unifying cloud
and carrier networks

Deliverable D3.2a

Network Function Forwarding Graph specification (Supplement to D3.2)

Dissemination level	Public
Version	1.0
Due date	
Version date	17.11.2015

This project is co-funded
by the European Union

Document information

Editors and Authors:

Editors: Pontus Sköldström (ACREO)

Contributing Partners and Authors:

ACREO	Pontus Sköldström
DTAG	Mario Kind
EAB	Wolfgang John
EHU	Jokin Garay, Jon Matias
ETH	David Jocha, Róbert Szabó
IMINDS	Wouter Tavernier

Project Coordinator

Dr. András Császár

Ericsson Magyarország Kommunikációs Rendszerek Kft. (ETH) AB

KONYVES KALMAN KORUT 11 B EP

1097 BUDAPEST, HUNGARY

Fax: +36 (1) 437-7467

Email: andras.csaszar@ericsson.com

Project funding

7th Framework Programme

FP7-ICT-2013-11

Collaborative project

Grant Agreement No. 619609

Legal Disclaimer

Table of contents

List of figures

List of Tables

1 Introduction

The main information element of the UNIFY architecture is the Network Function Forwarding Graph (NF-FG), introduced in [D2.2] and refined in [D3.1]. A range of different prototypes have adapted the initial NF-FG definitions, interpreted the requirements, and solved the problems discovered with the initial definition in different ways. This resulted in an evolution into two branches, each focused on solving separate problems for different use-cases and business models. At the time of the submission deadline of D3.2 the main content (service decomposition, scalable orchestration algorithms, scalable and resilient services, traffic steering and forwarding state, and scalable orchestration architectures) was ready and have been timely submitted. However, the discussions surrounding the NF-FG models was still ongoing, with the conclusion that merging these different branches into a single NF-FG data model was not possible without further evaluating the pros and cons of the different solutions. It became increasingly clear that the best way to continue the work was to allow the models to evolve separately in different prototypes and later evaluate how the different issues faced during prototyping were solved in the different NF-FG models. The current conclusion is that both of the definitions are valid.

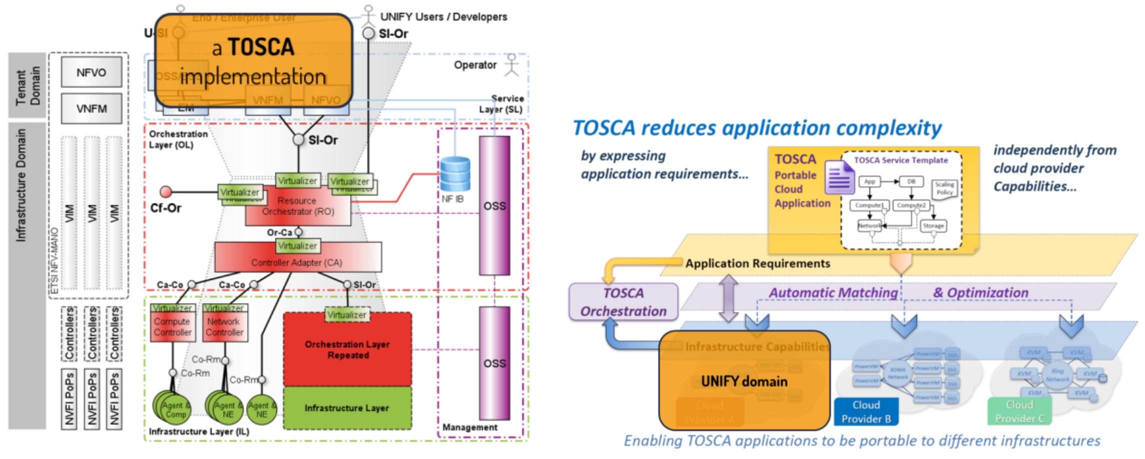
The two NF-FG models are documented in this document which consists of three parts: Section 2 documenting the state of the art outside of the UNIFY project with respect to NF-FG modelling, Section 3 documenting a Service-centric NF-FG model and Section 4 documenting a Virtualizer-based NF-FG model. Finally in Section 5 we evaluate the two different models, draw conclusions with respect to main differences, advantages and disadvantages of either approach as well as potential roadmaps in order to consolidate both.

2 State of the art

2.1 Topology and Orchestration Specification for Cloud Applications (TOSCA)

TOSCA addresses the automation of the application deployment and management lifecycle, in a portable, infrastructure-independent manner [TOSCA]. Compared to the UNIFY NF-FG, the TOSCA specification 1) describes application services, 2) is strong in describing relations between service components, and 3) is weak in networking description, as Layer 2 is not in scope while Layer 3 service components are assumed. The NF-FG models in UNIFY on the other hand focus not on service description but is strong in description forwarding, as Layer 2 bump-in-the-wire NFs are fully in scope.

TOSCA has the approach of substitution of Abstract Node Types by Service Templates, which corresponds to the high level service decomposition approach taken in UNIFY. To match a TOSCA and a UNIFY domain, the whole UNIFY framework could be a "cloud provider" for a TOSCA client. This way UNIFY's advanced resource orchestration acts as a cloud domain, while the advanced service related functionality is realized by the TOSCA implementation, as shown in Figure 2.1.



a TOSCA implementation
UNIFY Users / Developers
Sl-Or
Operator
Service Layer (SL)
Orchestration Layer (OL)
Cf-Or
Resource Orchestrator (RO)
Controller Adapter (CA)
Orchestration Layer Repeated
Infrastructure Layer
Infrastructure Layer (IL)
OSS
Management
NFVO
VNFM
Tenant Domain
Infrastructure Domain
TOSCA reduces application complexity
by expressing application requirements...
TOSCA Portable Cloud Application
TOSCA Service Template
independently from cloud provider Capabilities...
Application Requirements
TOSCA Orchestration
Automatic Matching & Optimization
Infrastructure Capabilities
UNIFY domain
Cloud Provider B
Cloud Provider C
Enabling TOSCA applications to be portable to different infrastructures

Figure 2.1: UNIFY and TOSCA. Left side: TOSCA in the UNIFY view. Right side: UNIFY in the TOSCA view

2.2 OpenStack HEAT

OpenStack's orchestration, HEAT, was designed to automate the configuration and setup of OpenStack resources and is therefore specific to OpenStack [OSheat]. HEAT has a template-driven engine called HEAT Orchestration Template (HOT) which describes and automates the deployment of infrastructure. A guide to HEAT Orchestration Template is available at [OShot], while an introduction to OpenStack can be found in [D3.1].

HEAT has an auto-scaling mechanism; however we see various limitations compared to what we need in UNIFY. First of all, we need transparent Layer 2 NFs, while HEAT assumes Layer 3; additionally the scaling actions in HEAT are limited to start/stop a VM.

Regarding the VM Placement constraints, HEAT doesn't provide more than what Nova (the OpenStack compute resource controller) supports, e.g. networking aspect are not taken into account at VM Placement. Regarding networking, HEAT doesn't support more than what Neutron (the OpenStack network controller) does, e.g. SDN-like forwarding is not supported [Nova, Neutron]. As a consequence, the shortcoming is that HEAT does not support the creation of Layer 2 forwarding rules between VMs. (A workaround could have been setting up L2 over IP tunnelling between VMs. However, only wait conditions are supported between HEAT resources, e.g. VM2 is started after VM1. To configure the tunnel endpoint in VM1, VM2 should already be up and running to get the current networking parameters, e.g. IP address). L2 tunnels are possible in Unify.

We see two possible connection points between UNIFY and HEAT, as shown in Figure 2.2 from both points of view.

In the first scenario (depicted on the left side of Figure 2.2) HEAT could provide a template as input to the Service Layer. The Service layer could then translate the received HEAT Orchestration Template to a UNIFY NF-FG, if a HEAT-capable infrastructure domain is available and advertised as being HEAT capable. However, HEAT is missing many important UNIFY concepts (like transport, L2 networking, multi-domain) and is very OpenStack specific,

In the second scenario (depicted on the right side of Figure 2.2) HEAT could be used to control an OpenStack infrastructure domain under UNIFY. Such an infrastructure domain would be a HEAT capable domain, which could advertise available NFs using HEAT Orchestration Templates. This could be useful because of some abstractions and features of HEAT, e.g. auto-scaling. However, this does not address Neutron's lack of Layer 2 forwarding support.

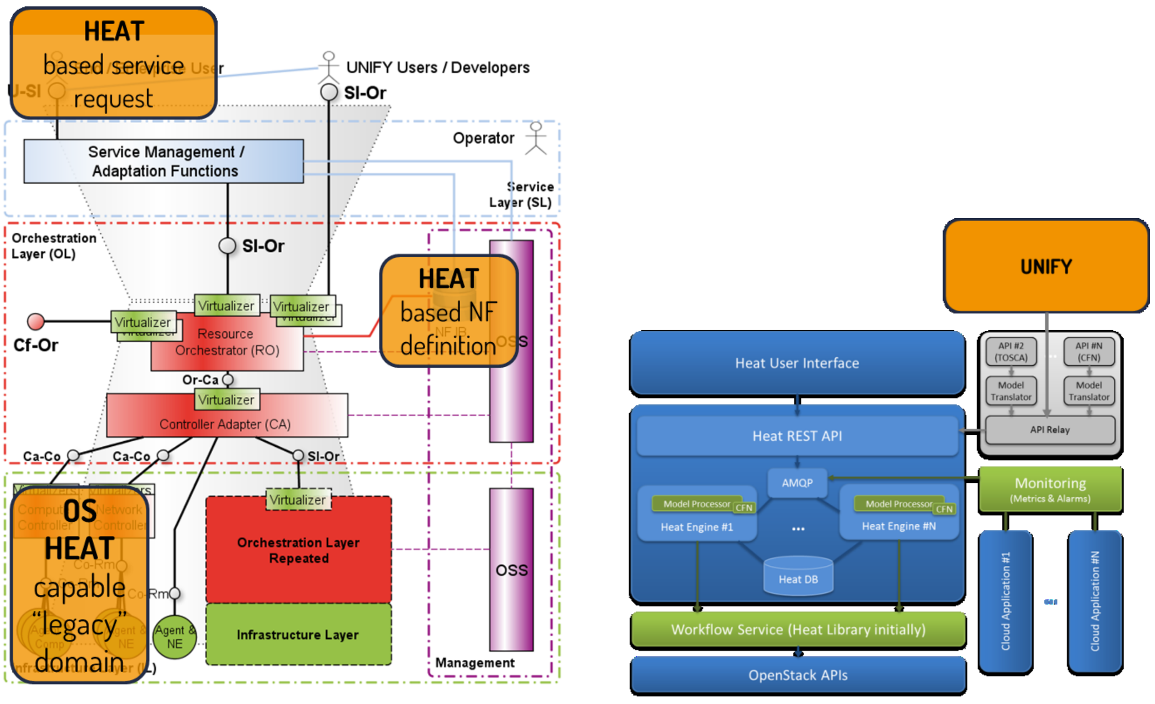


Figure 2.2: UNIFY and OpenStack Heat. Left side: Heat in the UNIFY view. Right side: UNIFY in the Heat view

2.3 ETSI NFV

ETSI NFV defines the Network Service (NS) as "composition of Network Functions and defined by its functional and behavioural specification". The Management and Orchestration framework [ETSIMANO] details the corresponding information element, which is composed by several sub-elements: Virtual Network Functions (VNFs), Physical Network Functions (PNFs), Virtual Links (VLs) interconnecting VNFs to PNFs and endpoints, and VNF Forwarding Graphs (VNFFGs) describing the topology of the NS (either the complete service or part of it). The VNFFG in turn contains Network Forwarding Paths that describe policies. Figure 2.3 depicts the elements included in a Network Service, Monitoring and other SP-DevOps related functionality is scattered through the different information elements in the ETSI NS. The UNIFY service-oriented NF-FG explicitly targets this adding the option to define KPIs and MEASURE controlled functionality in the main elements.

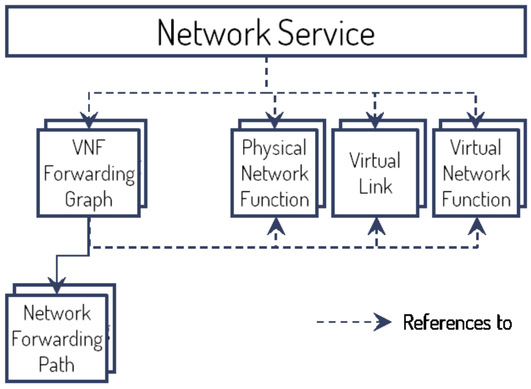


Figure 2.3: Network Service as defined in ETSI NFV MANO

Monitoring and other DevOps related functionality is scattered through the different information elements in the ETSI NS. The UNIFY service-oriented NF-FG explicitly targets this adding the option to define KPIs and MEASURE controlled functionality in the main elements.

Table 2.1 compares them to the UNIFY NF-FG and below the main differences between both models are summarized:

- NS model in ETSI only targets service description, whereas both UNIFY NF-FG models also covers resource description. The UNIFY Virtualizer-based NF-FG model represents deployment decisions to support the requested services (described in Service Graphs).
- Mapping of the service elements to the infrastructure is explicit and per-element in the UNIFY NF-FG. The ETSI model includes resource reservation for the overall NS and the reference of the Virtual Infrastructure Managers that will manage each Virtual Link.
- ETSI NFV differentiates between internal and external virtual links, whereas in the UNIFY NF-FG the same model covers both. Also the connectivity information is spread between different elements, with the VNFFG defining traffic flows and the NFP defining policies. In the UNIFY NF-FG models all the connectivity and traffic flow information is described in either the Service Links for the Service-centric model, or as forwarding information associated with the Infrastructure Nodes in the Virtualizer-based model.
- The service-oriented NF-FG in UNIFY also considers the scenarios with hierarchical orchestration, supporting a progressive refinement of the resource requirements. ETSI NFV is more oriented to a single layer approach with resource requirements described at the lowest level of detail (CPU, PCIe parameters, etc.).
- Monitoring and other DevOps related functionality is scattered through the different information elements in the ETSI NS. The UNIFY service-oriented NF-FG explicitly targets this adding the option to define KPIs and MEASURE controlled functionality in the main elements.

Table 2.1: Elements in the ETSI NFV MANO Network Service and comparison to the UNIFY service-oriented NF-FG

Element	Description	SC-NFFG	VB-NFFG
Network Service Descriptor (NSD)	Deployment template for a NS referencing all other descriptors which describe components that are part of that NS.	Service Graph	Virtualizer
VNF Forwarding Graph Descriptor (VNFFGD)	Deployment template which describes a topology of the NS or a portion of the NS, by referencing VNFs and PNFs and VLs that connect them. Also contains a NFP element.	Service Links with mapping to Infrastructure Links	Forwarding information in Infrastructure Nodes
Virtual Link Descriptor (VLD)	Deployment template which describes the resource requirements that are needed for a link between VNFs, PNFs and endpoints of the NS.	Resource requirements in the Service Links	Resource requirements for infrastructure (or virtual) Links
VNF Descriptor (VNFD)	Deployment template which describes a VNF in terms of its deployment and operational behaviour requirements. Also contains connectivity, interface and KPIs requirements.	Network Functions (single model for all types)	
PNF Descriptor (PNFD)	Describes the connectivity, Interface and KPIs requirements of VLs to an attached PNF.	Network Functions (single model for all types)	
Network Forwarding Path (NFP)	Include policies (e.g., MAC forwarding rules, routing entries, etc.) and references to Connection Points (e.g., virtual ports, virtual NIC addresses, etc.)	Service Links with mapping to Infrastructure Links, Flowrules assigned to Service Access Points and Service Links	Forwarding information in Infrastructure Nodes

3 Service-centric NF-FG model

3.1 Main changes since D3.1 and process overview

Based on the input from the prototypes, the NF-FG definition has been updated from the version described in [D3.1] to provide a joint model capable of covering service description as Service Graph (SG), resource information as Resource Graph (RG) and mapping of requests to resources as NF-FG. This is shown in Figure 3.1, where RGs are represented in green at the right-hand side, SG in blue at the top and the NF-FGs in the different levels with both green and blue elements (link mappings are not represented to improve readability).

The Service Graph (SG) defines the network functions composing the service and their logical connectivity, the access points to the service and the Service Level Specification to meet the Service Level Agreement. Most of the time, the SG will be coupled with an RG, except in the Service Layer, where it is used as an isolated element as there are no resources involved yet. It is composed by:

- Network Functions (NFs): one type of nodes in the SG (same as in the previous version of the NF-FG).
- Service Access Points (SAPs): another type of nodes in the SG that represent a reference point that defines the attachment of the SG to other elements outside in the context of the service (corresponding to endpoints in the previous version of the NF-FG). Examples could be "ACME Company office 1", "All users with the gold service", "Internet", etc.
- Service Links (SLs): edges in the SG (not included in the previous version of the NF-FG).

The Resource Graph (RG) describes the (virtual) resources exposed from the bottom layers that will be used to deploy the requested services. It provides a homogeneous representation of the (virtualized) infrastructure, in terms of both capacities and capabilities, at the defined abstraction level (for additional detail see section 3.4). For example, in domains with hierarchical orchestration processes, the RG in the higher level orchestrators has a wider scope and abstracts away the finer grain details of the underlying resources, whereas the RG in the lower level orchestrators has a fine grain detail of the resources. It is composed of the following elements:

- Infrastructure Nodes (INs): one type of nodes in the RG (corresponding to the Network Elements of the previous version of the NF-FG).
- Endpoints (EPs): another type of nodes in the RG that represent a reference point that defines the attachment of the RG to other elements outside in the context of the infrastructure (not included in the previous version of the NF-FG).
- Infrastructure Links (ILs): edges in the RG (not included in the previous version of the NF-FG).

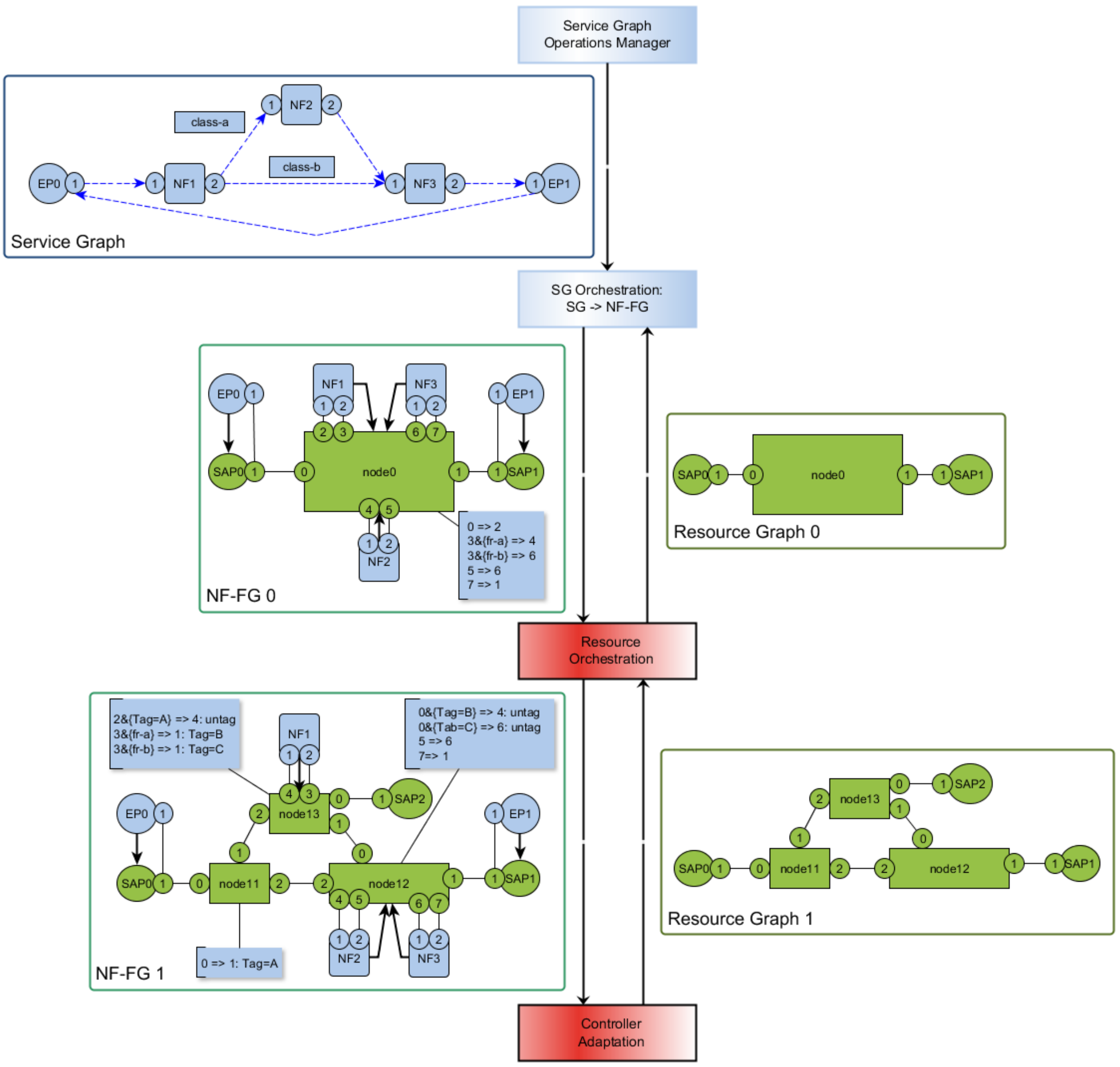


Figure 3.1: Example of Service Graph, Resource Graph and Network Function Forwarding Graph

The origin of the NF-FG is in the Service Layer, from a SG requested by the User Layer above and a RG provided by the Orchestration Layer below, where it will be deployed. The NF-FG extends the information in the SG with the assignment of its elements to the virtualized resources in the RG. The description of the NF-FG is based on the two main elements it is composed of, which follow a modular design aligned with the different process to be supported. As a result, the operations to be performed on the NF-FG as a result of each of those processes can be clearly defined.

The mapping between SG and RG elements is represented under a 'Resource Assignment' section of the SG elements (both nodes and edges) with the following considerations:

- NFs are mapped to Infrastructure Nodes. Several NFs can be mapped to the same Infrastructure Node if it has enough capacity but each NF is mapped to only one Infrastructure Node (N:1 mapping, see D3.2 for some considerations about resiliency). If during the decomposition process a NF is replaced by a group of NFs, each of them would be mapped also to single Infrastructure Node (could be the same for all or different nodes depending on the Virtual Network Embedding (VNE) process, but each resulting NF would be mapped to a single Infrastructure Node). The NFs can also be mapped to running NF instances provided they can be shared among different NF-FGs (i.e. a single running instance realizes a NF in different NF-FGs). This can be signalled either referencing the related NF-FG and NF including the NF instance to be shared or referencing the identifier of the running NF instance if provided by the lower layer. The configuration related to the NF to be shared that must be included in the NF-FG could vary depending on the mechanisms offered by the NF to support the sharing (e.g. the NF could mandate that the traffic corresponding to each NF-FG must reach and leave the NF through different ports).

