An introduction to Writing Transport SDN Applications (Hands-on)

Ricard Vilalta (CTTC/CERCA)

Karthik Sethuraman/Yuta Higuchi (NEC)

CTTC NEC Orchestrating a brighter world

Short course Materials

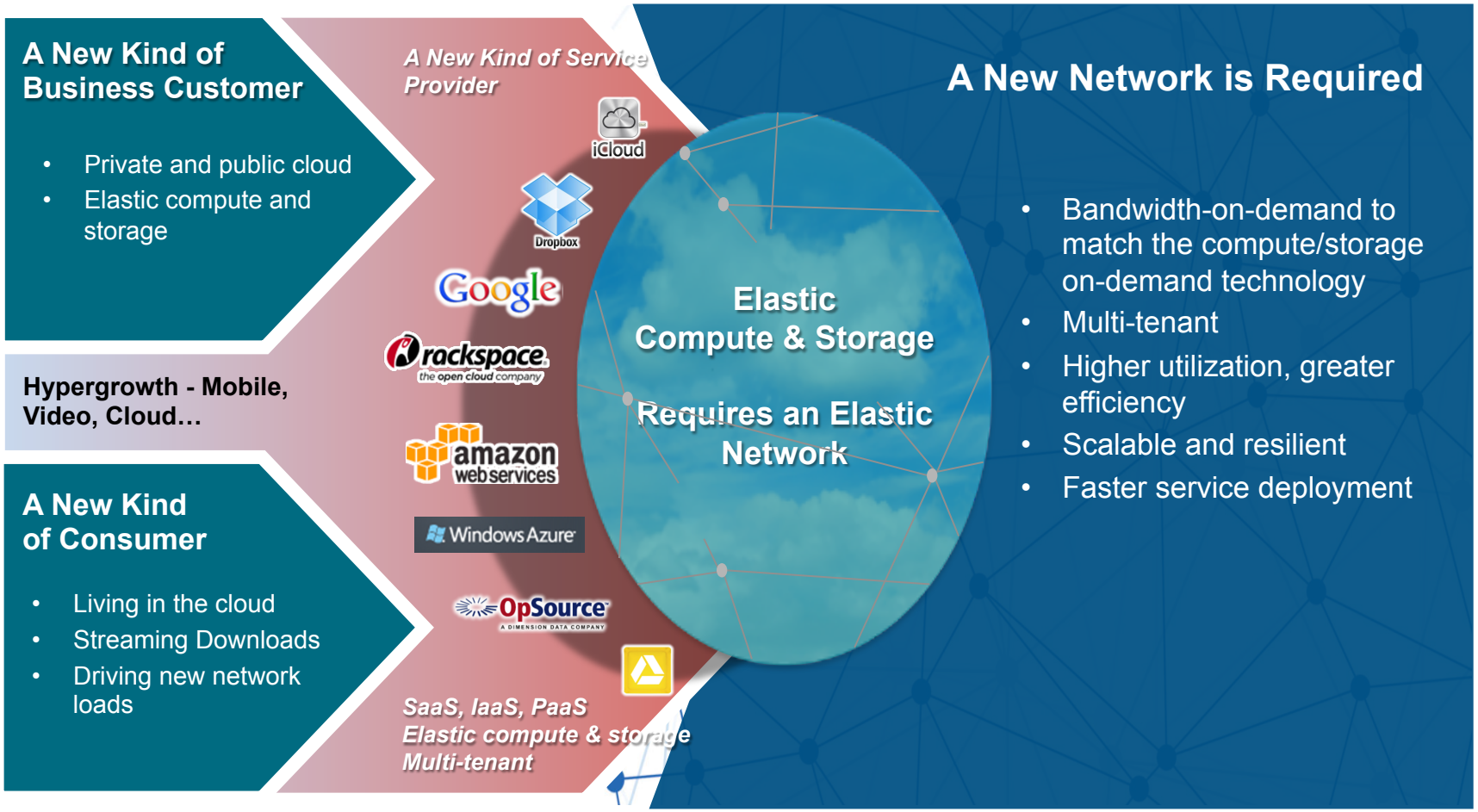
- Please go to github repository to get latest version:
  - https://github.com/rvilalta/OFC_TAPI_SC
- For a perfect hands-on experience, a VirtualBox VM image is needed. Please download the course VM from the link below and make sure the VM is installed and loads/starts up on your PC before travelling to OFC:
  - http://tiny.cc/ofc-tapi-sc

