Technical Disclosure Commons

Defensive Publications Series

May 17, 2017

NFV Dynamic and Adaptative Contextual Execution

Ignacio Aldama
Hewlett Packard Enterprise

Ruben Sevilla Giron
Hewlett Packard Enterprise

Javier GarcÃa Lopez
Hewlett Packard Enterprise

NFV Dynamic and adaptative Contextual execution

Abstract

Network function virtualization is an emerging technology that is changing the paradigm of operations within the telco industry. Associated with the virtualization benefits it also appear a growing complexity to manage and operate complex and dynamic scenarios.

In a traditional network all the elements are physical and so they are limited and very rigid. You may have N and only N elements and it is not easy, neither cheap and neither fast, to either increase or decrease that number.

In an NFV world you got one additional layer as the “physical elements” become virtual and run as software on top of one or more servers as a set of virtual machines (typically more than one) and on top of that those “physical elements” can “breath“ scaling in or out and increasing or decreasing over time the number of VMs assigned to perform the same function so operations over them has no longer a fix limit but a variable over time and context limit.

Problem Solved

In a traditional OSS operation the SW that operates the network elements has some form of workflow/process engine that processes a set of actions (some serial some even in parallel) and trigger branch workflows passing different arguments from the parent to the child. Typically each workflow performs the same set of operations over the given parameters.

If we make an analogy with a car in the traditional pre-NFV you use to have cars for cities cars for mountains, car for icy terrain etc. And so you have SW (workflows generally speaking) that perform several actions (like provision a VPN) but on a VERY particular context (VPN over MPLS network or VPN over ATM network)

This traditional way of operation has two main drawbacks:

- Traceability of complex operations: we have found tons of frameworks that always run the same set of operations Configure VPN / Start /Test and have a nice tracking (even in gant chart shape) .
    - In a car analogy it will be like a car that tells you how much you consumed in the last 100 KM.
  - We have found as well several frameworks able to trigger a variable set of operations even in parallel Configure VPN (if not configured) / Start (if not started) / Test (each user on the VPN) but those have a very poor tracking and they rely mainly on logs not so easy to follow.
    - Like an hybrid car electric , gas and fuel very efficient but at the end you do not know exactly how much you consumed on 100Km and why is so much or so less

- Re use of pieces as lego pieces: If you have a complex logic to execute several actions and you need to pass information (parameters) between sw elements then you face the well know problem of parameter matching where each process need to pass a set of variables/parameters to the child making difficult to re use the Childs with different parents (something like lego pieces) as the parameter names may or may not match and the names normally is tied to the coders discipline that is not so easy to achieve on large teams. As a result creating a new piece ends up in either creating a new piece (even if the piece already exist) or spending time into adapting it.
    - In the car analogy this will be like having different sets of engines that fits only in one or two car models and need re-adaptation to fit and be plugged into other car models

In the NFV space where the operations, the order of the operation and the parameters change rapidly as the technology is not yet mature an still suffers different redesigns be able to reuse lego pieces with the minimal changes possible becomes specially critical.

As well the NFV environment is dynamic so workflows/processes configuring a VNF cannot be tied to just configure the VMs of a particular VNF component as the VNF can scale and create or destroy vnf components or VMs or even networks and the next execution of the same process may need to operate over a different set of targets.

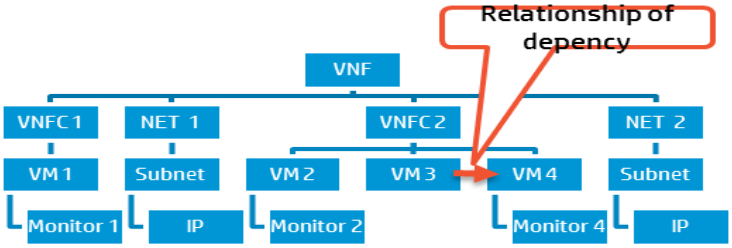
Description

We have developed a contextual execution engine that basically reduced all the input parameters to three: operation to be performed, context on which the operation should execute and execution target.

Being all the input/output parameters always the same allows the engine to easily trigger any configuration of workflow in any order and keep trace of their execution state. Any other parameter needed by any sub workflow will be stored in the context so each sub workflow does not need to receive any other parameter but the three main ones and if needed search within the given context for any relevant information. This will be like driving a car and specifying before start (operation) Drive, (context) over mountain, (target) to my house and any other needed information will be gathered by the sensors of the car as the car drives

This also allow the engine to search the context and instead to trigger one workflow with a complex logic based several parameters to trigger as many small workflows as needed based on the context.

For example it is not rare to find a VNF with two networks and two components and several VMs where some VMs have monitors and others not.

