

Simplification of the design, deployment, and testing of 5G vertical services

Mauro Femminella^{*†}, Matteo Pergolesi[†], and Gianluca Reali^{*†}

^{*}Department of Engineering, University of Perugia, Perugia, Italy

[†]Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Italy

Abstract—5G technology offers innovative network services to vertical players, able to boost their service portfolio in critical areas that could not be fully addressed with 4G systems. However, 5G networks are complex systems, massively relying on the virtualization of network and computing resources, which requires specific skills to fully exploit their potential. This may represent a hard barrier for vertical stakeholders to effectively test and use innovative services on 5G networks. Our solution addresses this issue, by simplifying the way vertical players approach 5G services through simple yet flexible service templates that allow not only to specify and deploy a service, but also to test it in a variety of realistic conditions. Vertical players interact with these templates named Experiment Blueprints, which are manipulated by our tool and then translated into network service descriptors to be deployed by a service orchestrator, keeping unaware the vertical experimenter of the underlying operations. We use a vertical service in the area of Industry 4.0 as a running example to ease the understanding of the proposed approach.

Index Terms—5G, intent, NFV, data models, blueprint

I. INTRODUCTION

The 5G air interface [1] promises to provide broadband services definitely outperforming current cellular service. Due to improved service features, 5G is expected to open to up new applications, from high-end mobile gaming to connected driverless vehicles. With these premises, 5G networks aim at tackling not only consumers, but also a large portion of non ICT industry, commonly known as vertical stakeholders. Up to now, these industries can leverage ICT services only deploying integrated vertical platforms, and relying on expensive and inflexible proprietary solutions. 5G technology will change this scenario significantly. Thanks to the innovations brought by the concepts of virtualization and slicing [2] in 5G networks, vertical users will see a true vertical network dedicated to the provision of their services [3], [4], including the computing part, by leveraging shared resources on public networks.

Motivation: These improvements in performance and flexibility do not come for free. In fact, managing a large amount of shared and virtualized resources requires complex management and orchestration (MANO) procedures, which require specific skills to interact with and to fully exploit their potential. Often, this would represent a hard barrier for verticals not expert in these topics, which could nullify huge opportunities and prevent the development of innovative

services. Thus, a solution to address this issue and simplify the interaction between 5G verticals and 5G service providers could be using techniques in the area of the so-called intent-based networking (IBN) [5]–[10]. We propose an IBN-like solution to simplify the way in which vertical players interact with 5G services. However, most IBN solutions target simplifying network administrators tasks, whereas very few of them try to use intents to simplify the interaction with verticals [9].

Original contribution: We propose a simple yet flexible procedure to build service templates that allows not only to specify a service, but also to test it in a variety of realistic conditions. The core of our proposal is the intent-driven building of a data structure named Experiment Blueprint (ExpB), defined in the framework of the 5G-EVE project [11]. The ExpB acts as a customizable template for an experiment, where the vertical players fill parameters with desired values before the deployment. The resulting model is translated into an ETSI network service descriptor (NSD) [12], [13] to be deployed and then executed by a service orchestrator. We define the tool to manipulate the blueprints, keeping unaware the vertical experimenter of underlying operations. The concept of service blueprint is not completely new in this area. The 5G-Trasformer project has done a first attempt to use it to define a vertical service in a simplified way [14]. However, we significantly advance the blueprint concept proposed in 5G-Transformer. First, our tool automatically compose blueprint and relevant NSD, whereas in [14] the production of blueprint and NSD is completely manual and static, as well as their mapping. Second, in the ExpB we include not only the service definition, but also a number of testing conditions (delay, background traffic, etc) allowing the vertical experimenter to validate the service under realistic settings. Our main innovations are summarized in Table I. To ease the understanding of the proposed approach, we use an Industry 4.0 vertical service use case as running example.

The paper is organized as follows. In section II, we illustrate background and related work. In section III, we illustrate the experiment blueprint data structure and the algorithm used to create and manipulate it. Section IV presents experiments on a real test case. We sketch our conclusion in section V.

II. BACKGROUND AND RELATED WORK

A. ETSI standards

ETSI Network Function Virtualization (NFV) is an initiative to virtualize network functions (NFs) and services

This work has been performed in the framework of the EU project 5G-EVE under grant agreement No. 815974.