Outline

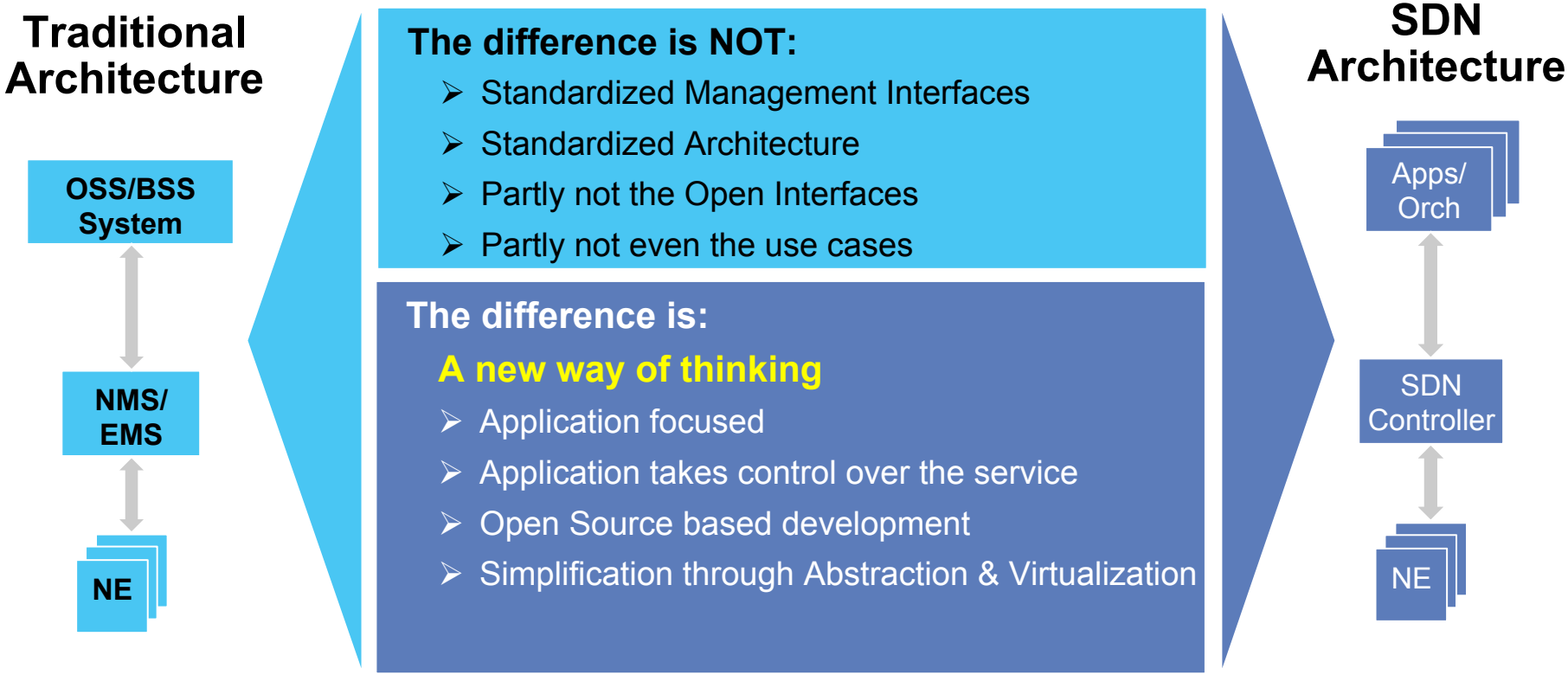
- 13:30 – 13:45 : Welcome, agenda and motivation
- 13:45 – 14:30 TAPI Overview and Architecture
  - Relevance and position of TAPI in SDN Architecture
- 14:30 – 15:30 : Hands-on : Using TAPI SDK Reference Implementation
  - Hands-on: TAPI Concepts – REST API, Context, Topology, Connectivity, etc
  - Hands-on: Writing and running a simple TAPI Topology & Connectivity client
  - Hands-on: Tooling: Generate an Python Server & Client from TAPI Open-API Spec
  - Hands-on: Write application logic and execute the application
- 15:30 - 16:00 : Break
- 16:00 – 17:00 Hands-On : Using ONOS with TAPI
  - Introduction to ONOS northbound REST API, Mininet config
  - Hands-on: Write ONOS client (topology & flows)
  - Hands-on: Write TAPI RI interacting with ONOS
  - Hands-on: Mininet configuration Scenario
- 17:00 – 17:30: Wrap-up & Conclusion
- Extras:
  - Hands-on: From TAPI UML to code (UML, YANG, OpenAPI/Swagger, Extensions/Specification)

TRANSPORT SDN - MOTIVATION

What we see in the market ?



