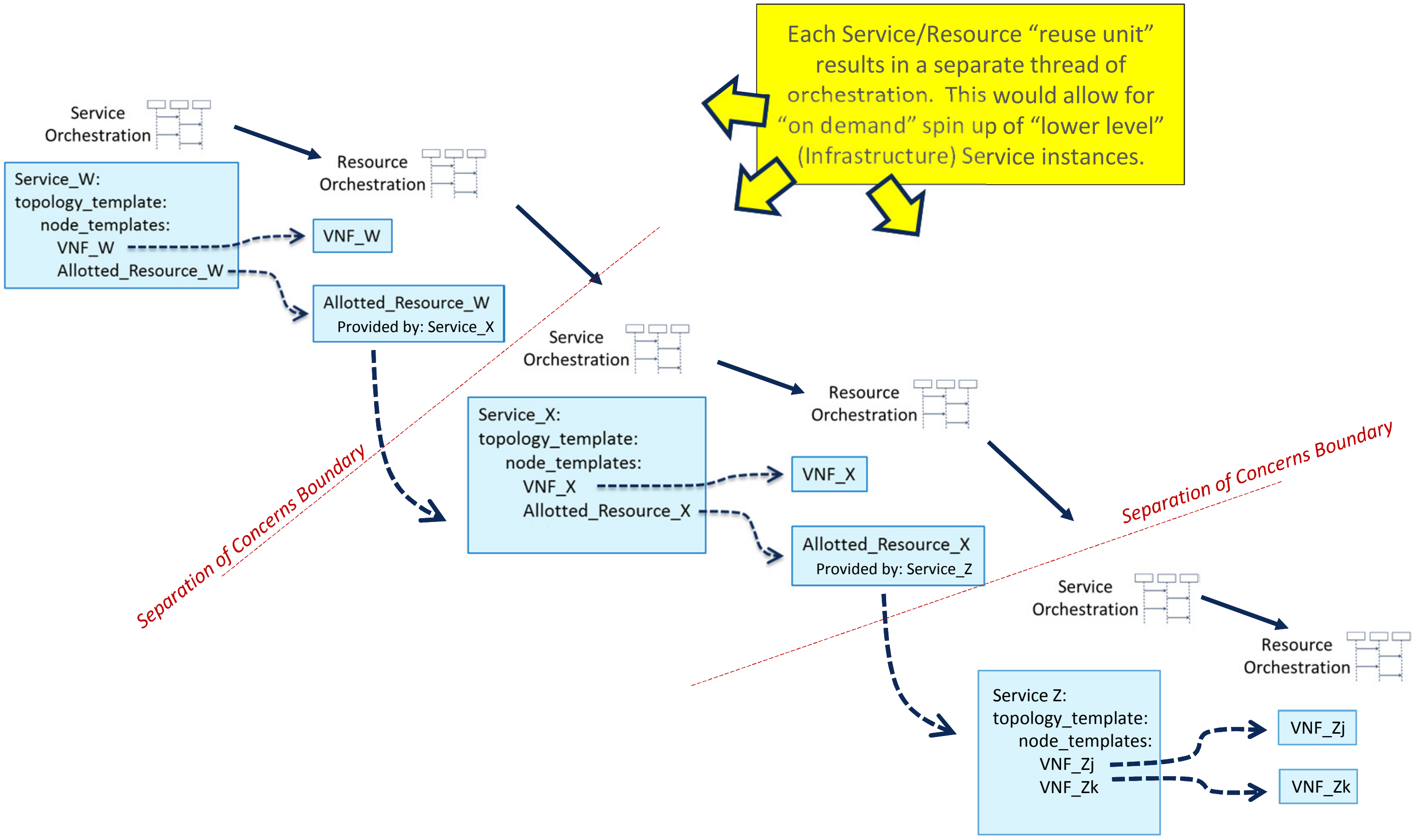
N-Level Run Time Nesting? Let The Service Providers Decide

Service_W Modeling Example 1



Each Service/Resource “reuse unit” results in a separate thread of orchestration. This would allow for “on demand” spin up of “lower level” (Infrastructure) Service instances.

Service Orchestration

Resource Orchestration

Service_W:
topology_template:
node_templates:
VNF_W
Allotted_Resource_W

VNF_W

Allotted_Resource_W
Provided by: Service_X

Separation of Concerns Boundary

Service Orchestration

Resource Orchestration

Service_X:
topology_template:
node_templates:
VNF_X
Allotted_Resource_X

VNF_X

Allotted_Resource_X
Provided by: Service_Z

Separation of Concerns Boundary

Service Orchestration

Resource Orchestration

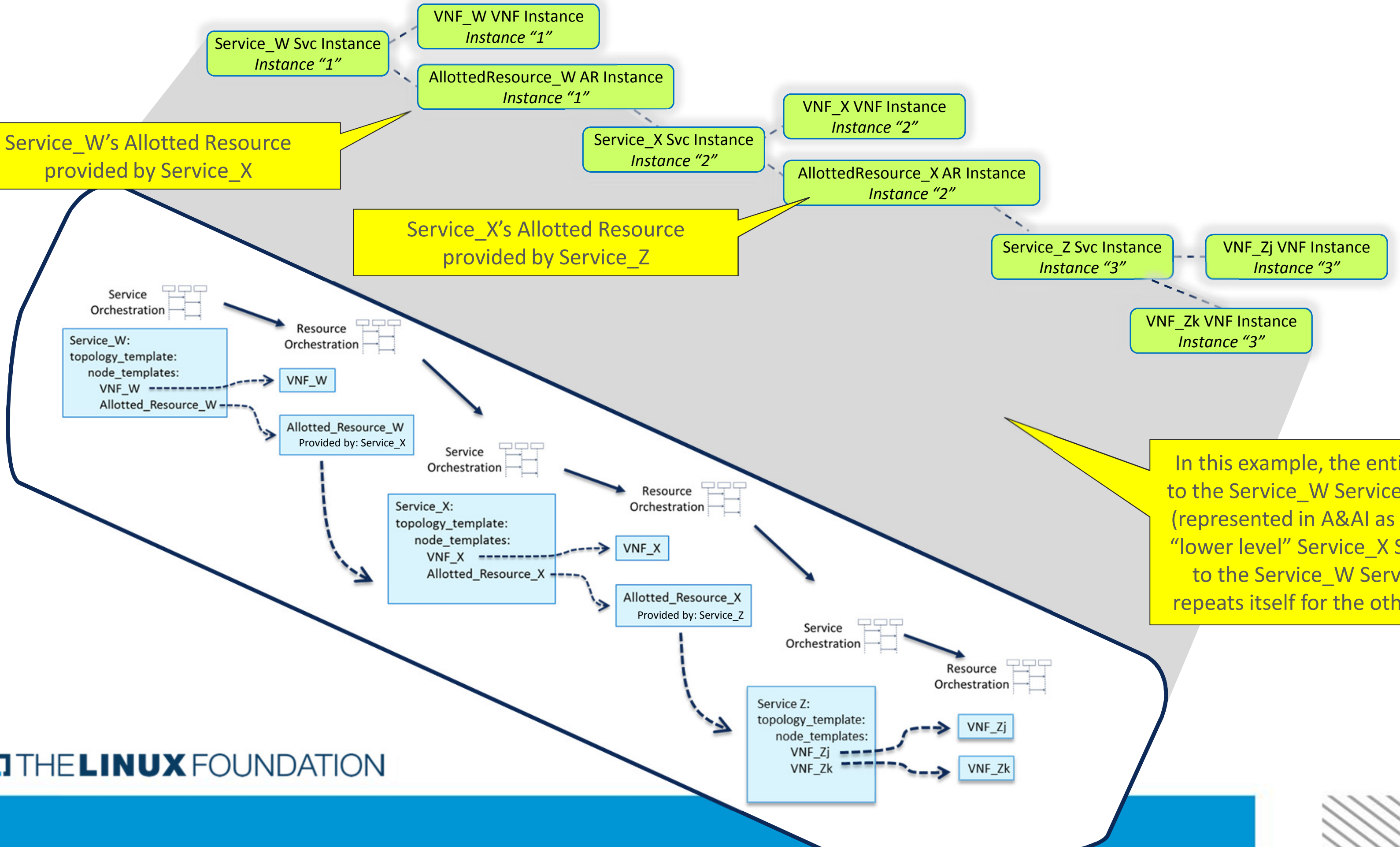
Service Z:
topology_template:
node_templates:
VNF_Zj
VNF_Zk

VNF_Zj

VNF_Zk

Could be extended to allow multiple “higher level Services” to each have a “share” of a “lower level Service’s” instance

A&AI Instance Representation of Service_W Example 1

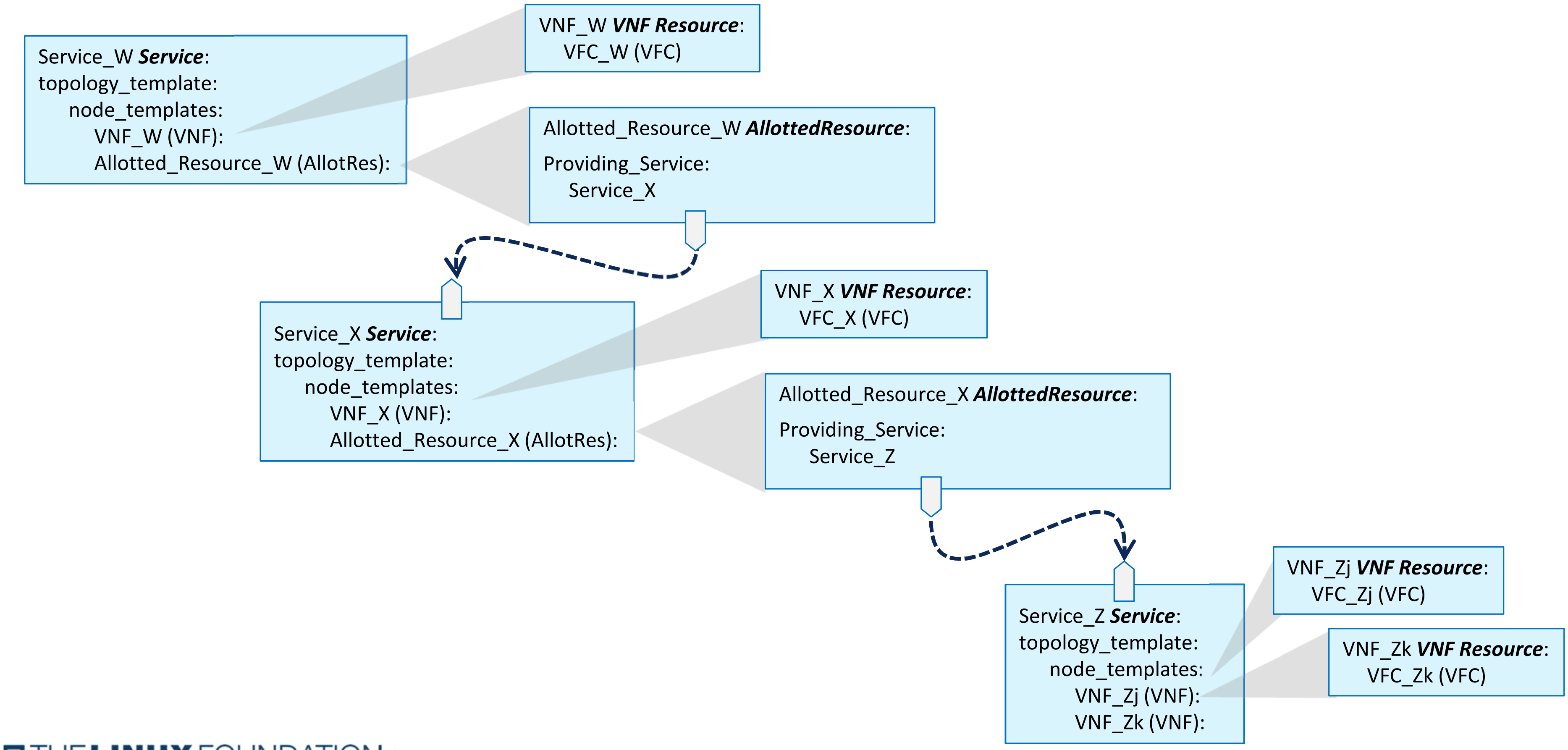


Service_W's Allotted Resource provided by Service_X

Service_X's Allotted Resource provided by Service_Z

In this example, the entirety of VNF_W is dedicated to the Service_W Service Instance, but only a portion (represented in A&AI as AllottedResource_W) of the "lower level" Service_X Service Instance is dedicated to the Service_W Service Instance. This pattern repeats itself for the other Service Instances shown.