Solution	Main functionality	Automated topology composition	Automated NSD generation	Compliance with ETSI NFV	High-Level descriptors	Experimentation support
OSM Composer	Graphical editor for composing custom OSM NSDs	No (manual GUI editing)	Yes	Partial (NSD custom format)	No	No
Cloudify Blueprint Composer	Graphical editor for creating TOSCA-based service descriptors	No (manual GUI editing)	Yes	No	No	No
5G-Transformer	High-level service templates with statically associated NSDs	No (manual definition)	No (static mapping)	Partial (OSM custom format for NSD)	Yes	No
Experiment Blueprint Builder	High-level service templates with dynamic composition and NSD generation	Yes	Yes	Yes	Yes	Yes

TABLE I: Experiment Blueprint builder vs. state of the art solutions. IBN approaches oriented solely to SDN are not reported.

traditionally run on dedicated hardware. NFs are packaged as virtual machines (VMs) and are instantiated on top of commodity hardware, introducing the concept of Virtual Network Functions (VNFs). A chain of multiple VNFs creates a network service (NS). These concepts are the basis to define the concept of a network slice, an enabling feature for 5G. ETSI proposes a complete reference architecture that defines functional blocks and interfaces to describe the NFV framework [15]. In particular, the top layer is the NFV MANO, which is responsible for the orchestration and management of the resources on which NSs are deployed. ETSI has defined information models to describe a VNF and a NS: NFV-IFA 011 [13] defines the Virtual Network Function Descriptor (VNFD), while NFV-IFA014 [13] defines the Network Service Descriptor (NSD). NFV-SOL 001 [16] is an implementation of NFV-IFA 014 using Topology and Orchestration Specification for Cloud Applications (TOSCA) Simple Profile in YAML.

B. NFV Orchestrators

Open Source MANO (OSM) [17] is an ETSI project targeting a production-quality open source stack aligned with ETSI NFV information models. OSM provides a full solution for NFV management and orchestration, covering both design (i.e. DevOps) and runtime phases of NSs lifecycle management. Unfortunately, descriptors supported by OSM are not fully aligned with those standardized by ETSI. OSM provides a GUI-based Composer to build NSDs starting from constituent VNFDs. Specific expertise on NFV is required to use it.

Cloudify [18] is an NFV orchestration project relying on TOSCA standard. It provides a set of design tools such as the Deployment Management UI and the Blueprint Composer for designing blueprints. However, a blueprint in Cloudify is basically a TOSCA-based YAML file that describes the service topology, and, even if visualization is simplified thanks to the Composer, it requires specific skills to build it, equivalent to manipulate an NFV-SOL 001 NSD.

C. Intent-Based Networking (IBN)

The IBN paradigm [8] has been conceived as an enabler for the interaction with network devices via high-level declarative statements (*intents*). Users can express what they desire from the network in a simple and abstract way with intents. While IBN approach is generic, most of proposed solutions are designed for simplifying the interaction with softwarized network solutions (aka SDN) [5]–[7], [10].

In 5G-Transformer [14], the *intent* is a high level, simplified description of the service: the blueprint. It has to be translated to a format, the NSD, understood by the underlying platform. However, this approach lacks about automation in both blueprint definition and translation to NSDs. In addition, NSDs used in [14] are not those standardized by ETSI [16]. We push this approach forward, providing verticals with the necessary level of automation in both definition and translation phases (blueprint and NSD are dynamically composed together), and also with the possibility to define testing conditions, metrics, and key performance indicators (KPIs).

III. EXPERIMENT BLUEPRINT INFORMATION MODEL

Our approach aims offering 5G services to vertical users in an easy way. To achieve this goal, we define high level information models for network services that are easily understandable by non-experienced users and partially customizable: the blueprint, a network service template proposed in 5G-Transformer [14]. It includes parameters to partially customize the service and allowing the vertical user to play with it, and to try it out under different configurations. Finally, the blueprint can be directly mapped to a network slice, specifying the desired Slice Service Type (SST) [19] and the resources to be allocated. We extend the blueprint concept according to the requirements of the 5G-EVE project, which provides a platform for extensive trial and validation of 5G network services, also adding improved automation levels for its creation and manipulation. The information elements are the following:

- Vertical Service Blueprint (VSB): a high-level representation of a network service template. The service is designed at high level by a vertical user, who also provides software artifacts like VMs including service-specific application packages. An experiment developer¹ creates the VSB representing the vertical service by using the tool presented in section III-E.
- Context Blueprint (CB): a high-level representation of context network elements. We define context as the environmental conditions (from a network point-of-view) that the service can experience in a real-world deployment, such as background traffic, packet delay, packet losses,

¹The experiment developer is a third-party actor having the task of interfacing with the VSB/ExpB management tool on the 5G-EVE portal as well as with the vertical customer. He is required to have some more ICT skills than vertical customer, mainly focused on high level service composition, without being an expert of 5G technology.

etc. Their representation as CBs enables the composition with the VSB and their easy re-use with multiple VSBs.

- Experiment Blueprint (ExpB): a high-level representation of an experiment template. It is generated from the composition of a VSB with one or more CBs, and represents the network service under specific operation conditions. The ExpB also includes information about tests to be run, metrics to be monitored, and KPIs to be validated.
- Test Case Blueprint (TCB): a template describing the actions to be performed during the experiment execution.
- Key Performance Indicator (KPI): its formalized description includes metrics involved and objectives to be met.
- Experiment Descriptor (ExpD): a high-level representation of a fully set-up experiment. A vertical experimenter configures an ExpB by assigning values to its parameters and by customizing tests to be performed. The procedure produces an ExpD, an information element containing all the information to execute the experiment and collect the relevant results using the 5G-EVE platform.

The sequence to compose different blueprints up to obtain the experiment descriptor is sketched in Fig. 1.

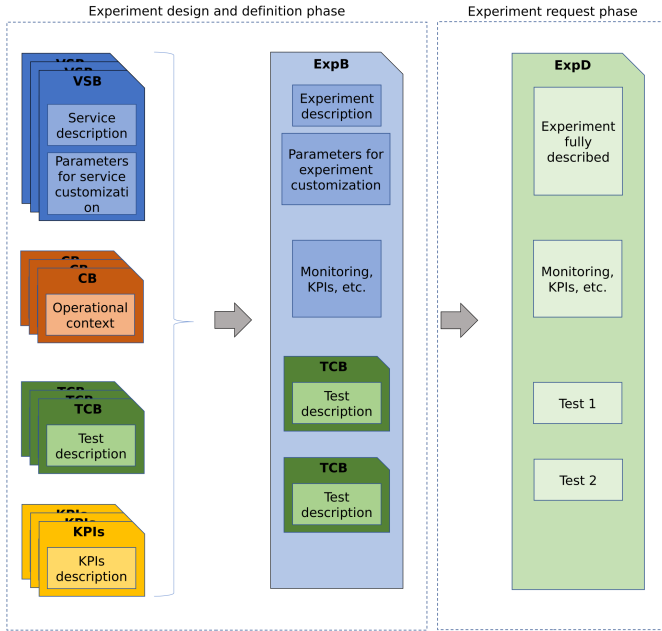


Fig. 1: Relationship between Blueprints and Descriptors.

Blueprints are implemented in YAML [20], a data serialization language easy to understand for both machines and humans, already adopted by ETSI for the implementation of their descriptors [16]. A code snippet of a YAML file representing a VSB is shown in Code 1.

In turn, the high-level information models have to be translated and associated to more detailed entities, i.e. the descriptors defined by ETSI NFV [12], [13], [16]. These descriptors, while containing all the information about the network service, are still technology agnostic and aim to be a generic standard for network orchestration. This makes our

```
1 # vsb
2 ---
3 vsBlueprintId: 'industry_4_vehicles'
4 version: '1.0'
5 name: 'Industry 4.0 autonomous vehicles.'
6 description: 'Blueprint for Industry 4.0'
7 imgUrl: ''
8 parameters:
9   - parameterId: 'number_of_vehicles'
10   [...]
11 atomicComponents:
12   - componentId: 'RAN'
13     serversNumber: 1
14     imagesUrls: []
15     endPointIds: ['SapRANVs', 'CpRANCtrl']
16     lifecycleOperations: {}
17   [...]
18 endPoints:
19   - endPointId: 'SapRANVs'
20     external: true
21     management: false
22     ranConnection: true
23   [...]
24 connectivityServices:
25   - endPointIds: ['CpRanCtrl', 'CpCtrlRan']
26     external: false
27   [...]
```

Code 1: Main sections of a YAML file for a VSB.

approach not tied to any specific NFV orchestrator requiring proprietary descriptors, thus increasing flexibility and applicability to multiple platforms.