- SAPs are mapped to one or multiple Endpoints. The mapping could be more straightforward and require a lighter VNE process than for NFs or Service links, such as just picking one element from a list of available Endpoints or even just a direct assignment, such as selecting the Endpoint corresponding to the requesting User. Nevertheless, the separation of the SAP and EP elements allows for a coherent and complete description of the SG and RG and prevents changes in the infrastructure (e.g. adding one more endpoint) impacting the service definition. For example: a SAP can be requested to be mapped to all available endpoints (so the service can be accessed by users connected to any endpoint). In this scenario, the addition of a new endpoint would just be a change in the infrastructure that would be handled by the RO (not having to modify the service to follow infrastructure changes and vice versa). Also, depending on the scenario the SAP to EP mapping could be impacted by the VNE process, for example if the infrastructure where the NF-FG will be deployed on has several possible EPs offering connection to the Internet (as a SAP example) the one selected must be reachable (optimally) from the Infrastructure Nodes the NFs are deployed in.

- Service Links are mapped to an Infrastructure Node internal connection (if both ends of the Service Link are mapped to the same Infrastructure Node), a single Infrastructure Link or a sequence of them forming a path. Several Service Links can be mapped to the same Infrastructure Link if it has enough capacity (N:M mapping) thus sharing it.

Besides the resource requirements at the individual elements, we also consider the possibility to detail networking requirements for the service to be deployed related to its end to end behaviour or between specific subsections of the service.

Figure 3.2 compares the SG to NF-FG mapping described in D2.2, based on the previous version of NF-FG, with the mapping according to the version described in this document. In the D3.2 version, blue dashed arrows represent the Service Links, black thick arrows the mapping of SG to RG elements and the black dashed lines the mapping of Service Links to paths in the RG.

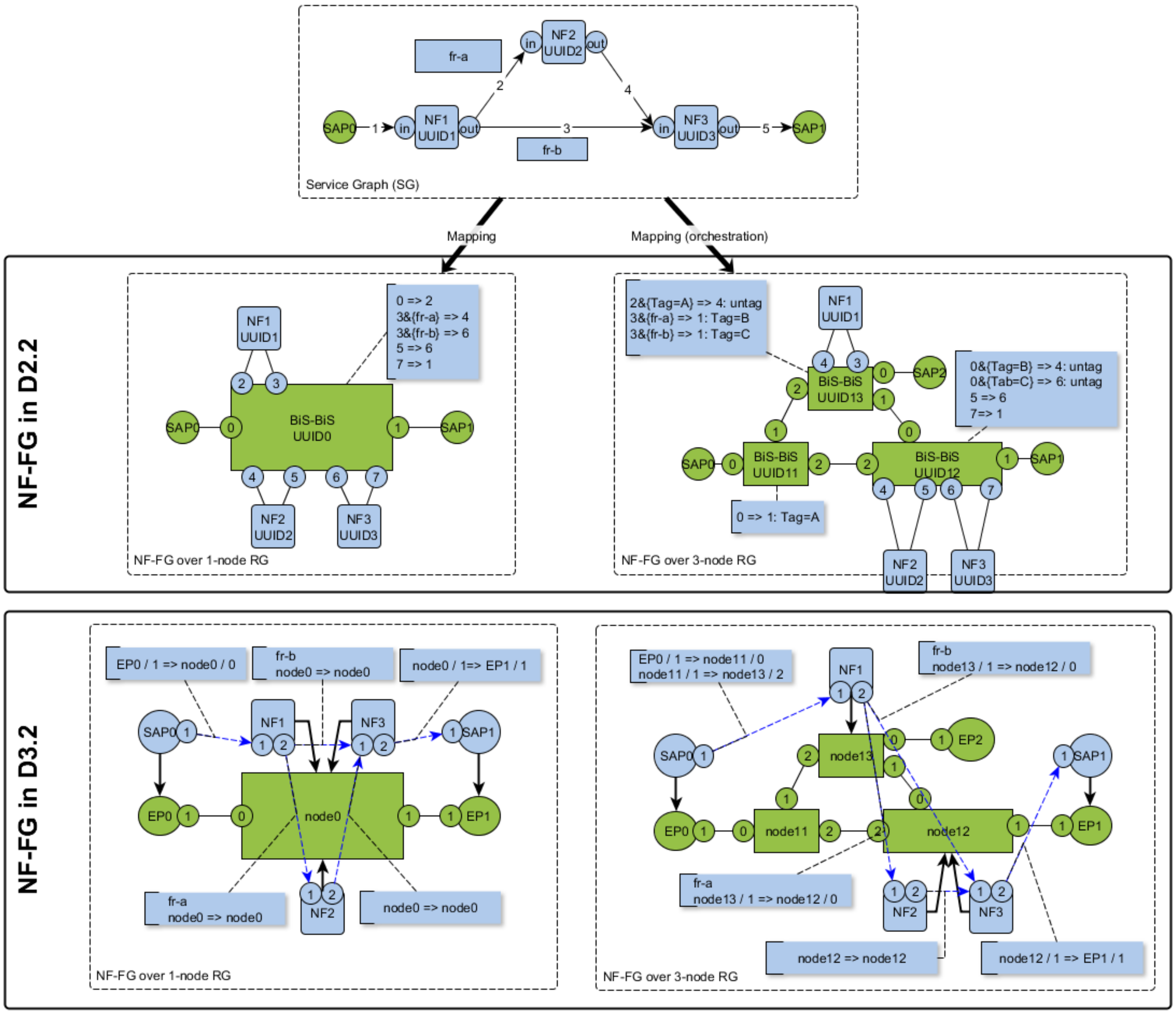


Figure 3.2: Comparison with D2.2 Exemplary mapping of a SG to NF-FGs (figure 6 in D2.2)

3.2 Support for UNIFY main components and features in the NF-FG

Deliverable [D2.2] identified the main components of the UNIFY architecture, as well as its main features and benefits. How the NF-FG relates to each of them is detailed next:

- Virtualizers: the Resource Graph contains the exposed virtualized view and it's explicitly included in the NF-FG, with the description of resource elements classified as nodes, links and endpoints. This structure also eases the translation of the NF-FG from the exposed RG to the underlying RG. In order to optimize the operations and transmissions of the NF-FG, the information of the RG could be substituted by a reference to the corresponding RG or reduced to only the elements used in the mapping.
- Service management and adaptation functions / Service decomposition: the definition of the Service Graph has been aligned with the definition in the NF-FG so a joint model covers both. The clear separation of the service information in the NF-FG supports an easier replacement of NFs by new sub graphs during the decomposition process.
- Resource Orchestrator / Recursive orchestration: the NF-FG now contains an explicit mapping of the elements of the Service Graph to the elements in the Resource Graph. Moreover, the modularity of the resource-related information in the SG elements allows using different levels of abstraction in hierarchical orchestration scenarios, where the resource description and requirements can be refined through the different layers. In these scenarios, the RG exposed to the upper layer would be constructed by the RGs received from the layers below. The explicit declaration of SG, RG and their mapping in the NF-FG allows clearly identifying the scope for the re-orchestration based on the mapping already done by the layer above (and contained in the NF-FG) and relation between the RG exposed to the layer above and the RG received from the layer below.
- Monitoring: the defined model considers the inclusion of KPIs in the SG that get transferred to the NF-FG, and MEASURE language annotations to support the DevOps processes with two different scopes: either related to specific elements (NFs, SAPs or SLs) or related to the whole NF-FG or arbitrary points of it (in which case the reference points must be specified in the definition of the KPI). The KPIs can be expressed as single values, range of values or probability distributions. Examples of KPIs could be: Maximum latency, Maximum jitter, Minimum throughput, Maximum Packet loss, Link protection / Availability, either as strict restriction or probability distribution of type X (e.g. 95% max 10 ms). KPIs transferred to the NF-FG are used during the orchestration process as constraints on the mapping of the NF-FG to the substrate (discussed in section 4 of [D3.2]). The MEASURE NF-FG annotations describe which monitoring functions are needed, how to configure them, and how to react on measurement results (MEASURE was initially described in [D4.1], with more details to follow in D4.2). Additionally, the RG (either as a standalone element or contained in a NF-FG) provides the mechanism for the lower layers to report resource-related information to the upper layers and support performance queries as described in section 3.4. The description of the resources can also include the probability distribution of the KPIs so they can be considered also in the embedding process. The difference between resources that must be allocated and KPIs that must be measured can be considered as a matter of timescale (for the allocation process the resource availability must also be measured, either at the request or periodically to update the availability information).

- Control and data plane split design: the definition of a common resource model for all infrastructure nodes allows for joint treatment of all NF-FG elements, be they related to the Control or Data plane.
- Cf-Or reference point: the NF-FG details which NFs will make use of this interface and which port has to be used for this purpose. The inclusion of the RG presented by the corresponding virtualizer in the NF-FG for the service provides the grounds for requesting self-modifications or new deployments based on the same RG. This simplifies the scaling process as the entity responsible for the scaling decision is aware of the current configuration.
- Scalable and resilient services: The approach defined for resource assignment allows for resiliency to be provided by either the Resource Orchestrator or the Infrastructure layer (or next Orchestration Layer in case of recursion).
    - Orchestrator-based resiliency: the output of the embedding process will determine both the primary resources for deployment of the SG as well as a set of secondary resources. All of them will be detailed in the corresponding 'Resource Assignment' section. The Orchestration Layer will be responsible for deploying the monitoring (e.g. through the MEASURE language described in WP4) and requesting the switch to the secondary resources in case of failure.
    - Infrastructure-based resiliency: the elements included in the Resource Graph offer resiliency and the details are hidden from the Resource Orchestrator (e.g. a single link in the RG represent multiple different links in the Infrastructure Layer so if anyone of them fails, the Infrastructure Layers switches to a backup link without this change being propagated to the Orchestration Layer). The Resource Orchestrator would select those resources offering the capability and signal the layer below that such capacity must be used and to which extent.

Also, the model support the Resource Orchestrator mapping a single Service Link into several Infrastructure Links (or paths) simultaneously to provide multipath links and offer more possibilities for the embedding. In such case, it must also define the flow space corresponding to each of the available paths which will be active at the same time (as opposed to resiliency where only a single link/path would be active).

Statistical multiplexing of different NF-FGs over shared resources can be handled either in or below an Orchestration layer:

- In the layer: In this scenario, the embedding process in the Resource Orchestrator of layer N considers the multiplexing gain to decide if a resource has enough capacity to deploy a NF-FG. Thus, the sum of all the NF-FGs mapped in layer N over the same RG would exceed the nominal capacity of the resources in the RG exposed by layer N-1.

- Below the layer: in this scenario, the Virtualizer in layer N considers the multiplexing gain to determine the capacity to be included in the RGs exposed to layer N+1. Thus, the sum of all the RGs exposed to layer N+1 above would exceed the nominal capacity of the resources in the RG constructed in layer N.

3.3 Updated NF-FG model

The updated Network Function Forwarding has been modelled in YANG and the detailed definition is included in **Annex** 1, as well as an example in JSON which has been selected as interchange format. In order to boost consolidation between the current prototypes, to hide the low level details of the NF-FG structure and to provide isolation to possible future changes to the NF-FG structure, a UNIFY NF-FG library module providing a common set of functionality for NF-FG parsing, interpretation and manipulation is being developed and will be covered in future WP3 deliverables. For the communication between the different modules there are still different potential options being considered (REST, messaging based on ZeroMQ, protocol over TCP/IP). The final decision will be taken aligned with Task 2.3 Integration of prototyping activities in WP2.

The main elements of the NF-FG structure are covered in Table 3.1.

Table 3.1: Details of the Service-centric NF-FG

NF-FG Header	
`+--rw nffg`	

`+--rw parameters`
`|  +--rw id`
`|  +--rw name?`
`|  +--rw version`
`|  +--rw tenant?`
`|  +--rw template?`
	NF-FG header information. All management information pertaining the NF-FG entity would be included here

`|  +--rw constraints`
`|     +--rw resiliency?`
`|     +--rw location?`
`|     +--rw privacy?`
	This section includes possible placement constraints affecting the whole NF-FG related, for example, to privacy (so resources are not shared among NF-FGs or isolation is guaranteed), geography, etc.

`+--rw monitoring`
`|  +--rw monitoring_params*`
`|     +--rw KPI_desc`
`|     +--rw KPI_value`
`|     +--rw scope`
`|     +--rw priority?`
`|     +--rw MEASURE?`
	Monitoring parameters applying to the whole NF-FG (end-to-end) or arbitrary subsections of it (i.e., those not applicable to single sub-elements) as, for example, Service availability, latency between SAPs or between two NFs not directly connected. Each KPI can also contain a priority to signal possible Orchestrators in the layers below which KPI they should optimize in the embedding process (if possible).

SG Information	
`+--rw sg`	
`|  +--rw nfs*`	Information of the NFs composing the service,

`|	+--rw specification` `|		+--rw deployment_type?` `|		+--rw image_uri?` `|		+--rw vnf_type?` `|	+--rw resources` `|		+--rw requirements` `|			+--rw compute` `|				+--rw cpu?` `|				+--rw memory?` `|				+--rw capacity?` `|			+--rw storage` `|				+--rw hdd?` `|			+--rw networking` `|				+--rw delay?` `|				+--rw bandwidth?` `|				+--rw programmability?` `|				+--rw isolation?` `|			+--rw constraints` `|			+--rw resiliency?` `|			+--rw location?` `|			+--rw privacy?` `|		+--rw assignment*` `|		+--rw inf_id` `|		+--rw inf_domain` `|		+--rw nffg_id?` `|		+--rw running_nf_id?` `|	+--rw ports` `|		+--rw ports*` `|		+--rw id` `|		+--rw property*` `|	+--rw nf_monitoring` `|		+--rw monitoring_params*` `|		+--rw KPI_desc` `|		+--rw KPI_value` `|		+--rw scope` `|		+--rw priority?` `|		+--rw MEASURE?` `|	+--rw id` `|	+--rw name?` `|	+--rw functional_type`	including: • Specification and deployment information. • Resource requirements and assignment including assignment to running NFs for sharing. • Ports for connectivity description. • Monitoring information for the NF as a single entity. As with the global KPIs, they can also include a priority and scope (for example, to specify the delay between two ports of the NF).
`	+--rw saps*` `		+--rw ports` `			+--rw ports*` `			+--rw flow_rules` `				+--rw flowrules*` `				+--rw match` `				+--rw action` `			+--rw id` `			+--rw property*` `		+--rw resources` `			+--rw requirements` `				+--rw compute` `					+--rw cpu?` `					+--rw memory?` `					+--rw capacity?` `				+--rw storage` `					+--rw hdd?` `				+--rw networking`	Information of the SAPs composing the service, including: • Resource requirements and assignment. • Ports for connectivity description with associated flow spaces as described in D3.1.

```
       |   |   |   |   |   +--rw delay?
       |   |   |   |   |   +--rw bandwidth?
       |   |   |   |   |   +--rw programmability?
       |   |   |   |   |   +--rw isolation?
       |   |   |   |   +--rw constraints
       |   |   |   |       +--rw resiliency?
       |   |   |   |       +--rw location?
       |   |   |   |       +--rw privacy?
       |   |   |   +--rw assignment*
       |   |   |       +--rw ep_id
       |   |   |       +--rw inf_domain
       |   |   |       +--rw ep_ports*
       |   |   |           +--rw ep_port
       |   |   +--rw id
       |   |   +--rw name?
```

```
       |   +--rw sls*
       |       +--rw resources
       |       |   +--rw requirements
       |       |   |   +--rw delay?
       |       |   |   +--rw bandwidth?
       |       |   +--rw assignment*
       |       |       +--rw source
       |       |       +--rw src_port
       |       |       +--rw target
       |       |       +--rw dst_port
       |       +--rw flowclass?
       |       +--rw source
       |       +--rw src_port
       |       +--rw target
       |       +--rw dst_port
```

Information of the links connecting the NFs and SAPs composing the service, including:

- Connectivity description.
- Resource requirements and assignment considering resiliency. Monitoring information for the link as a single entity.
- Flow spaces for infrastructure supported traffic steering.

RG Information

```
      +--rw rg
```

```
         +--rw infs*
         |   +--rw resources
         |   |   +--rw compute
         |   |   |   +--rw cpu?
         |   |   |   +--rw memory?
         |   |   |   +--rw capacity?
         |   |   +--rw storage
         |   |   |   +--rw hdd?
         |   |   +--rw networking
         |   |   |   +--rw delay?
         |   |   |   +--rw bandwidth?
         |   |   |   +--rw programmability?
         |   |   |   +--rw isolation?
         |   |   +--rw constraints
         |   |       +--rw resiliency?
         |   |       +--rw location?
         |   |       +--rw privacy?
         |   +--rw id
         |   +--rw domain
         |   +--rw name?
         |   +--rw type
         |   +--rw ports*
         |       +--rw id
         |       +--rw property*
```

Information of the Infrastructure Nodes including:

- Resource description.
- Ports for connectivity description.
- Monitoring information on the infrastructure provided by the layer below.

```
+--rw eps*
|  +--rw resources
|  |  +--rw compute
|  |  |  +--rw cpu?
|  |  |  +--rw memory?
|  |  |  +--rw capacity?
|  |  +--rw storage
|  |  |  +--rw hdd?
|  |  +--rw networking
|  |  |  +--rw delay?
|  |  |  +--rw bandwidth?
|  |  |  +--rw programmability?
|  |  |  +--rw isolation?
|  |  +--rw constraints
|  |     +--rw resiliency?
|  |     +--rw location?
|  |     +--rw privacy?
|  +--rw id
|  +--rw domain?
|  +--rw name?
|  +--rw type
|  +--rw ports*
|     +--rw id
|     +--rw property*
```
	Information of the Endpoints including:
	• Resource description.
	• Ports for connectivity description.
	• Monitoring information on the infrastructure provided by the layer below.

```
+--rw ils*
   +--rw resources
   |  +--rw delay?
   |  +--rw bandwidth?
   +--rw source
   +--rw src_port
   +--rw target
   +--rw dst_port
```
	Information of the Infrastructure Links including:
	• Resource description.
	• Monitoring information on the infrastructure provided by the layer below.

3.4 Topology and resource model

The UNIFY approach of describing the topology and resources based on a Resource Graph allows an easier integration of information, both for the top-down processes (the explicit relation between NF-FG and RG makes visible what service elements are deployed where) and the bottom-up (information of the status of the running infrastructure). Moreover, two complementary approaches are supported for the reporting of instantaneous resource information:

- Information related to the complete resource view (e.g. as an updated input for the VNE process with current resource availability) would be included in updates to the Resource Graph sent from the lower layers through the primitives related to resource information defined in [D2.2].
- Information related to the resources assigned to a specific NF-FG would be included in the NF-FG itself and be sent through the primitives related to observability information defined in [D2.2].

Based on the layers of the UNIFY architecture we envision the Resource Graph being subject to several aggregations (composing of RGs of different domains into a single RG) and abstractions (creation of new RGs to be exposed upwards hiding details from RGs received from below, Section 6.7.1 in D3.1 described different alternatives for abstraction) as it progresses up the layers:

- For scenarios without recursion (a single Orchestration layer) the Controller Adaptation would construct an aggregated Resource Graph based on the input of the different Infrastructure domains and present a homogenous view to the Resource Orchestrator hiding the details of the inter-domain connections, which will be reintroduced when an NF-FG must be split to be deployed in several domains. The Virtualizer in the Resource Orchestrator maps the Resource Graph received from the Controller Adapter to the Resource View defined for each consumer of its services (see D2.2, section 3.2.1 for more detail), potentially increasing its abstraction to hide the details of the infrastructure.
- For scenarios with recursion (multiple Orchestration layers), besides those considerations for scenarios without recursion, each Orchestration layer will aggregate the Resource Graphs of the different Orchestration layers below and further abstract the Resource Graph before exposing it to the layer above. In the top-down process, the Orchestration Layer would add again the required functions and features (e.g. the interconnection between different domains).

This approach for resource abstraction allows for two different models for resiliency support, which could be provided either in the upper layer (consumer of the Resource Graph) including in the NF-FG mapping multiple nodes (as primary and backups) or in the lower layer (provider of the Resource Graph) including it in the resource abstraction provided (e.g. nodes representing a cluster of nodes, links representing several disjoint paths, big switch abstraction for a network with multiple paths).

The resource model included in the Resource Graph is built around three main abstractions, compute, networking and storage, described in terms of capacities (which are finite and consumed by the requests) and capabilities (which further characterize the resource and are not consumed by the requests). Examples of capacities are the number of vCPUs an Infrastructure Node can handle or the bandwidth of an Infrastructure Link. Examples of capabilities are redundancy for a link, the delay matrix for a node abstracting a network or the presence of a hardware accelerator for SSL. Support for Hardware NFs, for example, is included in this model based on a capability describing the type of VNFs the Infrastructure Node can handle (aligned with the NF Functional and Deployment Types introduced in the previous version of the NF-FG). This way, a Universal Node would have the capability of running different Deployment Types (VMs, containers, etc.) without a priori restrictions in the Functional Type, whereas a Hardware NF would be restricted to a single Deployment Type (Physical NF) and the corresponding Functional Types. The elements in the Resource Graph are assigned to domains, which group all the elements managed by the same entity from the perspective of the layer handling the NF-FG and determine the splitting to be performed by the CA. For example, for an Orchestration Layer on top of three Universal Nodes, each of them would constitute a different domain, as each of them would manage independently the part of the NF-FG it has to deploy. As an opposite example, if an Orchestration Layer exposed to a higher level Orchestrator a RG with three nodes in it, all of them would belong to the same domain for the higher level Orchestrator, as a single entity would be managing them.

In order to efficiently support the recursive orchestration approach proposed in UNIFY, we envision this description to be different in each level, increasing the abstraction of the resource/infrastructure description in the higher layers, aligned with the service description. In this vision, the two extremes would be a descriptive / quantitative view where the resource description would be oriented to what the resources are (more meaningful in the lower layers, described in example A below) and a functional / qualitative view where the resource description would be oriented to what the resources can do (more meaningful in the upper layers, described in example B below).

- Example A: Resource Graph contains multiple "Virtual" hardware resources such as 4 vCPUs, 100 MB of vMem, 1 GB vStorage. As the RG goes up the layers these resources can be:
    - Completely aggregated considering all resources equal: 4 vCPUs from one node and 4 vCPUs from another domain or node equals 4+4=8 vCPUs in the next layer.
    - Clustered to show the relation between some resources: a node with a quad-core is 4 vCPU, two quad-core nodes are 2x(4 vCPU) (instead of 8 vCPUs).
- Example B: Resource Graph contains functional capabilities with per-function parameters such as "Firewall(100mbit, 100 users)". As the RG goes up the layers these resources can be:
    - Aggregated without considering additional requirements: two nodes each capable two of "Firewall(100mbit, 100 users)" are aggregated to "Firewall(200mbit, 200 users)".
    - Aggregated considering additional requirements: two nodes each capable two of "Firewall(100mbit, 100 users)" requires an additional "Loadbalancer(200 mbit)" to be combined into a "Firewall(200mbit, 200 users)".

In the non-recursive scenario, this change of view is present in the decomposition of the Service Graph, where abstract NFs with SLAs/KQIs (functional view) are transformed into deployable NFs with resource requirements (descriptive view). In recursive scenarios, this change could be realized gradually in the different layers, keeping the resource description from the Resource Graph aligned with the resource requirements in the NF-FG.