SDC Model View



Service_W ***Service***:
topology_template:
node_templates:
VNF_W (VNF):
Allotted_Resource_W (AllotRes):

VNF_W ***VNF Resource***:
VFC_W (VFC)

Allotted_Resource_W ***AllottedResource***:
Providing_Service:
Service_X

Service_X ***Service***:
topology_template:
node_templates:
VNF_X (VNF):
Allotted_Resource_X (AllotRes):

VNF_X ***VNF Resource***:
VFC_X (VFC)

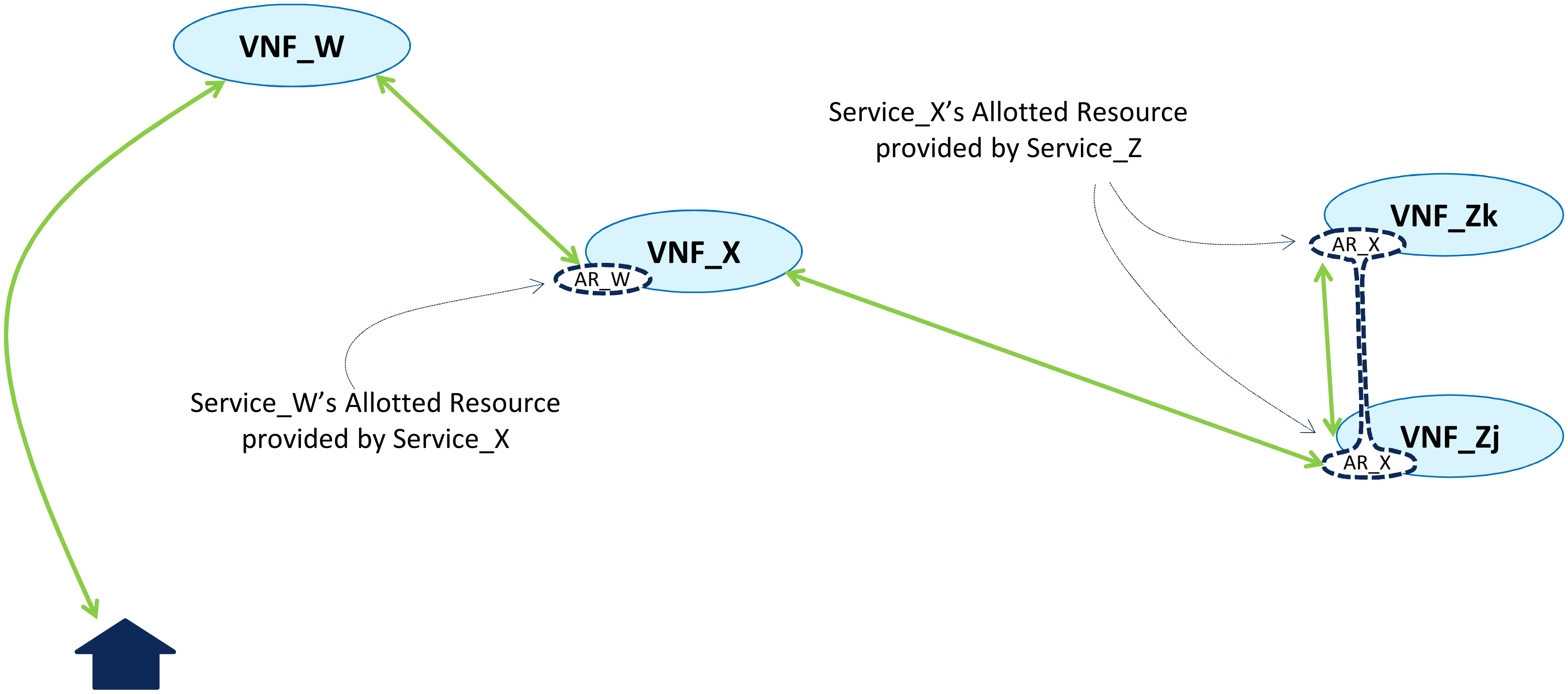
Allotted_Resource_X ***AllottedResource***:
Providing_Service:
Service_Z

Service_Z ***Service***:
topology_template:
node_templates:
VNF_Zj (VNF):
VNF_Zk (VNF):

VNF_Zj ***VNF Resource***:
VFC_Zj (VFC)

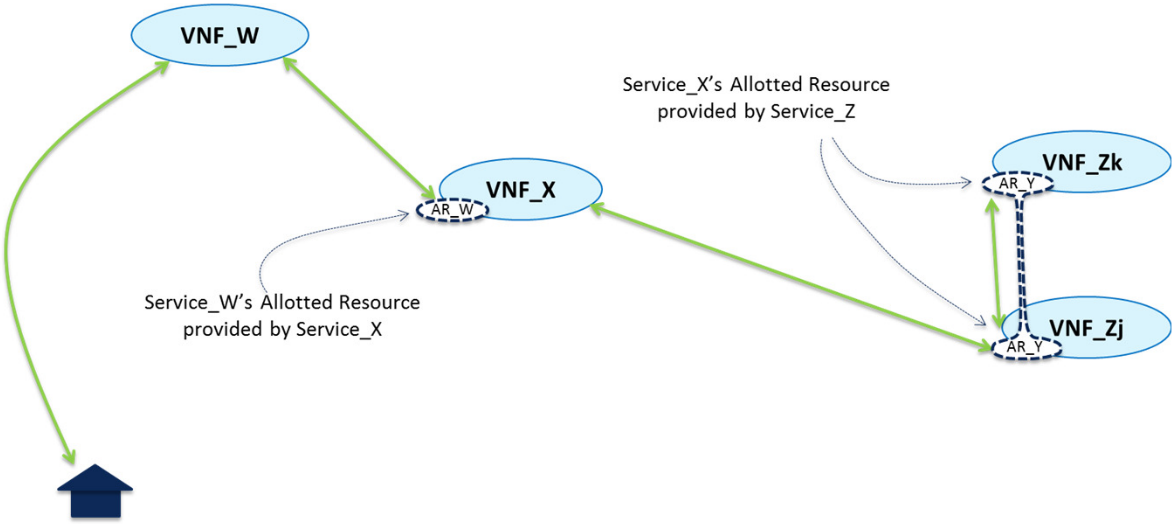
VNF_Zk ***VNF Resource***:
VFC_Zk (VFC)

“VNF Chaining” Data Flow for Service_W Example 1


VNF_W
Service_X’s Allotted Resource provided by Service_Z
VNF_Zk
AR_X
VNF_X
AR_W
Service_W’s Allotted Resource provided by Service_X
VNF_Zj
AR_X

Modeling Network Latency Homing Constraints for Allotted Resources

If Service_W is sensitive to network latency beween VNF_W and the VNF_X that hosts AR_W, then the homing algorithm will need to select only VNF_X instances that meet the Service_W constraint. However, we don't want to write any homing (or any other) policies for Service_W in terms of the internal structure of the underlying "lower order" Service type.


VNF_W
VNF_X
AR_W
Service_W's Allotted Resource provided by Service_X
Service_X's Allotted Resource provided by Service_Z
VNF_Zk
AR_Y
VNF_Zj
AR_Y

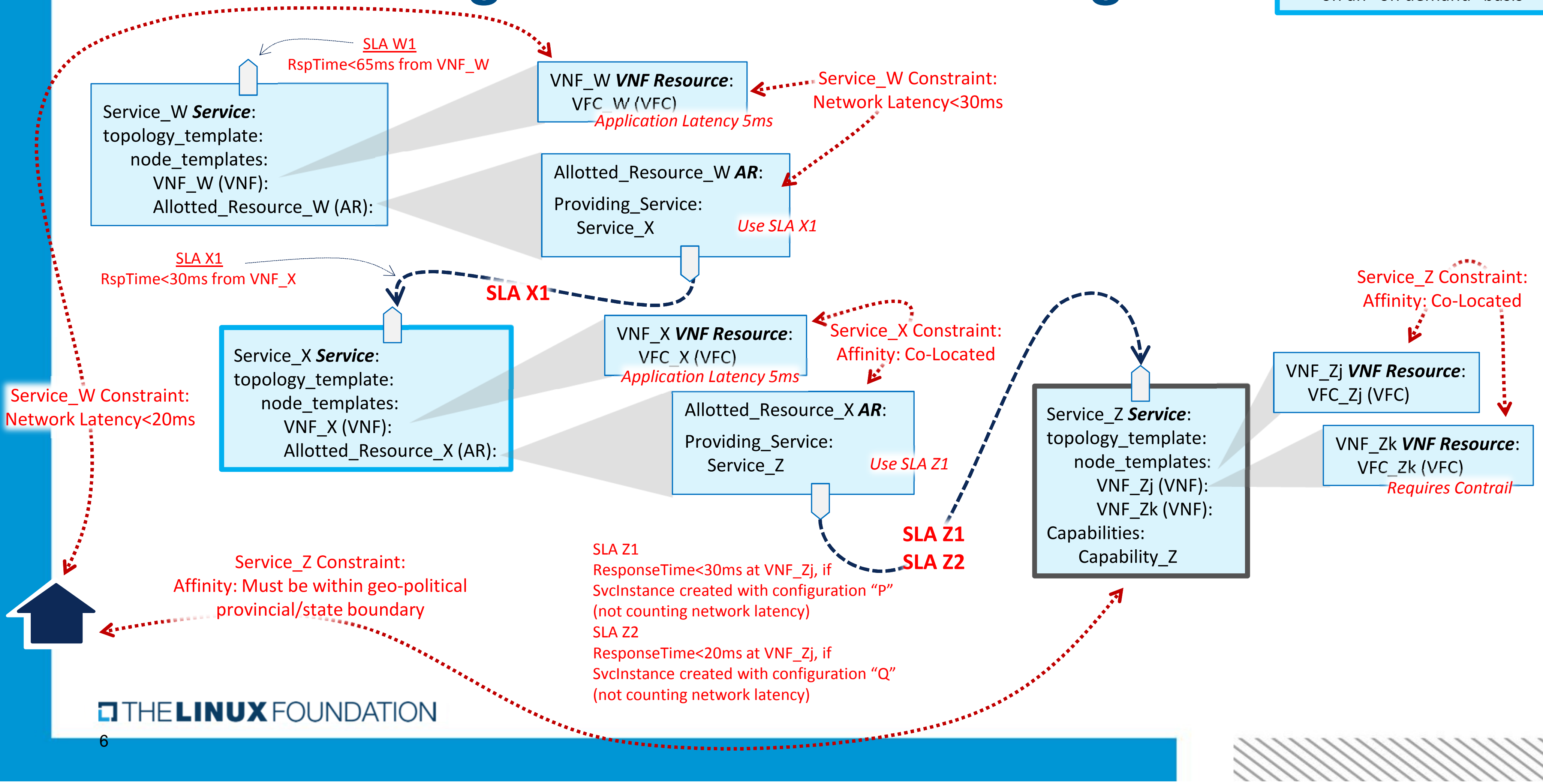
We can instead write the network latency constraint in terms of two policies, one a Service_W policy and one a Service_X policy.