Why is SDN different from traditional SDN Architectures?



Why do we need SDN in Transport?

Principles of SDN

Programmability:
- Programmable interfaces
- Applications focused architecture
- Abstraction & Virtualization
- Multi-Tenant capabilities

Openness:
- Open Stanadsrds & Interfaces
- Open Source SW

Integration focused:
- Multi-layer
- Multi-vendor

What it Enables in Transport Network

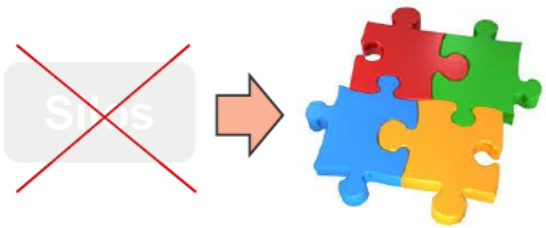
Innovation:
- Opens doors for new service models
- Service differentiation through new application

Simplified Architectures:
- Integrated E2E / Multi-layer service creation
- Automatic reaction on errors or any changes

Financial Benefits:
- Opex: efficient service setup
- Capex: fast ROI / hardware utilization
- New revenue opportunities

Keys to success

Break Silos

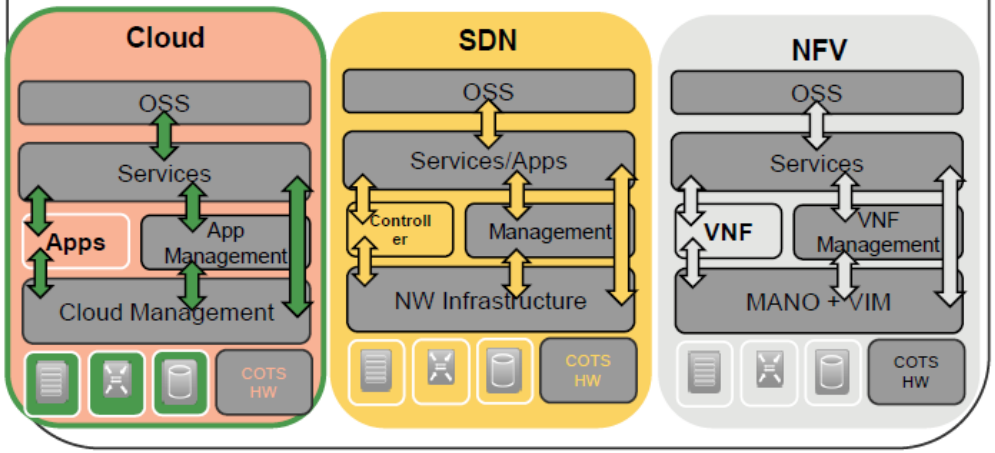


Metrics of Success

1) Multi-vendor Interoperability
2) Industry Adoption

Avoid moving from silos of HW to silos of SW