3.5 NF-FG examples

3.5.1 Service Graph

The first example corresponds to a Service Graph, as initially requested to the Service Layer, composed of three Network functions and to be provided between two SAPs. The NFs are traversed in the way forward (left to right) but not in the way back (right to left) The output from NF1 is split according to two defined traffic classes and forwarded to either NF2 or NF3.

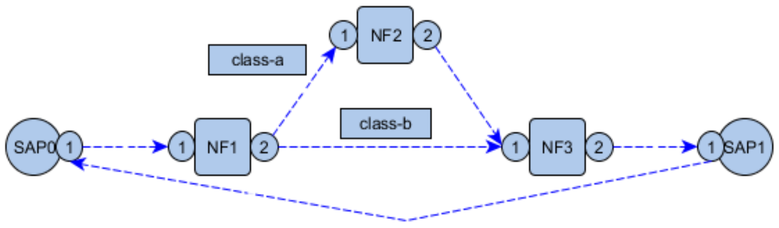


Figure 3.3: Service Graph with 3 Network Functions

Table 3.2: Service Graph

```
{ "sg": {
    "parameters" : { "template": 2,
             "version": "1.0",
             "id": 579,
             "name": "D3.2 Example",
             "tenant": 131124 }
    "sls":  [  { "resources": { "requirements": { "bandwidth": "1G",
                         "delay": "50ms" } },
           "source": 0, "src_port": 1,
           "target": 1,"dst_port": 1 },
           { "resources": { "requirements": { "bandwidth": "1G",
                         "delay": "50ms" } },
           "source": 1, "src_port": 2,
           "target": 2, "dst_port": 1,
           "flowclass": "class-a"  },
           { "resources": { "requirements": { "bandwidth": "1G",
                         "delay": "50ms" } },
           "source": 1, "src_port": 2,
           "target": 3, "dst_port": 1,
           "flowclass": "class-b" },
           { "resources": { "requirements": { "bandwidth": "1G",
                         "delay": "50ms" } },
           "source": 2, "src_port": 2,
           "target": 3, "dst_port": 1 },
           { "resources": { "requirements": { "bandwidth": "1G",
                         "delay": "50ms" } },
           "source": 3, "src_port": 2,
           "target": 4, "dst_port": 2 } ],
    "nfs": [ { "functional_type": 101,
           "id": 1,
           "monitoring": { "KQI_desc": "Numer of Users",
                   "KQI_value": 100 },
           "name": "NF1",
           "ports": [ { "id": 1, "property": "In port" },
                { "id": 2, "property": "Out port" }],
           "resources": { "requirements": {
                     "compute" : { "capacity": "medium",
                           "cpu": "1",
                           "memory": "512M" },
                     "networking" : { "bandwidth": "1G" },
                     "storage" : { "hdd": "10G" } } },
           "specification": { "deployment_type": 2201,
                    "image_uri": "/opt/elwud/2201.json",
                    "max_KQI_value": 250,
                    "vnf_type": 2 } },
         { "functional_type": 102,
           "id": 2,
           "monitoring": { "KQI_desc": "Number of connections",
                   "KQI_value": 10000 },
           "name": "NF2",
           "ports": [ { "id": 1, "property": "In port" },
                { "id": 2, "property": "Out port" }],
           "resources": { "requirements": {
                     "compute" : { "capacity": "medium",
                           "cpu": "1",
                           "memory": "512M" },
                     "networking" : { "bandwidth": "1G" },
                     "storage" : { "hdd": "10G" } } },
           "specification": { "deployment_type": 2202,
                    "image_uri": "/opt/elwud/2202.json",
                    "max_KQI_value": 10000,
                    "vnf_type": 2 } },
         { "functional_type": 103,
           "id": 3,
           "location": null,
           "monitoring": { "KQI_desc": "Number of connections",
                   "KQI_value": 10000 },
           "name": "NF3",
           "ports": [ { "id": 1, "property": "In port" },
                 { "id": 2, "property": "Out port" } ],
           "resources": { "requirements": {
                     "compute" : { "capacity": "medium",
                           "cpu": "1",
                           "memory": "512M" },
                     "networking" : { "bandwidth": "1G" },
                     "storage" : { "hdd": "10G" } } },
          "specification": { "deployment_type": 2203,
                   "image_uri": "/opt/elwud/2203.json",
                   "max_KQI_value": 10000,
                    "vnf_type": 2 } } ],
    "saps": [ { "id": 0,
          "name": "SAP0",
          "ports": [ { "id": 1, "property": "Inbound SAP" } ] },
        { "id": 4,
          "name": "SAP1",
          "ports": [ { "id": 1, "property": "Outbound SAP" } ] } ]
}  }
```

3.5.2 Single node infrastructure

The second example corresponds to a Resource Graph, composed of a single infrastructure node and two endpoints, all of them belonging to the same domain (domain_0).

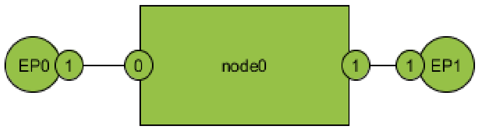


Figure 3.4: Resource Graph with a Single Node

Table 3.3: Single node Resource Graph

```
{ "rg": {
    "parameters" : { "id": 8,
             "name": "UNIFY 1 node",
             "version": "1.0" },
    "ils": [ { "resources": { "bandwidth": "1G",
                   "delay": "50ms" },
           "source": 0, "src_port": "1",
           "target": 1, "dst_port": "0" },
         { "resources": { "bandwidth": "1G",
                   "delay": "50ms" },
           "source": 1, "src_port": "1",
           "target": 2, "dst_port": "1" } ],
    "infs": [ { "domain": "domain_0",
          "id": 1,
          "name": "node0",
          "ports": [ { "id": "0", "property": "abstract" },
                 { "id": "1", "property": "abstract" } ],
          "resources": { "compute": { "capacity": "high",
                         "cpu": 60 },
                    "networking": { "bandwidth": "10G",
                            "delay": "50ms",
                            "isolation": "yes",
                            "programmability": "yes" } },
          "type": "BiS-BiS" } ]
    "eps": [ { "domain": "domain_0",
           "id": 0,
           "name": "ep0",
           "ports": [ { "id": "1", "property": "sap" } ],
           "type": "endpoint" },
         { "domain": "domain_0",
           "id": 2,
           "name": "ep1",
           "ports": [ { "id": "1", "property": "internet" } ],
           "type": "endpoint" } ]
}
```

3.5.3 Three node infrastructure

The third example corresponds to a Resource Graph, composed of three infrastructure nodes and three endpoints. One of the infrastructure nodes and endpoints (node 11 and EP0 respectively) belong to one domain (domain_a), whereas the rest of infrastructure nodes and endpoints belong to a different domain (domain_b).

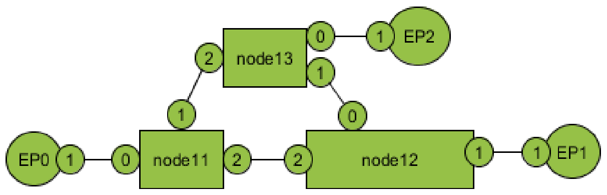


Figure 3.5: Resource Graph with 3 nodes

Table 3.4: Three node Resource Graph

```
{ "rg": {
    "parameters" : { "id": 9,
             "name": "UNIFY 3 node",
             "version": "1.0" },
    "ils": [ { "resources": { "bandwidth": "1G",
                  "delay": "50ms" },
           "source": 0, "src_port": "1",
           "target": 1, "dst_port": "0" },
         { "resources": { "bandwidth": "1G",
                  "delay": "50ms" },
           "source": 1, "src_port": "2",
           "target": 2, "dst_port": "2" },
       { "resources": { "bandwidth": "1G",
               "delay": "50ms" },
           "source": 1, "src_port": "1",
           "target": 3, "dst_port": "2" },
       { "resources": { "bandwidth": "1G",
               "delay": "50ms" },
           "source": 2, "src_port": "1",
           "target": 3, "dst_port": "0" },
       { "resources": { "bandwidth": "1G",
               "delay": "50ms" },
           "source": 2, "src_port": "1",
           "target": 4, "dst_port": "1" },
       { "resources": { "bandwidth": "1G",
               "delay": "50ms" },
           "source": 3, "src_port": "0",
           "target": 5, "dst_port": "1"} ],
    "infs": [ { "domain": "domain_a",
          "id": 1,
          "name": "node11",
          "ports": [ { "id": "0", "property": "abstract" },
                 { "id": "1", "property": "abstract" },
                 { "id": "2", "property": "abstract" } ],
          "resources": { "compute": { "capacity": "high",
                    "cpu": 20 },
                   "networking": { "bandwidth": "10G",
                           "delay": "50ms",
                           "isolation": "yes",
                           "programmability": "yes" } },
          "type": "BiS-BiS" },
          { "domain": "domain_b",
          "id": 2,
          "name": "node12",
          "ports": [ { "id": "0", "property": "abstract" },
                 { "id": "1", "property": "abstract" },
                 { "id": "2", "property": "abstract" } ],
          "resources": { "compute": { "capacity": "high",
                        "cpu": 20 },
                   "networking": { "bandwidth": "10G",
                           "delay": "50ms",
                           "isolation": "yes",
                           "programmability": "yes" } },
          "type": "BiS-BiS" },
          { "domain": "domain_b",
          "id": 3,
          "name": "node13",
          "ports": [ { "id": "0", "property": "abstract" },
                 { "id": "1", "property": "abstract" },
                 { "id": "2", "property": "abstract" } ],
          "resources": { "compute": { "capacity": "high",
                        "cpu": 20 },
                   "networking": { "bandwidth": "10G",
                           "delay": "50ms",
                           "isolation": "yes",
                           "programmability": "yes" } },
          "type": "BiS-BiS" } ]
    "eps": [ { "domain": "domain_a",
           "id": 0,
           "name": "ep0",
           "ports": [ { "id": "1", "property": "sap" } ],
           "type": "endpoint" },
         { "domain": "domain_b",
           "id": 4,
           "name": "ep1",
           "ports": [ { "id": "1", "property": "internet" } ],
           "type": "endpoint" },
         { "domain": "domain_b",
           "id": 5,
           "name": "ep2",
           "ports": [ { "id": "1", "property": "internet" } ],
           "type": "endpoint" } ]
}
```

3.5.4 Request over single node infrastructure

The fourth example corresponds to a NF-FG, resulting from the mapping of the example Service Graph over the single node Resource Graph. All examples consider a single link between RG nodes, so mapping of SLs to RG is represented as the sequence of RG nodes to be traversed. To support multiple links between RG nodes, the mapping would be expressed as a sequence of pairs {source node:source port, destination node:destination:port} thus fully identifying the IL to use (or triplets {source node, destination node, key} if a key is defined to identify each IL connecting the same pair of nodes).

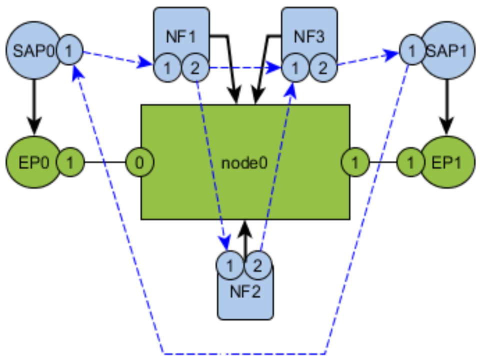


Figure 3.6: Request of NF-FG with 3NFs on a Resource Graph with a single node

Table 3.5: NF-FG with 3 NFs over single node Resource Graph

```
{ "nffg":
  { "sg": {
      "parameters" : { "template": 2,
               "version": "1.0",
               "id": 578,
               "name": "D3.2 Example",
               "tenant": 131124 }
      "sls":  [  { "resources": { "requirements": { "bandwidth": "1G",
                            "delay": "50ms" } },
                   "assignment": [ [ { "domain": "domain_0", "node": 0 },
                            { "domain": "domain_0", "node": 1 } ] ] },
             "source": 0, "src_port": 1,
             "target": 1,"dst_port": 1 },
             { "resources": { "requirements": { "bandwidth": "1G",
                            "delay": "50ms" } },
                   "assignment": [ [ { "domain": "domain_0", "node": 1 },
                            { "domain": "domain_0", "node": 1 } ] ] },
             "source": 1, "src_port": 2,
             "target": 2, "dst_port": 1 },
             { "resources": { "requirements": { "bandwidth": "1G",
                            "delay": "50ms" },
                   "assignment": [ [ { "domain": "domain_0", "node": 1 },
                            { "domain": "domain_0", "node": 1 } ] ] },
             "source": 1, "src_port": 2,
             "target": 3, "dst_port": 1},
             { "resources": { "requirements": { "bandwidth": "1G",
                            "delay": "50ms" },
                   "assignment": [ [ { "domain": "domain_0", "node": 1 },
                            { "domain": "domain_0", "node": 1 } ] ] },
             "source": 2, "src_port": 2,
             "target": 3, "dst_port": 1 },
             { "resources": { "requirements": { "bandwidth": "1G",
                            "delay": "50ms" },
                   "assignment": [ [ { "domain": "domain_0", "node": 1 },
                            { "domain": "domain_0", "node": 2 } ] ] },
             "source": 3, "src_port": 2,
             "target": 4, "dst_port": 1 },
             { "resources": { "requirements": { "bandwidth": "1G",
                            "delay": "50ms" },
                   "assignment": [ [ { "domain": "domain_0", "node": 2 },
                            { "domain": "domain_0", "node": 1 },
                            { "domain": "domain_0", "node": 0 } ] ] },
             "source": 4, "src_port": 1,
             "target": 0, "dst_port": 1 } ],
      "nfs": [ { "functional_type": 101,
             "id": 1,
             "monitoring": { "KQI_desc": "Numer of Users",
                     "KQI_value": 100 },
             "name": "NF1",
             "ports": [ { "id": 1, "property": "In port" },
                  { "id": 2, "property": "Out port" }],
             "resources": { "requirements": {
                       "compute" : { "capacity": "medium",
                             "cpu": "1",
                             "memory": "512M" },
                       "networking" : { "bandwidth": [
                                             {"port": 1, bandwidth: "1G" },
                                             {"port": 2, bandwidth: "1G" }]} ,
                       "storage" : { "hdd": "10G" } },
                   "assignment": [ { "domain": "domain_0",
                           "node": 1 } ] },
             "specification": { "deployment_type": 2201,
                   "image_uri": "/opt/elwud/2201.json",
                   "max_KQI_value": 250,
                   "vnf_type": 2 } },
        { "functional_type": 102,
             "id": 2,
             "monitoring": { "KQI_desc": "Number of connections",
                     "KQI_value": 10000 },
             "name": "NF2",
             "ports": [ { "id": 1, "property": "In port" },
                  { "id": 2, "property": "Out port" }],
             "resources": { "requirements": {
                       "compute" : { "capacity": "medium",
                             "cpu": "1",
                             "memory": "512M" },
                       "networking" : { "bandwidth": "1G" },
                       "storage" : { "hdd": "10G" } },
                   "assignment": [ { "domain": "domain_0",
                           "node": 1 } ] },
             "specification": { "deployment_type": 2202,
                   "image_uri": "/opt/elwud/2202.json",
                   "max_KQI_value": 10000,
                   "vnf_type": 2 } },
        { "functional_type": 103,
             "id": 3,
             "location": null,
             "monitoring": { "KQI_desc": "Number of connections",
                     "KQI_value": 10000 },
             "name": "NF3",
             "ports": [ { "id": 1, "property": "In port" },
                   { "id": 2, "property": "Out port" } ],
             "resources": { "requirements": {
                       "compute" : { "capacity": "medium",
                             "cpu": "1",
                             "memory": "512M" },
                       "networking" : { "bandwidth": "1G" },
                       "storage" : { "hdd": "10G" } },
                   "assignment": [ { "domain": "domain_0",
                           "node": 1 } ] },
           "specification": { "deployment_type": 2203,
                     "image_uri": "/opt/elwud/2203.json",
                     "max_KQI_value": 10000,
                   "vnf_type": 2 } } ],
   "saps": [ { "id": 0,
           "name": "SAP0",
           "ports": [ { "id": 1, "property": "Inbound SAP" } ],
           "resources": { "assignment": [ { "domain": "domain_0",
                             "endpoint": 0,
                             "port": "1" } ] } },
           { "id": 4,
           "name": "SAP1",
           "ports": [ { "id": 1, "property": "Outbound SAP" } ],
           "resources": { "assignment": [ { "domain": "domain_0",
                             "endpoint": 2,
                             "port": "1" } ] } } ]
  }  }
  { "rg": {
   "parameters" : { "id": 8,
            "name": "UNIFY 1 node",
            "version": "1.0" },
   "ils": [ ... ],
   "infs": [ ... ]
   "eps": [ ... ]
  }
}
```

3.5.5 Request over three node infrastructure

The fifth example corresponds to another NF-FG, resulting from the mapping of the example Service Graph over the three node Resource Graph. This could be the result of mapping the SG over a different Infrastructure domain, or a subsequent mapping of the previous NF-FG in a hierarchical orchestration scenario. As the Resource Graph contains elements from two different domains, the NF-FG must be subsequently split.

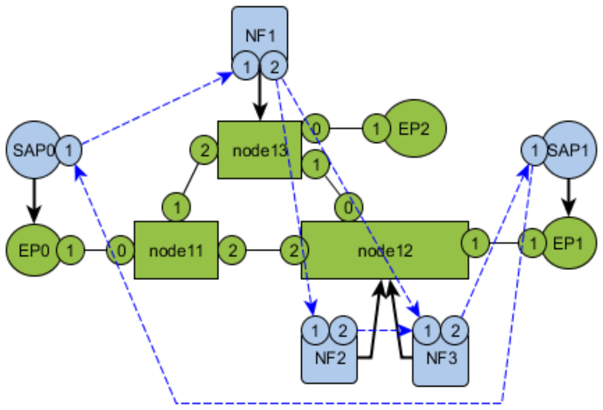


Figure 3.7: Request of NF-FG with 3NFs over a Resource Graph with 3 nodes

Table 3.6: NF-FG with 3 NFs over a three node Resource Graph

```
{ "nffg":
  { "sg": {
      "parameters" : { "template": 2,
               "version": "1.0",
               "id": 579,
               "name": "D3.2 Example",
               "tenant": 131124 }
      "sls":   [  { "resources": { "requirements": { "bandwidth": "1G",
                              "delay": "50ms" } },
                    "assignment": [ [ { "domain": "domain_a", "node": 0 },
                              { "domain": "domain_a", "node": 1 },
                              { "domain": "domain_b", "node": 3 } ] ] },
             "source": 0, "src_port": 1,
             "target": 1,"dst_port": 1 },
             { "resources": { "requirements": { "bandwidth": "1G",
                              "delay": "50ms" } },
                    "assignment": [ [ { "domain": "domain_b", "node": 3 },
                              { "domain": "domain_b", "node": 2 } ] ] },
             "source": 1, "src_port": 2,
             "target": 2, "dst_port": 1 },
             { "resources": { "requirements": { "bandwidth": "1G",
                              "delay": "50ms" },
                    "assignment": [ [ { "domain": "domain_b", "node": 3 },
                              { "domain": "domain_b", "node": 2 } ] ] },
             "source": 1, "src_port": 2,
             "target": 3, "dst_port": 1},
             { "resources": { "requirements": { "bandwidth": "1G",
                              "delay": "50ms" },
                    "assignment": [ [ { "domain": "domain_b", "node": 2 },
                              { "domain": "domain_b", "node": 2 } ] ] },
             "source": 2, "src_port": 2,
             "target": 3, "dst_port": 1 },
             { "resources": { "requirements": { "bandwidth": "1G",
                              "delay": "50ms" },
                    "assignment": [ [ { "domain": "domain_b", "node": 2 },
                              { "domain": "domain_b", "node": 4 } ] ] },
             "source": 3, "src_port": 2,
             "target": 4, "dst_port": 1 },
             { "resources": { "requirements": { "bandwidth": "1G",
                              "delay": "50ms" },
                    "assignment": [ [ { "domain": "domain_b", "node": 4 },
                              { "domain": "domain_b", "node": 2 },
                              { "domain": "domain_a", "node": 1 },
                              { "domain": "domain_a", "node": 0 } ] ] },
             "source": 4, "src_port": 1,
             "target": 0, "dst_port": 1 } ],
      "nfs": [ { "functional_type": 101,
             "id": 1,
             "monitoring": { "KQI_desc": "Numer of Users",
                     "KQI_value": 100 },
             "name": "NF1",
             "ports": [ { "id": 1, "property": "In port" },
                  { "id": 2, "property": "Out port" }],
             "resources": { "requirements": {
                       "compute" : { "capacity": "medium",
                             "cpu": "1",
                             "memory": "512M" },
                       "networking" : { "bandwidth": "1G" },
                       "storage" : { "hdd": "10G" } },
                    "assignment": [ { "domain": "domain_b",
                            "node": 3 } ] },
             "specification": { "deployment_type": 2201,
                      "image_uri": "/opt/elwud/2201.json",
                      "max_KQI_value": 250,
                      "vnf_type": 2 } },
           { "functional_type": 102,
             "id": 2,
             "monitoring": { "KQI_desc": "Number of connections",
                     "KQI_value": 10000 },
             "name": "NF2",
             "ports": [ { "id": 1, "property": "In port" },
                  { "id": 2, "property": "Out port" }],
             "resources": { "requirements": {
                       "compute" : { "capacity": "medium",
                             "cpu": "1",
                             "memory": "512M" },
                       "networking" : { "bandwidth": "1G" },
                       "storage" : { "hdd": "10G" } },
                    "assignment": [ { "domain": "domain_b",
                            "node": 2 } ] },
             "specification": { "deployment_type": 2202,
                      "image_uri": "/opt/elwud/2202.json",
                      "max_KQI_value": 10000,
                      "vnf_type": 2 } },
           { "functional_type": 103,
             "id": 3,
             "location": null,
             "monitoring": { "KQI_desc": "Number of connections",
                     "KQI_value": 10000 },
             "name": "NF3",
             "ports": [ { "id": 1, "property": "In port" },
                   { "id": 2, "property": "Out port" } ],
             "resources": { "requirements": {
                       "compute" : { "capacity": "medium",
                             "cpu": "1",
                             "memory": "512M" },
                       "networking" : { "bandwidth": "1G" },
                       "storage" : { "hdd": "10G" } },
                    "assignment": [ { "domain": "domain_b",
                            "node": 2 } ] },
            "specification": { "deployment_type": 2203,
                     "image_uri": "/opt/elwud/2203.json",
                     "max_KQI_value": 10000,
                      "vnf_type": 2 } } ],
      "saps": [ { "id": 0,
            "name": "SAP0",
            "ports": [ { "id": 1, "property": "Inbound SAP" } ],
            "resources": { "assignment": [ { "domain": "domain_a",
                             "endpoint": 0,
                             "port": "1" } ] } },
            { "id": 4,
            "name": "SAP1",
            "ports": [ { "id": 1, "property": "Outbound SAP" } ],
            "resources": { "assignment": [ { "domain": "domain_b",
                             "endpoint": 4,
                             "port": "1" } ] } } ]
  }  }
  { "rg": {
      "parameters" : { "id": 9,
               "name": "UNIFY 3 node",
               "version": "1.0" },
      "ils": [ ... ],
      "infs": [ ... ]
      "eps": [ ... ]
  }
}
```