Specifically, we will define the concept of an "SLA" that the lower order service will advertise. We will give the "higher order" Service a policy as to which SLA it requires from the "lower order" Service type. We will have the "lower order" Service type have a policy which indicates from which VNF the SLA is measured (mirroring the data path)

SDC Modeling Tool for Service Designer

Key

"Lower Level Service Type" that can be instantiated in real time on an "on demand" basis



SLA W1
RspTime<65ms from VNF_W

Service_W *Service*:
topology_template:
node_templates:
VNF_W (VNF):
Allotted_Resource_W (AR):

VNF_W ***VNF Resource***:
VFC_W (VFC)
Application Latency 5ms

Service_W Constraint:
Network Latency<30ms

Allotted_Resource_W ***AR***:
Providing_Service:
Service_X
Use SLA X1

SLA X1
RspTime<30ms from VNF_X

SLA X1

Service_X *Service*:
topology_template:
node_templates:
VNF_X (VNF):
Allotted_Resource_X (AR):

VNF_X ***VNF Resource***:
VFC_X (VFC)
Application Latency 5ms

Service_X Constraint:
Affinity: Co-Located

Allotted_Resource_X ***AR***:
Providing_Service:
Service_Z
Use SLA Z1

Service_W Constraint:
Network Latency<20ms

SLA Z1
SLA Z2

Service_Z *Service*:
topology_template:
node_templates:
VNF_Zj (VNF):
VNF_Zk (VNF):
Capabilities:
Capability_Z

Service_Z Constraint:
Affinity: Co-Located

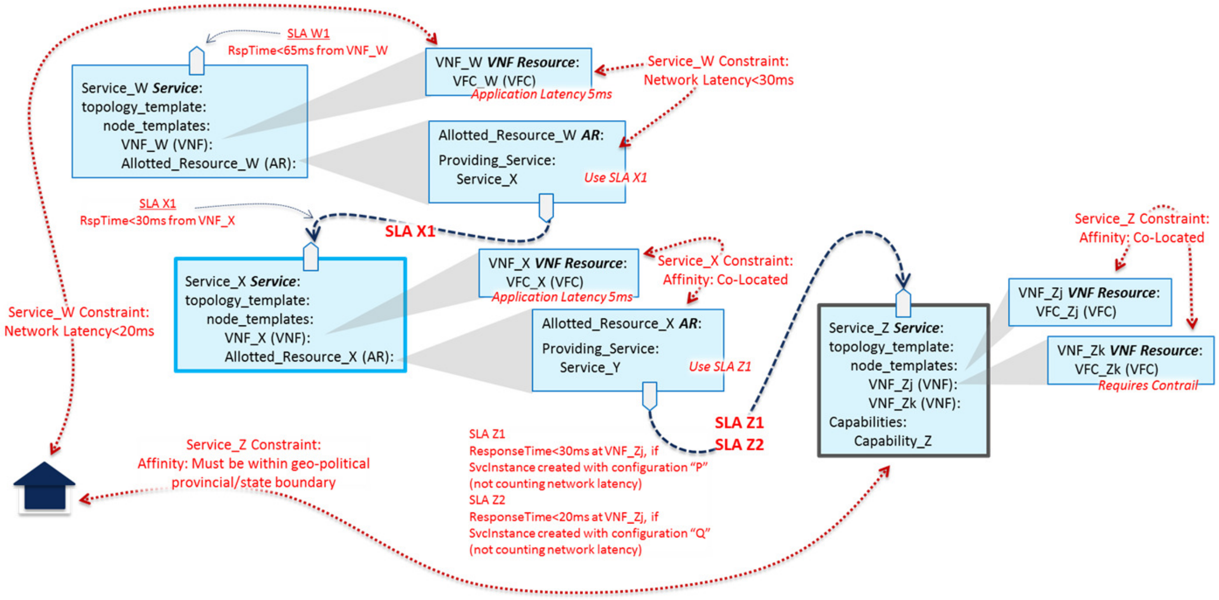
VNF_Zj ***VNF Resource***:
VFC_Zj (VFC)

VNF_Zk ***VNF Resource***:
VFC_Zk (VFC)
Requires Contrail

Service_Z Constraint:
Affinity: Must be within geo-political provincial/state boundary

SLA Z1
ResponseTime<30ms at VNF_Zj, if SvcInstance created with configuration "P" (not counting network latency)
SLA Z2
ResponseTime<20ms at VNF_Zj, if SvcInstance created with configuration "Q" (not counting network latency)

SDC Modeling Tool for Service Designer



SLA W1
RspTime<65ms from VNF_W

Service_W ***Service***:
topology_template:
node_templates:
VNF_W (VNF):
Allotted_Resource_W (AR):

VNF_W ***VNF Resource***:
VFC_W (VFC)
Application Latency 5ms

Service_W Constraint:
Network Latency<30ms

Allotted_Resource_W ***AR***:
Providing_Service:
Service_X
Use SLA X1

SLA X1
RspTime<30ms from VNF_X

SLA X1

Service_W Constraint:
Network Latency<20ms

Service_X ***Service***:
topology_template:
node_templates:
VNF_X (VNF):
Allotted_Resource_X (AR):

VNF_X ***VNF Resource***:
VFC_X (VFC)
Application Latency 5ms

Service_X Constraint:
Affinity: Co-Located

Allotted_Resource_X ***AR***:
Providing_Service:
Service_Y
Use SLA Z1

SLA Z1
SLA Z2

SLA Z1
ResponseTime<30ms at VNF_Zj, if
SvcInstance created with configuration “P”
(not counting network latency)
SLA Z2
ResponseTime<20ms at VNF_Zj, if
SvcInstance created with configuration “Q”
(not counting network latency)

Service_Z Constraint:
Affinity: Must be within geo-political
provincial/state boundary

Service_Z ***Service***:
topology_template:
node_templates:
VNF_Zj (VNF):
VNF_Zk (VNF):
Capabilities:
Capability_Z

Service_Z Constraint:
Affinity: Co-Located

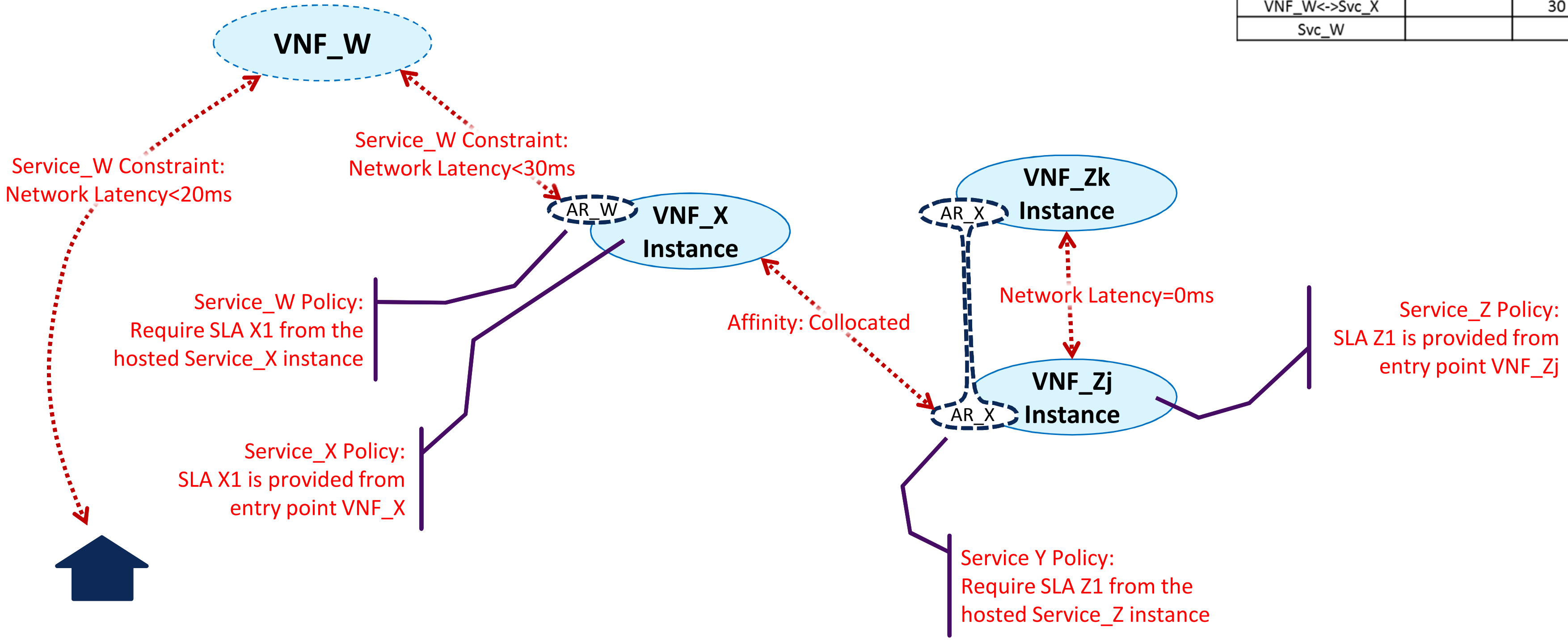
VNF_Zj ***VNF Resource***:
VFC_Zj (VFC)

VNF_Zk ***VNF Resource***:
VFC_Zk (VFC)
Requires Contrail

	Application Latency	Network Latency	Cumulative Latency	Advertised SLA
VNF_Zk				
VNF_Zj				
VNF_Zk <-> VNF_Zj		0		
Svc_Z	Unknown			20
VNF_X	5			
VNF_X<->Svc_Z		0		
Svc_X			25	30
VNF_W	5			
VNF_W<->Svc_X		30		
Svc_W			65	65

SDC Homing Policy Calculator

	Application Latency	Network Latency	Cumulative Latency	Advertised SLA
VNF_Zk				
VNF_Zj				
VNF_Zk <-> VNF_Zj		0		
Svc_Z	Unknown			20
VNF_X	5			
VNF_X<->Svc_Z		0		
Svc_X			25	30
VNF_W	5			
VNF_W<->Svc_X		30		
Svc_W			65	65



Service_W Constraint:
Network Latency<20ms

Service_W Constraint:
Network Latency<30ms

Service_W Policy:
Require SLA X1 from the
hosted Service_X instance

Service_X Policy:
SLA X1 is provided from
entry point VNF_X

Affinity: Collocated

Network Latency=0ms

Service_Z Policy:
SLA Z1 is provided from
entry point VNF_Zj

Service Y Policy:
Require SLA Z1 from the
hosted Service_Z instance

Decomposition and Homing Approach

Note that, from a Service_W perspective, homing involves finding a cloud instance suitable for a new VNF_W instance such that the constraint:

Latency: [geographic point on map] <-> VNF_W < 20 ms

(where the geographic point is the location of the residence), and such that the "Network Latency" constraint of "VNF_W <-> AR_W < 30ms" is met. This involves knowing that the Providing Service for AR_W is Service_X. This processing would require decomposition to have created the Service_W rows in the decomposition example. If an appropriate cloud instance and Service_X service instance is found, then homing is complete.