Avoid new silos of Management



ONF TRANSPORT API (TAPI) OVERVIEW AND ARCHITECTURE

Transport API – Simple Problem Statement

APP

Network Orchestrator

Open Standard T-API

ONOS APIs

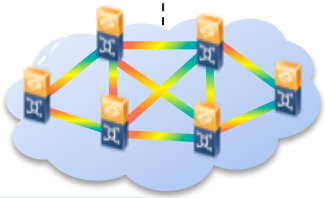
ODL APIs

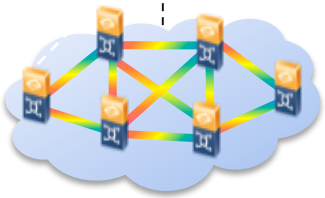
Proprietary APIs

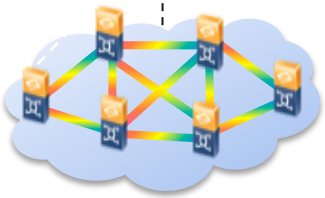
ONOS

ODL

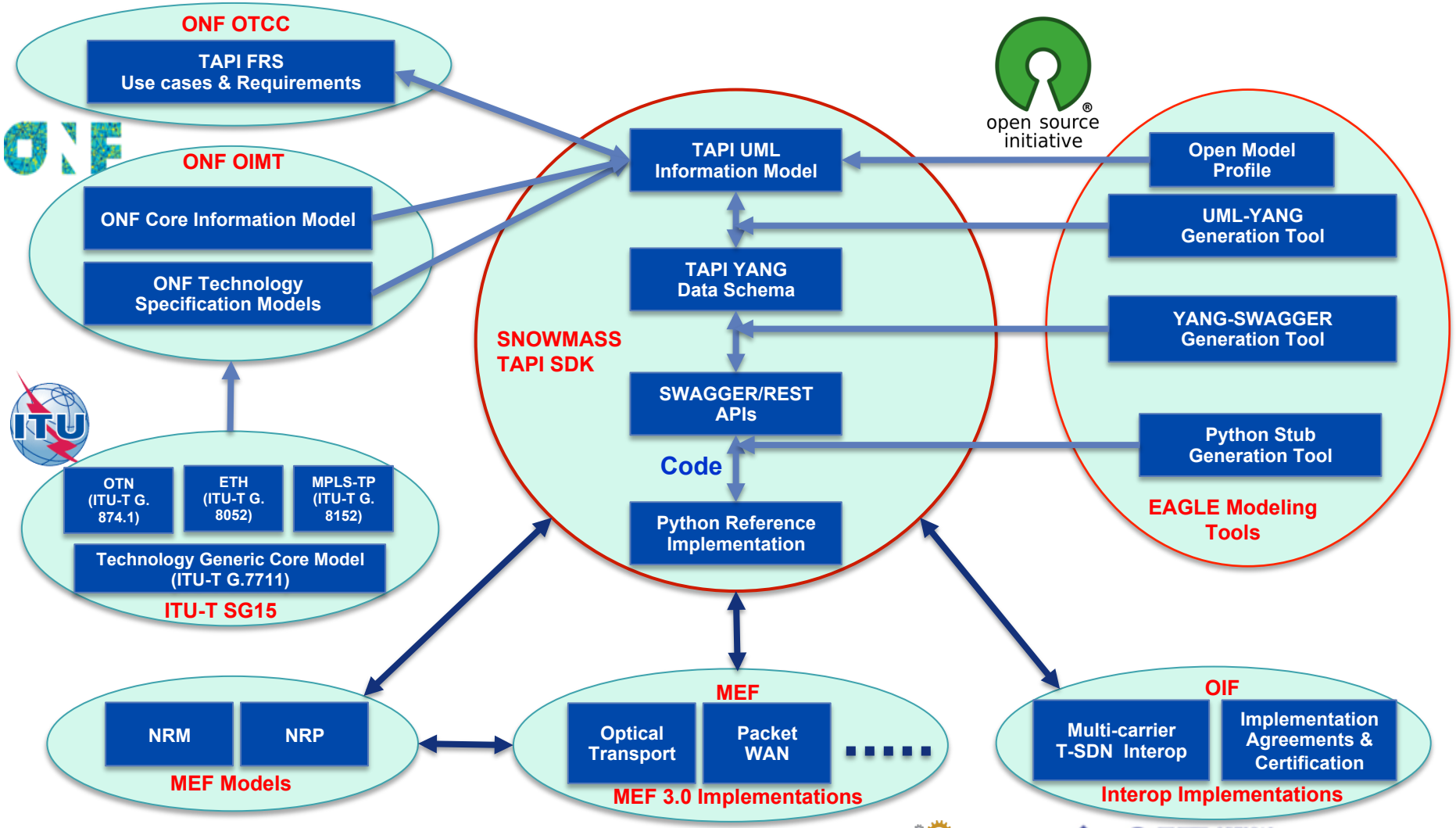
Proprietary



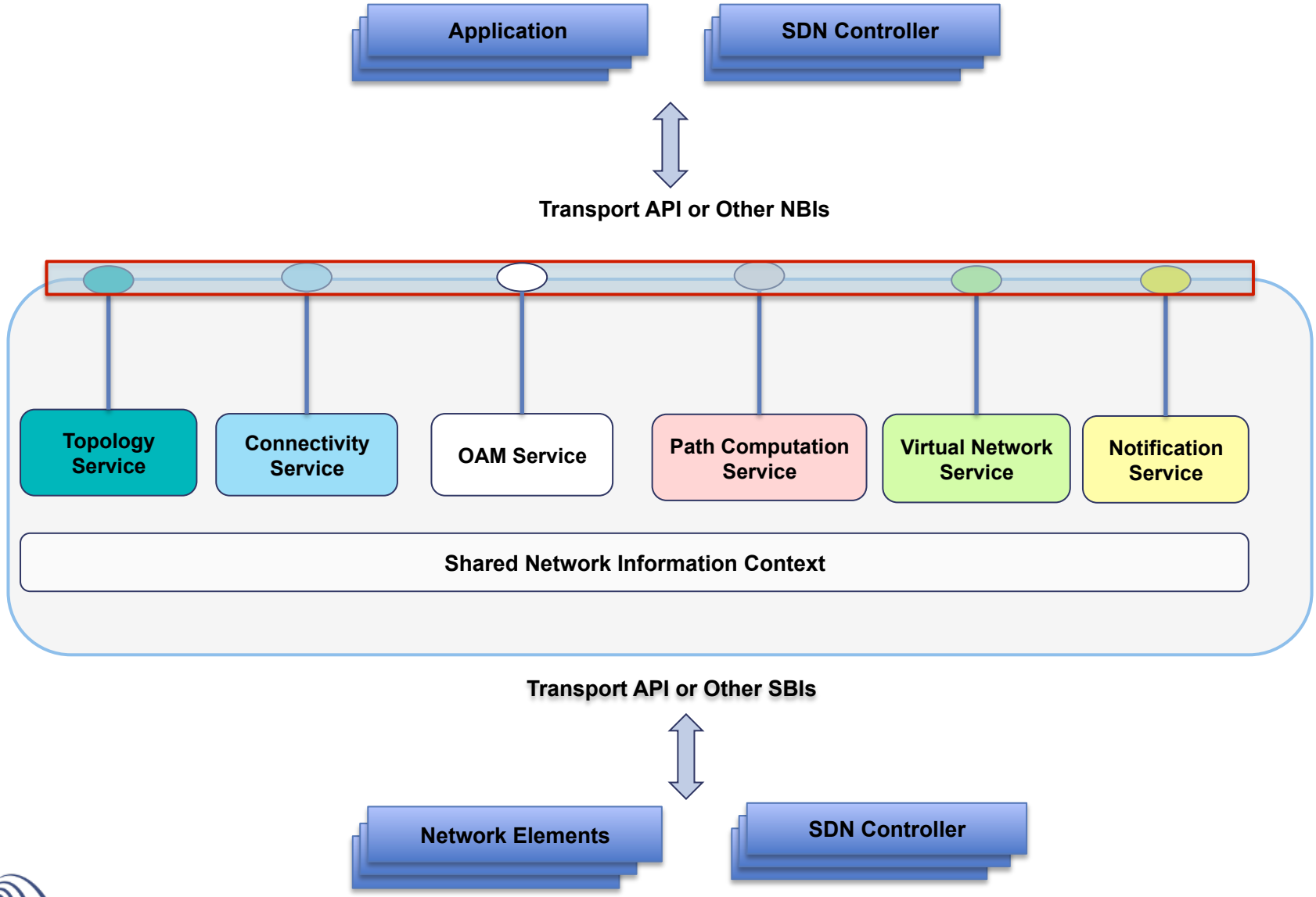




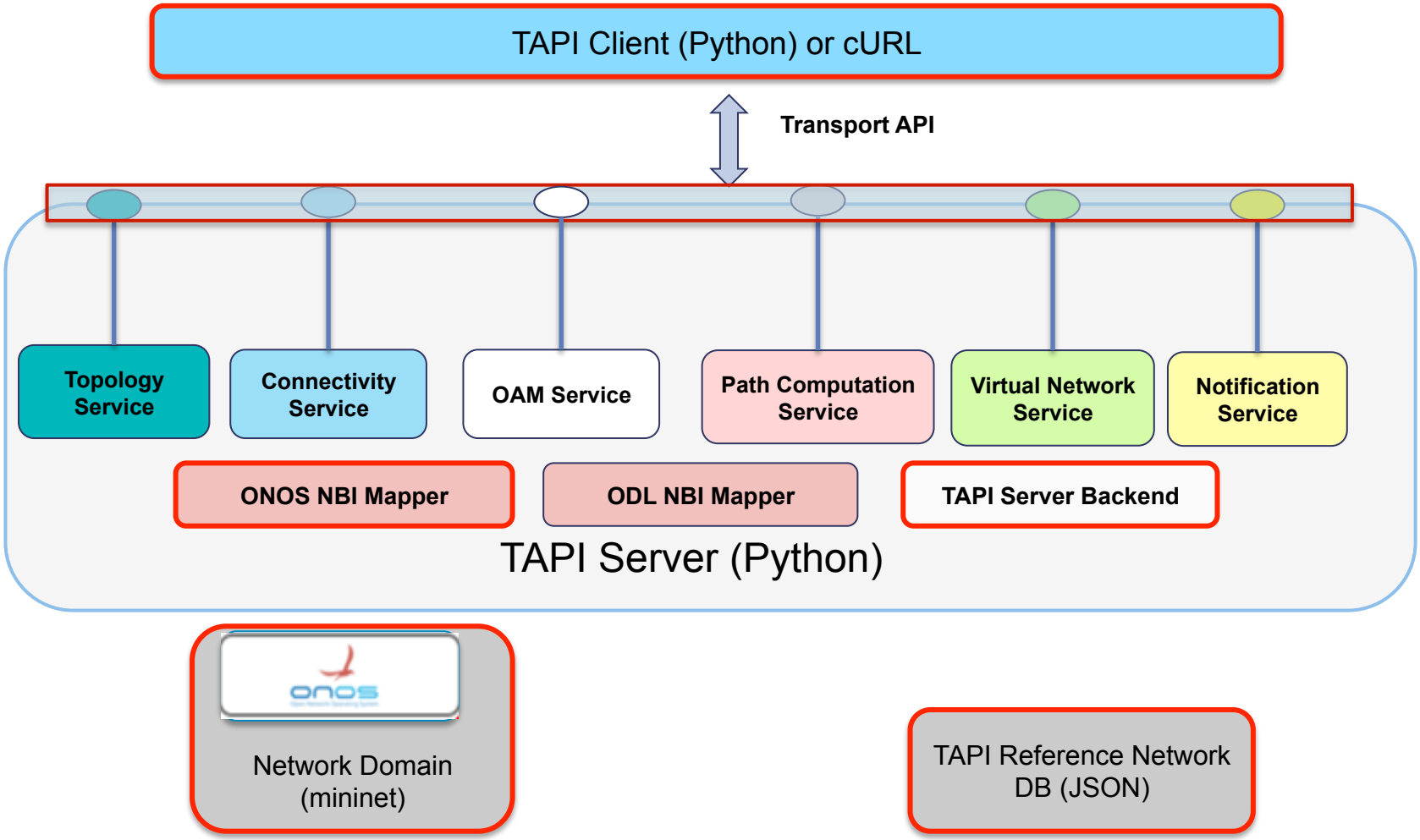
Confluence of Standards and Open Source



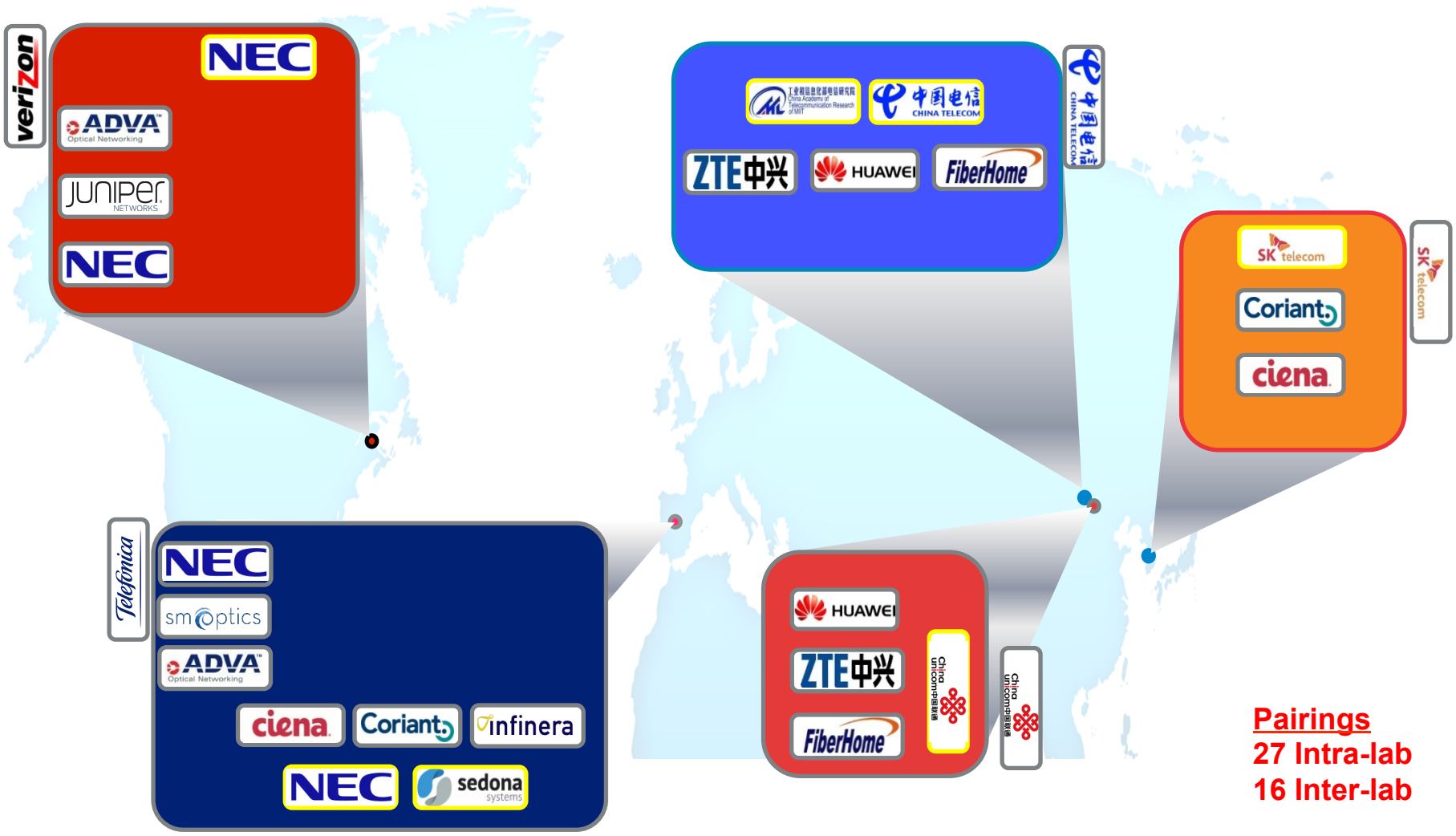
ONF Transport API (TAPI): Functional Architecture