Table 3.7: NF-FG with 3 NFs over three node Resource Graph: sub graph handled to Domain A

```
{ "nffg":
  { "sg": {
      "parameters" : { "template": 2,
               "version": "1.0",
               "id": 579,
               "name": "D3.2 Example",
               "tenant": 131124 }
      "sls":  [  { "resources": { "requirements": { "bandwidth": "1G",
                          "delay": "50ms" } },
                   "assignment": [ [ { "domain": "domain_a", "node": 0 },
                          { "domain": "domain_a", "node": 1 },
                          { "domain": "domain_a", "node": 11 } ] ] },
              "source": 0, "src_port": 1,
              "target": 11, "dst_port": 1 },
              { "resources": { "requirements": { "bandwidth": "1G",
                          "delay": "50ms" },
                   "assignment": [ [ { "domain": "domain_a", "node": 12 },
                          { "domain": "domain_a", "node": 1 },
                          { "domain": "domain_a", "node": 0 } ] ] },
              "source": 12, "src_port": 1,
              "target": 0, "dst_port": 1 } ],
      "nfs": [],
      "saps": [ { "id": 0,
            "name": "SAP0",
            "ports": [ { "id": 1, "property": "Inbound SAP" } ],
            "resources": { "assignment": [ { "domain": "domain_a",
                             "endpoint": 0,
                             "port": "1" } ] } },
            { "id": 11,
            "name": "xSAP1",
            "ports": [ { "id": 1, "property": "Cross domain SAP" } ],
            "resources": { "assignment": [ { "domain": "domain_a",
                             "endpoint": 11,
                             "port": "1" } ] } },
            { "id": 12,
            "name": "xSAP2",
            "ports": [ { "id": 1, "property": " Cross domain SAP" } ],
            "resources": { "assignment": [ { "domain": "domain_a",
                             "endpoint": 12,
                             "port": "1" } ] } } ]
  }  }
  { "rg": {
      "parameters" : { "id": 19,
               "name": "UNIFY Domain A",
               "version": "1.0" },
      "ils": [ ... ],
      "infs": [ ... ]
      "eps": [ { "domain": "domain_a",
              "id": 0,
              "name": "ep0",
              "ports": [ { "id": "1", "property": "sap" } ],
              "type": "endpoint" },
            { "domain": "domain_a",
              "id": 11,
              "name": "xepA1",
              "ports": [ { "id": "1", "property": "cross-domain" } ],
              "type": "endpoint" },
            { "domain": "domain_a",
              "id": 12,
              "name": "xepA2",
              "ports": [ { "id": "1", "property": " cross-domain " } ],
              "type": "endpoint" } ]
  }
}
```

Table 3.8: NF-FG with 3 NFs over three node Resource Graph: sub graph handled by Domain B

```
{ "nffg":
  { "sg": {
      "parameters" : { "template": 2,
               "version": "1.0",
               "id": 579,
               "name": "D3.2 Example",
               "tenant": 131124 }
      "sls":  [  { "resources": { "requirements": { "bandwidth": "1G",
                        "delay": "50ms" } },
              "assignment": [ [ { "domain": "domain_b", "node": 11 },
                        { "domain": "domain_b", "node": 3 } ] ] },
             "source": 11, "src_port": 1,
             "target": 1,"dst_port": 1 },
               { "resources": { "requirements": { "bandwidth": "1G",
                        "delay": "50ms" } },
                "assignment": [ [ { "domain": "domain_b", "node": 3 },
                          { "domain": "domain_b", "node": 2 } ] ] },
         "source": 1, "src_port": 2,
         "target": 2, "dst_port": 1 },
           { "resources": { "requirements": { "bandwidth": "1G",
                        "delay": "50ms" },
                "assignment": [ [ { "domain": "domain_b", "node": 3 },
                          { "domain": "domain_b", "node": 2 } ] ] },
         "source": 1, "src_port": 2,
         "target": 3, "dst_port": 1},
           { "resources": { "requirements": { "bandwidth": "1G",
                        "delay": "50ms" },
                "assignment": [ [ { "domain": "domain_b", "node": 2 },
                          { "domain": "domain_b", "node": 2 } ] ] },
         "source": 2, "src_port": 2,
         "target": 3, "dst_port": 1 },
           { "resources": { "requirements": { "bandwidth": "1G",
                        "delay": "50ms" },
                "assignment": [ [ { "domain": "domain_b", "node": 2 },
                          { "domain": "domain_b", "node": 4 } ] ] },
         "source": 3, "src_port": 2,
         "target": 4, "dst_port": 1 },
           { "resources": { "requirements": { "bandwidth": "1G",
                        "delay": "50ms" },
                "assignment": [ [ { "domain": "domain_b", "node": 4 },
                          { "domain": "domain_b", "node": 2 },
                          { "domain": "domain_b", "node": 12 } ] ] },
         "source": 4, "src_port": 1,
         "target": 12, "dst_port": 1 } ],
  "nfs": [ { "functional_type": 101,
         "id": 1,
         "monitoring": { "KQI_desc": "Numer of Users",
                 "KQI_value": 100 },
         "name": "NF1",
         "ports": [ { "id": 1, "property": "In port" },
              { "id": 2, "property": "Out port" }],
         "resources": { "requirements": {
                   "compute" : { "capacity": "medium",
                         "cpu": "1",
                         "memory": "512M" },
                   "networking" : { "bandwidth": "1G" },
                   "storage" : { "hdd": "10G" } },
                "assignment": [ { "domain": "domain_b",
                        "node": 3 } ] },
         "specification": { "deployment_type": 2201,
                  "image_uri": "/opt/elwud/2201.json",
                  "max_KQI_value": 250,
                  "vnf_type": 2 } },
       { "functional_type": 102,
         "id": 2,
         "monitoring": { "KQI_desc": "Number of connections",
                 "KQI_value": 10000 },
         "name": "NF2",
         "ports": [ { "id": 1, "property": "In port" },
              { "id": 2, "property": "Out port" }],
         "resources": { "requirements": {
                   "compute" : { "capacity": "medium",
                         "cpu": "1",
                              "memory": "512M" },
                        "networking" : { "bandwidth": "1G" },
                        "storage" : { "hdd": "10G" } },
                    "assignment": [ { "domain": "domain_b",
                                "node": 2 } ] },
             "specification": { "deployment_type": 2202,
                       "image_uri": "/opt/elwud/2202.json",
                       "max_KQI_value": 10000,
                       "vnf_type": 2 } },
           { "functional_type": 103,
             "id": 3,
             "location": null,
             "monitoring": { "KQI_desc": "Number of connections",
                     "KQI_value": 10000 },
             "name": "NF3",
             "ports": [ { "id": 1, "property": "In port" },
                   { "id": 2, "property": "Out port" } ],
             "resources": { "requirements": {
                        "compute" : { "capacity": "medium",
                              "cpu": "1",
                              "memory": "512M" },
                        "networking" : { "bandwidth": "1G" },
                        "storage" : { "hdd": "10G" } },
                    "assignment": [ { "domain": "domain_b",
                                "node": 2 } ] },
            "specification": { "deployment_type": 2203,
                      "image_uri": "/opt/elwud/2203.json",
                      "max_KQI_value": 10000,
                       "vnf_type": 2 } } ],
      "saps": [ { "id": 11,
            "name": "xSAP1",
            "ports": [ { "id": 1, "property": "Inbound SAP" } ],
            "resources": { "assignment": [ { "domain": "domain_b",
                              "endpoint": 11,
                              "port": "1" } ] } },
            { "id": 12,
            "name": "xSAP2",
            "ports": [ { "id": 1, "property": "Inbound SAP" } ],
            "resources": { "assignment": [ { "domain": "domain_b",
                              "endpoint": 12,
                              "port": "1" } ] } },

            { "id": 4,
            "name": "SAP1",
            "ports": [ { "id": 1, "property": "Outbound SAP" } ],
            "resources": { "assignment": [ { "domain": "domain_b",
                              "endpoint": 4,
                              "port": "1" } ] } } ]
  }  }
  { "rg": {
      "parameters" : { "id": 21,
               "name": "UNIFY Domain B",
               "version": "1.0" },
      "ils": [ ... ],
      "infs": [ ... ]
      "eps": [ { "domain": "domain_a",
             "id": 0,
             "name": "ep0",
             "ports": [ { "id": "1", "property": "sap" } ],
             "type": "endpoint" },
           { "domain": "domain_b",
             "id": 11,
             "name": "xepB1",
             "ports": [ { "id": "1", "property": "cross-domain" } ],
             "type": "endpoint" },
           { "domain": "domain_b",
             "id": 12,
             "name": "xepB2",
             "ports": [ { "id": "1", "property": " cross-domain " } ],
             "type": "endpoint" } ]
  }
}
```

3.5.6 Request with shared NF

The sixth and final example is a variation of the last example, where one of the NFs (shown with a stronger blue in the figure) is shared with another NF-FG.

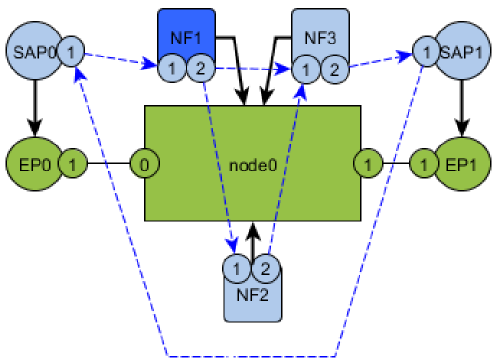


Figure 3.8: Request of NF-FG with 3NFs where one is shared

Table 3.9: NF-FG with 3 NFs where one is shared

```
{ "nffg":
  { "sg": {
      "parameters" : { "template": 2,
                       "version": "1.0",
                       "id": 578,
                       "name": "D3.2 Example",
                       "tenant": 131124 }
      "sls":  [  { "resources": { "requirements": { "bandwidth": "1G",
                                 "delay": "50ms" } },
                     "assignment": [ [ { "domain": "domain_0", "node": 0 },
                                 { "domain": "domain_0", "node": 1 } ] ] },
              "source": 0, "src_port": 1,
              "target": 1,"dst_port": 1 },
              { "resources": { "requirements": { "bandwidth": "1G",
                                 "delay": "50ms" } },
                     "assignment": [ [ { "domain": "domain_0", "node": 1 },
                                 { "domain": "domain_0", "node": 1 } ] ] },
              "source": 1, "src_port": 2,
              "target": 2, "dst_port": 1 },
              { "resources": { "requirements": { "bandwidth": "1G",
                                 "delay": "50ms" },
                     "assignment": [ [ { "domain": "domain_0", "node": 1 },
                                 { "domain": "domain_0", "node": 1 } ] ] },
              "source": 1, "src_port": 2,
              "target": 3, "dst_port": 1},
              { "resources": { "requirements": { "bandwidth": "1G",
                                 "delay": "50ms" },
                     "assignment": [ [ { "domain": "domain_0", "node": 1 },
                                 { "domain": "domain_0", "node": 1 } ] ] },
              "source": 2, "src_port": 2,
              "target": 3, "dst_port": 1 },
              { "resources": { "requirements": { "bandwidth": "1G",
                                 "delay": "50ms" },
                     "assignment": [ [ { "domain": "domain_0", "node": 1 },
                                 { "domain": "domain_0", "node": 2 } ] ] },
              "source": 3, "src_port": 2,
              "target": 4, "dst_port": 1 },
              { "resources": { "requirements": { "bandwidth": "1G",
                                 "delay": "50ms" },
                     "assignment": [ [ { "domain": "domain_0", "node": 2 },
                                 { "domain": "domain_0", "node": 1 },
                                 { "domain": "domain_0", "node": 0 } ] ] },
              "source": 4, "src_port": 1,
              "target": 0, "dst_port": 1 } ],
       "nfs": [ { "functional_type": 101,
              "id": 1,
              "monitoring": { "KQI_desc": "Numer of Users",
                      "KQI_value": 100 },
              "name": "NF1",
              "ports": [ { "id": 1, "property": "In port" },
                   { "id": 2, "property": "Out port" }],
              "resources": { "requirements": {
                         "compute" : { "capacity": "medium",
                               "cpu": "1",
                               "memory": "512M" },
                         "networking" : { "bandwidth": "1G" },
                         "storage" : { "hdd": "10G" } },
                     "assignment": [ { "domain": "domain_0", "node": 1,
                                  "nffg": 576, "running_nf": 1 } ] },
              "specification": { "deployment_type": 2201,
                        "image_uri": "/opt/elwud/2201.json",
                        "max_KQI_value": 250,
                        "vnf_type": 2 } },
            { "functional_type": 102,
              "id": 2,
              "monitoring": { "KQI_desc": "Number of connections",
                      "KQI_value": 10000 },
              "name": "NF2",
              "ports": [ { "id": 1, "property": "In port" },
                   { "id": 2, "property": "Out port" }],
              "resources": { "requirements": {
                         "compute" : { "capacity": "medium",
                               "cpu": "1",
                               "memory": "512M" },
                         "networking" : { "bandwidth": "1G" },
                         "storage" : { "hdd": "10G" } },
                     "assignment": [ { "domain": "domain_0",
                              "node": 1 } ] },
              "specification": { "deployment_type": 2202,
                       "image_uri": "/opt/elwud/2202.json",
                       "max_KQI_value": 10000,
                       "vnf_type": 2 } },
            { "functional_type": 103,
              "id": 3,
              "location": null,
              "monitoring": { "KQI_desc": "Number of connections",
                      "KQI_value": 10000 },
              "name": "NF3",
              "ports": [ { "id": 1, "property": "In port" },
                    { "id": 2, "property": "Out port" } ],
              "resources": { "requirements": {
                        "compute" : { "capacity": "medium",
                              "cpu": "1",
                              "memory": "512M" },
                        "networking" : { "bandwidth": "1G" },
                        "storage" : { "hdd": "10G" } },
                     "assignment": [ { "domain": "domain_0",
                          "node": 1 } ] },
             "specification": { "deployment_type": 2203,
                      "image_uri": "/opt/elwud/2203.json",
                      "max_KQI_value": 10000,
                       "vnf_type": 2 } } ],
       "saps": [ { "id": 0,
             "name": "SAP0",
             "ports": [ { "id": 1, "property": "Inbound SAP" } ],
             "resources": { "assignment": [ { "domain": "domain_0",
                              "endpoint": 0,
                              "port": "1" } ] } },
             { "id": 4,
             "name": "SAP1",
             "ports": [ { "id": 1, "property": "Outbound SAP" } ],
             "resources": { "assignment": [ { "domain": "domain_0",
                              "endpoint": 2,
                              "port": "1" } ] } } ]
   }  }
   { "rg": {
       "parameters" : { "id": 8,
                "name": "UNIFY 1 node",
                "version": "1.0" },
       "ils": [ ... ],
       "infs": [ ... ]
       "eps": [ ... ]
   }
}
```

4 Virtualizer-based NF-FG model and mapped resource requests

In this section we present both the Virtualizer and the mapping of the resource request (also known the NF-FG) with a YANG model. This model is based on the principals laid down in D2.2 [D2.2]. The model includes (i) infrastructure virtualization (referred to as Resource Graph –RG in previous discussions) combined with (ii) the mapping of resource request(s). The virtualization together with the mapping constructs the NF-FG.

In what follows, we gradually introduce the different concepts with figures and examples. We will use a four-domain UNIFY scenario to elaborate on the concept as shown in Figure 4.1. The corresponding message sequence chart is show in Figure 4.5. We will focus on the clients of the virtualizers, namely Service Orchestrator (SO), Resource Orchestrators (ROs) and Controller Adaptor (CA). Furthermore, T1 and T2 denotes tenants of Service Provider 0 (SP0), who operates UNIFY Domain 0. The resources of UNIFY Domain 0 are built of UNIFY Domain 1 and 2 (not detailed further); UNIFY Domain 0 provides wholesale resources to UNIFY Domain 3 (U3), who is a retail provider. However, technically, both the SO and U3 are similar clients of the UNIFY Domain 0 as we will see later.

Figure 4.2, Figure 4.3 and Figure 4.4 show the views of the different components (actors) of the system with regard to the scoping or responsibility of their operation. Figure 4.2 shows that the Service Orchestrator within the service layer only sees its own tenants (Tenant-1 and Tenant-2) and the infrastructure view that is presented to it. The service graph orchestration from its tenants will be performed like operating over physical infrastructure resources directly. Figure 4.3 shows the view corresponding to the Unify Resource Orchestrator. The RO sees only two clients: the Service Orchestrator of the service layer and the client representing the external UNIFY domain. The tenants of the service layer (Tenant-1 and Tenant-2) do not appear in the RO's view, as the service graph requests are handled (orchestrated) by the Service Orchestrator. It is only the aggregated view of the tenant requests that is conveyed as updates to the infrastructure use. Both, from operation and reference point of view there is absolutely no difference between the two clients of the RO. With this respect the RO multiplexes the requests of its clients. Finally, Figure 4.4 shows the view corresponding to the Controller Adaptor (CA). The CA creates a global resource view to the RO via its Virtualizer based on the infrastructure views presented by the underlying domains. The CA has no knowledge of whether it operates over physical or virtualized resources, though it converts the resource management operations into the northbound API of the underlying domains. For example, the operation could be executed as OpenFlow 1.3 commands over a physical switch or over a BigSwitch abstraction, however, neither makes any difference operation wise in the CA.

Last but not least, Figure 4.5 shows an example message chart related to topology of Figure 4.1 (message sequence numbers are denoted with {#}). We assume NETCONF operations (see Section 4.9):

- {1}-{2}: SO discovers the infrastructure view
- {3}: Tenant-1's service request (SG1)

- {4}: Service Orchestration – mapping of SG1 to the infrastructure view
- {5}: SO edits the resource configuration as a result of the SG1 mapping
- {6}: RO is notified of the resource configuration updates
- {7}: RO maps the updated resource allocation of its Virtualizer-1 to the underlying infrastructure view (Virtualization-0) and updates the resource configuration at Virtualizer-0 {8}
- The CA is notified in {9} of the changes and splits the configuration changes according to the two distinct resource sets to its south; updates the corresponding configurations in {11} and {12}
- In {13}-{16} the SO processes Tenant-2's service request. It is important to note that both Tenant-1 and Tenant-2 requests are mapped to the same Virtualizer-1. Therefore, neither Virtualizer-1 nor the RO know if the update in {15/16} comes as a result of a single or multiple tenant requests.
- {16}-{22} are executed as {6}-{12}
- The resource requests from an external domain {25} are handled exactly the same as the request from the SO {5}-{12} and {15}-{22}.

The NF-FG model appears throughout all communication in Figure 4.5 when a Virtualizer is involved, i.e., all communication of Figure 4.5 but in the Service Graph requests marked with blue.

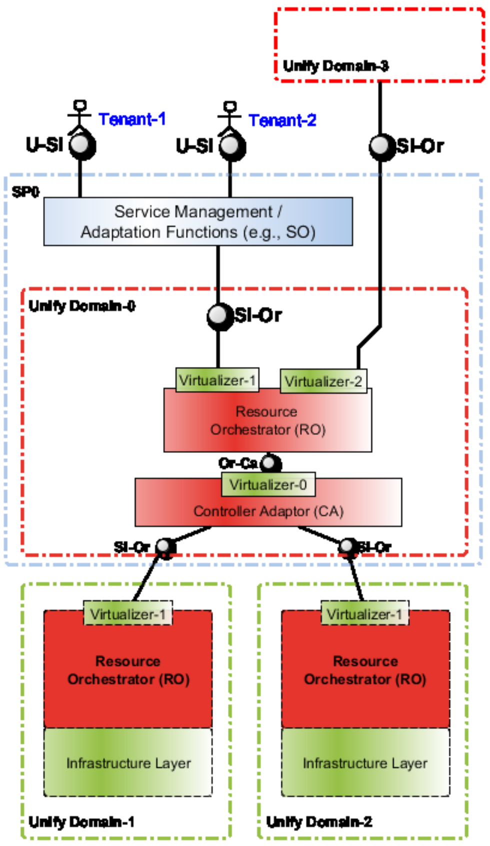


Figure 4.1: Example topology for Virtualizer message chart

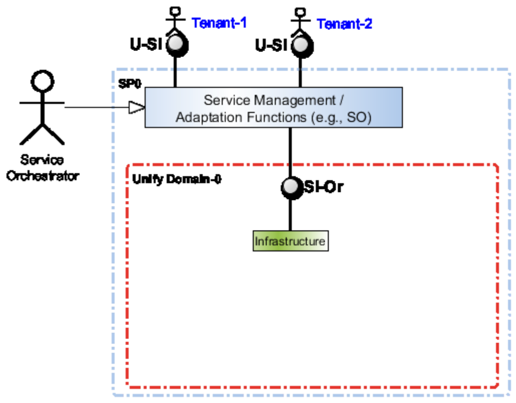


Figure 4.2: Service Orchestrator's view of the topology shown in Figure 4.1.

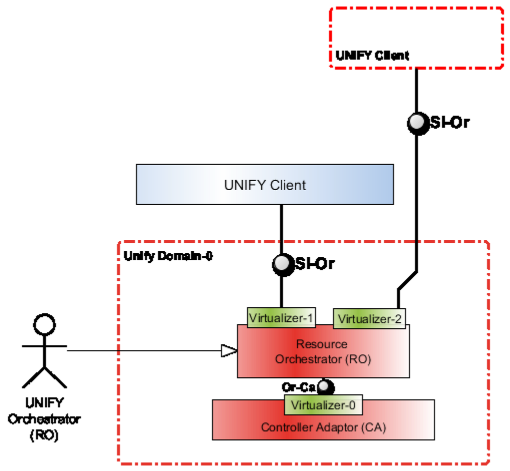


Figure 4.3: Resource Orchestrator's view of the topology shown in Figure 4.1.

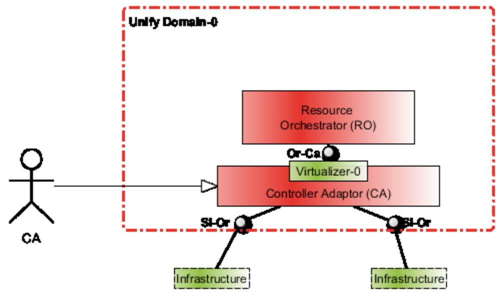


Figure 4.4: Controller Adaptor's view of the topology shown in Figure 4.1.

