n_j*.*

Based on 5, SPARQL query in 1, retrieves each node along with all other nodes reachable from that node. After the request is validated, it is passed to OpenNaaS for provisioning. Moreover, the request is saved in RDF to Virtuoso triplestore (Erling and Mikhailov, 2007) for later use. That helps in report generation and for system utilization analysis, i.e. it helps in tracking OpenNaaS utilization. After performing this two-level validation on the user's request, a list of errors and/or warnings, in case one exits, is returned to the user. An error represents a violation of an NML rule, which prevents the request from fulfillment. Thus the request is neither passed to OpenNaaS, nor stored in the triplestore. In case an unreachability between two or more nodes is detected, it is just returned as a warning to the user in order for him/her to fix.

Monitoring Controller Component. Once the request is sent, OpenNaaS determines whether the requested resources can be granted immediately or the user should wait till those resources can be granted. During normal operation, a resource passes through several states, which represent its current state, e.g. active. However, the resource may experience failure conditions as well, e.g. network connectivity failure. All those states should also be tracked and monitored, thus the system administrator can later analyze and calibrate those problems.
Whenever a change occurs in the resource status, SemNaaS tracks that change. So that the SemNaaS data always reflects the most recent status of OpenNaaS resources. OpenNaaS has a core component called "Resource Manager" via which the user can create, update, and/or delete resource. Furthermore, the user can retrieve and/or change the resource status. The resource manager provides all of its service as RESTful web services, which enables querying those services online. For example, in order to get the current status of a resource you can use http://www.i2cat.net/resources/resourceId/status, and just replace `resourceId` with the local resource ID.
SemNaaS can utilize this interface, to regularly contact OpenNaaS to get the status of each resource reserved in a request. Thus, SemNaaS data always reflects the most up-to-date state of the whole system. This also assists in determining whether there are free resources OpenNaaS has, which can be used to fulfill a new request.
It is worth noting that there is a minimal dependency from SemNaaS on OpenNaaS web services to retrieve the status of various resources. Consequently, if OpenNaaS is replaced by another NaaS provider, it should provide an interface to SemNaaS for resource status retrieval.

```
1 SELECT DISTINCT ?StartNode ?EndNode
2 WHERE { ?StartNode nml:hasOutboundPort ?
  StartPort.
3 ?StartPort (nml:isSource/nml:
  hasSinkNetworkObject)+ ?EndPort.
4 ?EndNode nml:hasInboundPort ?EndPort }
5 ORDER BY ?StartNode ?EndNode
```

Listing 1: SPARQL query to check the connectivity between the various nodes of a sample topology.

```
1 SELECT ?networkResource ?startTime
2 WHERE {?networkResource nml:existsDuring  ?
  lifetime. ?lifetime time:hasBeginning ?
  startTimeInstant. ?startTimeInstant time:
  inXSDDateTime ?startTime.
3 FILTER((?startTime > "2014-10-01T00:00:00Z"^^
  xsd:dateTime) && (NOT EXISTS {?lifetime time:
  hasEnd ?endtime}))}
4 ORDER BY ?startTime
```

Listing 2: SPARQL query to list all active resources reserved after the beginning of October.

Report Generator Component. SemNaaS empowers OpenNaaS, and any other NaaS framework if engaged in place of OpenNaaS, to create complex reports about the whole resource reservation process. Those reports enable the system administrator to identify the problematic resources of OpenNaaS, monitor the resource failure conditions, and identify the lifetime of any resource, i.e. the time from its creation until release.

Definition 6 (Unreleased Resources Check)**.** $\forall$ $r \in \mathbb{R}$, $\exists$ $l \in \mathbb{L}$, *nml: existsDuring(r, l)* $\wedge$

time:hasBeginning$(l,t_s) \wedge \neg$ *time:hasBeginning*(l,t_e) *where* $\mathbb{R}$ *is the network resource set,* $\mathbb{L}$ *is the lifetime set, and* t_s, t_e *are the start and end times respectively.*

```
nml-resource:host1 a nml:Node ;
    nml:existsDuring nml-resource:
ReservationLifeTime ;
    nml:hasInboundPort nml-resource:port_A ;
    nml:hasOutboundPort nml-resource:port_A .
nml-resource:sw1 a nml:Node ;
    nml:existsDuring nml-resource:
ReservationLifeTime ;
    nml:hasInboundPort nml-resource:port_B;
    nml:hasOutboundPort nml-resource:port_B .
nml-resource:sw2 a nml:Node ;
    nml:hasInboundPort nml-resource:port_C ;
    nml:hasOutboundPort nml-resource:port_C .
nml-resource:port_A a nml:Port ;
    nml:isSink nml-resource:link1, nml-
resource:link2 ;
    nml:isSource nml-resource:link1, nml-
resource:link2 .
nml-resource:ReservationLifeTime a nml:
Lifetime ;
    time:hasEnd nml-resource:startTime ;
    time:hasBegining nml-resource:enTime .
nml-resource:startTime a time:Instant ;
     time:inXSDDateTime "2014-10-05T14
:00:00+00:00"^^xsd:dateTime .
nml-resource:enTime a time:Instant ;
    time:inXSDDateTime "2014-10-05T15
:30:00+00:00"^^xsd:dateTime .
```

Listing 3: Example topology expressed in Turtle (prefixes are omitted).

