Inferring Cloud-Network Slice's Requirements from Non-Structured Service Description

Rafael Pasquini*, Javier Baliosian†‡, Joan Serrat‡, Juan-Luis Gorricho‡, Augusto Neto§¶ and Fábio Verdi‖
* Federal University of Uberlândia (UFU), Uberlândia, Brazil – Email: rafael.pasquini@ufu.br
† Universidad de la República (UdelaR), Montevideo, Uruguay – Email: baliosian@fing.edu.uy
‡ Universitat Politècnica de Catalunya (UPC), Barcelona, Spain – Email: {joan.serrat, juanluis}@entel.upc.edu
§ Federal University of Rio Grande do Norte (UFRN), Natal, Brazil – Email: augusto@dimap.ufrn.br
¶ Instituto de Telecomunicações (IT), Aveiro, Portugal
‖ Federal University of São Carlos (UFSCar), Sorocaba, Brazil – Email: verdi@ufscar.br

***Abstract*—To support future 5G computing and communication scenarios, cloud-network management tools should deploy cloud-network services adopting uncomplicated ways, reducing not only the time to market but also broadening the community capable of deploying new services. In this paper, we present the support of NECOS Platform, an EU-Brazil jointly funded project, towards slice-as-a-service creation from non-structured service description. We describe how NECOS architecture allows such functionality during the slice creation loop, and we present the initial efforts we took for structuring such a mechanism.**

***Index Terms*—Cloud-networks, Natural Language Processing, Cloud-network Slicing.**

I. Introduction

The advances in new technologies should allow ever-growing access to the most recent Internet and cloud services. In order to support future scenarios, such as industry 4.0, the deployment of services should be done in an uncomplicated manner, reducing not only the time to market, but also broadening the community capable of creating such new services.

The NECOS project is devoted to design and implement a cloud-network slice platform to automate via software components the entire slice-as-a-service process across multiple-administrative domains. It defines different slice request possibilities, ranging from a detailed-resources-requirement-description up to an abstract-service-description.

Currently, the community around cloud-network environments is mainly composed of experts, capable of precisely describing what the required resources to offer a given service are. As a next step, such environment needs to strive for less-detail-demanding support, as the abstract-service-description effort present in NECOS. This has the potential to ease the inclusion of other communities to the cloud-network environment, fostering concepts like pay-as-you-go and elasticity.

In this paper, we present the NECOS' work towards slice-as-a-service creation from non-structured service description. The objective is to describe how NECOS includes such possibility in the slice creation loop. Also, we will present the initial efforts taken for structuring such a mechanism. In practical terms, there is a module in NECOS named Slice Specification Processor (SSP), and we present a proposal for its deployment based upon Structured Output Learning [1], a machine learning umbrella to infer dependencies in between arbitrary inputs and outputs. The ultimate goal is to allow a NECOS tenant to place a slice request describing key aspects of its service and, at the end of the process, to receive its slice up and running.

II. Related Work

The work presented in [2] introduces a novel intent-refinement process that uses machine learning and feedback from the operator to translate the operator's utterances into network configurations. As pointed out by the authors, the accuracy of the learning mechanism depends on the universe of different technologies to which natural language shall be mapped. In this way, they propose Nile, an intent definition language close to natural language. It exhibits structural flexibility that works well as the target for the learning algorithm and allows translation to different target networks. By using a user-friendly chat, the user can use natural language to specify requests to its network, for example, *"add firewall and intrusion detection from gateway to backend for client B, with latency less than 10 ms and 100 Mbps of bandwidth, and allow HTTPS only, everyday from 09:00 to 18:00"*.

It is also true that current efforts on translating natural language or at least very high-level descriptions of a networked service to technology-dependant configurations are not the first ones. The idea of "refining" business objectives written in almost natural language existed since the beginning of the policy-based management research. For example, [3] and [4], are related to our proposal. The authors in [3] present an approach to policy refinement that allows the inference of the low-level operations that satisfy a high-level goal by making use of existing techniques in goal-based requirements elaboration and the Event Calculus [5]. They show that the formal representation of a system, can be used together with abductive reasoning techniques [6] to infer a sequence of events that need to occur to reach a desired goal.