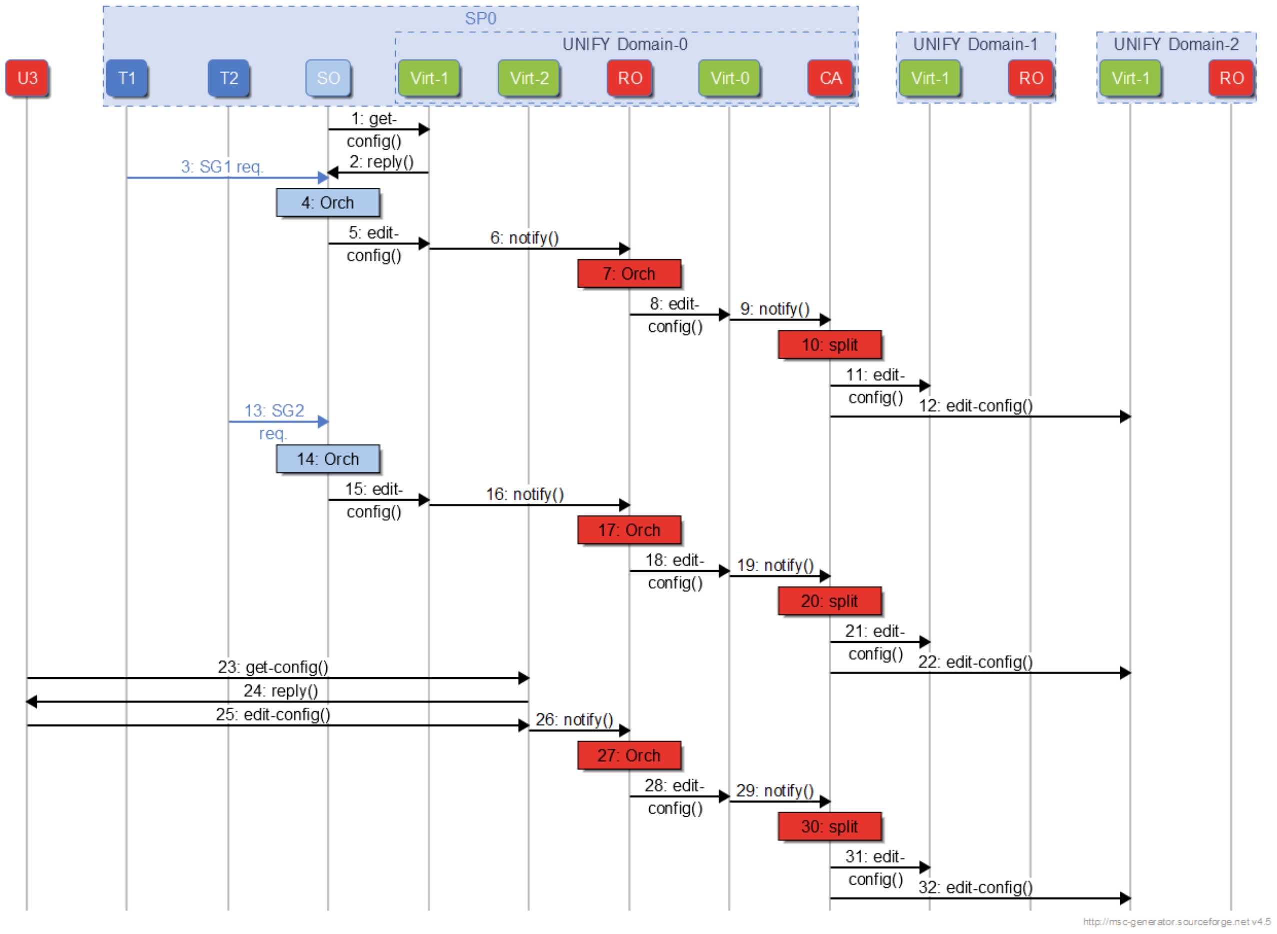


Figure 4.5: Virtualizer example message chart

We will incrementally introduce the data model in the following sections.

4.1 Simple infrastructure view

First we focus on step 2 of Figure 4.5, which is the step where the Virtualizer reports the infrastructure. To report the available (virtualized) infrastructure to the upper layer we need the notion of

- BiS-BiS entities, their resources, and
- Links interconnecting BiS-BiS nodes.

Figure 4.6 shows a simple example topology, for which the information model (the tree representation of the YANG model) is shown in Table 4.1.

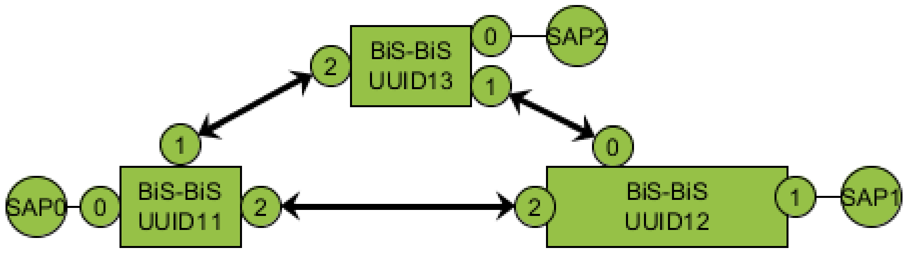


Figure 4.6: Simple infrastructure example

Table 4.1: Simple infrastructure model tree view

```
+--ro virtualizer
  +--ro id?       string
  +--ro name?     string
  +--ro nodes
  |  +--ro node* [id]
  |     +--ro id                 string
  |     +--ro name?              string
  |     +--ro type               string
  |     +--ro ports
  |     |  +--ro port* [id]
  |     |     +--ro id              string
  |     |     +--ro name?           string
  |     |     +--ro port_type?      string
  |     +--ro resources
  |     |  +--ro cpu         string
  |     |  +--ro mem         string
  |     |  +--ro storage     string
  +--ro links
        +--ro link* [src dst]
              +--ro id?            string
              +--ro name?          string
              +--ro src            port-ref
              +--ro dst            port-ref
              +--ro resources
                 +--ro delay?         string
                 +--ro bandwidth?     string
```

There are common elements to all objects, like the Id and the Name of the object.

The Infrastructure view consists of BiS-BiS entities, which are represented as Nodes. The nodes have Ports and Resources (like available `CPU` or `memory`). The Ports can either be simple (called abstract) or Service Access Point (SAP) type.

The infrastructure connections between the BiS-BiS nodes are represented by Links in the Virtualizer view. These Links are defined between Ports of the nodes, and can have a resource description like available bandwidth or delay.

Further examples are given in A.2.2 and A.2.3.

4.2 Advanced infrastructure view

A single BiS-BiS node from the Virtualizer view may represent the abstract view of a complex underlying infrastructure (e.g. data centres, transport network or both). Figure 4.7 shows a network topology abstraction into single BiS-BiS node. There may be a need to express some additional limitations of the infrastructure beyond the already discussed connectivity. Therefore we introduce optional Links within the Nodes, to be able to describe e.g. delay between ports of a BiS-BiS. Note that this new Link of the Node (shown in red in Table 4.2) has the same structure as the previously introduced infrastructure Link (shown in green). It is important to note here, that there may be multiple virtualizers presenting different (abstraction) views of the same underlying infrastructure towards different clients, e.g. 1 BiS-BiS node view, 3 BiS-BiS node view, 1 BiS-BiS node view with internal links, etc. This enables high flexibility.

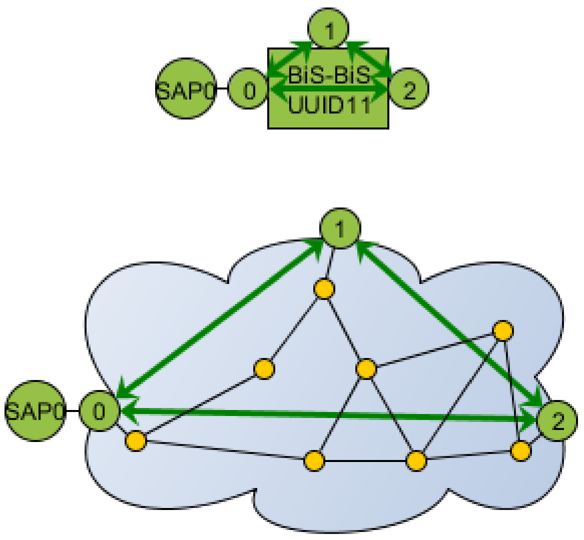


Figure 4.7: Topology abstraction as internal link characteristics

Table 4.2: Advanced infrastructure model tree view

```
+--ro virtualizer
  +--ro id?      string
  +--ro name?    string
  +--ro nodes
  |  +--ro node* [id]
  |     +--ro id                string
  |     +--ro name?             string
  |     +--ro type              string
  |     +--ro ports
  |     |  +--ro port* [id]
  |     |     +--...
  |     +--ro links
  |     |  +--ro link* [src dst]
  |     |     +--ro id?         string
  |     |     +--ro name?       string
  |     |     +--ro src         port-ref
  |     |     +--ro dst         port-ref
  |     |     +--ro resources
  |     |        +--ro delay?       string
  |     |        +--ro bandwidth?   string
  |     +--ro resources
  |     |  +--...
  +--ro links
        +--ro link* [src dst]
              +--ro id?         string
              +--ro name?       string
              +--ro src         port-ref
              +--ro dst         port-ref
              +--ro resources
                 +--ro delay?       string
                 +--ro bandwidth?   string
```

Example is given in A.2.4, with a domain abstracted to a BiS-BiS node, and reported by the underlying Virtualizer.

4.3 Simple Request

Now we move our focus to the description of a request, corresponding to step 5 of Figure 4.5, where a mapping of NS to infrastructure nodes as well as the forwarding rules have to be given. In the request the previously reported infrastructure is not repeated, however it's referred to by the IDs of, e.g., BiS-BiS Nodes.

A simple request consists of assigning NFs to Nodes, and configuring the forwarding between BiS-BiS and/or NF ports. An example is shown in Figure 4.8 and the information model is extended according to Table 4.3.

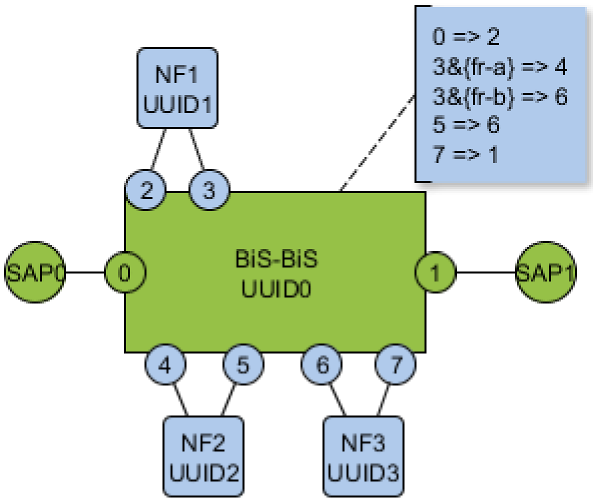


Figure 4.8: Simple request of 3 NFs mapped to a single BiS-BiS node

Table 4.3: Simple request model tree view

```
+--rw virtualizer
   +--rw id?      string
   +--rw name?    string
   +--rw nodes
   |  +--ro node* [id]
   |     +--ro id                 string
   |     +--ro name?              string
   |     +--ro type               string
   |     +--ro ports
   |     |  +--ro port* [id]
   |     |     +--ro id               string
   |     |     +--ro name?            string
   |     |     +--ro port_type?       string
   |     |     +--ro (port-type)?
   |     +--ro resources
   |     |  +--ro cpu         string
   |     |  +--ro mem         string
   |     |  +--ro storage     string
   |     +--rw NF_instances
   |     |  +--rw node* [id]
   |     |     +--rw id              string
   |     |     +--rw name?           string
   |     |     +--rw type            string
   |     |     +--rw ports
   |     |     |  +--rw port* [id]
   |     |     |     +--rw id              string
   |     |     |     +--rw name?           string
   |     |     |     +--rw port_type?      string
   |     +--rw flowtable
   |        +--rw flowentry* [port match action]
   |           +--rw port       port-ref
   |           +--rw match      string
   |           +--rw action     string
```

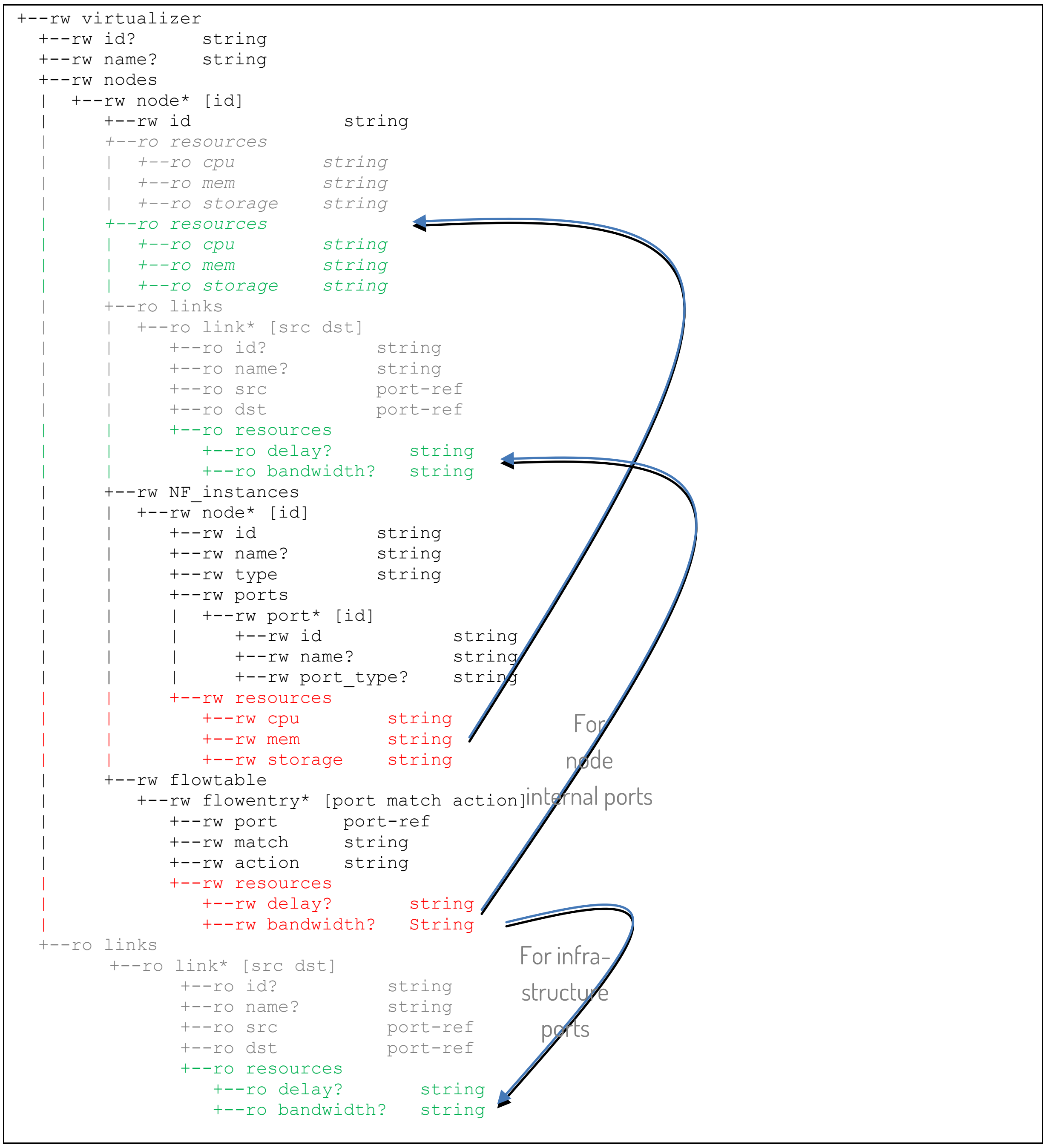
Please note, that the description of NFs (red colour) of the Nodes is equivalent to the existing description of Nodes themselves (shown in grey colour, but only for reference: details of nodes are not repeated in a request). The ports in this case represent the ports of the NF, while the resources represent the required resources of the NF. The flow entries can refer to (infrastructure) Ports and NF Ports of the Node (shown in purple).

See A.2.5 for further examples.

4.4 Requests with resource demands

A request may contain computes resources (e.g., CPU, memory) for the NFs and networking resources (e.g., delay, bandwidth) for the forwarding overlay among NFs and SAPs. This is described as extra parameters to the NF instances or to the flow entries (shown in red in Table 4.4). We use the same resource fields that have been used for the infrastructure description at nodes, external and internal links.

Table 4.4: Advanced request with flow requirements model tree view



```
+--rw virtualizer
  +--rw id?      string
  +--rw name?    string
  +--rw nodes
  |  +--rw node* [id]
  |     +--rw id              string
  |     +--ro resources
  |     |  +--ro cpu        string
  |     |  +--ro mem        string
  |     |  +--ro storage    string
  |     +--ro resources
  |     |  +--ro cpu        string
  |     |  +--ro mem        string
  |     |  +--ro storage    string
  |     +--ro links
  |     |  +--ro link* [src dst]
  |     |     +--ro id?          string
  |     |     +--ro name?        string
  |     |     +--ro src          port-ref
  |     |     +--ro dst          port-ref
  |     |     +--ro resources
  |     |        +--ro delay?       string
  |     |        +--ro bandwidth?   string
  |     +--rw NF_instances
  |     |  +--rw node* [id]
  |     |     +--rw id           string
  |     |     +--rw name?        string
  |     |     +--rw type         string
  |     |     +--rw ports
  |     |     |  +--rw port* [id]
  |     |     |     +--rw id            string
  |     |     |     +--rw name?         string
  |     |     |     +--rw port_type?    string
  |     |     +--rw resources
  |     |        +--rw cpu        string
  |     |        +--rw mem        string
  |     |        +--rw storage    string
  |     +--rw flowtable
  |        +--rw flowentry* [port match action]
  |           +--rw port      port-ref
  |           +--rw match     string
  |           +--rw action    string
  |           +--rw resources
  |              +--rw delay?       string
  |              +--rw bandwidth?   String
  +--ro links
        +--ro link* [src dst]
               +--ro id?          string
               +--ro name?        string
               +--ro src          port-ref
               +--ro dst          port-ref
               +--ro resources
                  +--ro delay?       string
                  +--ro bandwidth?   string
```

See A.2.6 for further examples.

Furthermore, requirements may be formulated on the NF instance itself, for example the requested delay or bandwidth between the NF's ports. Such requirements are input to the orchestrator to select appropriate NF type and resource assignment to deliver the requested performance characteristics. Furthermore, an NF may be decomposed, hence a compound NF requirement may be mapped to a topology of interconnected NFs. Note however, that defining NF requirements is an alternative to assigning resources to the NF, i.e., an orchestrator either use one or the other as both might create a conflict unless clear precedence is assigned. Such NF requirements can be described similarly to interconnection links between NF, but the scope is the internal connection of the NF (see Figure 4.9 for an example). The extended data model is shown with red in Table 4.5. Naturally, we reused the existing link description format from the infrastructure report (shown grey colour, which is actually not part of a request).

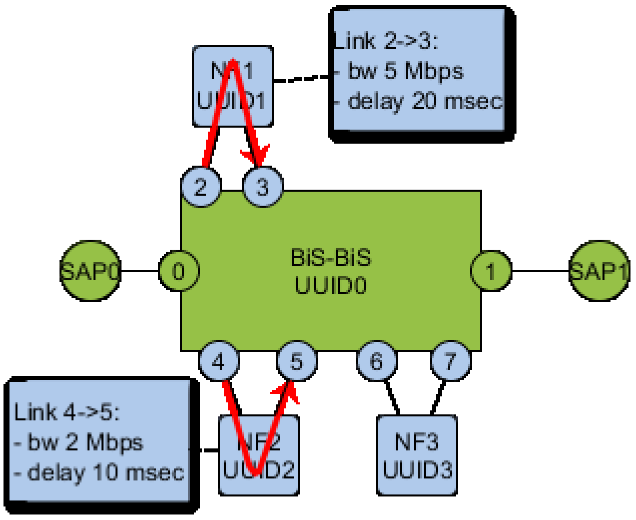


Figure 4.9: Request with NF (internal) requirements

Table 4.5: Advanced request with NF requirements model tree view

```
+--rw virtualizer
  +--rw id?       string
  +--rw name?     string
  +--rw nodes
   |  +--rw node* [id]
   |     +--rw id              string
   |     +--rw links
   |     |  +--rw link* [src dst]
   |     |     +--rw id?          string
   |     |     +--rw name?        string
   |     |     +--rw src          port-ref
   |     |     +--rw dst          port-ref
   |     |     +--rw resources
   |     |        +--rw delay?       string
   |     |        +--rw bandwidth?   string
   |     +--rw NF_instances
   |     |  +--rw node* [id]
   |     |     +--rw id           string
   |     |     +--rw name?        string
   |     |     +--rw type         string
   |     |     +--rw ports
   |     |     |  +--...
   |     |     +--rw links
   |     |     |  +--rw link* [src dst]
   |     |     |     +--rw id?          string
   |     |     |     +--rw name?        string
   |     |     |     +--rw src          port-ref
   |     |     |     +--rw dst          port-ref
   |     |     |     +--rw resources
   |     |     |        +--rw delay?       string
   |     |     |        +--rw bandwidth?   string
   |     |     +--rw resources
   |     |        +--...
   |     +--rw flowtable
   |        +--...
```

See A.2.7 for further examples.

4.5 Capability reporting

Finally, BiS-BiS nodes may report their capabilities related to their ability to host different NF types. The goals are twofold, (i) once to report the list of supported NF types and (ii) to provide resource figures for the client service or resource orchestrator. This later can be considered as some discrete benchmarking values for hosting an NF, for example, NF type A with an in/out port can be instantiated with 1 CPU and 600Mbyte memory to handle up to 20Mbps input traffic rate within 20msec in-out port delay; or 2 CPU with 1Gbyte memory to handle 30Mbps traffic within 25msec delay. This report is optional and can be used in a case where the NF developer has provided these performance values for the given infrastructure, or maybe in a case, where the infrastructure has this information from previous measurements. We provide here the way to represent this information once it's available. This information is of course domain specific, different values may apply for the same NF in a different infrastructure domain. (Note: similar performance to resource "conversion" rules may be contained in the NFIB service decomposition rules, to be elaborated in D3.3) The capability reporting reuses the data model of NF instances (see Table 4.6).

Table 4.6: Capability reporting tree view

```
+--rw virtualizer
   +--ro id?      string
   +--ro name?    string
   +--ro nodes
   |  +--ro node* [id]
   |     +--...
   |     +--rw NF_instances
   |     |  +--rw node* [id]
   |     |     +--rw id            string
   |     |     +--rw name?         string
   |     |     +--rw type          string
   |     |     +--rw ports
   |     |     |  +--rw port* [id]
   |     |     |     +--rw id            string
   |     |     |     +--rw name?         string
   |     |     |     +--rw port_type?    string
   |     |     +--rw links
   |     |     |  +--rw link* [src dst]
   |     |     |     +--rw id?          string
   |     |     |     +--rw name?        string
   |     |     |     +--rw src          port-ref
   |     |     |     +--rw dst          port-ref
   |     |     |     +--rw resources
   |     |     |        +--rw delay?       string
   |     |     |        +--rw bandwidth?   string
   |     |     +--rw resources
   |     |        +--rw cpu        string
   |     |        +--rw mem        string
   |     |        +--rw storage    string
   |     +--ro capabilities
   |     |  +--ro supported_NFs
   |     |     +--ro node* [id]
   |     |        +--ro id            string
   |     |        +--ro name?         string
   |     |        +--ro type          string
   |     |        +--ro ports
   |     |        |  +--ro port* [id]
   |     |        |     +--ro id            string
   |     |        |     +--ro name?         string
   |     |        |     +--ro port_type?    string
   |     |        +--ro links
   |     |        |  +--ro link* [src dst]
   |     |        |     +--ro id?          string
   |     |        |     +--ro name?        string
   |     |        |     +--ro src          port-ref
   |     |        |     +--ro dst          port-ref
   |     |        |     +--ro resources
   |     |        |        +--ro delay?       string
   |     |        |        +--ro bandwidth?   string
   |     |        +--ro resources
   |     |           +--ro cpu        string
   |     |           +--ro mem        string
   |     |           +--ro storage    string
```

4.6 Constraints handling

Figure 4.10 shows and example of presenting the infrastructure by Virtualizers in a multi-layer scenario. In this example the physical infrastructure (consisting of multiple networking nodes, Data Centres and links) is abstracted to a 3-node view by a Virtualizer. Then this 3-node view is further abstracted to a single node view by an upper layer Virtualizer.