Various network components are connected to `ReservationLifeTime` which is of type `Lifetime`, to indicate the lifetime of those nodes. This represents a reservation of network resources for certain duration. For example, in 3 the reservation is for one and half hours between `14:00:00` and `15:30:00`. Based on this data, a system administrator can view all unreleased network resources which have been reserved after the beginning of October and are still active, in order to release them. Unreleased resources can be checked as in 6, which is translated into SPARQL in 2.

6 SemNaaS FEATURES

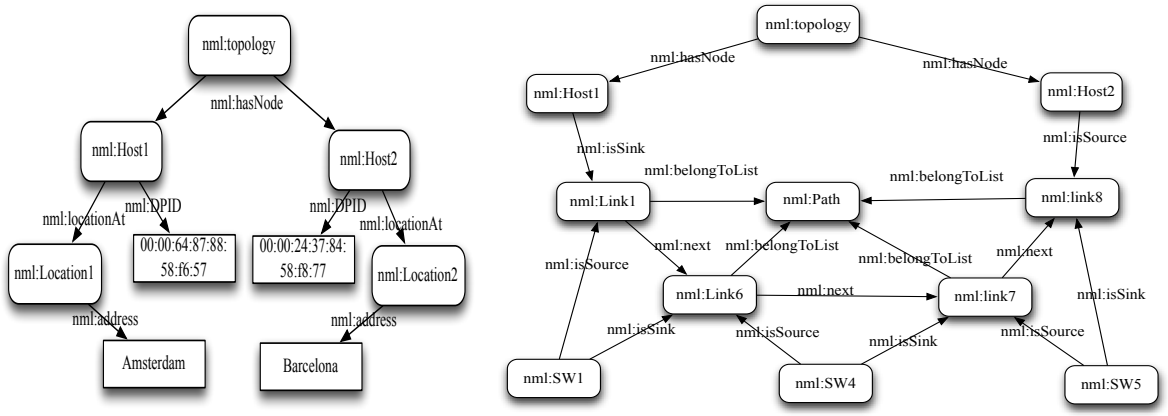
Interconnection of Distributed NaaS Instances. One significant feature that Semantic Web adds to NaaS, is the support of establishing multiple distributed NaaS instances and interconnecting their respective components with ease. One major challenge OpenNaaS has faced before SemNaaS, was how to maintain the uniqueness of the IDs assigned to the various network components, in order to interconnect them whenever required. In other words, each network component, e.g. network node, has an ID that is only unique within the boundaries of the OpenNaaS instance in which it resides. That introduces an obstacle in connecting the components of two remote OpenNaaS instances. Semantic Web depends on using URIs (Uniform Resource Identifier) as an enabling technology. Since we use the NML ontology to describe networks provisioned by OpenNaaS, we should assign a unique URI to each component, which plays the role of a global identifier.
In our case we use `http://ivi.fnwi.uva.nl/sne/` and `http://www.i2cat.net/` as base namespaces for the OpenNaaS instances hosted in the University of Amsterdam and i2CAT respectively. So, for `host1` as an example, the respective URIs will be `http://ivi.fnwi.uva.nl/sne/resource/host1`, and `http://www.i2cat.net/resource/host1`. Thus, those nodes instantiated by distant OpenNaaS instances can be interconnected with no problem.

URIs Minting. As mentioned before, URIs enable interconnecting several NaaS instances together. Therefore, the process of minting those URIs should be handled very carefully. SemNaaS also provides a RESTful web service for creating URIs, which can be used for referencing individual objects in the network OpenNaaS provides.
The generated URI is based on the domain name currently registered for SemNaaS. Thus, if the NaaS provider can provide a *locally* unique identifiers, e.g. link1, the service can generate a URI like http://ivi.fnwi.uva.nl/sne/resource/link1. On the other hand, if the NaaS provider does not provide such an identifier, the service can create the URI from scratch with the help of the current timestamp. Example URI generated is http://ivi.fnwi.uva.nl/sne/resource/1412684412000.