III. Overview of NECOS Project

This section briefly introduces the main NECOS Architectural subsystems and modules used during the provision of a new slice to a tenant. It also details the NECOS Information

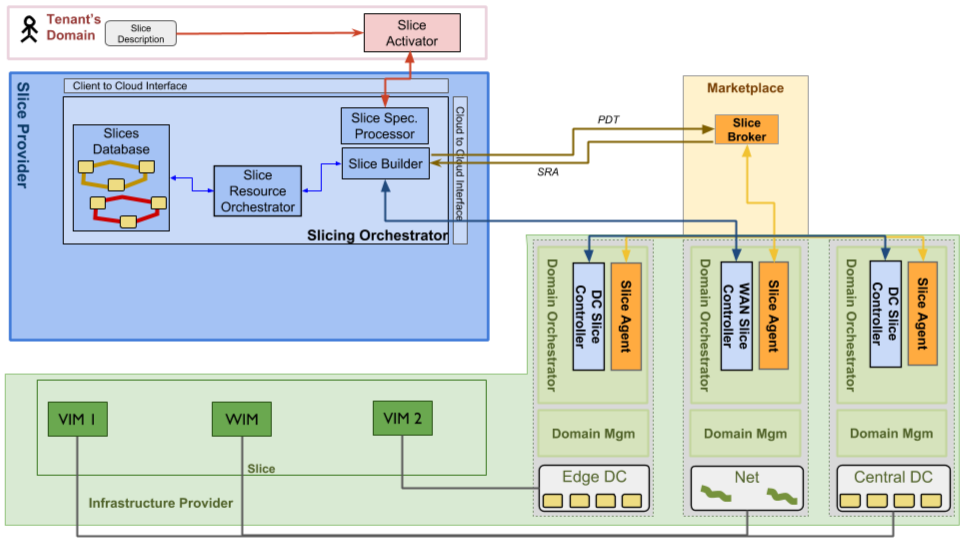

Fig. 1. NECOS Architectural subsystems and modules required to support the slice-as-a-service paradigm.

Model and the APIs (Client-to-Cloud and Cloud-to-Cloud) required to support a slice request.

A. Key NECOS Architectural Subsystems and Modules

The NECOS system architecture is designed to provide slices under a new paradigm defined as slice-as-a-service. Slices are provided across a shared infrastructure of different administrative domains in a federation. The slices are orchestrated by NECOS to fulfill on-demand end-to-end service level agreements (SLAs) [7].

There are four subsystems defined in the NECOS architecture: Tenant's Domain; Slice Provider; Marketplace; and Infrastructure Providers. All these subsystems support the proposed slice-as-a-service paradigm of NECOS, offering on-demand slices to tenants. Figure 1 depicts a subset of the NECOS Architecture, in which the four subsystems along with the modules for slice creation are highlighted. The full architectural description, with all modules can be found in [7]. In the sequence, each subsystem is briefly described:

- **Tenant's Domain:** In order to have a slice, the tenant may provide the description in three formats: (i) fully-described low-level specification of the slice, focusing on resource aspects; (ii) a slice requirements specification; or (iii) a service specification;
- **Slice Provider:** The entrance domain (cloud or network operator) to which the tenant presents its slice request;
- **Marketplace:** It is a subsystem in which all domains, part of a NECOS federation, can advertise the resources they can provide. The service provider (entrance domain) will consult the marketplace to find the options available in the federation to create the requested slices;
- **Infrastructure Providers:** All domains in a given NECOS federation. Such domains offer resources to compose slices under the slice-as-a-service paradigm proposed in the NECOS project. Examples include network operators, core cloud and edge cloud providers.