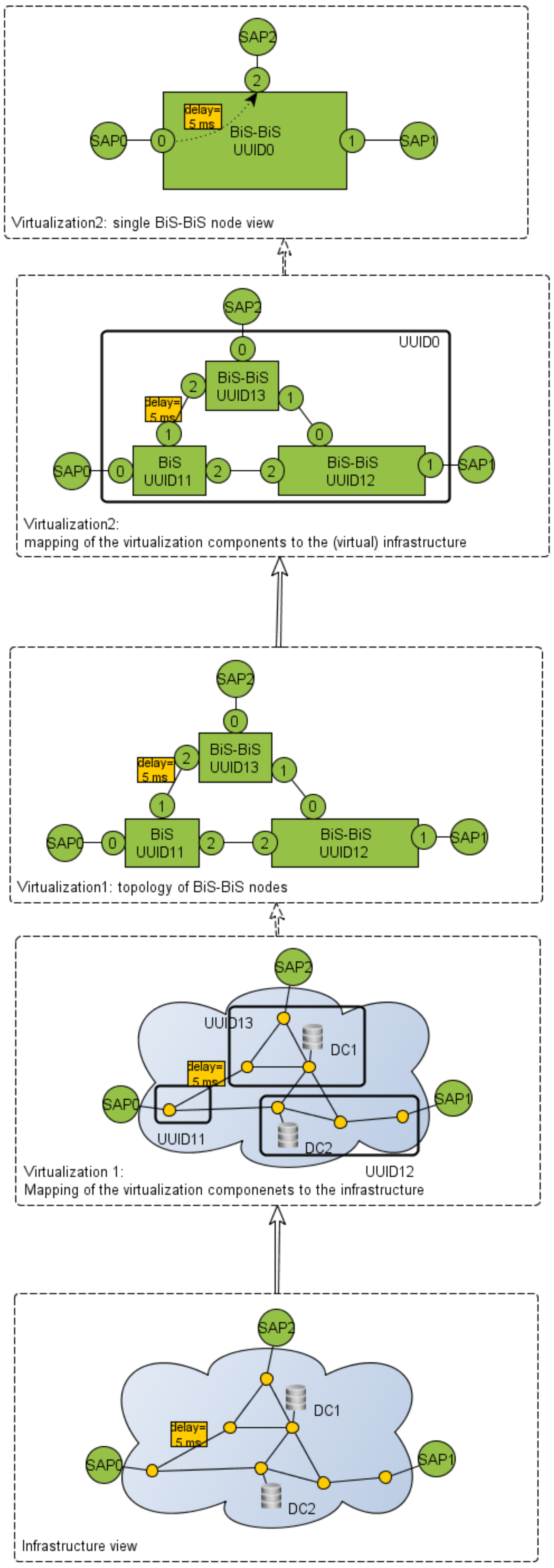


Figure 4.10: Physical resources and Multiple-level Virtualizations

Figure 4.11 shows an example service request, containing 3 Network Functions (NFs). Figure 4.12 shows an advanced version of the request containing delay requirements on both the whole service and on some components. We will use this example in the rest of the document.

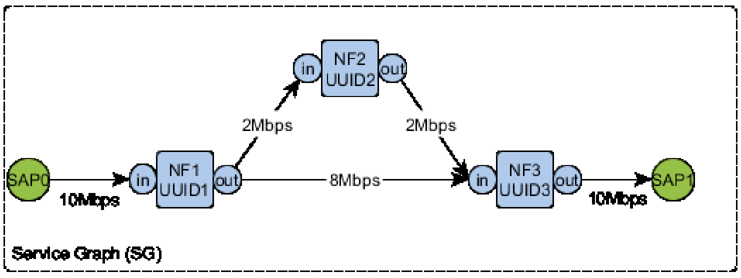


Figure 4.11: Example service graph

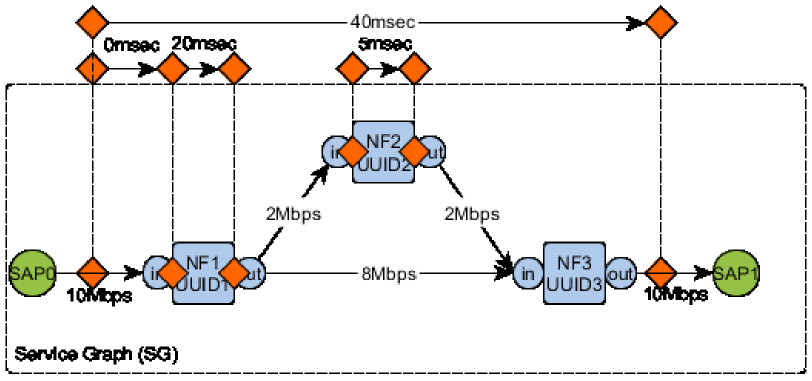


Figure 4.12: Service requirements in a service graph

Service requests must be mapped to the virtualization views, e.g., Figure 4.10 presented to the clients. However, some of the constraints cannot be / may not be necessarily mapped to the infrastructure view due to lack of details (information hiding with the virtualization) or on purpose to delegate final orchestration decision. For example, in Figure 4.10 the top Virtualizer shows a single node view, while the bottom one has a 3 node view.

Mapping of software and networking requirements must match (and must be in pair) of the corresponding software and networking resources. Software resources, e.g., compute and storage, are assigned to the BiS-BiS nodes. Networking resources, e.g. bandwidth and delay, are assigned to external links among BiS-BiS nodes (e.g. bottom part of Figure 4.10 link between nodes UUID11 and UID13) and between each port pair of a BiS-BiS node. However, allocation of Network Functions (NFs) to BiS-BiS introduces additional ports representing the point of the NF attachment to the BiS-BiS node (e.g. bottom part of Figure 4.10 ports 4 of node UUID13).

Therefore, we propose to map the resource requirements of the service to

- Networking resources, e.g., delay and bandwidth, within a BiS-BiS
    - from an external port to a NF port (e.g. bottom part of Figure 4.13 node UUID12, between ports 0 and 4)
    - from an NF port to another NF port (e.g. bottom part of Figure 4.13 node UUID12, between ports 5 and 6)
    - from an NF port to an external port (e.g. bottom part of Figure 4.13 node UUID12, between ports 7 and 1)
- Networking resources, e.g., delay and bandwidth, within an NF
    - for each in port out port relation (e.g. bottom part of Figure 4.13 node UUID12, NF UUID2, between ports 4 and 5)
- Software resources to each NF allocation
    - (e.g. bottom part of Figure 4.13 node UUID13, NF UUID1)

Here networking resources could be bandwidth, delay, priority / pre-emption, and traffic class, whereas software resources could be compute (e.g. CPU), storage (e.g. memory, disk), and I/O capabilities.

Beyond the explicit allocation of resources, we propose to allow variable binding to any of the above resource instances. Constraints related to algebraic operations, min/max and alike can be used to formulate unresolved but global constraints.

Figure 4.13 shows a recursive mapping of the Service Graph with the requirements presented in Figure 4.12. Figure 4.13 shows both explicit resource assignment and variable binding to resources and global constraints, e.g., expressed as delay to be smaller than the sum of different delay variables.

The example contains a requirement of 0 ms delay. This should be interpreted as "as close as possible". We could allocate a new variable e.g. "R" to this delay, include it to the sum-constraint, and add a further one to "minimize R".

To describe Networking resources (e.g. delay and bandwidth) within a BiS-BiS, we propose to extend the OpenFlow-like forwarding rules deployed to BiS-BiS nodes (consisting of a matching rule and an action) with fields containing requirements for the given entry like bandwidth and delay.

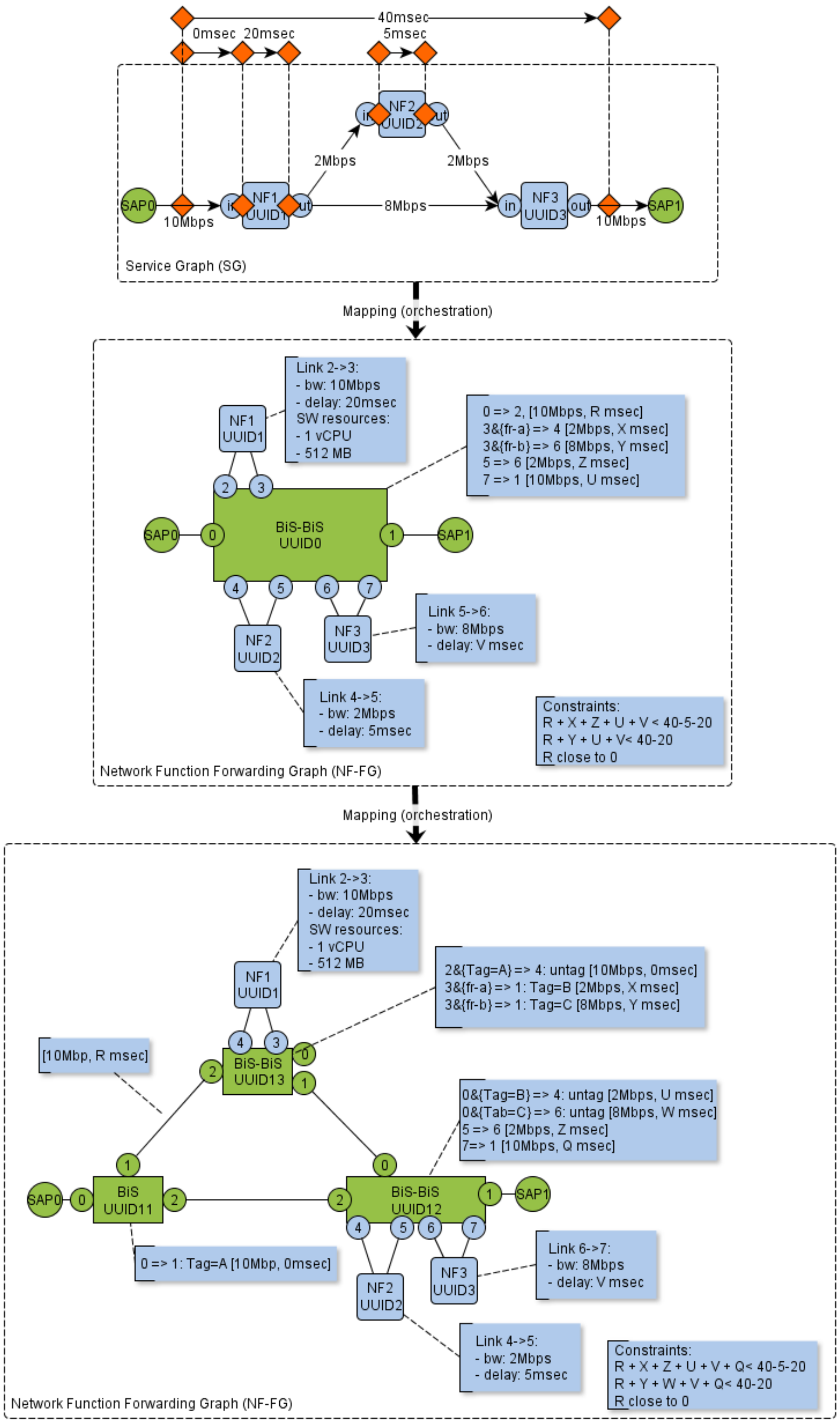


Figure 4.13: Mapping of Service Level Requirements to the BiS-BiS Virtualization View

4.7 Resource sharing

The proposed Virtualizer model supports describing resource sharing, e.g. link sharing or NF sharing. Figure 4.14 contains an infrastructure example (left part), where the upcoming service requests (right part) will be mapped.

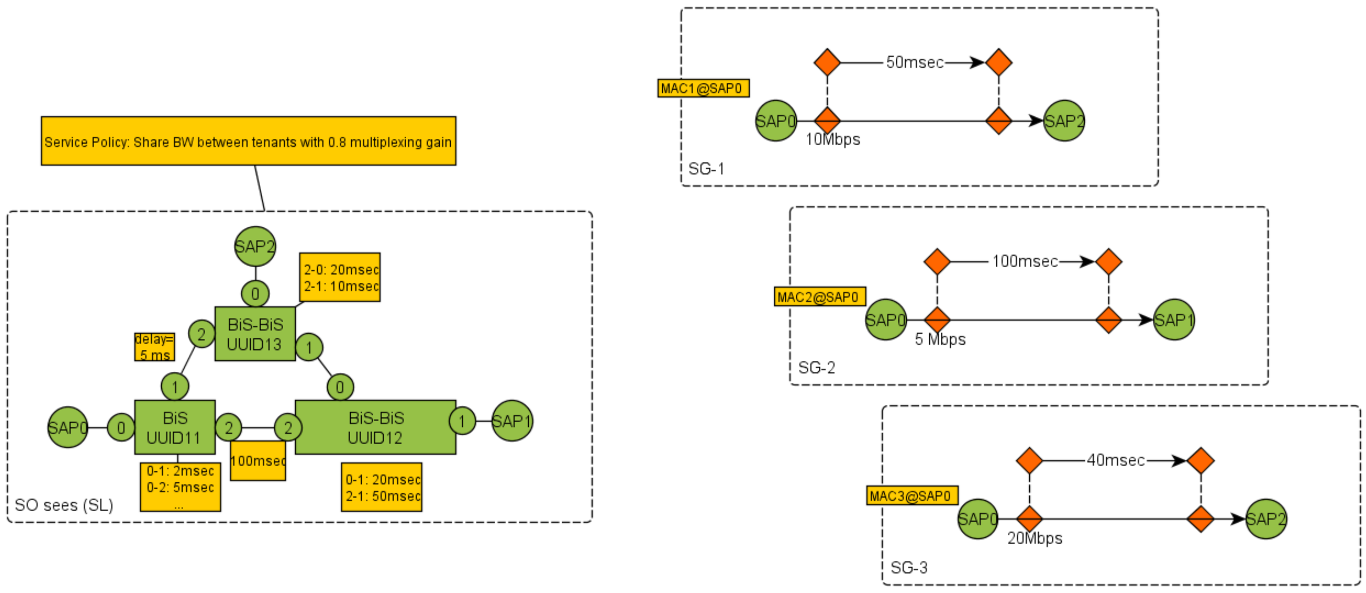


Figure 4.14: Infrastructure (left), and Service graph requests (right) for the resource sharing example

The corresponding messages describing the resource sharing can be found in 0.

The resulting configuration of the infrastructure after each update is shown in Figure 4.15.

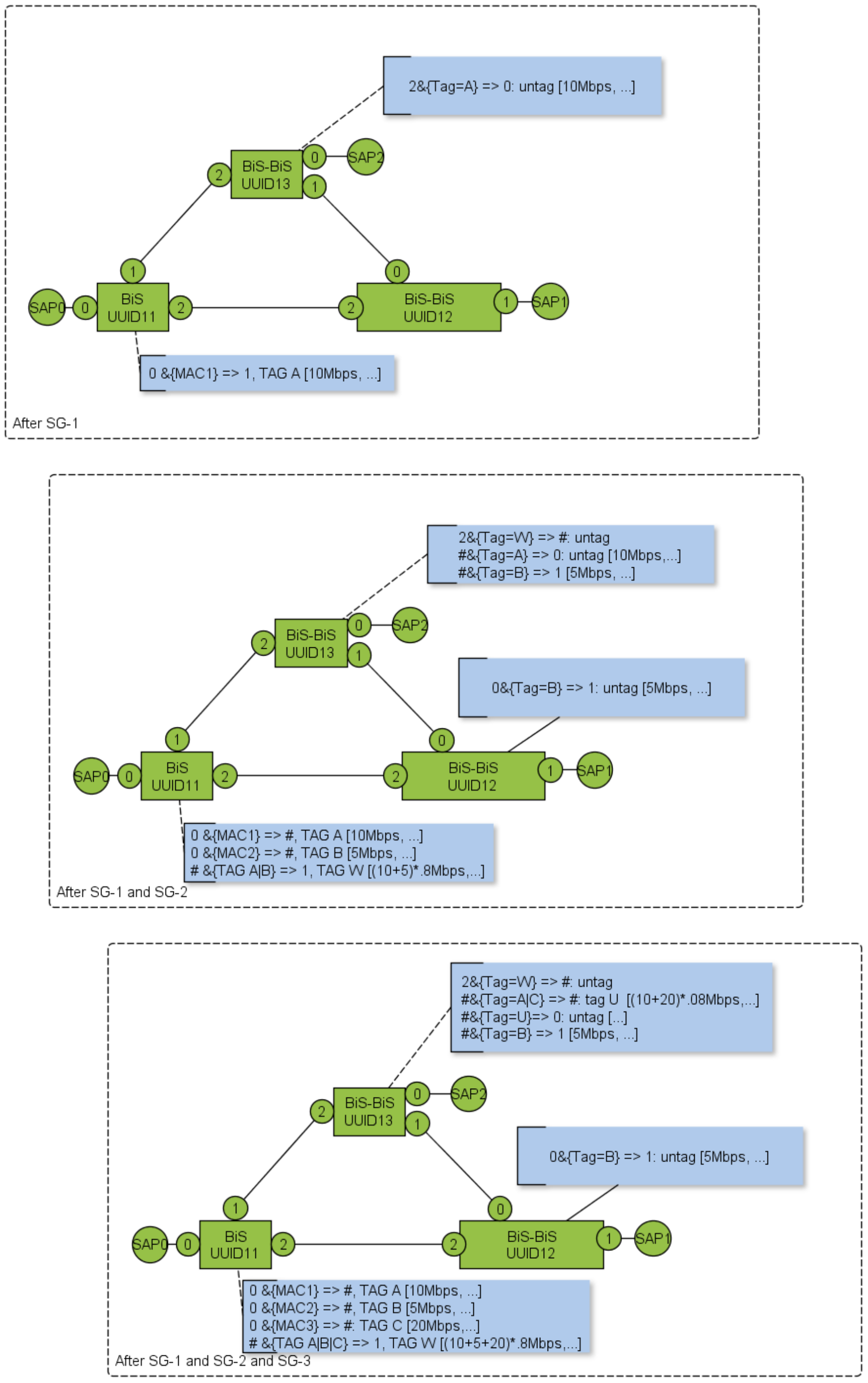

2&{Tag=A} => 0: untag [10Mbps, ...]
BiS-BiS UUID13
SAP2
BiS UUID11
SAP0
BiS-BiS UUID12
SAP1
0 &{MAC1} => 1, TAG A [10Mbps, ...]
After SG-1
2&{Tag=W} => #: untag
#&{Tag=A} => 0: untag [10Mbps,...]
#&{Tag=B} => 1 [5Mbps, ...]
0&{Tag=B} => 1: untag [5Mbps, ...]
0 &{MAC1} => #, TAG A [10Mbps, ...]
0 &{MAC2} => #, TAG B [5Mbps, ...]
# &{TAG A|B} => 1, TAG W [(10+5)*.8Mbps,...]
After SG-1 and SG-2
2&{Tag=W} => #: untag
#&{Tag=A|C} => #: tag U [(10+20)*.08Mbps,...]
#&{Tag=U}=> 0: untag [...]
#&{Tag=B} => 1 [5Mbps, ...]
0&{Tag=B} => 1: untag [5Mbps, ...]
0 &{MAC1} => #, TAG A [10Mbps, ...]
0 &{MAC2} => #, TAG B [5Mbps, ...]
0 &{MAC3} => #: TAG C [20Mbps,...]
# &{TAG A|B|C} => 1, TAG W [(10+5+20)*.8Mbps,...]
After SG-1 and SG-2 and SG-3

Figure 4.15: Result of subsequent requests in the Infrastructure: a resource sharing example

The examples above show link sharing. NF sharing can be described similarly.

4.8 The full data model

The combined YANG model containing the various options described in the previous sections can be found in the appendix A.2.1. The simplified tree view of the model is shown in Table 4.7.

Table 4.7: Combined model tree view

`+--rw virtualizer`
`   +--rw id?      string`
`   +--rw name?    string`
	NF-FG header information

`   +--rw nodes`
`   |  +--rw node* [id]`
`   |     +--ro id              string`
`   |     +--ro name?           string`
`   |     +--ro type            string`
	Infrastructure nodes (BiS-BiS)

`   |     +--ro ports`
`   |     |  +--ro port* [id]`
`   |     |     +--ro id            string`
`   |     |     +--ro name?         string`
`   |     |     +--ro port_type?    string`
`   |     |     +--ro (port-type)?`
	Infrastructure ports of the BiS-BiS.
	
	Port type can be SAP

`   |     +--ro links`
`   |     |  +--ro link* [src dst]`
`   |     |     +--ro id?          string`
`   |     |     +--ro name?        string`
`   |     |     +--ro src          port-ref`
`   |     |     +--ro dst          port-ref`
`   |     |     +--ro resources`
`   |     |        +--ro delay?       string`
`   |     |        +--ro bandwidth?   string`
	Virtual internal links of the infrastructure BiS-BiS to express e.g. delays between ports

`   |     +--ro resources`
`   |     |  +--ro cpu        string`
`   |     |  +--ro mem        string`
`   |     |  +--ro storage    string`
	Available resources of the BiS-BiS

`   |     +--rw NF_instances`
`   |     |  +--rw node* [id]`
`   |     |     +--rw id           string`
`   |     |     +--rw name?        string`
`   |     |     +--rw type         string`
	NFs actually running or to be deployed on the BiS-BiS

`   |     |     +--rw ports`
`   |     |     |  +--rw port* [id]`
`   |     |     |     +--rw id            string`
`   |     |     |     +--rw name?         string`
`   |     |     |     +--rw port_type?    string`
	Ports of the NF

`   |     |     +--rw links`
`   |     |     |  +--rw link* [src dst]`
`   |     |     |     +--rw id?          string`
`   |     |     |     +--rw name?        string`
`   |     |     |     +--rw src          port-ref`
`   |     |     |     +--rw dst          port-ref`
`   |     |     |     +--rw resources`
`   |     |     |        +--rw delay?       string`
`   |     |     |        +--rw bandwidth?   string`
	Virtual internal links of the NF to express e.g. delay requirement between NF ports

`   |     |     +--rw resources`
`   |     |        +--rw cpu        string`
`   |     |        +--rw mem        string`
`   |     |        +--rw storage    string`
	Required resources of the NF

`   |     +--ro capabilities`
`   |     |  +--ro supported_NFs`
`   |     |     +--ro node* [id]`
`   |     |        +--ro id           string`
`   |     |        +--ro name?        string`
`   |     |        +--ro type         string`
`   |     |        +--ro ports`
	Capabilities of the BiS-BiS, e.g. which NFs are supported. The whole NF structure is re-used here, but e.g. internal Links are likely

```
|      |         |  +--ro port* [id]
|      |         |     +--ro id             string
|      |         |     +--ro name?          string
|      |         |     +--ro port_type?     string
|      |         +--ro links
|      |         |  +--ro link* [src dst]
|      |         |     +--ro id?           string
|      |         |     +--ro name?         string
|      |         |     +--ro src           port-ref
|      |         |     +--ro dst           port-ref
|      |         |     +--ro resources
|      |         |        +--ro delay?       string
|      |         |        +--ro bandwidth?   string
|      |         +--ro resources
|      |            +--ro cpu        string
|      |            +--ro mem        string
|      |            +--ro storage    string
```
	not to be used.

```
|     +--rw flowtable
|        +--rw flowentry* [port match action]
|           +--rw port         port-ref
|           +--rw match        string
|           +--rw action       string
|           +--rw resources
|              +--rw delay?       string
|              +--rw bandwidth?   string
```
	Flow entries of/for the BiS-BiS, between infrastructure or NF ports. For a request, resources can express e.g. delay requirement between ports.

```
+--ro links
      +--ro link* [src dst]
            +--ro id?           string
            +--ro name?         string
            +--ro src           port-ref
            +--ro dst           port-ref
            +--ro resources
               +--ro delay?       string
               +--ro bandwidth?   string
```
	Infrastructure Links between ports of different BiS-BiS'

4.9 Relation to NETCONF

The above presented model is used in both directions of the communication. There are read-only (ro) marked fields, which are to be modified only by the Virtualizer, and read by the above layer. These fields belong to the infrastructure view representation, used e.g. in steps 2 and 24 of Figure 4.5. The other type of fields, read-write (rw) can be written by the above layer and represent a request towards the Virtualizer, corresponding to steps 5 and 25 of Figure 4.5. These second type fields, however, can also be parts of an infrastructure report, e.g. when reporting back the deployed, previously requested NFs.