7 EVALUATION AND EXPERIMENT

To demonstrate the advantages of SemNaaS, we discuss an example on how SemNaaS is used to meet the specific challenges in a NaaS use case. The use case shows how SemNaaS improves the functionality of network resource provisioning and path finding of OpenNaaS. This request passes several phases to get fulfilled:

1. The user sends a request with the required details, e.g. source and destination nodes, as illustrated in Figure 6(a). The request is formulated in NML



(a) (b)

Figure 6: An example request (a), and reply (b) in the virtual routing function use case.

and user should use the resource URIs, generated via SemNaaS to refer to various network elements;

2. SemNaaS receives the request, validates it, and uses SPARQL to check the connectivity of the nodes, in order to make sure there is a certain path between those nodes (cf. Section 5);
3. SemNaaS forwards the path finding request to OpenNaaS for fulfillment, as discussed in the Section 7.1;
4. After OpenNaaS calculates the path between nodes, SemNaaS formulates the output response in NML using `Path` class.

7.1 Virtual Routing Function

OpenNaaS interworks several types of OpenFlow controllers to orchestrate network services on top of OpenFlow infrastructures. An OpenFlow controller is an application that manages flow control in a SDN environment.

The Virtual Routing Function use case aims to implement inter-domain routing through OpenNaaS over an OpenFlow infrastructure. VRF finds out a routing path according to the routing mode such as static, Dijkstra, and then invoke the controllers to dynamically create flow tables of the switches on the path.

We developed a prototype to show the VRF functionality of OpenNaaS. The prototype includes a web client GUI, which communicates with the capabilities of OpenNaaS through REST APIs. We emulated a Mininet network with a topology of 5 OpenFlow switches and 3 hosts, shown in 4. The switches had the same network capacity and controlled by Floodlight controllers.

7 presents the sequence diagram of use case in our emulated environment. A provider (SemNaaS in our case) creates a set of abstract OpenFlow resources and loads the OpenFlow capabilities for the resources using OpenNaaS (*Step 1*). After creating the resources, the provider sets the controller information for the network to build the connections between switches and the controllers (*Step 2*). After that, the provider sets the routing mode e.g. static or Dijkstra (*Step 3*). At this point the provider submits a routing request to OpenNaaS for a path between a source and a destination (*Step 4*). OpenNaaS calculates the routing path according to the specific routing mode (*Step 4.1*). Then OpenNaaS interacts with the OFCs to create flow forwarding rules in the switches (*Step 4.2*). The SDN controller finishes the flow configuration by executing the modification of the Route Table of switches. In the end, the unstructured route path information is instantiated by NML and returns to the provider if the flow rules are successfully created (*Step 4.3 and 5*).

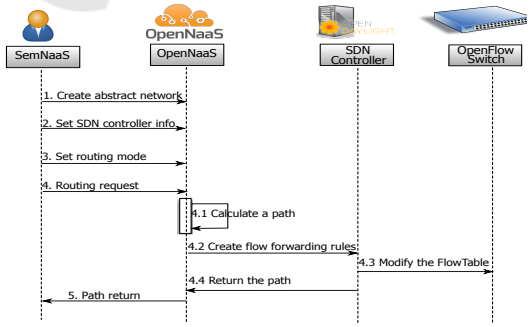


Figure 7: Sequence diagram for VRF functionality using SemNaaS.

Both VRF request and reply require an information model describing resources. SemNaaS can use NML to describe resources in the request and reply. Figure 6(a) shows an example of a possible basic request

in the VRF use case. The request contains a start host and a destination host. The nodes have different properties, e.g. location and IP address. As Figure 6(b) shows, the reply from OpenNaaS components formulated in NML, which is a routing path consisting of a set of links and switches. Besides the description of request and reply, SemNaaS also validates the request for users. It also avoids the VRF use case from receiving wrong request and performing serious behavior.

8 CONCLUSIONS AND FUTURE WORK

In this paper we described SemNaaS, an open source Semantic Web based system for supporting Network as a Service (NaaS). NaaS is a modern Cloud Computing paradigm which gives users easy control over the networking resources. SemNaaS utilizes the Semantic Web to add new features to NaaS systems, e.g. connectivity checking. It fuses Semantic Web with Network as a Service to develop an intelligent NaaS system.
The current version of SemNaaS uses OpenNaaS for network provisioning, a free and open source NaaS system. However, SemNaaS uses OpenNaaS only as a pluggable component, i.e. it can be replaced by any other NaaS provisioning framework. Furthermore, we have also elaborated a use case, to show SemNaaS in action, and demonstrate its various features for NaaS provisioning.
We plan to continue our work on SemNaaS, and extend it in several directions. First, SemNaaS uses URIs for uniquely identifying the network components, which opens up the door for interlinking those components to the Linking Open Data (LOD) cloud (Bizer et al., 2009). This is quite impressive because using the LOD cloud, the metadata of a network resource is enriched dramatically. For instance, for a router resource when linked to the LOD cloud, the specifications registered in SemNaaS will increase significantly. Internet Topology Data Kit (ITDK)[8] is an example potential dataset for interlinking.
Moreover, we are contributing to the OpenMultinet initiative (Willner et al.,), which leverages ontologies for interlinking heterogeneous networks[9]. That enables describing various interconnected computing infrastructures and cloud systems in RDF, thus helping SemNaaS to control and manage heterogeneous networks. It can also support the federation of multiple testbeds, which might be in distant geographical locations.