A. Vertical Service Blueprint (VSB)

A VSB is a high-level information model acting as a vertical service template. It includes a high-level description of the atomic functional components of the service and their interconnections. The VSB is used to describe in a formal and structured way the service that a vertical wants to test. The VSB is written by the experiment developer, who interacts with the vertical to gather information about the service and relevant experiments to run. The VSB defines a set of parameters that allow customizing the service in some parts. Usually, they are used to correctly size the service in terms of resources. We define translation rules that take one or more parameter values as input and select the appropriate NSD, DF (deployment flavour), and IL (instantiation level) [12].

The VSB can be graphically rendered to visualize the service topology. In this way, the experiment developer can use the graphical rendering to check with the vertical if the blueprint represents the service correctly. The VSB is composed of atomic functional components, end points, and connectivity services. Atomic functional components represent the building blocks of the service. Each atomic component corresponds to a VNF, a PNF, or a nested NS. End points are the anchor points to connect the atomic components. They are listed in the description of the related atomic component but they are also described separately, as they can have specific properties. Finally, connectivity services describe how end points, and thus atomic components, are connected to each other via virtual links.

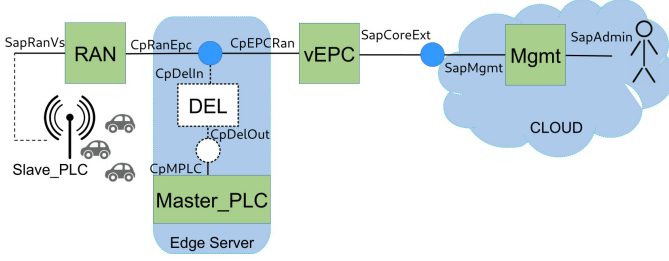


Fig. 2: Representation of a network service for autonomous guided vehicles. The green components are part of the VSB, while the white box is a CB added to create the ExpB.

Fig. 2, white components excluded, shows a graphical representation of a VSB for the control of autonomous guided vehicle in an Industry 4.0 plant. The green boxes are the atomic functional components, the black connectors are the endpoints, and the blue circles are the connectivity services. The vehicles are shown on the left and they can be considered PNFs. The Master_PLC component is a VNF included in an edge computing server to provide low latency commands to the vehicles. The Mgmt component is an interface with management functions to be operated by the vertical user.

B. Context Blueprint (CB)

Similar to a VSB, CBs are templates for network elements describing an operational contexts for the network service. CBs define experimental conditions for the service described in the VSB, like artificial background traffic and delay. The CB can include some parameters to customize its components. They can be generic, like traffic delay, or service specific, e.g. used to emulate the behaviour of multiple users in a service.

A CB is usually consists of only atomic components and their endpoints. The CB must be combined with a VSB as it cannot be deployed by itself.

C. Experiment Blueprint (ExpB)

An ExpB describes a template for an experiment. The ExpB is created from the composition of a VSB and a custom set of CBs, and it includes all their parameters. Furthermore, the ExpB includes information about monitoring, KPIs, and tests to run for their assessment. The parameters allow for some customization of the experiment. The ExpB is compiled by the experiment developer with the support of our semi-automated tool. It takes as input a VSB and a selected set of CBs. Then, it combines the selected elements respecting some constraints and it proposes the resulting topology to the experiment developer. The combination of colored and white components in Fig. 2 is the visualization of an ExpB example. The DEL component is a CB to introduce artificial packet delay on the control node in the edge server.

D. Graph representation of the Network Topology

From the reading of YAML code it is not easy to understand the network topology and the connections between the atomic components. To ease the composition of network experiments,

we programmatically build a graph structure by parsing the blueprint YAML file.