Figure 1 also depicts key NECOS modules for slice creation. Such components are briefly described as follows:

- **Slice Activator:** Located at Tenant's Domain, it is responsible to inject the slice specification (low-level or high-level description) from the tenant towards the NECOS Slice Provider;
- **Slice Specification Processor:** It admits the slice description at the Slice Provider. The main role of this module relates to translating abstract slice descriptions to actual infrastructure resources. This module delivers a Partially Defined Template (PDT) message to the Slice Builder. This message defines the requirements for the new slice, but does not indicate to which infrastructure providers the resources are associated;
- **Slice Builder:** As the name suggests, it builds the slices. In order to do it, the Slice Builder communicates with the Marketplace using the Cloud-to-Cloud API to gather options for the slice provision. By receiving a Slice Resource Alternatives (SRA) message from the Marketplace, the Slice Builder defines the best arrangement for the new slice;
- **Slice Broker:** Located at the Marketplace, its responsibility is to create the inventory of resources available in the federation by interacting with Data Centers and WAN Slice Agents;
- **Slice Agents:** Located at Infrastructure Providers, they perform internal inventories of resources and expose them in the Marketplace using the NECOS Information Model;

B. NECOS Information Model

The NECOS Information Model provides a way to describe (i) all infrastructure resources with their properties, and (ii) the slice components and service elements that could be deployed within slices. A thorough description of it can be found in [8].

The NECOS information model aims at capturing the main resource and network elements available in datacenter and transport networks. To this aim, main entities represented in the model are: Infrastructure, Domain, Network Element, Router, Switch, Access Point, Host, CPU, Controller, Link, Port, Queue and Path.

Each entity has a set of attributes. As an example, the Host entity has the following properties: Host ID, Hostname, Availability, Location, CPU, Memory, Storage, Number of ports, Monitoring parameters for the host, other service-specific host capabilities (energy-measurement hardware, SAS disks optimized for storage nodes, etc.).

A Fully-Described Low-Level Slice Specification presented by a tenant is generated in conformance with the NECOS Information Model. On the other hand, abstract slice descriptions are mapped to the NECOS Information Model by the Slice Specification Processor module.

IV. Mapping From Abstract Service Descriptions to Slice's Infrastructure Requirements

In this section, we present the problem statement and briefly introduce the background required to support it. Figure 2 illustrates the overall scenario investigated in this paper. As can be seen, the process starts with a tenant providing a service description to our learning mechanism, using abstract (non-infrastructure-specific) information. The learning mechanism implements a function $f(x)$, capable of predicting structured descriptions of infrastructure requirements for new slices.

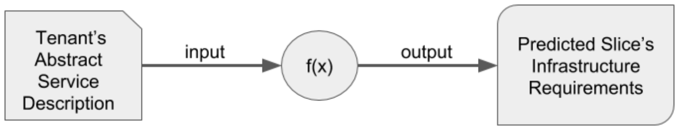
Tenant's Abstract Service Description
input
f(x)
output
Predicted Slice's Infrastructure Requirements

Fig. 2. Learning infrastructure requirements for new slices based upon abstract service descriptions provided by tenants.

In order to tackle this challenge, we leverage the concepts of Structured Output Learning (SOL) [1], a supervised learning umbrella usefull for inferring functional dependencies between arbitrary input and output domains. Rather than predicting discrete elements or real numbers, SOL predicts structured objects that must satisfy rules/constraints specified in its domain. The basic definition of the problem addressed by this work is stated in the following equation:

$$\hat{y} = \underset{y \in Y}{\operatorname{argmax}} f(x, y)$$

Where x is the input information, Y is the set of all possible outputs and $f(x, y)$ is a compatibility function to evaluate how accurate is the fit of y to x. The prediction $\hat{y}$ is the element of set Y, output of the trained model, that better fits x. Therefore, the objective is to find a model $M : x_i \rightarrow \hat{y}_i$, such that $\hat{y}_i$ closely approximates y_i for a given x_i request.

Although Y is finite, it can be very large, making unfeasible the adoption of classifiers and other techniques that exhaustively try different values for y. Common techniques adopted in the literature, and also currently available in tools like PyStruct [9], include energy functions or conditional random fields (CRFs) for building $f(x, y)$ [10].

A. An example of mapping

Suppose that a logistics industry needs to perform asset tracking. With asset tracking an enterprise should be able to easily locate and monitor key assets such as materials, products, or containers, in order to optimize logistics, maintain inventory levels, monitor quality and detect theft.