There is a single data store (NETCONF terminology for the database) of a Virtualizer, meaning that all infrastructure elements and all requests are located in the same structure, i.e., a single instance of Virtualizer, containing many elements according to Table 4.7. However, the full instance doesn't have to be communicated each time, because reads (get-config) and writes (edit-config) can be scoped to a part of the whole data store.

5 Conclusion

Although both models have been evolving based on prototype experiments, some provisional conclusions can be made at this point in time. Potential updates resulting from the integration effort and final conclusions on the models will be given in the future Deliverable D3.5. Below we give a short overview between both models focusing on the core parts of the UNIFY programming and orchestration architecture.

- Data-model differences
    - The service-centric **(SC) model** stores *service link information as well as requirements on these links within the NF-FG data structure,* while the Virtualizer-based **(VB) model** does not. In the latter approach, the mapping (and budgeting) between the service links and the NF-FG attributes is locally maintained in the Service (Adaptation) Layer.
    - The **SC model** does introduce a *difference between SAPs and EPs*. EPs are available in the infrastructure layer; SAPs are service constructs dependent on the particular service. A mapping/translation between both is supposed to happen at the Service (Adaptation) Layer.
    - The **SC model** only maps service links to a sequence of BiS-BiS components which the links are expected to cross. *BiS-BiS components are not configured as switches*, how traffic steering is implemented when crossing the BiS-BiSes is the responsibility of the CA and network control components, once the RO has determined routing.
    - In general, the philosophy of the **SC model** seems to be more *declarative*, storing SG attributes in the NF-FG to *enable maximal flexibility for lower layers* to benefit from this information. The **VB model** philosophy on the other hand follows *a more prescriptive,* s*tep-wise refining approach,* focusing on the isolation of responsibilities between layers and scalability of the overall hierarchic approach.
- Impact on KPI fulfilment process
    - In the **SC model**, the Service (Layer) Orchestration is limited to mapping NFs to BiS-BiSes and mapping service links to sequences of BiS-BiSes. KPI requirements are merely re-formulated over NFs and service links, even when they cross multiple domains/BiS-BiSes.
    - In the **VB model** initial concept, Service (Layer) Orchestration also the *translation of groups of service attributes into NF-FG components* that are mapped on a Virtualizer. The *latter involves splitting of KPI requirements* related to service components when, for example service links with particular delay requirements are split over multiple links between BiS-BiSes in the considered virtualizer. This became relaxed with the introduction of constraints handling.

- The **VB model** initial philosophy *encourages a pre-planned scheduled orchestration process* where accurate knowledge of infrastructure capabilities is available in order to maximally guarantee fast and successful orchestration. The **SC model** philosophy is to delay the splitting decision on KPI requirements as long as possible, with the intention of avoiding premature suboptimal decisions, potentially resulting in a more trial-and-error, iterative orchestration and provisioning approach.

- Impact on resource orchestration processes and capabilities
  - From an *algorithmic point of view the difference between both approaches is limited*, because service-driven constraints and KPI's are possible in both models.
  - **SC model view**: The RO and subsequent layers in the SC model might perform (re-)optimizations relying on service attributes available in the NF-FG data structure without requiring interaction with higher layers.
  - **VB model view:** The RO and subsequent layers by design should be service-agnostic and focus on the orchestration of resources (not services). This reduces information/constraint overload, improves scalability of the processes and ensures a pure client-server relationship between layers. Lower-layer providers/domains (e.g., UN) could be part of different business entities, and should not be aware about which service/customer exactly they serve. Their focus should be on delivering requested resources.
- Impact on monitoring and troubleshooting
  - Monitoring and troubleshooting processes are mainly impacted with respect to service-related monitoring. In order to monitor or troubleshoot service-related parameters, the **SC model** might *reduce inter layer interaction*, because service-related information is available in the NF-FG structure across different layers, while the **VB model** by design *enforces isolation of information* (separation of concerns) between layers. Troubleshooting services at service-level does *might require to retrieve appropriate mappings across layers*.
- Impact on sharing or resources
  - *Both models enable for sharing of resources*, although the **VB model** is more *prescriptive* in how this must be achieved (e.g., enable link sharing through the use of forwarding rules on the BiS-BiS), whereas the **SC model** follows a more *declarative* approach (merely stating that it must be shared).

    - Sharing of resources seems to be largely independent of the model, but rather on how underlying blocks implement it.

- Current use of the models in prototype implementations
    - The **SC model** is used in ELwUD and ESCAPEv1,
    - The **VB model** approach is used in ESCAPEv2, FROG, and OS-ODL

6 References

[D2.2] "*Deliverable 2.2: Functional Architecture*" Tech. rep. UNIFY Project, 2014.

[D3.1] "*Deliverable 3.1: Programmability framework*". Tech. rep. UNIFY Project, 2014.

[D4.1] "*Deliverable 4.1: Initial requirements for the SP-DevOps concept, universal node capabilities and proposed tools*". Tech. rep. UNIFY Project, 2014.

[ETSIMANO] ETSI GS NFV-MAN 001,"*Network Functions Virtualisation (NFV); Management and Orchestration*", Accessed 2015-05-06, <http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf>

[Neutron] OpenStack, "*Neutron – OpenStack*", Accessed: 2015-04-27, https://wiki.openstack.org/wiki/Neutron

[Nova] OpenStack, "*Nova – OpenStack*", Accessed: 2015-04-27, https://wiki.openstack.org/wiki/Nova

[OSheat] OpenStack, "*Heat-OpenStack Orchestration*", Accessed: 2015-04-27, <https://wiki.openstack.org/wiki/Heat>

[OShot] OpenStack. "*Heat Orchestration Template (HOT) Guide*." OpenStack Homepage. Accessed: 2015-04-27 <http://docs.openstack.org/developer/heat/template_guide/hot_guide.html>.

[TOSCA] OASIS Standard, "*Topology and Orchestration Specification for Cloud Applications Version 1.0*", 2013, Accessed: 2015-04-27, http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf

Annex 1 Service-oriented and Virtualizer-based NF-FG models

A.1.1 Service-oriented NF-FG model defined in YANG

Table A.1: Service-oriented NF-FG YANG Model

```
module nffg {
  namespace "https://gitlab.fp7-unify.eu/wp2/nf-fg";
  prefix nffg;

  organization "UPV/EHU";
  contact
    "Jokin Garay <jokin.garay@ehu.eus>,
     Jon Matias <jon.matias@ehu.eus>,";
  description
    "Network Function Forwarding Graph (NF-FG) data model";

  revision 2015-06-30 {
    reference
      "Data model updated for D3.2a";
  }
  revision 2015-04-22 {
    reference
      "Initial version";
  }

  grouping link {
    description
      "Link definition";
    leaf source {
      type string;
      mandatory true;
      description
       "Source node";
    }
    leaf src_port {
      type string;
      mandatory true;
      description
       "Source port";
    }
    leaf target {
      type string;
      mandatory true;
      description
       "Target node";
    }
    leaf dst_port {
      type string;
      mandatory true;
      description
       "Target port";
    }
  }

  grouping ports {
    description
     "Port list";
    list ports {
      key "id";
      min-elements 1;
      description
       "Ports";
      leaf id {
        type string;
        mandatory true;
        description
         "Port id";
      }
      leaf-list property {
        type string;
        min-elements 1;
        description
         "Port properties";
      }
    }
  }

  grouping ports_flows {
    description
     "Port list with flows";
    list ports {
      key "id";
      min-elements 1;
      description
       "Port list";
      container flow_rules {
        uses flow_rules;
        description
         "Flow rules";
      }
      leaf id {
        type string;
        mandatory true;
        description
         "Port id";
      }
      leaf-list property {
        type string;
        min-elements 1;
        description
         "Port properties";
      }
    }
  }

  grouping flow_rules {
    description
     "Flow rule list";
    list flowrules {
      key "match action";
      description
       "Flow rule";
      leaf match {
        type string;
        mandatory true;
        description
         "Flow space to match";
      }
      leaf action {
        type string;
        mandatory true;
        description
         "Action to perform";
      }
    }
  }

  grouping node_resource {
    description
     "Node resource description";
    container compute {
      description
       "Compute resource description";
      leaf cpu {
        type string;
        description
         "CPU";
      }
      leaf memory {
        type string;
        description
         "Memory";
      }
      leaf capacity {
        type string;
        description
         "Capacity";
      }
    }
    container storage {
      description
       "Storage resource description";
      leaf hdd {
        type string;
        description
         "HDD";
      }
    }
    container networking {
      description
       "Networking resource description";
      uses link_resource;
      leaf programmability {
        type string;
        description
         "Programmability";
      }
      leaf isolation {
        type string;
        description
         "Isolation";
      }
    }
    uses res_constraints;
  }

  grouping res_constraints {
    description
     "Resource constraint grouping";
    container constraints {
      description
       "Resource constraints";
      leaf resiliency {
        type string;
        description
         "Resiliency constraints";
      }
      leaf location {
        type string;
        description
         "Location constraints";
      }
      leaf privacy {
        type string;
        description
         "Privacy constraints";
      }
    }
  }

  grouping link_resource {
    description
     "Link resource description";
    leaf delay {
      type string;
      description
       "Delay";
    }
    leaf bandwidth {
      type string;
      description
       "Bandwidth";
    }
  }

  grouping monitoring {
    description
     "Monitoring group";
    list monitoring_params {
      key "KPI_desc KPI_value";
      description
       "Monitoring parameters";
      leaf KPI_desc {
        type string;
        mandatory true;
        description
         "KPI description";
      }
      leaf KPI_value {
        type string;
        mandatory true;
        description
         "KPO value";
      }
      leaf scope {
        type string;
        mandatory true;
        description
         "KPI scope";
      }
      leaf priority {
        type string;
        description
         "KPI priority";
      }
      leaf MEASURE {
        type string;
        description
         "MEASURE script";
      }
    }
  }

  container nffg {
    description
      "Network Function Forwarding Graph container";
    container parameters {
      description
       "NF-FG general parameters";
      leaf id {
        type string;
        mandatory true;
        description
         "NF-FG id";
      }
      leaf name {
        type string;
        description
         "NF-FG name";
      }
      leaf version {
        type string;
        mandatory true;
        description
         "NF-FG version";
      }
      leaf tenant {
        type string;
        description
         "NF-FG tenant";
      }
      leaf template {
        type string;
        description
         "NF-FG template";
      }
      uses res_constraints;
    }
    container monitoring {
      description
        "Monitoring parameters for whole NF-FG";
      uses monitoring;
    }
    container sg {
      description
       "Service Graph";
      list nfs {
        key "id";
        description
          "Network Function (NF) nodes in the Service Graph";
        container specification {
          description
           "NF specification";
          leaf deployment_type {
            type string;
            description
             "NF deployment type";
          }
          leaf image_uri {
            type string;
            description
             "NF image URI";
          }
          leaf vnf_type {
            type string;
            description
             "VNF type";
          }
        }
        container resources {
          description
            "Resource information of NF";
          container requirements {
            uses node_resource;
            description
             "Resource requirements of NF";
          }
          list assignment {
            key "inf_id";
            description
             "Resource assigment to NF";
            leaf inf_id {
              type string;
              mandatory true;
              description
               "Assigned Inf Node id";
            }
            leaf inf_domain {
              type string;
              mandatory true;
              description
               "Assigned Endpoint domain";
            }
            leaf nffg_id {
              type string;
              description
               "Assigned NF-FG for NF sharing";
            }
            leaf running_nf_id {
              type string;
              description
               "Assigned NF for NF sharing";
            }
          }
        }
        container ports {
          uses ports;
          description
           "NF ports";
        }
        container nf_monitoring {
          description
            "Monitoring parameters for single NFs";
          uses monitoring;
        }
        leaf id {
          type string;
          mandatory true;
          description
           "NF id";
        }
        leaf name {
          type string;
          description
           "NF name";
        }
        leaf functional_type {
          type string;
          mandatory true;
          description
           "NF functional type";
        }
      }
      list saps {
        key "id";
        min-elements 1;
        description
          "SAP nodes in the Service Graph";
        container ports {
          uses ports_flows;
          description
           "SAP ports";
        }
        container resources {
          description
            "Resource information of SAP";
          container requirements {
            uses node_resource;
            description
             "Resource requirements of SAP";
          }
          list assignment {
            key "ep_id";
            description
             "Resource assigment to SAP";
            leaf ep_id {
              type string;
              mandatory true;
              description
               "Assigned Endpoint id";
            }
            leaf inf_domain {
              type string;
              mandatory true;
              description
               "Assigned Endpoint domain";
            }
            list ep_ports {
              key "ep_port";
              description
               "Assigned Endpoint ports";
              leaf ep_port {
                type string;
                mandatory true;
                description
                 "Ednpoint port";
              }
            }
          }
        }
        leaf id {
          type string;
          mandatory true;
          description
           "SAP id";
        }
        leaf name {
          type string;
          description
           "SAP Name";
        }
      }
      list sls {
        key "source src_port";
        description
         "Service links";
        container resources {
          description
            "Resource information of Service link";
          container requirements {
            uses link_resource;
            description
             "Resource requirements of Service link";
          }
          list assignment {
            key "source src_port";
            description
             "Resource assigment to Service link";
            uses link;
          }
        }
        leaf flowclass {
          type string;
          description
           "Flowclass of the Service link";
        }
        uses link;
      }
    }
    container rg {
      description
       "Resource Graph";
      list infs {
        key "id";
        min-elements 1;
        description
          "Infrastructure nodes in the Resource Graph";
        container resources {
          description
           "Resource description of Infrastucture Nodes";
          uses node_resource;
        }
        leaf id {
          type string;
          mandatory true;
          description
           "Inf Node id";
        }
        leaf domain {
          type string;
          mandatory true;
          description
           "Inf Node domain";
        }
        leaf name {
          type string;
          description
           "Inf Node name";
        }
        leaf type {
          type string;
          mandatory true;
          description
           "Inf Node type";
        }
        list ports {
          key "id";
          min-elements 1;
          description
           "Inf Node ports";
          leaf id {
            type string;
            mandatory true;
            description
             "Inf Node port id";
          }
          leaf-list property {
            type string;
            min-elements 1;
            description
              "Port properties";
          }
        }
      }
      list eps {
        key "id";
        min-elements 1;
        description
          "Endpoint nodes in the Resource Graph";
        container resources {
          description
            "Endpoint resource description";
          uses node_resource;
        }
        leaf id {
          type string;
          mandatory true;
          description
            "Endpoint id";
        }
        leaf domain {
          type string;
          description
            "Endpoint domain";
        }
        leaf name {
          type string;
          description
            "Endpoint name";
        }
        leaf type {
          type string;
          mandatory true;
          description
            "Endpoint type";
        }
        list ports {
          key "id";
          min-elements 1;
          description
            "Endpoint ports";
          leaf id {
            type string;
            mandatory true;
            description
            "Endpoint port id";
          }
          leaf-list property {
            type string;
            min-elements 1;
            description
              "Port properties";
          }
        }
      }
      list ils {
        key "source src_port";
        description
          "Infrastructure Links in Resource Graph";
        container resources {
          description
            "Resource description of Inf Links";
          uses link_resource;
        }
        uses link;
      }
    }
  }
}
```

Annex 2 Virtualizer-based NF-FG model

This Annex contains the proposed virtualizer-based NF-FG model and gives usage examples.

A.2.1 Virtualizer-based NF-FG model defined in YANG

The formal specification of the virtualizer-based NF-FG model is given below in Table A.2, containing element to describe both infrastructure reports and deployment requests.

Table A.2: YANG model of the Virtualizer

```
module virtualizer {
  namespace "urn:unify:virtualizer";
  prefix "virtualizer";

  revision 2015-05-07
  {
    description "Virtualizer's data model";
  }

  //======================== REUSABLE GROUPS ================================

  grouping id-name {
    leaf id { type string; }
    leaf name { type string;}
  }

  grouping id-name-type {
    uses id-name;
    leaf type {
      type string;
      mandatory true;
    }
  }

  // ------------ PORTS -------

  typedef port-ref {
    type string;
  }

  grouping port-sap-vxlan {
    leaf vxlan {type string;} // for example
    // container vx_lan {
    //   leaf remote_ip { type string; }
    //   leaf local_ip { type string; }
    //   leaf tunnel_key { type uint32; }
    // }
  }

  grouping port-type {
    leaf port_type {type string;} // TODO: enumerated
    choice port-type {
      description "Different port types: abstract and SAPs";
      case port-abstract {
        leaf capability { type string; }
      }
      case port-sap {
        choice sap-type {
          case vx-lan { // for example
            uses port-sap-vxlan;
          }
        }
      }
    }
  }

  grouping port {
    uses id-name;
    uses port-type;
  }

  // ------------ FLOW CONTROLS -------

  grouping flowentry {
    leaf port {
      type port-ref;
      mandatory true;
    }
    leaf match {
      type string;
      mandatory true;
    }
    leaf action {
      type string;
      mandatory true;
    }
    container resources{
      uses link-resource;
    }

  }

  grouping flowtable {
    container flowtable {
      list flowentry {
        key "port match action";
        uses flowentry;
      }
    }
  }

  // ------------ LINKS  -------

  grouping link-resource {
    leaf delay {
      type string;
      mandatory false;
    }
    leaf bandwidth {
      type string;
      mandatory false;
    }
  }

  grouping link {
    uses id-name;
    leaf src {
      type port-ref;
    }
    leaf dst {
      type port-ref;
    }
    container resources{
      uses link-resource;
    }
  }

  grouping links {
    container links {
      list link {
        key "src dst";
        uses link;
      }
    }
  }

  // ---------- CAPABILITIES  -------------------

  grouping capabilities {
    container supported_NFs { // if supported NFs are enumerated
      list node{
        key "id";
        uses node;
      }
    }
    // TODO: add other capabilities
  }

  // ---------- NODE -------------------

  grouping software-resource {
    leaf cpu {
      type string;
      mandatory true;
    }
    leaf mem {
      type string;
      mandatory true;
    }
    leaf storage {
      type string;
      mandatory true;
    }
  }

  grouping node {
    description "Any node: infrastructure or NFs";
    uses id-name-type;
    container ports {
      list port{
        key "id";
        uses port;
      }
    }
    uses links;
    container resources{
      uses software-resource;
    }
  }

  grouping infra-node { // they can contain other nodes (as NFs)
    uses node;
    container NF_instances {
      list node{
        key "id";
        uses node;
      }
    }
    container capabilities {
      uses capabilities;
    }
    uses flowtable;
  }



  //=============== NF-FG: Virtualizer and the Mapped request ====================

  container virtualizer {
    description "Container for a single virtualizer";
    uses id-name;

    container nodes{
      list node{ // infra nodes
        key "id";
        uses infra-node;
      }
    }
    uses links; // infra links
  }
}
```

The above described model can be visualized as shown in Figure A2-6.1.

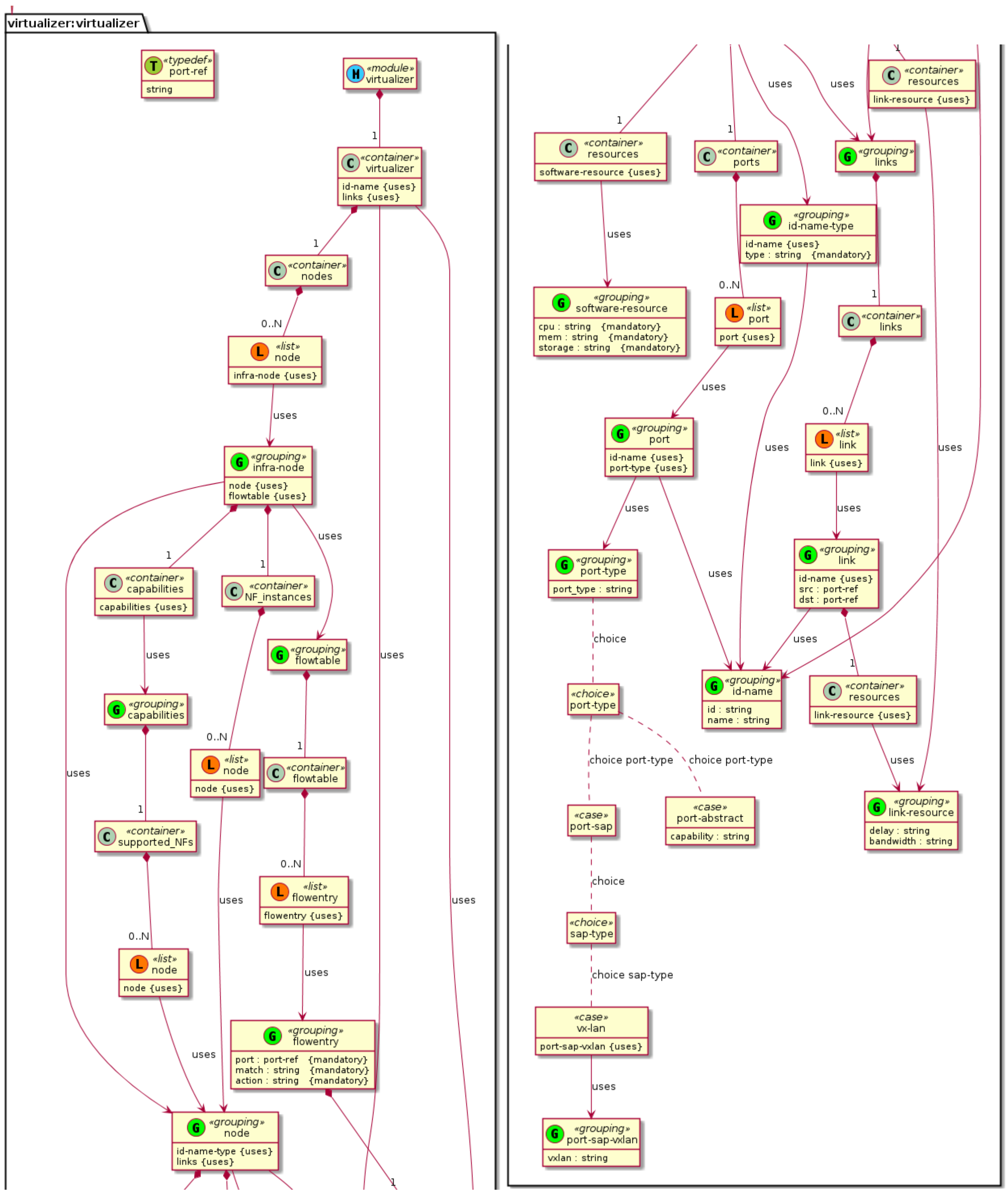


Figure A2-6.1: UML diagram of the YANG model

A.2.2 Single node infrastructure report example

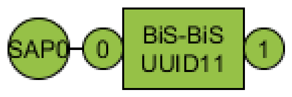


Figure A2-6.2: Single node

The infrastructure of Figure A2-6.2 can have the following XML description according to our YANG model, see Table A.3.