Graphs are well established mathematical and data structures in computer science. They provide an effective way to represent data elements and relationships between them. For these reasons, they appear perfectly suitable as an alternative representation of the network service to focus on the connections between network components. Using graphs also enables the easy visualization of the network topology, as many third party libraries are available for graph drawing.

A graph is defined as a pair $G = (V, E)$, where V is the set of vertices, while E is the set of edges. To represent a blueprint as a graph, we need to map elements described in the textual representation to vertices and edges. Atomic components and virtual links (the elements composing the `serviceConnectivity` field) are mapped onto vertices of the graph. The building of the edge set is trickier: we need to match the endpoint identifiers described by virtual links definition with the ones declared by atomic components. For each matching, we obtain an undirected edge going from an atomic component to a virtual link. We treat external endpoints (service access points, SAPs) as vertices and we add an edge linking the SAP vertex to the relevant virtual link. The result of the procedure is a connected graph.

E. Composition Algorithm

The main goal of the proposed procedure is to allow experiment developers to easily create new experiments from a VSB. By using pre-defined, reusable CBs, the generation of the ExpB can be automated, so avoiding a tedious writing of a YAML file. Furthermore, the automated procedure avoids human mistakes and inconsistent network service definitions.

As described in section III-B, a CB can include one or two atomic components and information on how to connect them to the elements of a VSB. At the moment, we identify two main composition strategies based on the CB type: *connect* to a virtual link and *passthrough*. The first strategy implies that the atomic components of the CB can be composed with the network topology of the VSB by connecting them to an existing virtual link. This strategy does not involve any modification to the VSB network topology. As an example, let us consider a CB to generate artificial background traffic. The CB would be made of two atomic components, a traffic source and a traffic sink. At the composition stage, the source and the sink are connected to two distinct virtual links of the VSB (see also Fig. 3.a). The second strategy (*passthrough*) implies that the atomic components of the CB should intercept, alter, and forward the traffic of the VSB components. So, they are placed between a virtual link and an atomic component of the VSB, causing the modification of the VSB network topology. An example of this kind is a CB to introduce artificial delay in the VSB (see also Fig. 3.b). The atomic component of the CB intercepts network traffic directed to a component of the VSB and delays packets for a customizable amount of time before forwarding them.

The composition algorithm shown in Code 2 handles all the operations to compose a VSB with a list of CBs. It uses the graph representation of the VSB network topology to ensure consistent modifications in the application of the two composition strategies. The algorithm takes as input a VSB and a list of CBs and returns an ExpB as output. From line 4 to line 9 the algorithm builds graph representations of VSB and CB. At line 10, a copy of the VSB graph is created to store the ExpB graph. Then, the algorithm starts cycling on the list of the CB graphs (line 11) and evaluates for each of them the composition strategy (CASE statement at line 12). If the CB specifies the *connect* strategy (line 13), the algorithm cycles on each atomic component of the CB graph and for each of them executes the sequence from lines 15 to 18. Fig. 3.a supports the explanation of the operations. A new vertex (v_c) for the atomic component is added to the ExpB graph and the experiment developer is asked to select a virtual link (v_l) to connect this new vertex to (left picture of Fig. 3.a). A new edge connecting v_c to v_l is added to the ExpB graph. Instead, if the CB specifies the *passthrough* strategy (line 19), the algorithm cycles on each component of the CB graph and for each of them executes the sequence from line 21 to line 29. Fig. 3.b supports the explanation of the operations. A new vertex (v_c) for the atomic component and a new vertex for a new virtual link (v_{lNew}) are added to the ExpB graph. Experiment developer is asked to select an edge (e) where the atomic component should be placed (left part of Fig. 3.b), or this can be specified by predefined rules, which is our choice. The algorithm obtains the atomic component vertex (eVC) and the virtual link vertex eVL connected by e . Three new edges are created as described at lines 26-28 and finally the edge e is removed from the graph. The resulting graph is still a valid network topology (see right part of 3.b). If no valid composition strategy is found, the algorithm raises an exception (lines 31-32). Finally, the algorithm generates an ExpB from the resulting graph and returns it (lines 35-36).