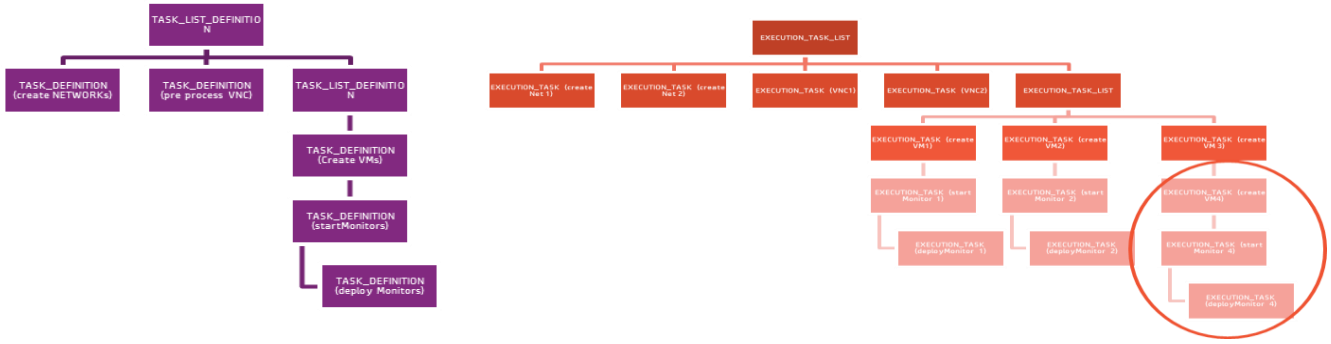


Example VNF target

Instead of trigger a fix set of workflows the engine can read the configuration of the operation and see which groups of task (Task List definition of TLD) need to execute and how many tasks (Task Definition or TD) are involved in each group of tasks.

In the car analogy this is like saying drive as fast as the signals allow you so the car keeps reading the signals on the road (context) and adjust the speed as it drive (part of the way it goes at 100KM/h other part it goes at 30 km/h)

The trigger of the task is contextual driven so a task saying create VMs will trigger as many workflows to create a single VM as number of VMs found on the given target creating from each Task definition a set of corresponding Execution Tasks (ET) and from each Task List



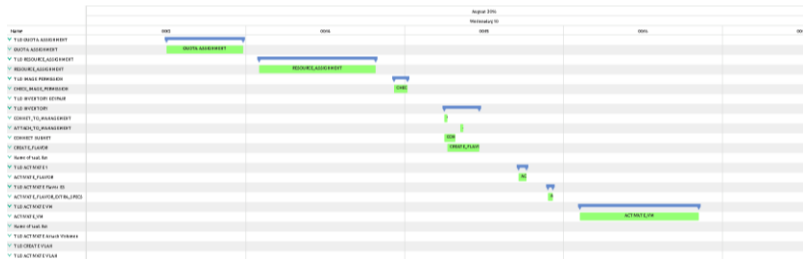
definition an Execution Task List (ETL)

Example operation defining 1 task list and several child tasks and the corresponding execution

The result of the execution can be easily traced and even displayed on a GUI showing a Gant chart with the details of each task.

In the car analogy this will be like displaying the consumption of each engine type (electrical / gas / fuel) for each part of the road.

Also the jobs can listed and filtered by the typical parameters (user, date) and by the input parameters operation, target and context



Execution task list visualization

Prior Solutions

There exist several workflow engines and even BPM (business process engine) liker concepwave or even oracle bpel ending even with open source solutions like JBPM and activity

All the traditional workflow/process engines are very parameter dependent when starting Childs from a given parent and none of the above has been able to successfully define a common context.