Table A.3: Single node infrastructure report example

```
<virtualizer xmlns="urn:unify:virtualizer">
    <id>UUID001</id>
    <name>Single node simple infrastructure report</name>
    <nodes>
        <node>
            <id>UUID11</id>
            <name>single BiS-BiS node</name>
            <type>BisBis</type>
            <ports>
                <port>
                    <id>0</id>
                    <name>SAP0 port</name>
                    <port_type>port-sap</port_type>
                    <vxlan>...</vxlan>
                </port>
                <port>
                    <id>1</id>
                    <name>North port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
                <port>
                    <id>2</id>
                    <name>East port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
            </ports>
            <resources>
                <cpu>20</cpu>
                <mem>64 GB</mem>
                <storage>100 TB</storage>
            </resources>
        </node>
    </nodes>
</virtualizer>
```

A.2.3 3-node infrastructure report example

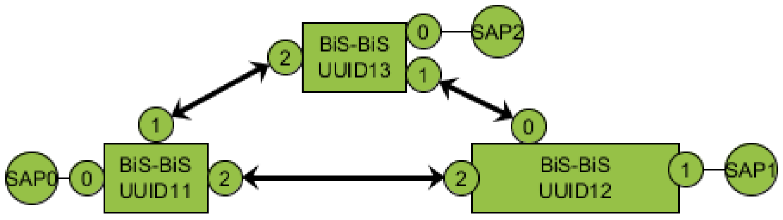


Figure A2-6.3: 3 nodes

The infrastructure of Figure A2-6.3 can have the following XML description according to our YANG model, see Table A.4.

Table A.4: 3-node infrastructure report example

```
<virtualizer xmlns="urn:unify:virtualizer">
    <id>UUID002</id>
    <name>3-node simple infrastructure report</name>
    <nodes>
        <node>
            <id>UUID11</id>
            <name>West BiS-BiS node</name>
            <type>BiSBiS</type>
            <ports>
                <port>
                    <id>0</id>
                    <name>SAP0 port</name>
                    <port_type>port-sap</port_type>
                    <vxlan>...</vxlan>
                </port>
                <port>
                    <id>1</id>
                    <name>North port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
                <port>
                    <id>2</id>
                    <name>East port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
            </ports>
            <resources>
                <cpu>20</cpu>
                <mem>64 GB</mem>
                <storage>100 TB</storage>
            </resources>
        </node>
        <node>
            <id>UUID12</id>
            <name>East BiS-BiS node</name>
            <type>BiSBiS</type>
            <ports>
```

```
                <port>
                    <id>1</id>
                    <name>SAP1 port</name>
                    <port_type>port-sap</port_type>
                    <vxlan>...</vxlan>
                </port>
                <port>
                    <id>0</id>
                    <name>North port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
                <port>
                    <id>2</id>
                    <name>West port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
            </ports>
            <resources>
                <cpu>10</cpu>
                <mem>32 GB</mem>
                <storage>100 TB</storage>
            </resources>
        </node>
        <node>
            <id>UUID13</id>
            <name>North BiS-BiS node</name>
            <type>BiSBiS</type>
            <ports>
                <port>
                    <id>0</id>
                    <name>SAP2 port</name>
                    <port_type>port-sap</port_type>
                    <vxlan>...</vxlan>
                </port>
                <port>
                    <id>1</id>
                    <name>East port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
                <port>
                    <id>2</id>
                    <name>West port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
            </ports>
            <resources>
                <cpu>20</cpu>
                <mem>64 GB</mem>
                <storage>1 TB</storage>
            </resources>
        </node>
    </nodes>
    <links>
        <link>
            <id>0</id>
            <name>Horizontal link</name>
            <src>../../nodes/node[id=UUID11]/ports/port[id=2]</src>
            <dst>../../nodes/node[id=UUID12]/ports/port[id=2]</dst>
            <resources>
                <delay>2 ms</delay>
                <bandwidth>10 Gb</bandwidth>
            </resources>
        </link>
        <link>
            <id>1</id>
            <name>West link</name>
            <src>../../nodes/node[id=UUID11]/ports/port[id=1]</src>
            <dst>../../nodes/node[id=UUID13]/ports/port[id=2]</dst>
            <resources>
                <delay>5 ms</delay>
                <bandwidth>10 Gb</bandwidth>
            </resources>
        </link>
        <link>
            <id>2</id>
            <name>East link</name>
            <src>../../nodes/node[id=UUID12]/ports/port[id=0]</src>
            <dst>../../nodes/node[id=UUID13]/ports/port[id=1]</dst>
            <resources>
                <delay>2 ms</delay>
                <bandwidth>5 Gb</bandwidth>
            </resources>
        </link>
    </links>
</virtualizer>
```

A.2.4 Single node with delay matrix example

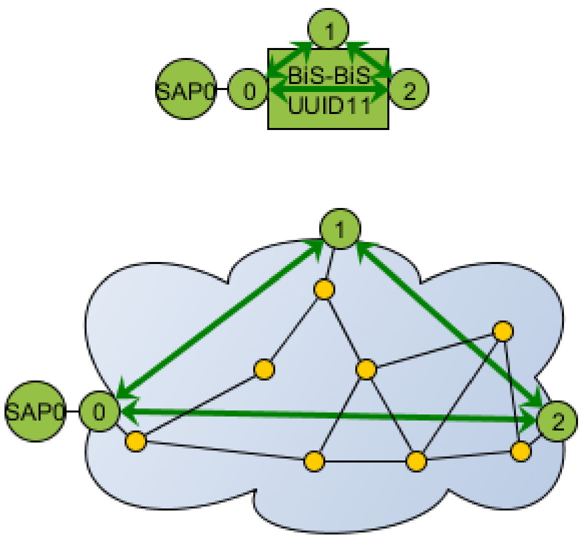


Figure A2-6.4: Delay matrix

The infrastructure of the lower part of Figure A2-6.4, represented by the Virtualizer as the upper part of Figure A2-6.4, can have the following XML description according to our YANG model, see Table A.5.

Table A.5: Delay matrix infrastructure report example

```
<virtualizer xmlns="urn:unify:virtualizer">
    <id>UUID001</id>
    <name>Single node with link internal delays infrastructure report</name>
    <nodes>
        <node>
            <id>UUID11</id>
            <name>single BiS-BiS node</name>
            <type>BiSBiS</type>
            <ports>
                <port>
                    <id>0</id>
                    <name>SAP0 port</name>
                    <port_type>port-sap</port_type>
                    <vxlan>...</vxlan>
                </port>
                <port>
                    <id>1</id>
                    <name>North port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
                <port>
                    <id>2</id>
                    <name>East port</name>
                    <port_type>port-abstract</port_type>
                    <capability>...</capability>
                </port>
            </ports>
            <links>
                <link>
                    <id>int0</id>
                    <name>internal horizontal</name>
                    <src>../../ports/port[id=0]</src>
                    <dst>../../ports/port[id=2]</dst>
                    <resources>
                        <delay>1 ms</delay>
                        <bandwidth>40 Gb</bandwidth>
                    </resources>
                </link>
                <link>
                    <id>int1</id>
                    <name>internal left</name>
                    <src>../../ports/port[id=0]</src>
                    <dst>../../ports/port[id=1]</dst>
                    <resources>
                        <delay>5 ms</delay>
                        <bandwidth>10 Gb</bandwidth>
                    </resources>
                </link>
                <link>
                    <id>int2</id>
                    <name>internal right</name>
                    <src>../../ports/port[id=1]</src>
                    <dst>../../ports/port[id=2]</dst>
                    <resources>
                        <delay>2 ms</delay>
                        <bandwidth>81 Gb</bandwidth>
                    </resources>
                </link>
            </links>
            <resources>
                <cpu>20</cpu>
                <mem>64 GB</mem>
                <storage>100 TB</storage>
            </resources>
        </node>
    </nodes>
</virtualizer>
```

A.2.5 Single request example

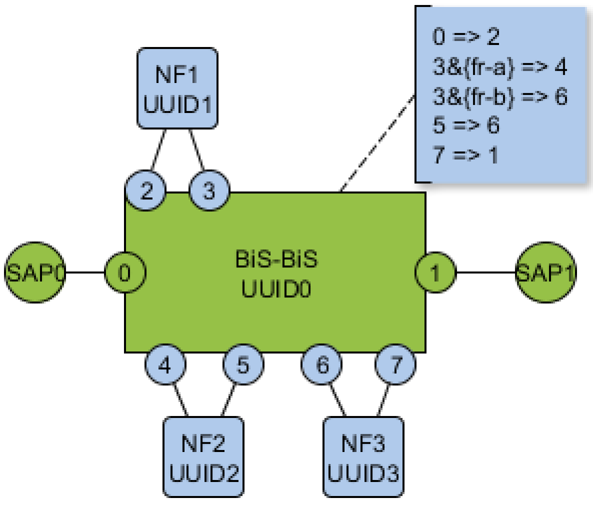


Figure A2-6.5: Simple request of 3NFs on a single BiS-BiS

The infrastructure of Figure A2-6.5 can have the following XML description according to our YANG model, see Table A.6.

Table A.6: Single request example

```
<virtualizer xmlns="urn:unify:virtualizer">
    <id>UUID001</id>
    <name>Single node simple request</name>
    <nodes>
        <node>
            <id>UUID11</id>
            <NF_instances>
                <node>
                    <id>NF1</id>
                    <name>first NF</name>
                    <type>Parental control B.4</type>
                    <ports>
                        <port>
                            <id>2</id>
                            <name>in</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                        <port>
                            <id>3</id>
                            <name>out</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                    </ports>
                    <!-- example <resources> could come here -->
                </node>
                <node>
                    <id>NF2</id>
                    <name>cache</name>
                    <type>Http Cache 1.2</type>
                    <ports>
                        <port>
                            <id>4</id>
                            <name>in</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                        <port>
                            <id>5</id>
                            <name>out</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                    </ports>
                    <!-- example <resources> could come here -->
                </node>
                <node>
                    <id>NF3</id>
                    <name>firewall</name>
                    <type>Stateful firewall C</type>
                    <ports>
                        <port>
                            <id>6</id>
                            <name>in</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                        <port>
                            <id>7</id>
                            <name>out</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                    </ports>
                    <!-- example <resources> could come here -->
                </node>
            </NF_instances>
            <flowtable>
                <flowentry>
                    <port>../../ports/port[id=0]</port>
                    <match>*</match>

<action>output:../../NF_instances/node[id=NF1]/ports/port[id=2]</action>
                </flowentry>
                <flowentry>
                    <port>../../NF_instances/node[id=NF1]/ports/port[id=3]</port>
                    <match>fr-a</match>

<action>output:../../NF_instances/node[id=NF2]/ports/port[id=4]</action>
                </flowentry>
                <flowentry>
                    <port>../../NF_instances/node[id=NF1]/ports/port[id=3]</port>
                    <match>fr-b</match>

<action>output:../../NF_instances/node[id=NF3]/ports/port[id=6]</action>
                </flowentry>
                <flowentry>
                    <port>../../NF_instances/node[id=NF2]/ports/port[id=5]</port>
                    <match>*</match>
                    <action>output:../../ports/port[id=1]</action>
                </flowentry>
                <flowentry>
                    <port>../../NF_instances/node[id=NF3]/ports/port[id=7]</port>
                    <match>*</match>
                    <action>output:../../ports/port[id=1]</action>
                </flowentry>
            </flowtable>
        </node>
    </nodes>
</virtualizer>
```

A.2.6 Request with BiS-BiS internal links example

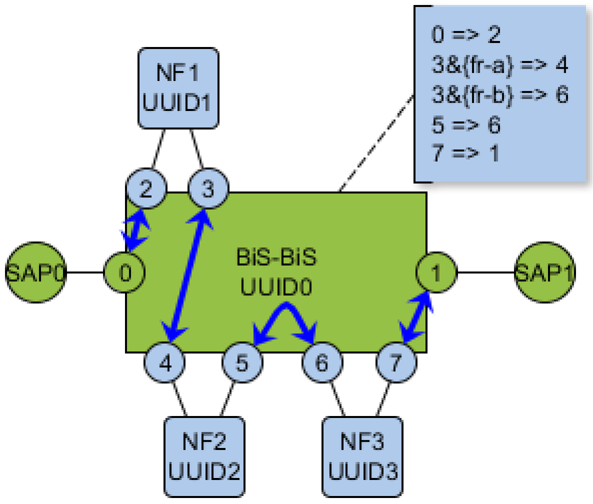


Figure A2-6.6: Request with BiS-BiS internal requirements

The infrastructure of Figure A2-6.6 can have the following XML description according to our YANG model, see Table A.7.

Table A.7: Request with Bis-Bis internal requirements example

```
<virtualizer xmlns="urn:unify:virtualizer">
    <id>UUID001</id>
    <name>Single node simple request with inter-virtual port delays</name>
    <nodes>
        <node>
            <id>UUID11</id>
            <NF_instances>
                <node>
                    <id>NF1</id>
                    <name>first NF</name>
                    <type>Parental control B.4</type>
                    <ports>
                        <port>
                            <id>2</id>
                            <name>in</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                        <port>
                            <id>3</id>
                            <name>out</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                    </ports>
                    <!-- example may contain <resources> here -->
                </node>
                <node>
                    <id>NF2</id>
                    <name>cache</name>
                    <type>Http Cache 1.2</type>
                    <ports>
                        <port>
                            <id>4</id>
                            <name>in</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                        <port>
                            <id>5</id>
                            <name>out</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                    </ports>
                    <!-- example may contain <resources> here -->
                </node>
            </NF_instances>
            <flowtable>
                <flowentry>
                    <port>../../ports/port[id=0]</port>
                    <match>*</match>

<action>output:../../NF_instances/node[id=NF1]/ports/port[id=2]</action>
                    <resources>
```

```
                        <delay>50 ms</delay>
                        <bandwidth>1 Mb</bandwidth>
                    </resources>
                </flowentry>
                <flowentry>
                    <port>../../NF_instances/node[id=NF1]/ports/port[id=3]</port>
                    <match>fr-a</match>

<action>output:../../NF_instances/node[id=NF2]/ports/port[id=4]</action>
                    <resources>
                        <delay>25 ms</delay>
                        <bandwidth>2 Mb</bandwidth>
                    </resources>
                </flowentry>
                <flowentry>
                    <port>../../NF_instances/node[id=NF1]/ports/port[id=3]</port>
                    <match>fr-b</match>

<action>output:../../NF_instances/node[id=NF3]/ports/port[id=6]</action>
                </flowentry>
                <flowentry>
                    <port>../../NF_instances/node[id=NF2]/ports/port[id=5]</port>
                    <match>*</match>
                    <action>output:../../ports/port[id=1]</action>
                </flowentry>
                <flowentry>
                    <port>../../NF_instances/node[id=NF3]/ports/port[id=7]</port>
                    <match>*</match>
                    <action>output:../../ports/port[id=1]</action>
                </flowentry>
            </flowtable>
        </node>
    </nodes>
</virtualizer>
```

A.2.7 Request with NF internal links example

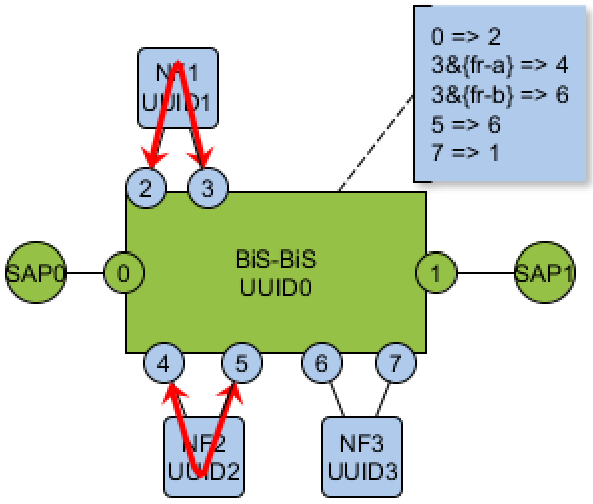


Figure A2-6.7: Request with NF internal requirements

The infrastructure of Figure A2-6.7 can have the following XML description according to our YANG model, see Table A.8.

Table A.8: Request with NF internal requirements example

```
<virtualizer xmlns="urn:unify:virtualizer">
    <id>UUID001</id>
    <name>Single node simple request with intra-NF virtual link requirements</name>
    <nodes>
        <node>
            <id>UUID11</id>
            <NF_instances>
                <node>
                    <id>NF1</id>
                    <name>first NF</name>
                    <type>Parental control B.4</type>
                    <ports>
                        <port>
                            <id>2</id>
                            <name>in</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                        <port>
                            <id>3</id>
                            <name>out</name>
                            <port_type>port-abstract</port_type>
                            <capability>...</capability>
                        </port>
                    </ports>
                    <links>
                        <link>
                            <id>012345</id>
                            <name>requirement on NF delay</name>
                            <src>../../ports/port[id=2]</src>
                            <dst>../../ports/port[id=3]</dst>
                            <resources>
                                <delay>20 ms</delay>
                                <bandwidth>1 Mb</bandwidth>
                            </resources>
                        </link>
                    </links>
                    <!-- example <resources> could come here -->
                </node>
            </NF_instances>
            <!-- <flowtable> omitted here -->
        </node>
    </nodes>
</virtualizer>
```

A.2.8 Link sharing example

Requests from Section 4.7 can have the following XML description according to our YANG model, see Table A.9, Table A.10, and Table A.11.

Table A.9: Request with link sharing example, step 1.

```
<virtualizer>
    <id>UUID002-step1</id>
    <name>update request from SG-1</name>
    <nodes>
        <node>
            <id>UUID11</id>
            <flowtable>
                <flowentry>
                    <port>../../ports/port[id=0]</port>
                    <match>{MAC1}</match>
                    <action>Tag A; output:../../ports/port[id=1]</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>10 Mbps</bandwidth>
                    </resources>
                </flowentry>
            </flowtable>
        </node>
        <node>
            <id>UUID13</id>
            <flowtable>
                <flowentry>
                    <port>../../ports/port[id=2]</port>
                    <match>Tag=A</match>
                    <action>Untag A; output:../../ports/port[id=0]</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>10 Mbps</bandwidth>
                    </resources>
                </flowentry>
            </flowtable>
        </node>
    </nodes>
</virtualizer>
```

Table A.10: Request with link sharing example, step 2.

```
<virtualizer>
    <id>UUID002-step2</id>
    <name>update request after SG-2</name>
    <nodes>
        <node>
            <id>UUID11</id>
            <flowtable>
                <flowentry>
                    <port>../../ports/port[id=0]</port>
                    <match>{MAC1}</match>
                    <action>Tag A; output: #</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>10 Mbps</bandwidth>
                    </resources>
                </flowentry>
                <flowentry>
                    <port>../../ports/port[id=0]</port>
                    <match>{MAC2}</match>
                    <action>Tag B; output: #</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>5 Mbps</bandwidth>
                    </resources>
                </flowentry>
                <flowentry>
                    <port>../../ports/port[id=#]</port>
                    <match>Tag A|B</match>
                    <action>Tag W; output:../../ports/port[id=1]</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>(10+5)*.8 Mbps</bandwidth>
                    </resources>
                </flowentry>
            </flowtable>
        </node>
        <node>
            <id>UUID13</id>
            <flowtable>
                <flowentry>
                    <port>../../ports/port[id=2]</port>
                    <match>Tag W</match>
                    <action>Untag W; output:../../ports/port[id=#]</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>(10+5)*.8 Mbps</bandwidth>
                    </resources>
                </flowentry>
                <flowentry>
                    <port>../../ports/port[id=#]</port>
                    <match>Tag=A</match>
                    <action>Untag A; output:../../ports/port[id=0]</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>10 Mbps</bandwidth>
                    </resources>
                </flowentry>
                <flowentry>
                    <port>../../ports/port[id=#]</port>
                    <match>Tag=B</match>
                    <action>output:../../ports/port[id=1]</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>5 Mbps</bandwidth>
                    </resources>
                </flowentry>
            </flowtable>
        </node>
        <node>
            <id>UUID12</id>
            <flowtable>
```

```
                <flowentry>
                    <port>../../ports/port[id=0]</port>
                    <match>Tag B</match>
                    <action>Untag B; output:../../ports/port[id=1]</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>5 Mbps</bandwidth>
                    </resources>
                </flowentry>
            </flowtable>
        </node>
    </nodes>
</virtualizer>
```

Table A.11: Request with link sharing example, step 3.

```
<virtualizer>
    <id>UUID002-step3</id>
    <name>update request after SG-3</name>
    <nodes>
        <node>
            <id>UUID11</id>
            <flowtable>
                <flowentry>
                    <port>../../ports/port[id=0]</port>
                    <match>{MAC3}</match>
                    <action>Tag C; output: #</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>20 Mbps</bandwidth>
                    </resources>
                </flowentry>
                <flowentry>
                    <port>../../ports/port[id=#]</port>
                    <match>Tag A|B</match>
                    <action>Tag W; output:../../ports/port[id=1]</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>(10+5+20)*.8 Mbps</bandwidth>
                    </resources>
                </flowentry>
            </flowtable>
        </node>
        <node>
            <id>UUID13</id>
            <flowtable>
                <flowentry>
                    <port>../../ports/port[id=2]</port>
                    <match>Tag W</match>
                    <action>Untag W; output:../../ports/port[id=#]</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>(10+20)*.8 Mbps</bandwidth>
                    </resources>
                </flowentry>
                <flowentry>
                    <port>../../ports/port[id=#]</port>
                    <match>Tag=A|C</match>
                    <action>Tag U; output:../../ports/port[id=#]</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>10 Mbps</bandwidth>
                    </resources>
                </flowentry>
                <flowentry>
                    <port>../../ports/port[id=#]</port>
                    <match>Tag=U</match>
                    <action>Untag U output:../../ports/port[id=0]</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>...</bandwidth>
                    </resources>
                </flowentry>
                <flowentry>
                    <port>../../ports/port[id=#]</port>
                    <match>Tag=B</match>
                    <action>output:../../ports/port[id=1]</action>
                    <resources>
                        <delay>...</delay>
                        <bandwidth>5 Mbps</bandwidth>
                    </resources>
                </flowentry>
            </flowtable>
        </node>
    </nodes>
</virtualizer>
```