Due to space limitations, we can not go into the details on how to replicate the graph modifications on the ExpB information model. Here we want to emphasize how working with graphs to build network services makes the reasoning and implementation easier. Starting from this, it is possible to map graph modifications to atomic operations for the modification of the ExpB information model. Furthermore,

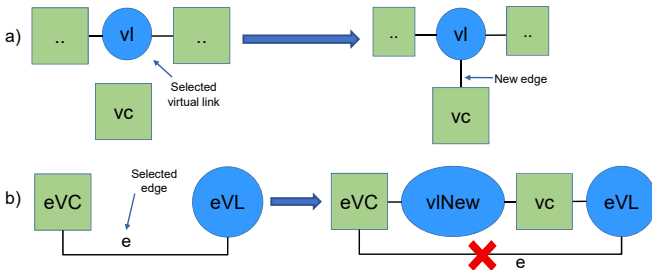


Fig. 3: Strategy to a) connect to a virtual link, and b) to add a passthrough node.

```

1 INPUT VSB, cbList := list of CB
2 OUTPUT ExpB
3
4 vsbG := build graph from VSB
5 cbGList := empty list
6 FOR each CB in cbList
7   cbG := build graph from CB
8   add cbG' to cbGList
9 ENDFOR
10 expbG := copy of vsbG
11 FOR each cbG in cbGList
12   CASE composition strategy OF
13     connect to VL:
14       FOR each atomic component vertex vc in cbG
15         vl := ask user to select a virtual link vertex
16         add vertex vc to expbG
17         add edge to expbG connecting vc and vl
18       ENDFOR
19     passthrough:
20       FOR each atomic component vertex vc
21         e := ask user to select an edge
22         add vertex vc to expbG
23         vlNew := create new virtual link vertex
24         add vertex vlNew to expbG
25         eVC, eVL := get nodes of edge e
26         add edge to expbG connecting eVC and vlNew
27         add edge to expbG connecting vlNew and vc
28         add edge to expbG connecting vc and eVL
29         remove edge e from expbG
30       ENDFOR
31     OTHERS:
32       RAISE Invalid composition strategy exception
33   ENDCASE
34 ENDFOR
35 ExpB := generate blueprint from expbG
36 RETURN ExpB

```

Code 2: Pseudo-code for the algorithm used to compose the ExpB from VSB and CBs.

the same approach can be used also to compose the VSB. In fact, as some specific macro-functions (e.g., the virtual EPC network function) are the result of an often complex chaining of several VNFs/PNFs, they can be provided to the experiment developer as atomic components. At the same time, at the ETSI descriptor level, these complex macro-functions are realized by means of a nested NSD, with external connection point to easily connect it to other NSDs. Atomic components at the ExpB level can be easily mapped to nested NSDs at the lower level, easing and automating also the generation of the final NSD using the same algorithm illustrated in Code 2.

IV. IMPLEMENTATION AND EXPERIMENTATION

We describe the implementation of a Java library to serialize and validate blueprints, as well as a graph implementation to better represent and manipulate the network topology.

A. Blueprint library in Java

We map data elements to Plain Old Java Objects (POJOs). Blueprint YAML files can be easily parsed and serialized to POJOs and deserialized back to textual format. This makes the handling and manipulation of data more predictable (methods to modify data elements are specific and not generic as in

a generic parser) and easy to use (methods have meaningful names). Furthermore, it allows effortless validation of the YAML files: if serialization (with some additional checks) fails, the YAML files is not valid.

The main drawback of using POJOs implies an initial development effort to map all the complex data elements to Java classes. Some interesting tools exist [21], [22] to automatically generate Java source code from YAML files. Anyway, as we needed specific customization in the Java classes, we could not rely on automated but very minimal generated sources. Furthermore, some IDEs (e.g., IntelliJ IDEA [23]) have powerful source generation functions that enormously help this potentially tedious phase of the development.

We developed a Java library for all the blueprints and related entities. Each class defines instance variables and relevant getter and setter methods. For data serialization and deserialization, we use Jackson [24]. It provides Java annotations that can be used on instance variables to govern their inclusion in the generation of YAML files. For basic validation we use Hibernate [25], a certified implementation of the Bean validation 2.0 [26]. As Jackson, also Hibernate provides annotations to specify basic constraints on values that instance variables can assume, e.g. `@NotBlank` for strings or `@NotEmpty` for lists. With this two libraries, we can perform basic validation of blueprints in YAML format in two steps: (i) serialization with Jackson succeeds only if the blueprint schema is respected. Missing or extra fields trigger an error, and (ii) Bean validation with Hibernate succeeds only if there are no violations in the value constraints defined by the POJO annotations. Each step can be implemented as one line of code, avoiding complex custom code to check the correctness of the blueprint data. An easy to use command line validator is also available to support developers writing blueprints.