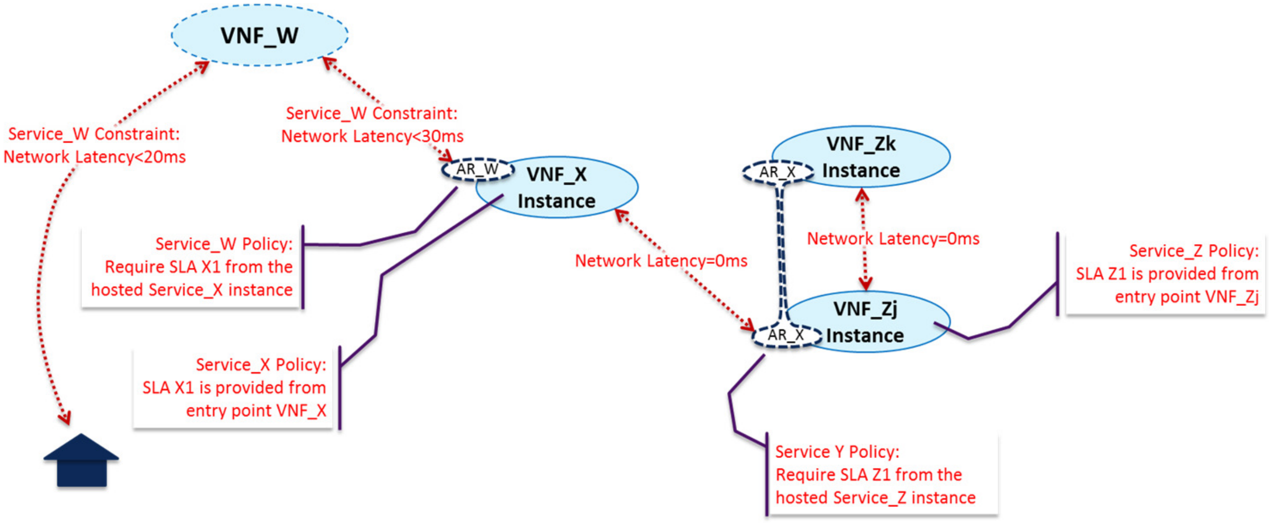
However, if no such Service_X instance exists (i.e., OOF Service_X homing thread returns an exception), homing can determine that a new one should be created "on demand." In such a case, we want to take a separation of concerns approach whereby the Service_W homing thread can delegate down to a Service_X homing thread for further solutioning.

Homing of a new Service_X instance would similarly require that decomposition of Service_X had been performed (i.e., the Service_X rows oof the table). One goal of this homing is to find a cloud instance suitable for a new VNF_X instance such that the Service_W constraint that VNF_W <-> AR_W < 30ms is met. However, we don't want to violate separation of concerns between the Service_W and the Service_X processing, so we will have the Service_W homing thread pass to the Service_X homing thread a constraint that is written in terms that Service_X can understand:

Latency: [geographic point on map] <-> Service_X < 30 ms

(where the geographic point is a "proposed" location of the VNF_W yet to be created). Because the optimal location of VNF_W has not yet been determined, this will likely require that the Service_W homing thread spawns multiple Service_X homing thread to solve the overall homing problem. Step by step processing can be seen on the following slides.

Homing Example Flow



VNF_W

Service_W Constraint:
Network Latency<20ms

Service_W Constraint:
Network Latency<30ms

AR_W

VNF_X
Instance

Service_W Policy:
Require SLA X1 from the
hosted Service_X instance

Service_X Policy:
SLA X1 is provided from
entry point VNF_X

Network Latency=0ms

AR_X

VNF_Zk
Instance

Network Latency=0ms

AR_X

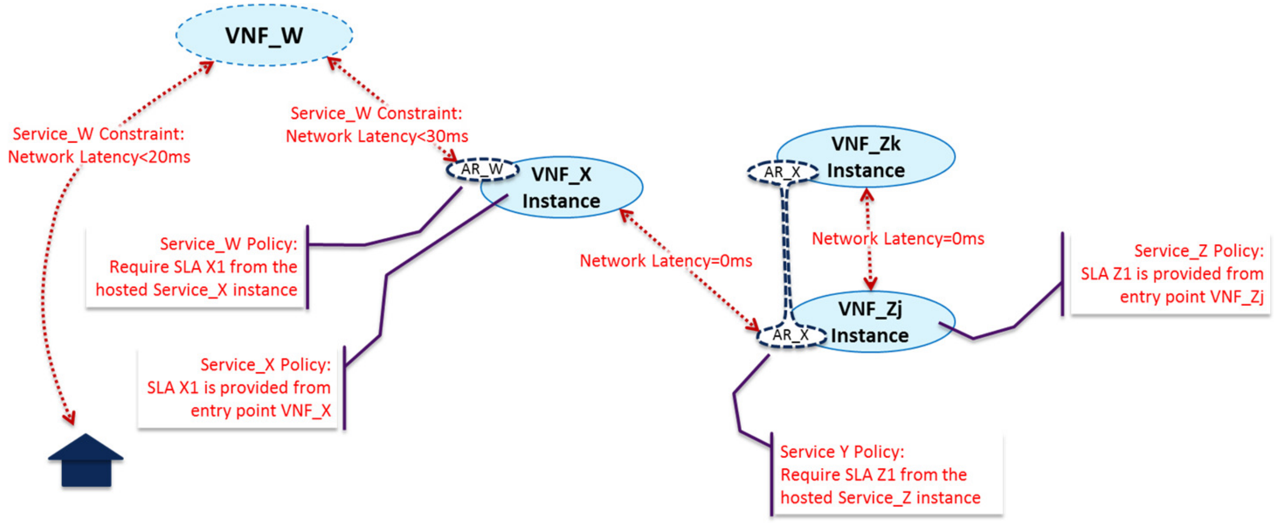
VNF_Zj
Instance

Service_Z Policy:
SLA Z1 is provided from
entry point VNF_Zj

Service Y Policy:
Require SLA Z1 from the
hosted Service_Z instance

SO sends OOF a Service_W homing request, providing as an input constraint the geographic location of the residence. OOF Service_W homing will comprise homing for VNF_W and AR_W. OOF homing for VNF_W will find eligible VNF_W cloud instances that meet the 20ms latency constraint with the residence. OOF homing for AR_W will, for each eligible VNF_W cloud instance, want to find the set of Service_X instances to provide that AR_W functionality that meet the 30ms latency constraint with that cloud instance.

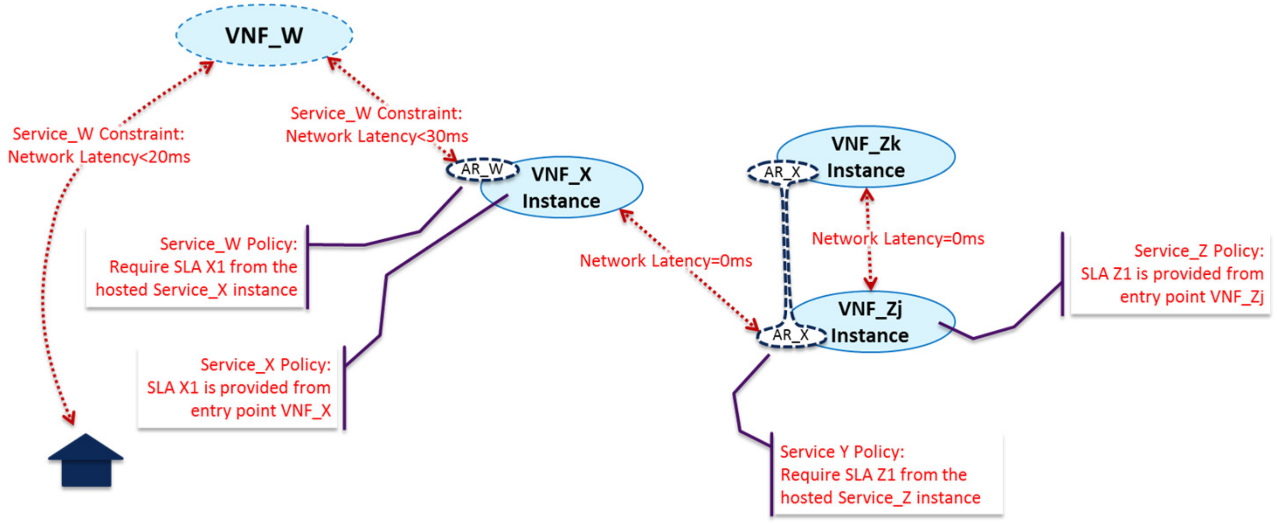
Homing Example Flow (Cont'd)



However, we want to maintain a separate of concerns approach, and the Service_W processing thread shouldn't know the implementation of AR_W such that it can measure latency to it. (This can be best seen in the Service_Z example to the right.) Thus, we will have the Service W OOF homing request thread delegate selection of the optimal Service_X instance to a subtending Service_X OOF thread. Thus, OOF can be seen as (logically) calling itself in parallel with multiple Service_X homing requests. Each such request can be seen as providing as input constraints the geographic location of the associated eligible VNF_W cloud instance and the SLA needed, in this case SLA X1.

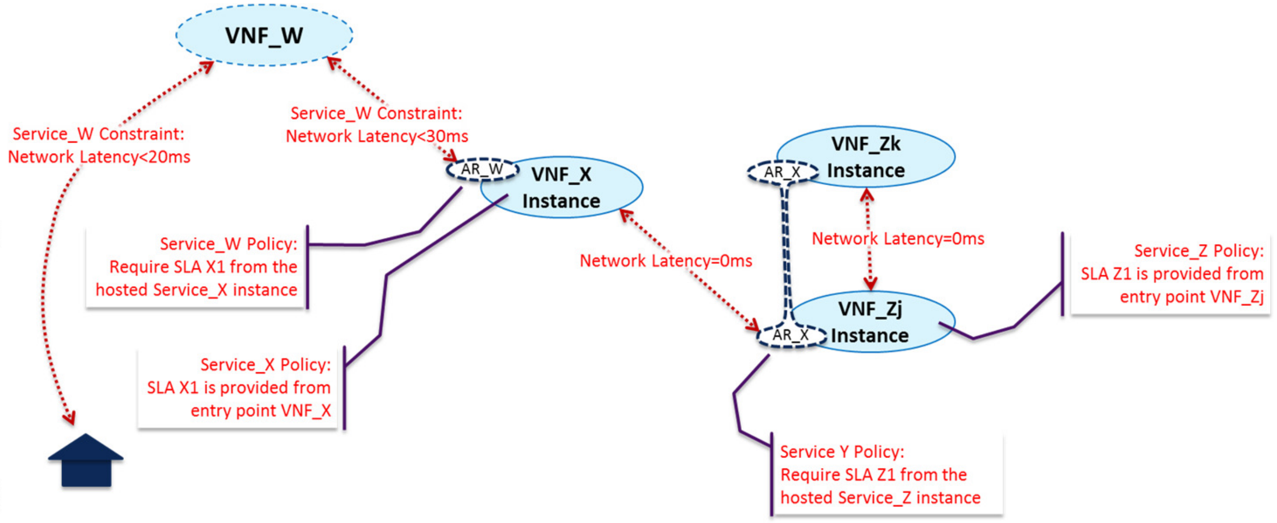
Homing Example Flow (Cont’d)

Service_X homing knows that SLA X1 is measured from an entry point on VNF_X. Thus Service_X homing is comprised of looking for the optimal Service_X instance whose VNF_X instance is within 30ms of the input geographic location. If at least one such Service X instance is found, homing is done (except for optimization).