Although it may be a large tenant, a maritime shipping company, for example, may not want to get into the technological implementation details of a cloud-network slice that supports its assets IoT-based tracking systems. Sensors may help to track the location of a ship at sea, and they can provide the status and temperature of cargo containers. So the shipping company may want to describe its slicing needs as follows:

Provide a cloud-network slice to serve an asset-tracking service. The service must run on the ports of Barcelona, Thessaloniki and Santos. The service must automatically adapt to the demand which will change depending on the time of the year. It is expected to track around 50,000 containers at the same time, with peaks of 80,000 at a frequency of 12 samples per minute. The collected data must be stored inside European Union. Information system's availability should be 99.998%, and data collection availability must be 99.9% as a minimum. Queries to the data, from any city, should have a response time of 100ms or less and will occur at a peak rate of 20 per minute.

To translate those requirements into a structured slice-requirement, i.e., a PDT message, the mapping process has to perform several tasks. They will be described with more detail in the next section, but in a sketchy way they are: i) to identify the service (or services) to be provided, ii) to determine the expected workload, and iii) to identify the constraints.

Each of these tasks may have different sub-tasks, some of them regarding quite different domains. The identification of the service is not a complicated task. Once the service is recognized from the description, it should be matched against known services setups and its particular requirements on CPU, memory, storage, and network must be identified.

Then, the service's workload has to be understood. This case might be more complicated. It is not so easy to verify if the workload characterizing the service relates to the number of shipping containers to track, or the query rate that the database has to manage. In this case, we can say that the workload is mainly given by the number of tracked shipping containers per time unit, but it is not so clear from the point of view of an automatic text processing algorithm.

Finally, service constraints have to be identified: geographical restrictions –in this case the mentioned cities and ports–, service-performance constraints –response time for the queries–, and availability for the data collection and information subsystem. NECOS will, then, create a high-level specification like the one in Figure 4. There, we just depict a fragment of the specification (the complete one is too long for this paper), but it provides an idea of how natural language

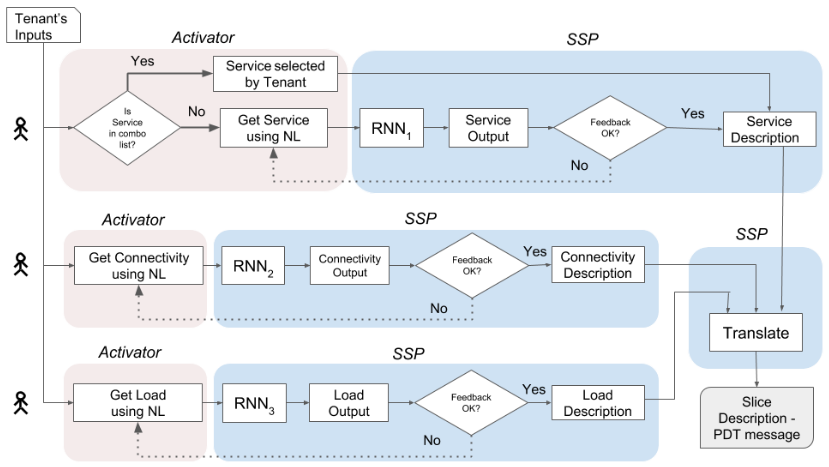

Fig. 3. Example for mapping tenant's inputs into slice descriptions using information forms structured in three fields.

```
...
slice-constraints:
    geographic:
        continent:
        country:
        city: [Barcelona, Thessaloniki, Santos]
        dc-slice-parts: 3
        net-slice-parts: 2
slice-requirements:
    elasticity: true
        target:
            service_metric: query_response_time
            value: {lower_than_equal: 100ms}
    reliability:
        enabled: true
        value: logical-backup
service:
    service-function:
        service-type: dojot
...
```

Fig. 4. Fragment of the high-level YAML-based slice specification.

expressions such as *"Queries to the data, from any city, should have a response time of 100ms or less"*.