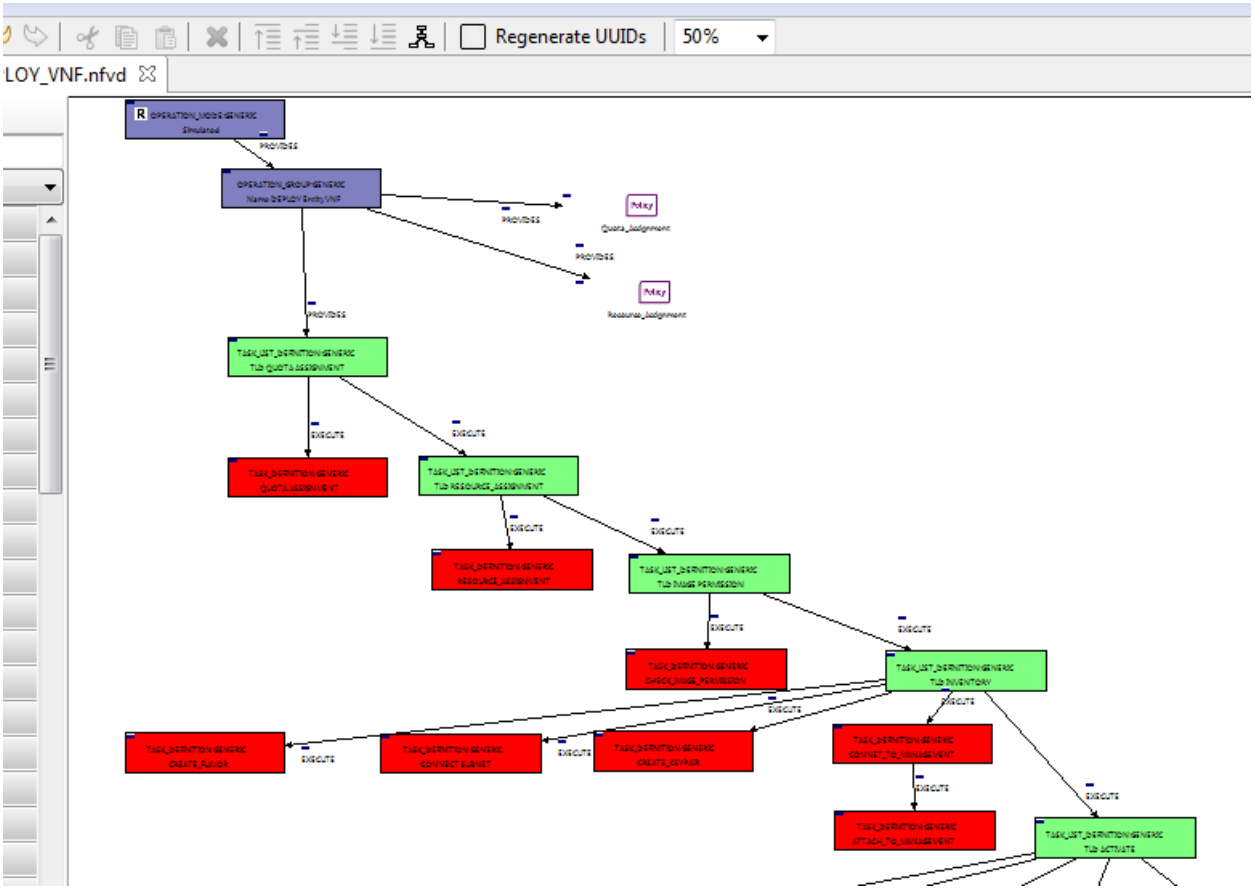
As we mentioned before we found dynamic frameworks with poor tracking and very dependent or sensible to parameters changes and also very rigid process with nice tracking and no flexibility to adapt to the context

In the end we have achieved all three aspect highly dynamic based on the context, low parameter dependence and good tracking

Within NFV Director we have reused the graph oriented NFV model (previously patented) and use the whole model as the context so at any given time the execution process is located in one of the branches of the graph model and can navigate from that point to find out if the needed parameters has been stored previously by any other execution or not or just to find within the target how many items are candidates (based on its state) to trigger a child workflow.

Advantages

This solution includes definition of new operations by an advanced operator using a graphical tool to create modes (each mode will group different set of operations), operations (each operation will group several task list and task definitions) and of course tasks list definitions and



tasks

Each task can be configured to:

- Trigger a workflow and to rollback on error with another workflow
- Lock the context based on a given target to manage parallelism
- Retry several times the same task
- Order all the dynamic generated tasks based on an parameter that must be found on the context
- Find data in the given context following a particular path
- Set the target to a given status when execution is successful

Each task list allow the parallel execution of several tasks and after the parallel execution of several task list

Each particular task will trigger as many workflows as targets found on the context but all the found ones can be order by a custom parameter (that must be found on the context)

This solution allow a nice and well defined tracking , allowing every process triggered to be tracked, secured and managed if a race condition happen

This solution allows high reusability as the same WF (for example create VM) can be reused in several task, with several context even with different configuration (with or without rollback, with or without re-tries, with order or without order)

As the workflow can be the same several times the regression tests are less and the quality improve over time as if a bug if found and fixed on a workflow it will be fixed for all the task that use it.

Packing the workflow into tasks and execution task list allow to create dynamic execution plan that due to its visual nature is easy to track re-design and reuse. As well the packaging allow to re use already existing workflow into this new way of operation.

Summary

The contextual engine developed allows a high re usability with less regression testing and improve the quality overtime providing an easy way to create new operations just by combining the already existing pieces in a different execution plan

This contextual approach allows not only to approach the dynamic NFV ecosystem with the right system view but as well solve other traditional problems on the typical oss space where process tend to be rigid with long cycles of development and high risk of decreasing the quality over time.

Disclosed by Ignacio Aldama, Ruben Sevilla Giron and Javier GarcÃ-a Lopez, Hewlett Packard Enterprise