If no such Service_X instance can be found, then homing will determine whether the Service_X service definition allows for dynamic instantiation of new Service_X instances. In this case we will assume “yes”, so OOF would determine whether a new Service_X could be instantiated such that all constraints can be met.

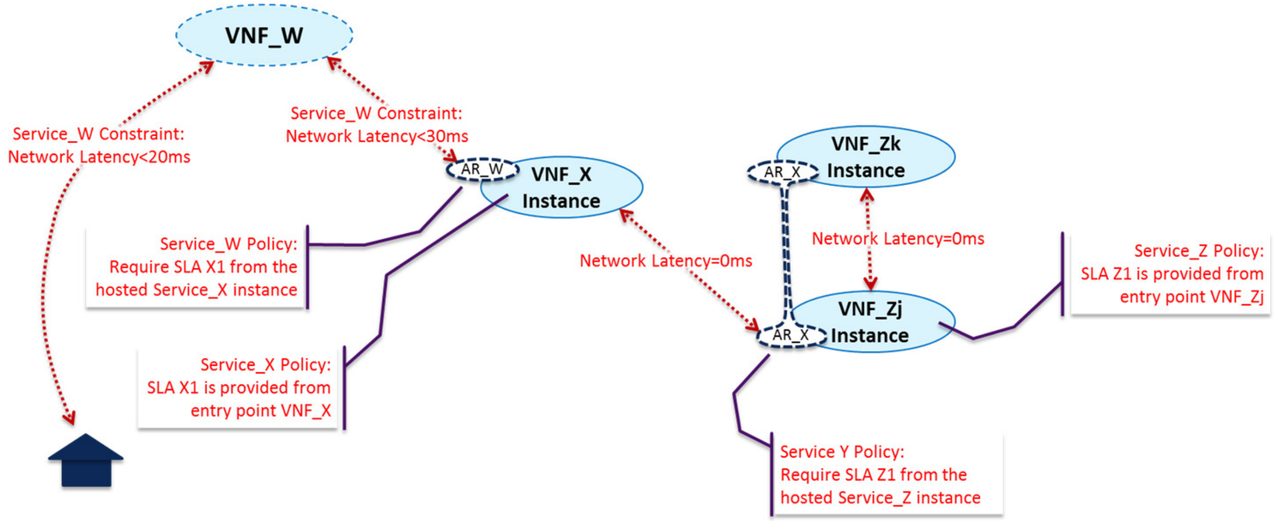
Homing Example Flow (Cont’d)



OOF Service_X homing will comprise homing for VNF_X and AR_X. OOF homing for VNF_X will find eligible VNF_X cloud instances that meet the 30ms latency constraint with the input geographic location. OOF homing for AR_X will, for each eligible VNF_X cloud instance, want to find the set of Service_Z instances to provide that AR_X functionality that meet the 0ms latency constraint with that cloud instance. The pattern recurs, however, that Service_X has no business understanding whether the 0ms latency constraint should be measured from VNF_Zj or VNF_Zk, or even in fact that there exists a VNF_Zj or VNF_Zk. In order to maintain separation of concerns, homing of AR_X will be delegated to a subtending request thread delegate selection of the optimal Service_Z instance to a subtending Service_Z OOF thread.

Homing Example Flow (Cont'd)

Service_Z homing will thus search for eligible Service_Z instances such that the 0ms constraint is measured from the input geographical location (in this case the potential cloud instance location for VNF_X) to an available VNF_Zj instance (the point from which the Service_Z SLA is measured.



If no such Service_Z instance can be found, then homing will determine whether the Service_Z service definition allows for dynamic instantiation of new Service_Z instances. In this case we will assume "no", so the OOF Service_Z homing thread would return an exception to the calling Service_X homing thread. Such an exception would likely not result in failure of the entire Service_W homing, but rather simply result in pruning a branch of the overall potential homing solution tree.

Decomposition Structure for Service_W Example 1

Svc Type	Rsc Type	AR Prov Svc	Advertised SLA	Homing Constraints	Capab Svc Struct
Service_W			W1: RspTime 65ms end to end	Ntw Latency: VNF_W <-> AR_W < 30ms	
Service_W	VNF_W			Ntw Latency: Residence <-> VNF_W< 20ms	
Service_W	AR_W	Service_X		Require SLA X1 from Service_X instance	

AR_W Provider’s Svc Struct

Svc Type	Rsc Type	AR Prov Svc	Advertised SLA	Homing Constraints	Capab Svc Struct
Service_X			X1: RspTime 30ms end to end	Affinity: VNF_X, AR_X Co-Located	
Service_X	VNF_X				
Service_X	AR_X	Service_Z		Require SLA Y1 from Service_Z instance	

AR_X Provider’s Svc Struct

Svc Type	Rsc Type	AR Prov Svc	SLA Policies	Homing Constraints	Capab Svc Struct
Service_Z			Z1: <30ms with config “Q” Z2: <20ms with config “P”	Affinity: VNF_Zj, VNF_Zk Co-Located	
Service_Z	VNF_Zj				
Service_Z	VNF_Zk				

Homing Solution Example for Service_W Example 1

Service Type	Resource Type	Allotted Resource Provider Service	Provider Service Struct	Homing Solution
Service_W	VNF_W			Cloud_Region_1
Service_W	Allotted_Resource_W	Service_X		Instantiation_Needed

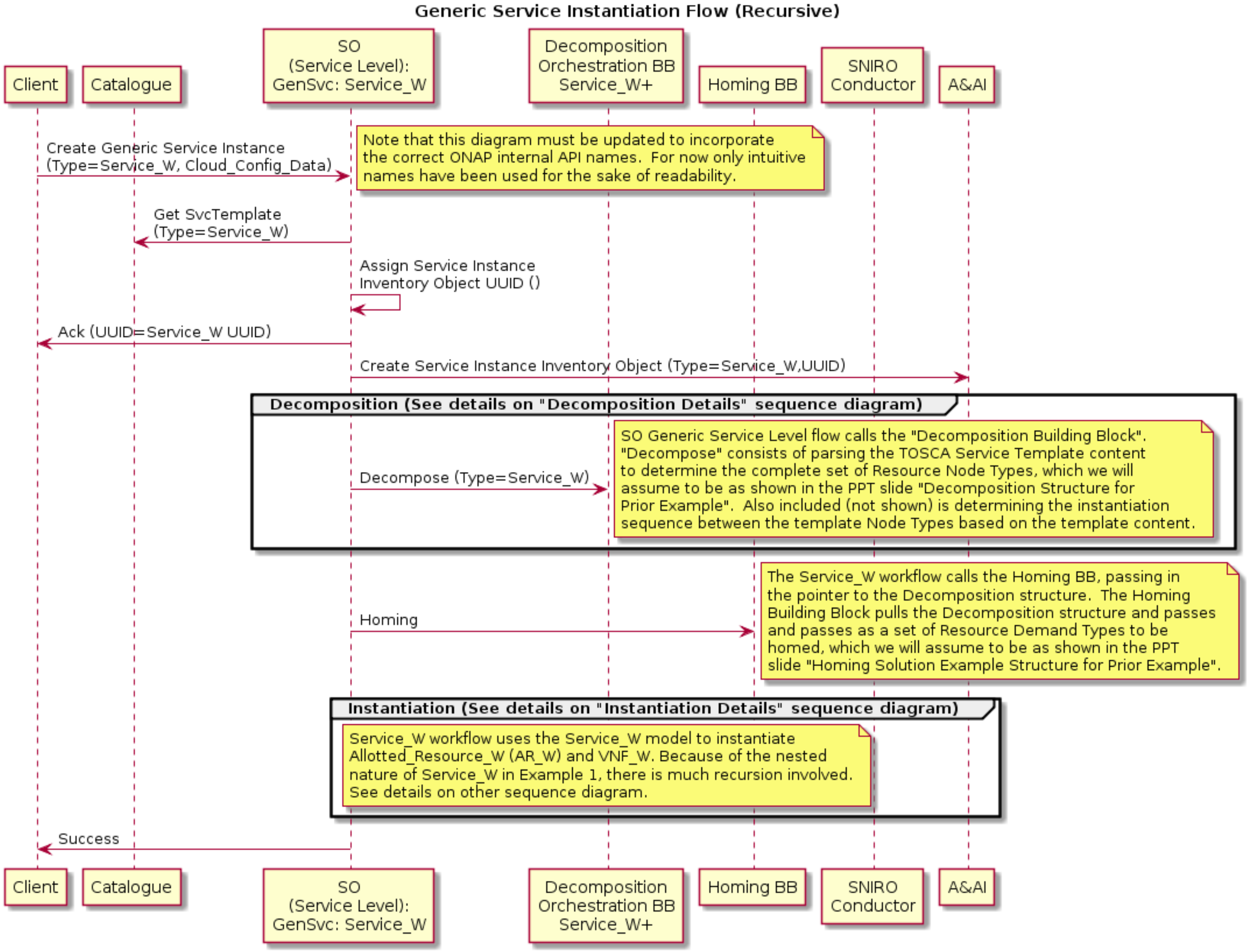
AR_W Homing Structure

Service Type	Resource Type	Allotted Resource Provider Service	Provider Service Struct	Homing Solution
Service_X	VNF_X			Cloud_Region_2
Service_X	Allotted_Resource_X	Service_Z		Service Z Instance Id

AR_X Homing Structure

Service Type	Resource Type	Allotted Resource Provider Service	Homing Solution
Service_Z	VNF_Zj		As Exists
Service_Z	VNF_Zk		As Exists

Generic Service Level Flow for Service_W Example 1


Generic Service Instantiation Flow (Recursive)
Client
Catalogue
SO (Service Level): GenSvc: Service_W
Decomposition Orchestration BB Service_W+
Homing BB
SNIRO Conductor
A&AI
Create Generic Service Instance (Type=Service_W, Cloud_Config_Data)
Get SvcTemplate (Type=Service_W)
Assign Service Instance Inventory Object UUID ()
Ack (UUID=Service_W UUID)
Create Service Instance Inventory Object (Type=Service_W,UUID)
Decomposition (See details on "Decomposition Details" sequence diagram)
Decompose (Type=Service_W)
Homing
Instantiation (See details on "Instantiation Details" sequence diagram)
Success
Client
Catalogue
SO (Service Level): GenSvc: Service_W
Decomposition Orchestration BB Service_W+
Homing BB
SNIRO Conductor
A&AI

Note that this diagram must be updated to incorporate the correct ONAP internal API names. For now only intuitive names have been used for the sake of readability.