V. Exemplification of Usage

In order to allow tenants to provide service descriptions using natural language, our proposal is to adopt Recurrent Neural Networks (RNN). RNNs are a family of neural networks for processing sequential data. Its connections between nodes form a directed graph along a chain. This allows to display a temporary dynamic behavior during a time sequence. Unlike neural feedback networks, RNNs use their internal state to process input sequences. This has made them successful on tasks such as handwriting or voice recognition. RNNs have been successful, for example, in the learning sequence and tree structures in natural language processing. This success makes them an obvious candidate to pursue our goal.

An exemplification on how to adopt RNNs to instantiate our proposal of Figure 2 is depicted in Figure 3. Basically, tenants can provide information using, for example, input forms. In Figure 3, tenants provide information structured in three fields.

The first field receives information, using natural language, that describes the service requested by the tenants. The tenant can also select a service among options in a combo list. The combo selection goes directly to a service description, while textual information passes through a RNN to identify the service requested by the tenant.

The second field receives information that describes all connectivity/geographic aspects of the requested slice. Such information passes through a second RNN, trained to translate such aspects of the request into connectivity description.

The third field receives information that describes load aspects of the requested slice. Such information passes through another RNN, trained, in this case, to translate such aspects into load description.

The treatment of information by RNNs can be seen as a first phase within SSP. The outputs of this phase is then used as a controlled input towards a second phase, in which, for example Structured Output Learning mechanism described earlier is instantiated. SOL translates such controlled input of first SSP phase into PDT messages of NECOS. The first phase can be seen as a way of taming the complexity of such translation.

VI. Conclusions and Future Work

This work presented a proposal for mapping high-level service description to low-level infrastructure resources in NECOS Project. Such mapping is challenging and gained strong attention to attend the raising of new services to be deployed over 5G. The approach introduced in this paper is based on the Structured Output Learning (SOL) and RNNs, that when used together, are capable of dealing with dependencies between arbitrary inputs and outputs. We are instantiating this proposal in the NECOS Project, specifically for implementing the Slice Specification Processor (SSP) module.

Acknowledgements

This research was supported by the H2020 4th EU-BR Collaborative Call (Grant Agreement no. 777067 – Novel Enablers for Cloud Slicing).

References

[1] G. BakIr, T. Hofmann, B. Schölkopf, A. J. Smola, and B. Taskar, *Predicting structured data*. MIT press, 2007.

[2] A. S. Jacobs, R. J. Pfitscher, R. A. Ferreira, and L. Z. Granville, “Refining network intents for self-driving networks,” in *Proceedings of the Afternoon Workshop on Self-Driving Networks*, 2018, pp. 15–21.

[3] A. K. Bandara, E. C. Lupu, J. Moffett, and A. Russo, “A goal-based approach to policy refinement,” in *Proceedings. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, June 2004, pp. 229–239.

[4] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, and G. Pavlou, “A methodological approach toward the refinement problem in policy-based management systems,” *IEEE Communications Magazine*, vol. 44, no. 10, pp. 60–68, Oct 2006.

[5] R. Kowalski and M. Sergot, “A logic-based calculus of events,” *New Generation Computing*, vol. 4, no. 1, pp. 67–95, 1986.

[6] G. Paul, “Approaches to abductive reasoning: an overview,” *Artificial Intelligence Review*, vol. 7, no. 2, pp. 109–152, Apr 1993.

[7] *NECOS Deliverable D3.1: NECOS System Architecture and Platform Specification*, 2018 (accessed December 20, 2018). [Online]. Available: http://www.h2020-necos.eu/documents/deliverables/

[8] *NECOS Deliverable D4.1: Provisional API and Information Model Specification*, 2018 (accessed December 20, 2018). [Online]. Available: http://www.h2020-necos.eu/documents/deliverables/

[9] A. C. Müller and S. Behnke, “pystruct - learning structured prediction in python,” *Journal of Machine Learning Research*, vol. 15, pp. 2055–2060, 2014. [Online]. Available: http://jmlr.org/papers/v15/mueller14a.html

[10] J. Lafferty, A. McCallum, and F. C. Pereira, “Conditional random fields: Probabilistic models for segmenting and labeling sequence data,” 2001.