[8]http://datahub.io/dataset/itdk
[9]http://open-multinet.info

ACKNOWLEDGMENTS

This work was supported by the Dutch national program COMMIT and by funding from the European Community Seventh Framework Programme under grant agreement n. 605243 (GN3plus).

REFERENCES

(2013). Deliverable D4.2 Software Development Report. Project Deliverable Mantychore.

Arenas, M., Conca, S., and Pérez, J. (2012). Counting Beyond a Yottabyte, or How SPARQL 1.1 Property Paths will Prevent Adoption of the Standard. In *Proceedings of the 21st international conference on World Wide Web*, pages 629–638. ACM.

Aznar, J. I., Jara, M., Rosello, A., Wilson, D., and Figuerola, S. (2013). OpenNaaS Based Management Solution for Inter-data Centers Connectivity. In *IEEE 5th International Conference on Cloud Computing Technology and Science, CloudCom 2013, Bristol, United Kingdom, December 2-5, 2013, Volume 2*.

Bizer, C., Heath, T., and Berners-Lee, T. (2009). Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5.

Costa, P., Migliavacca, M., Pietzuch, P., and Wolf, A. L. (2012). Naas: Network-as-a-service in the cloud. In *Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, Hot-ICE*, volume 12.

Dastjerdi, A. V., Tabatabaei, S. G. H., and Buyya, R. (2010). An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), 2010*. IEEE.

Erling, O. and Mikhailov, I. (2007). RDF Support in the Virtuoso DBMS. In *Conference on Social Semantic Web*, volume 113 of *LNI*. GI.

Haak, S. and Grimm, S. (2011). Towards Custom Cloud Services - Using Semantic Technology to Optimize Resource Configuration. In *Proceedings of the 8th Extended Semantic Web Conference, ESWC 2011*.

Haase, P., Mathäß, T., Schmidt, M., Eberhart, A., and Walther, U. (2010). Semantic Technologies for Enterprise Cloud Management. In *International Semantic Web Conference*.

Harris, S. and Seaborne, A. (2013). SPARQL 1.1 query language. *W3C Recommendation*.

Hobbs, J. R. and Pan, F. (2006). Time Ontology in OWL. Working draft.

Junghans, M., Agarwal, S., and Studer, R. (2010). Towards Practical Semantic Web Service Discovery. In *ESWC (2)*, volume 6089 of *Lecture Notes in Computer Science*, pages 15–29. Springer.

Minoves, P., Frendved, O., Peng, B., Mackarel, A., and Wilson, D. (2012). Virtual CPE: Enhancing CPE's

deployment and operations through Virtualization. In *CloudCom*.

Ngan, L. D. and Kanagasabai, R. (2012). OWL-S Based Semantic Cloud Service Broker. In *2012 IEEE 19th International Conference on Web Services, Honolulu, HI, USA, June 24-29, 2012*.

Pérez, J., Arenas, M., and Gutierrez, C. (2006). Semantics and Complexity of SPARQL. In *The Semantic Web-ISWC 2006*, pages 30–43. Springer.

Santana-Pérez, I. and Perez-Hernández, M. S. (2012). A semantic scheduler architecture for federated hybrid clouds. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. (2007). Pellet: A Practical OWL-DL Reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2).

Stollberg, M., Keller, U., Lausen, H., and Heymans, S. (2007). Two-Phase Web Service Discovery Based on Rich Functional Descriptions. In Franconi, E., Kifer, M., and May, W., editors, *Proceedings of the 4th European Semantic Web Conference (ESWC)*, volume 4519 of *Lecture Notes in Computer Science*, pages 99–113. Springer.

van der Ham, J., Dijkstra, F., Łapacz, R., and Zurawski, J. (2013). The Network Markup Language (NML) A Standardized Network Topology Abstraction for Interdomain and Cross-layer Network Applications. In *Proceedings of the 13th Terena Networking Conference*.

Willner, A., Papagianni, C., Giatili, M., Grosso, P., Morsey, M., Al-Hazmi, Y., and Baldin, I. The Open-Multinet Upper Ontology Towards the Semantic-based Management of Federated Infrastructures. In *Proceedings of 10th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TRIDENTCOM2015)*.