SO Generic Service Level flow calls the "Decomposition Building Block". "Decompose" consists of parsing the TOSCA Service Template content to determine the complete set of Resource Node Types, which we will assume to be as shown in the PPT slide "Decomposition Structure for Prior Example". Also included (not shown) is determining the instantiation sequence between the template Node Types based on the template content.

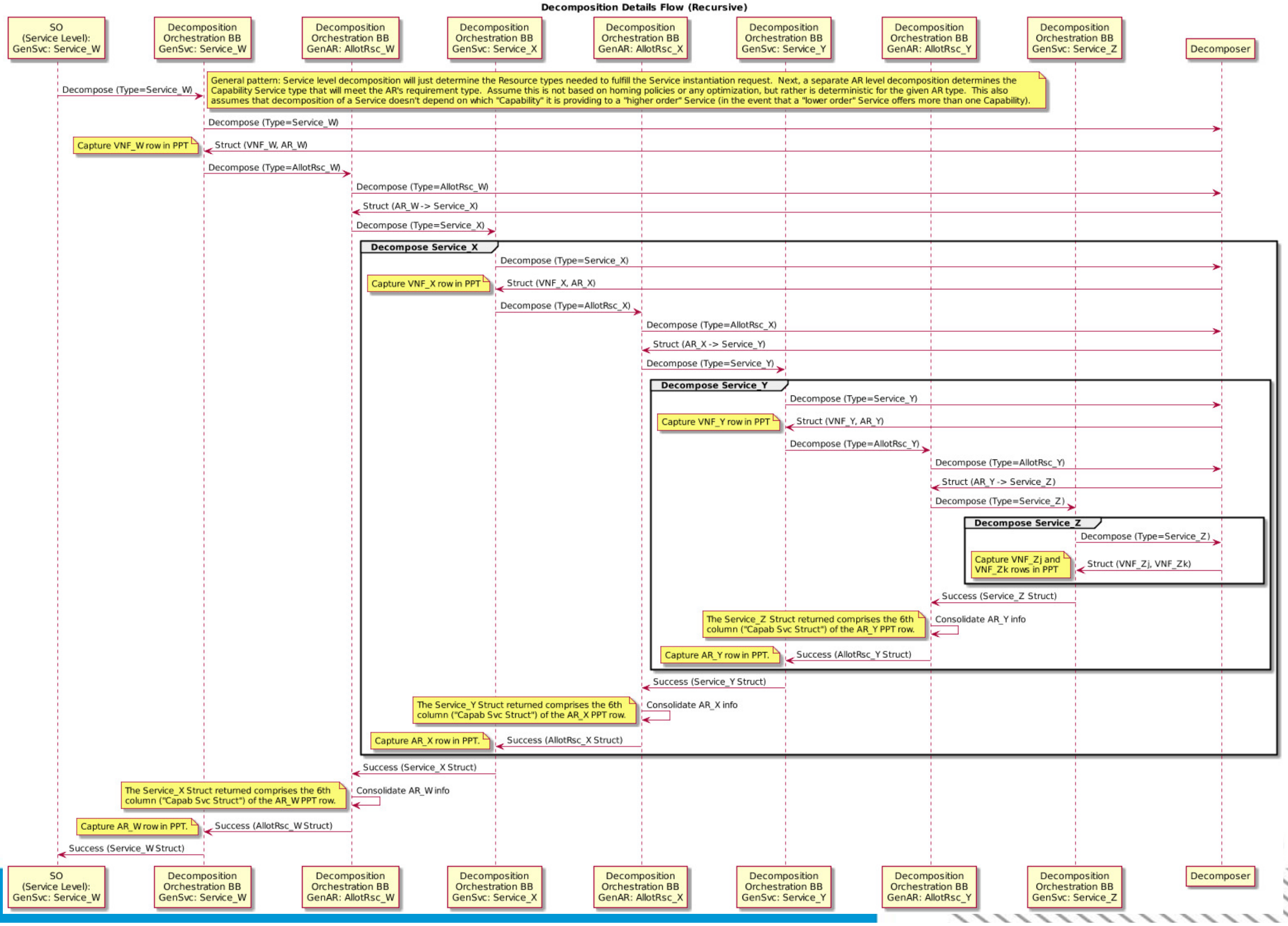
The Service_W workflow calls the Homing BB, passing in the pointer to the Decomposition structure. The Homing Building Block pulls the Decomposition structure and passes and passes as a set of Resource Demand Types to be homed, which we will assume to be as shown in the PPT slide "Homing Solution Example Structure for Prior Example".

Service_W workflow uses the Service_W model to instantiate Allotted_Resource_W (AR_W) and VNF_W. Because of the nested nature of Service_W in Example 1, there is much recursion involved. See details on other sequence diagram.

onap_uc_Generic_Service_Recursive_p1.html

Decomposition Detail Flow for Prior Example

onap_uc_Generic_Service_Decomp_p1.html


Decomposition Details Flow (Recursive)

General pattern: Service level decomposition will just determine the Resource types needed to fulfill the Service instantiation request. Next, a separate AR level decomposition determines the Capability Service type that will meet the AR's requirement type. Assume this is not based on homing policies or any optimization, but rather is deterministic for the given AR type. This also assumes that decomposition of a Service doesn't depend on which "Capability" it is providing to a "higher order" Service (in the event that a "lower order" Service offers more than one Capability).

The Service_Z Struct returned comprises the 6th column ("Capab Svc Struct") of the AR_Y PPT row.

The Service_Y Struct returned comprises the 6th column ("Capab Svc Struct") of the AR_X PPT row.

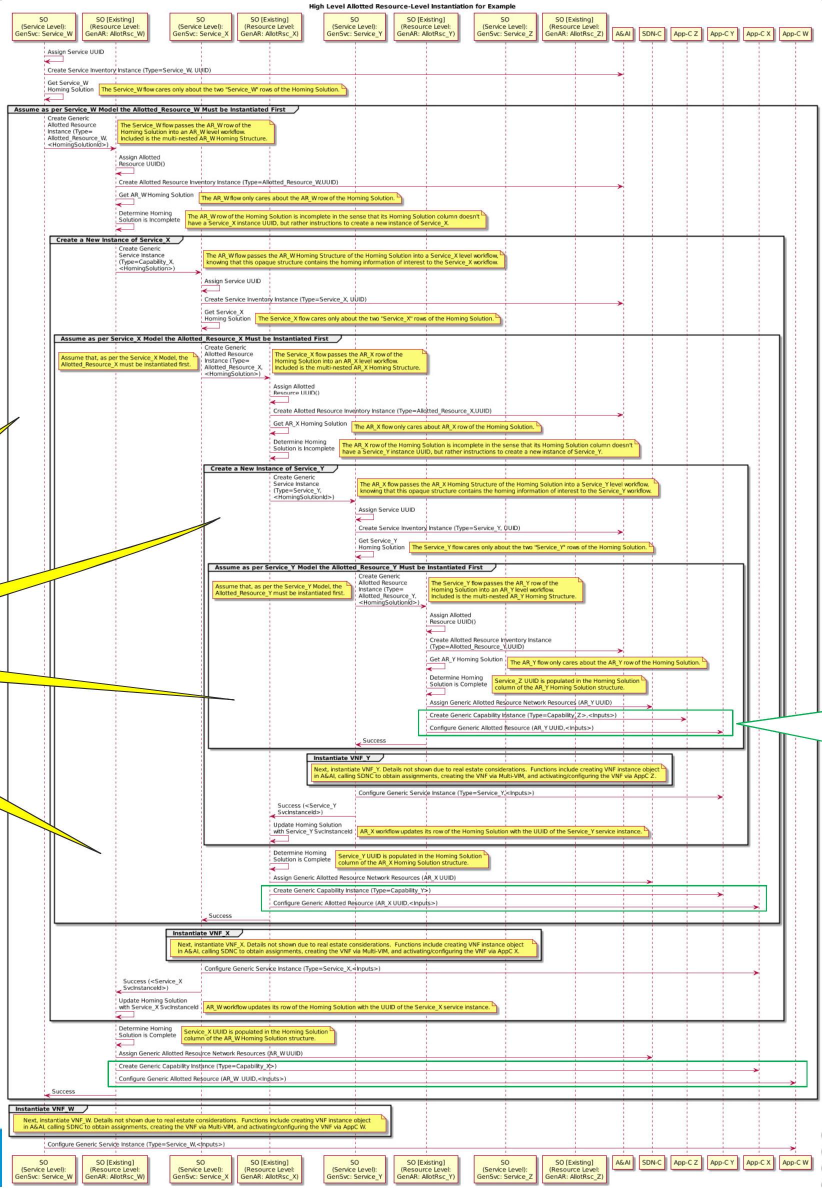
The Service_X Struct returned comprises the 6th column ("Capab Svc Struct") of the AR_W PPT row.