B. Graph implementation

We use JGraphT [27], a popular and full-featured Java library for working with graphs. It provides developers with utility methods for quickly accessing, modifying, and generating graphs, plus a large catalog of algorithms for graphs manipulation. Furthermore, it allows using any Java object for vertex and edge types, although with some restrictions.

As described in Section III-D, we have three types of vertices in the graph representation of a blueprint. We discuss them with reference to the VSB. We create three different java classes for them: `AtomicComponentVertex`, `VsbLinkVertex`, and `SapVertex`. All these classes have an instance field to refer the relevant entity they represent in the blueprint (e.g., the `AtomicComponentVertex` has a reference to a `AtomicComponent` instance that contains all the relevant data). All the three vertex types implement a Java interface, `VsbVertex`. This is necessary as JGraphT accepts only one vertex type when creating a new graph instance. As we do not need to store any additional data in edges, we use the simple Java `String` type for them.

The class `VsbGraph` is used to create graph instances, using a blueprint instance as input. The class also declares getter and

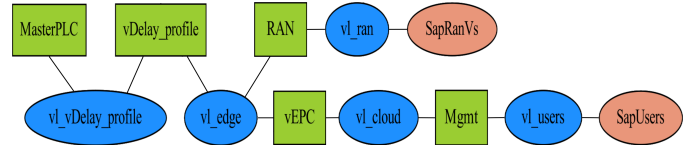


Fig. 4: Graphviz visualization for composed experiment. The `vDelay_profile` is added to the VSB of the considered sample use case, as in Fig. 2.

setter methods for vertices and methods to export the graph into Graphviz [28] format for visualization.

C. Algorithm implementation and example

The algorithm is implemented in a dedicated class called `Composer`. The constructor takes as input a VSB and a list of CBs and instantiates relevant graphs for these objects. A method named `buildExperiment` implements the pseudo code described in Code 2. It handles the composition strategy selection and the user input but it delegates all the operations about adding and removing vertices and edges to the `VsbGraph` class. This conceptually separates the tasks of the two classes and makes methods scope more limited for easier testing.

As an example, we show a prototype visualization of graphs generated by the algorithm and rendered with Graphviz. The composition of green and white components shown in Fig. 2 results in the generation of the graph in Fig. 4. The algorithm applies the *passthrough* composition strategy to include the artificial delay CB into the VSB.

V. CONCLUSION

In this paper, we have presented an intent-based solution for defining experiments for 5G vertical services ready to be deployed and run in a virtualized infrastructure. We have highlighted the main innovations we brought in the 5G research arena, and namely the usage of flexible and automated service template composition able to define not only service components, but also additional modules used to emulate realistic conditions. This is of great importance in order to assess if and under which conditions vertical customer expectations can be met. Finally, we have presented not only the core algorithm used for building the ExpB template, but also a first implementation with some results to illustrate the benefit of our proposal.

The main lesson learned is that, by taking into account some reasonable constraints on placement of specific components, the composition can be carried out smoothly. The possibility of visualizing the obtained service graph allows for a first assessment from the vertical user and/or experiment designer himself, before testing the obtained NSD by means of the deployment through a service orchestrator. Despite this proposal is framed in the 5G-EVE project, it is general enough to be easily applied to other frameworks.

Future work will present not only additional results, but also the workflows describing system operation, from service definition at the blueprint level to deployment/configuration and finally to execution with data collection.