Instantiation Detail Flow for Service_W Example 1

Scale PPT to 300% to view detail. ☺

Note recursion in the process

onap_uc_Generic_Resource_VNF_Recursive.html

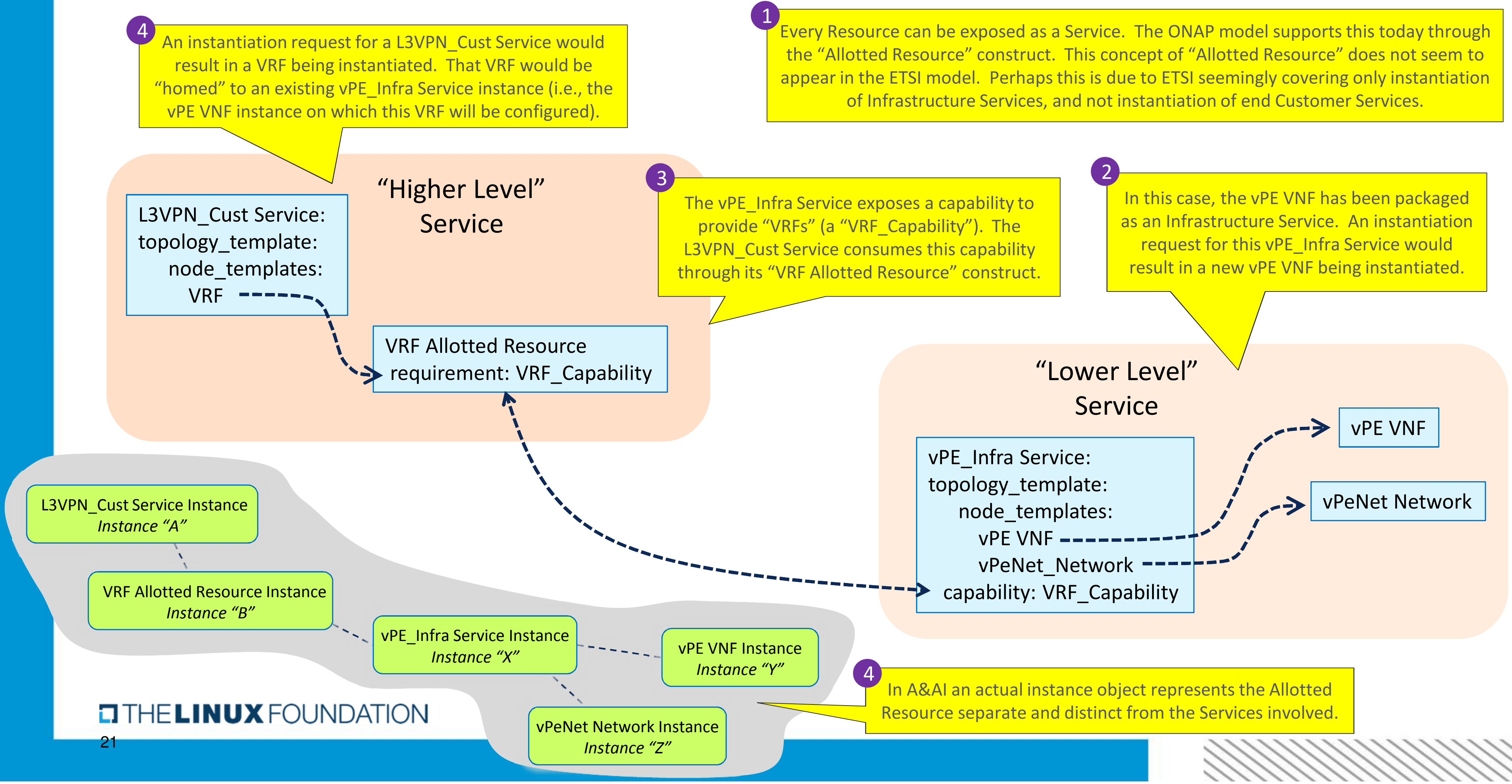

High Level Allotted Resource-Level Instantiation for Example
SO (Service Level): GenSvc: Service_W
SO [Existing] (Resource Level: GenAR: AllotRsc_W)
SO (Service Level): GenSvc: Service_X
SO [Existing] (Resource Level: GenAR: AllotRsc_X)
SO (Service Level): GenSvc: Service_Y
SO [Existing] (Resource Level: GenAR: AllotRsc_Y)
SO (Service Level): GenSvc: Service_Z
SO [Existing] (Resource Level: GenAR: AllotRsc_Z)
A&AI
SDN-C
App-C Z
App-C Y
App-C X
App-C W
Assign Service UUID
Create Service Inventory Instance (Type=Service_W, UUID)
Get Service_W Homing Solution
The Service_W flow cares only about the two "Service_W" rows of the Homing Solution.
Assume as per Service_W Model the Allotted_Resource_W Must be Instantiated First
Create Generic Allotted Resource Instance (Type= Allotted_Resource_W, <HomingSolutionId>)
The Service_W flow passes the AR_W row of the Homing Solution into an AR_W level workflow. Included is the multi-nested AR_W Homing Structure.
Assign Allotted Resource UUID()
Create Allotted Resource Inventory Instance (Type=Allotted_Resource_W,UUID)
Get AR_W Homing Solution
The AR_W flow only cares about the AR_W row of the Homing Solution.
Determine Homing Solution is Incomplete
The AR_W row of the Homing Solution is incomplete in the sense that its Homing Solution column doesn't have a Service_X instance UUID, but rather instructions to create a new instance of Service_X.
Create a New Instance of Service_X
Create Generic Service Instance (Type=Capability_X, <HomingSolution>)
The AR_W flow passes the AR_W Homing Structure of the Homing Solution into a Service_X level workflow, knowing that this opaque structure contains the homing information of interest to the Service_X workflow.
Assign Service UUID
Create Service Inventory Instance (Type=Service_X, UUID)
Get Service_X Homing Solution
The Service_X flow cares only about the two "Service_X" rows of the Homing Solution.
Assume as per Service_X Model the Allotted_Resource_X Must be Instantiated First
Assume that, as per the Service_X Model, the Allotted_Resource_X must be instantiated first.
Create Generic Allotted Resource Instance (Type= Allotted_Resource_X, <HomingSolution>)
The Service_X flow passes the AR_X row of the Homing Solution into an AR_X level workflow. Included is the multi-nested AR_X Homing Structure.
Assign Allotted Resource UUID()
Create Allotted Resource Inventory Instance (Type=Allotted_Resource_X,UUID)
Get AR_X Homing Solution
The AR_X flow only cares about AR_X row of the Homing Solution.
Determine Homing Solution is Incomplete
The AR_X row of the Homing Solution is incomplete in the sense that its Homing Solution column doesn't have a Service_Y instance UUID, but rather instructions to create a new instance of Service_Y.
Create a New Instance of Service_Y
Create Generic Service Instance (Type=Service_Y, <HomingSolutionId>)
The AR_X flow passes the AR_X Homing Structure of the Homing Solution into a Service_Y level workflow, knowing that this opaque structure contains the homing information of interest to the Service_Y workflow.
Assign Service UUID
Create Service Inventory Instance (Type=Service_Y, UUID)
Get Service_Y Homing Solution
The Service_Y flow cares only about the two "Service_Y" rows of the Homing Solution.
Assume as per Service_Y Model the Allotted_Resource_Y Must be Instantiated First
Assume that, as per the Service_Y Model, the Allotted_Resource_Y must be instantiated first.
Create Generic Allotted Resource Instance (Type= Allotted_Resource_Y, <HomingSolutionId>)
The Service_Y flow passes the AR_Y row of the Homing Solution into an AR_Y level workflow. Included is the multi-nested AR_Y Homing Structure.
Assign Allotted Resource UUID()
Create Allotted Resource Inventory Instance (Type=Allotted_Resource_Y,UUID)
Get AR_Y Homing Solution
The AR_Y flow only cares about the AR_Y row of the Homing Solution.
Determine Homing Solution is Complete
Service_Z UUID is populated in the Homing Solution column of the AR_Y Homing Solution structure.
Assign Generic Allotted Resource Network Resources (AR_Y UUID)
Create Generic Capability Instance (Type=Capability_Z>,<Inputs>)
Configure Generic Allotted Resource (AR_Y UUID,<Inputs>)
Success
Instantiate VNF_Y
Next, instantiate VNF_Y. Details not shown due to real estate considerations. Functions include creating VNF instance object in A&AI, calling SDNC to obtain assignments, creating the VNF via Multi-VIM, and activating/configuring the VNF via AppC Z.
Configure Generic Service Instance (Type=Service_Y,<Inputs>)
Success (<Service_Y SvcInstanceId>)
Update Homing Solution with Service_Y SvcInstanceId
AR_X workflow updates its row of the Homing Solution with the UUID of the Service_Y service instance.
Determine Homing Solution is Complete
Service_Y UUID is populated in the Homing Solution column of the AR_X Homing Solution structure.
Assign Generic Allotted Resource Network Resources (AR_X UUID)
Create Generic Capability Instance (Type=Capability_Y>)
Configure Generic Allotted Resource (AR_X UUID,<Inputs>)
Success
Instantiate VNF_X
Next, instantiate VNF_X. Details not shown due to real estate considerations. Functions include creating VNF instance object in A&AI, calling SDNC to obtain assignments, creating the VNF via Multi-VIM, and activating/configuring the VNF via AppC X.
Configure Generic Service Instance (Type=Service_X,<Inputs>)
Success (<Service_X SvcInstanceId>)
Update Homing Solution with Service_X SvcInstanceId
AR_W workflow updates its row of the Homing Solution with the UUID of the Service_X service instance.
Determine Homing Solution is Complete
Service_X UUID is populated in the Homing Solution column of the AR_W Homing Solution structure.
Assign Generic Allotted Resource Network Resources (AR_W UUID)
Create Generic Capability Instance (Type=Capability_X>)
Configure Generic Allotted Resource (AR_W UUID,<Inputs>)
Success
Instantiate VNF_W
Next, instantiate VNF_W. Details not shown due to real estate considerations. Functions include creating VNF instance object in A&AI, calling SDNC to obtain assignments, creating the VNF via Multi-VIM, and activating/configuring the VNF via AppC W.
Configure Generic Service Instance (Type=Service_W,<Inputs>)
SO (Service Level): GenSvc: Service_W
SO [Existing] (Resource Level: GenAR: AllotRsc_W)
SO (Service Level): GenSvc: Service_X
SO [Existing] (Resource Level: GenAR: AllotRsc_X)
SO (Service Level): GenSvc: Service_Y
SO [Existing] (Resource Level: GenAR: AllotRsc_Y)
SO (Service Level): GenSvc: Service_Z
SO [Existing] (Resource Level: GenAR: AllotRsc_Z)
A&AI
SDN-C
App-C Z
App-C Y
App-C X
App-C W

Potential insertion points for Cloudify? Do we want to do that as part of this POC? Or just stick with Camunda?

THE LINUX FOUNDATION

Backup Slides

Allotted Resources – vPE/VRF Example



1 Every Resource can be exposed as a Service. The ONAP model supports this today through the “Allotted Resource” construct. This concept of “Allotted Resource” does not seem to appear in the ETSI model. Perhaps this is due to ETSI seemingly covering only instantiation of Infrastructure Services, and not instantiation of end Customer Services.

2 In this case, the vPE VNF has been packaged as an Infrastructure Service. An instantiation request for this vPE_Infra Service would result in a new vPE VNF being instantiated.

3 The vPE_Infra Service exposes a capability to provide “VRFs” (a “VRF_Capability”). The L3VPN_Cust Service consumes this capability through its “VRF Allotted Resource” construct.

4 An instantiation request for a L3VPN_Cust Service would result in a VRF being instantiated. That VRF would be “homed” to an existing vPE_Infra Service instance (i.e., the vPE VNF instance on which this VRF will be configured).

4 In A&AI an actual instance object represents the Allotted Resource separate and distinct from the Services involved.

“Higher Level” Service

L3VPN_Cust Service:
topology_template:
node_templates:
VRF

VRF Allotted Resource
requirement: VRF_Capability

“Lower Level” Service

vPE_Infra Service:
topology_template:
node_templates:
vPE VNF
vPeNet_Network
capability: VRF_Capability

vPE VNF

vPeNet Network

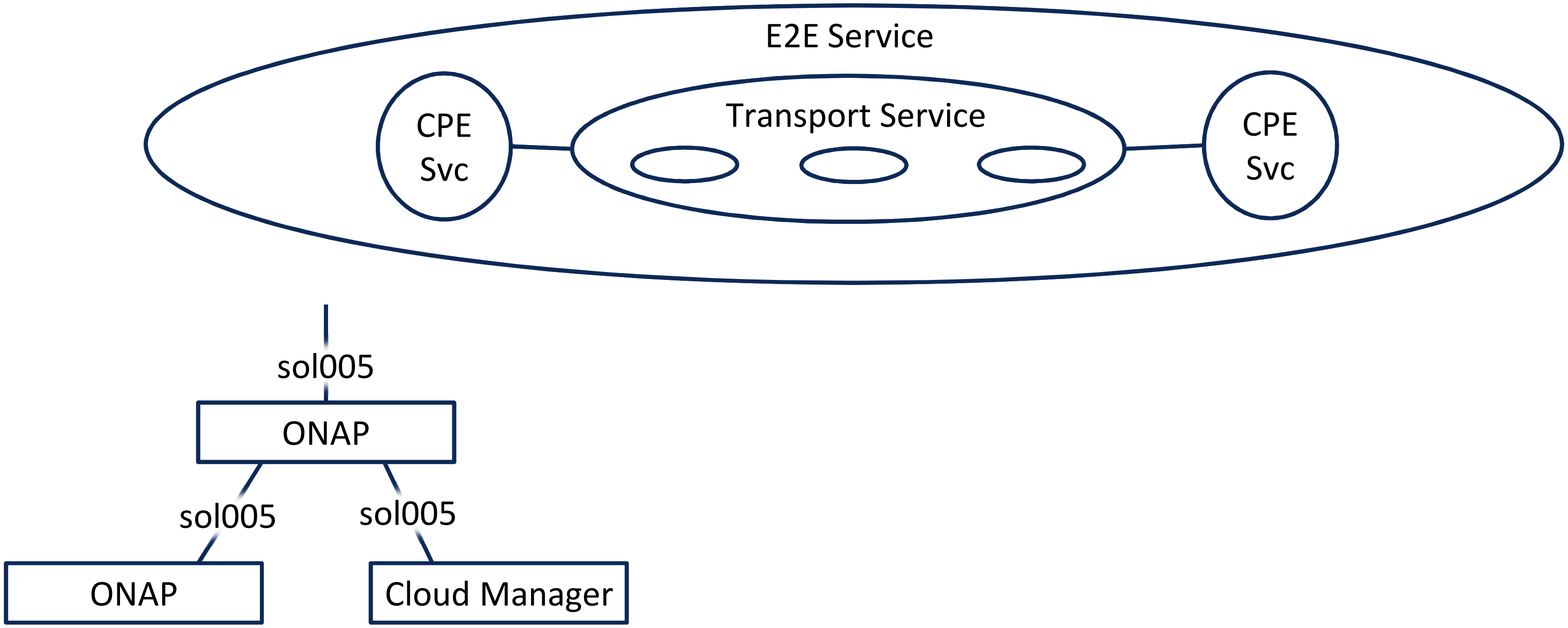
L3VPN_Cust Service Instance
Instance “A”