REFERENCES

- [1] 3GPP, “Release description; Release 16 - Draft”, 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 21.916, version 0.1.0. [Online]. Available: <https://www.3gpp.org/release-16>
- [2] A. de la Oliva, X. Li, X. Costa-Perez, C. J. Bernardos, P. Bertin, P. Iovanna, T. Deiss, J. Mangues, A. Mourad, C. Casetti, J. E. Gonzalez, and A. Azcorra, “5G-TRANSFORMER: Slicing and Orchestrating Transport Networks for Industry Verticals”, *IEEE Commun. Mag.*, vol. 56, no. 8, pp. 78–84, Aug. 2018.
- [3] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, D. Aziz, and H. Bakker, “Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks”, *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, May 2017.
- [4] GSMA, “From Vertical Industry Requirements to Network Slice Characteristics”, Tech. Rep., August 2018.
- [5] M. Beshley, A. Pryslupskyi, O. Panchenko, and H. Beshley, “SDN/Cloud Solutions for Intent-Based Networking”, in *2019 3rd International Conference on Advanced Information and Communications Technologies (AICT)*, July 2019, pp. 22–25.
- [6] M. Riftadi and F. Kuipers, “P4I/O: Intent-Based Networking with P4”, in *2019 IEEE Conference on Network Softwarization (NetSoft)*, June 2019, pp. 438–443.
- [7] D. Comer and A. Rastegatnia, “OSDF: An Intent-based Software Defined Network Programming Framework”, in *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, Oct 2018, pp. 527–535.
- [8] A. Elhabbash, G. S. Blair, G. Tyson, and Y. Elkhatib, “Adaptive Service Deployment using In-Network Mediation”, in *2018 14th International Conference on Network and Service Management (CNSM)*, Nov 2018, pp. 170–176.
- [9] Y. Elkhatib, G. Coulson, and G. Tyson, “Charting an intent driven network”, in *2017 13th International Conference on Network and Service Management (CNSM)*, Nov 2017, pp. 1–5.
- [10] ONF, “Intent NBI – Definition and Principles”, Tech. Rep. ONF TR-523, October 2016.
- [11] 5G-EVE project: European 5G validation platform for extensive trials. [Online]. Available: <https://www.5g-eve.eu/>
- [12] *Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; Network Service Templates Specification - ETSI GS NFV-IFA 014 V2.5.1.* European Telecommunications Standards Institute, August 2018.
- [13] *Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; VNF Descriptor and Packaging Specification - ETSI GS NFV-IFA 011 V2.5.1.* European Telecommunications Standards Institute, August 2018.
- [14] C. Casetti, C. F. Chiasserini, T. Deis, P. A. Frangoudis, A. Ksentini, G. Landi, X. Li, N. Molner, and J. Mangues, “Network slices for vertical industries”, in *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. Barcelona: IEEE, Apr. 2018, pp. 254–259.
- [15] *Network Functions Virtualisation (NFV); Management and Orchestration - ETSI GS NFV-MAN 001 V1.1.1.* European Telecommunications Standards Institute, December 2014.
- [16] *Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors based on TOSCA specification - ETSI GS NFV-SOL 001 V2.5.1.* European Telecommunications Standards Institute, December 2018.
- [17] OSM: open source MANO. [Online]. Available: <https://osm.etsi.org/>
- [18] Cloudify: Cutting edge orchestration. [Online]. Available: <https://cloudify.co/>
- [19] T. ETSI, “123 501 V15. 2.0 (2018-06) 5G”, *System Architecture for the 5G System (Release 15)*.
- [20] C. C. Evans, “YAML 1.2”, 2009.
- [21] (2019) jsonschematopojo. [Online]. Available: <http://www.jsonschema2pojo.org/>
- [22] (2019) Project lombok. [Online]. Available: <https://projectlombok.org/>
- [23] (2019) IntelliJ idea. [Online]. Available: <https://www.jetbrains.com/idea/>
- [24] (2019) Jackson project home @github. [Online]. Available: <https://github.com/FasterXML/jackson>
- [25] (2019) Hibernate. [Online]. Available: <https://hibernate.org>
- [26] (2019) Bean validation. [Online]. Available: <https://beanvalidation.org/2.0/>
- [27] D. Michail, J. Kinable, B. Naveh, and J. V. Sichi, “JGraphT-A Java library for graph data structures and algorithms”, *arXiv preprint arXiv:1904.08355*, 2019.
- [28] E. R. Gansner and S. C. North, “An open graph visualization system and its applications to software engineering”, *SOFTWARE - PRACTICE AND EXPERIENCE*, vol. 30, no. 11, pp. 1203–1233, 2000.