VRF Allotted Resource Instance
Instance “B”

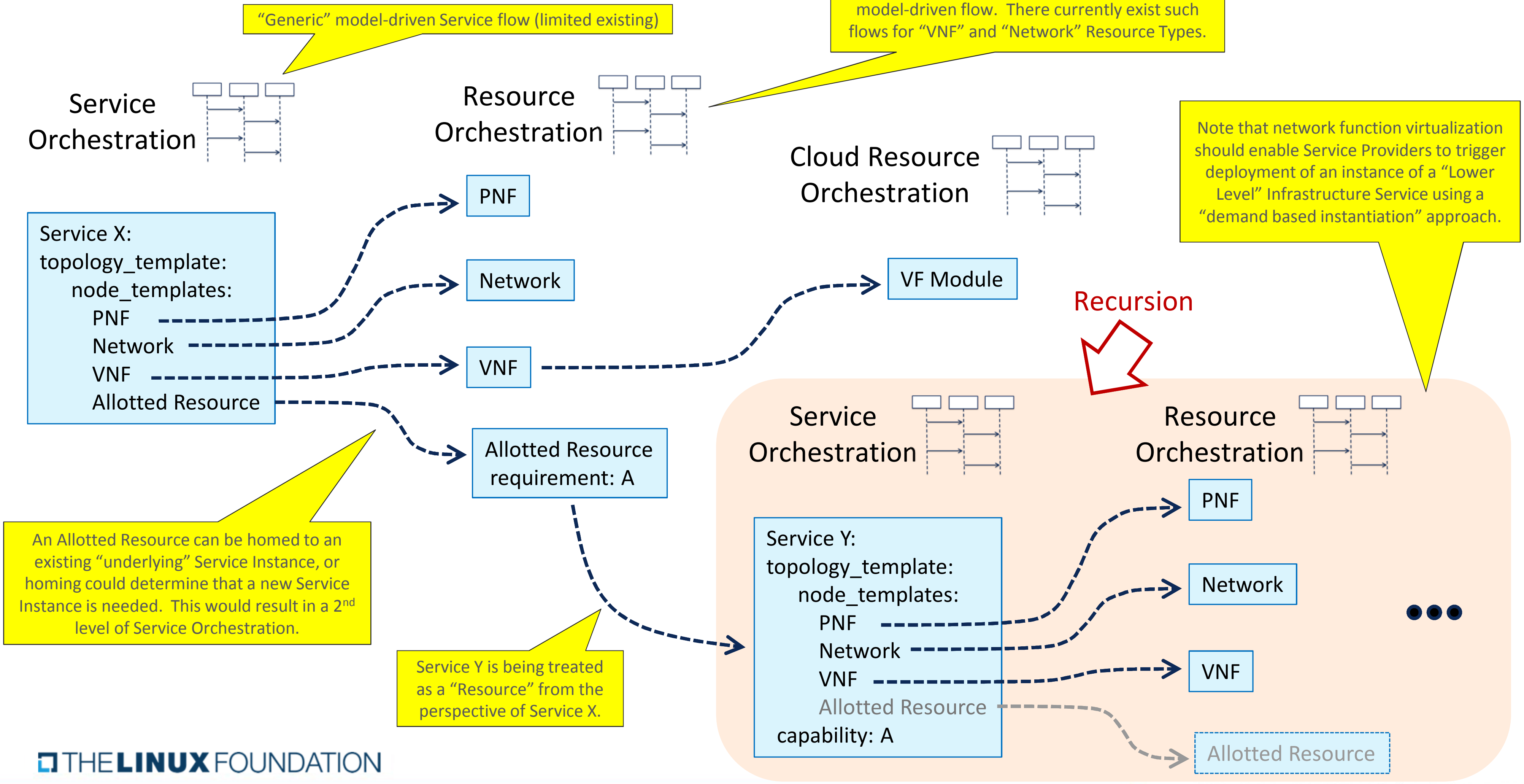
vPE_Infra Service Instance
Instance “X”

vPE VNF Instance
Instance “Y”

vPeNet Network Instance
Instance “Z”


E2E Service
CPE Svc
Transport Service
CPE Svc
sol005
ONAP
sol005
sol005
ONAP
Cloud Manager

Model-Driven Orchestration



"Generic" model-driven Service flow (limited existing)

Each Resource Type has its own "Generic" model-driven flow. There currently exist such flows for "VNF" and "Network" Resource Types.

Note that network function virtualization should enable Service Providers to trigger deployment of an instance of a "Lower Level" Infrastructure Service using a "demand based instantiation" approach.

Service Orchestration

Resource Orchestration

Cloud Resource Orchestration

Service X:
topology_template:
node_templates:
PNF
Network
VNF
Allotted Resource

PNF

Network

VNF

VF Module

Allotted Resource requirement: A

Recursion

Service Orchestration

Resource Orchestration

Service Y:
topology_template:
node_templates:
PNF
Network
VNF
Allotted Resource
capability: A

PNF

Network

VNF

Allotted Resource

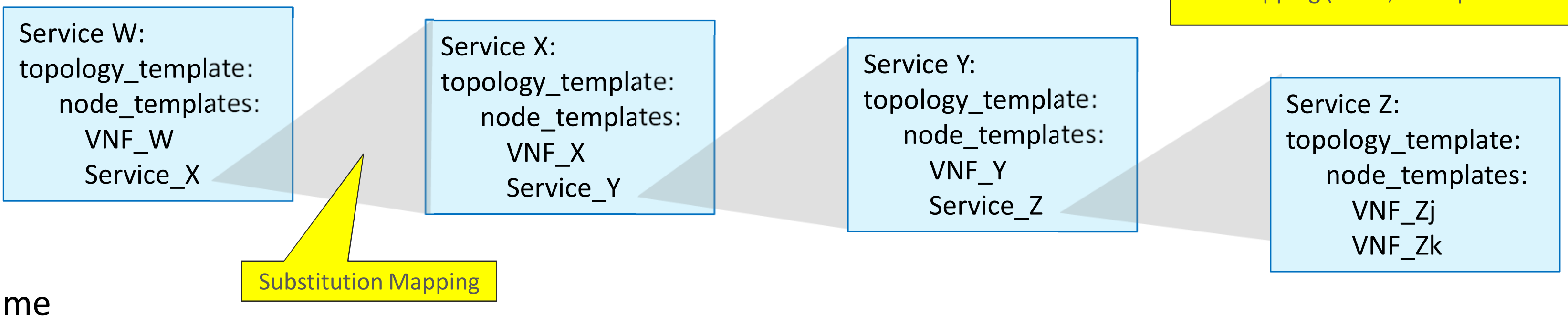
An Allotted Resource can be homed to an existing "underlying" Service Instance, or homing could determine that a new Service Instance is needed. This would result in a 2nd level of Service Orchestration.

Service Y is being treated as a "Resource" from the perspective of Service X.

N-Level Run Time Nesting? Let The Service Providers Decide

Service_W Modeling Example 2

For the case whereby a "higher level Service" consumes the entirety of a "lower level Service's" instance, SDC should support the Design Time ability to construct an "upper" Service Definition from other Services definitions via substitution mapping (a.k.a., "Compile Time Nesting")



Service W:
topology_template:
node_templates:
VNF_W
Service_X

Service X:
topology_template:
node_templates:
VNF_X
Service_Y

Service Y:
topology_template:
node_templates:
VNF_Y
Service_Z

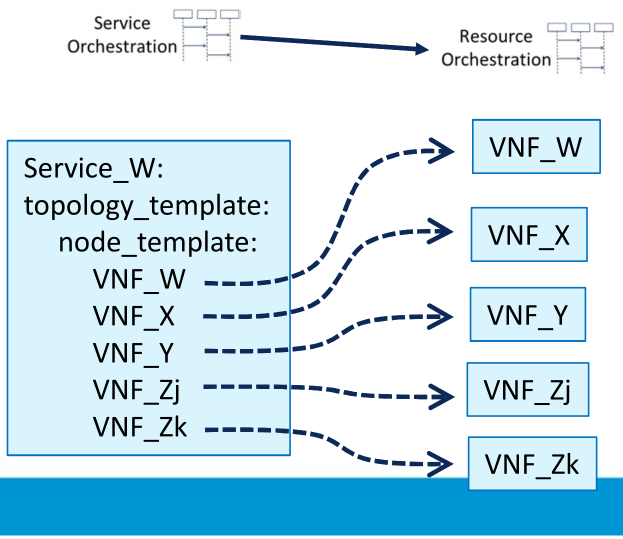
Service Z:
topology_template:
node_templates:
VNF_Zj
VNF_Zk

Substitution Mapping

Design Time

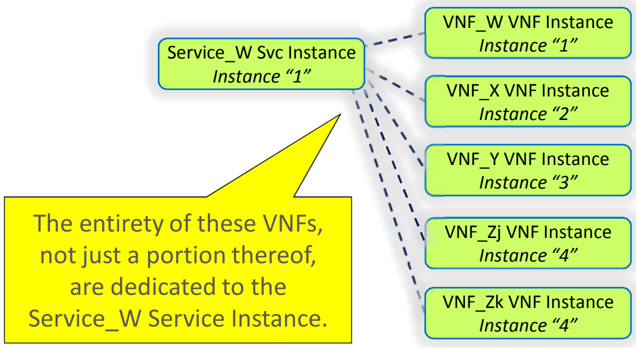
Distribution Time & Run Time

The "lower level Services" would not be visible at Distribution Time. Hence a "flattening" of the run-time orchestration would result.



Service_W:
topology_template:
node_template:
VNF_W
VNF_X
VNF_Y
VNF_Zj
VNF_Zk

A&AI Instance Representation of Service_W Modeling Example 2



The entirety of these VNFs, not just a portion thereof, are dedicated to the Service_W Service Instance.