TAPI Snowmass RI - Prototyping Controller-agnostic API

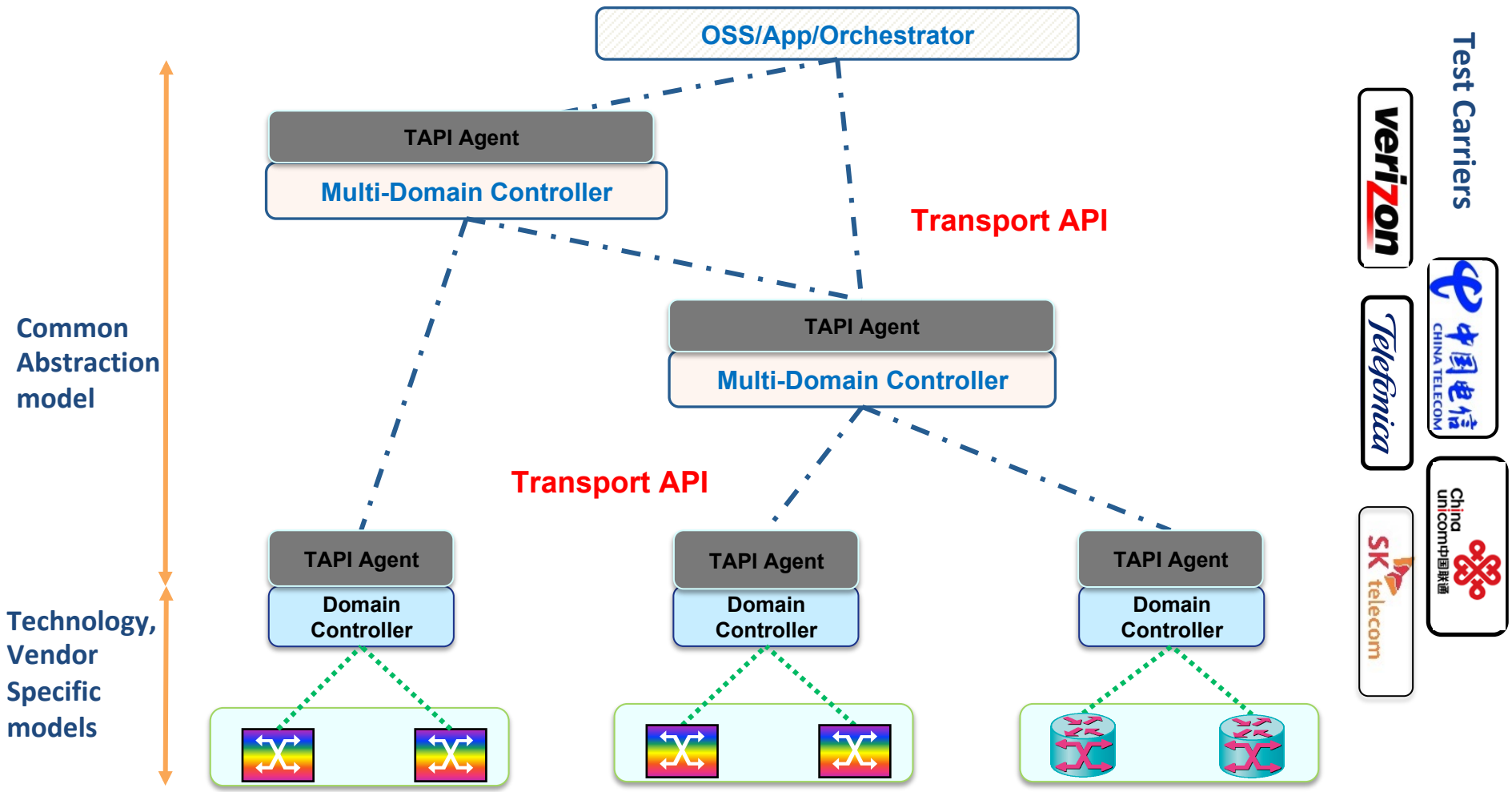


TAPI in 2016 OIF-ONF Interoperability Demonstration

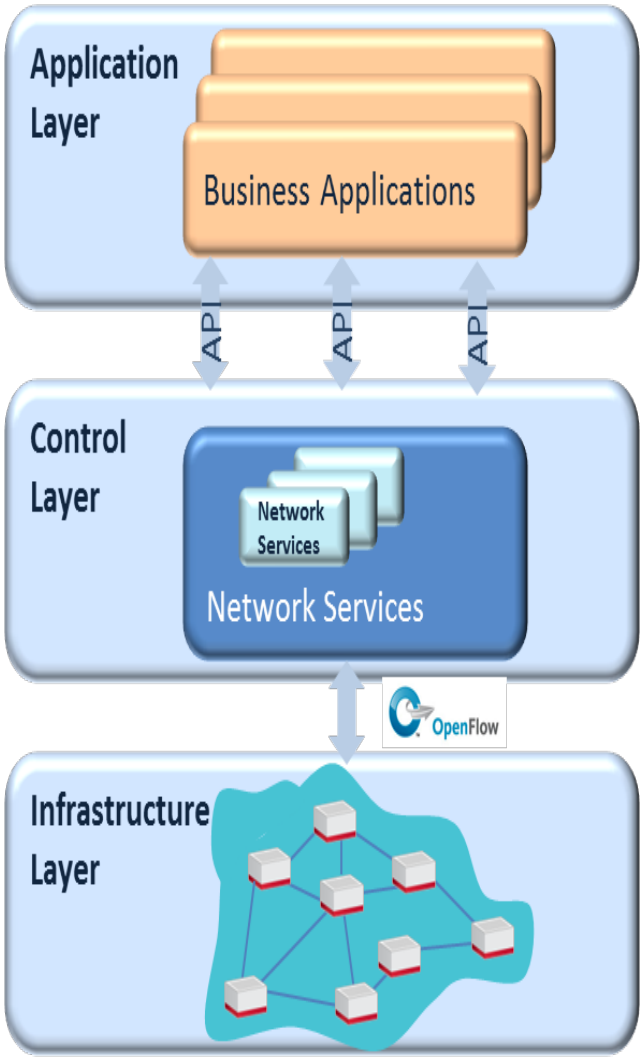


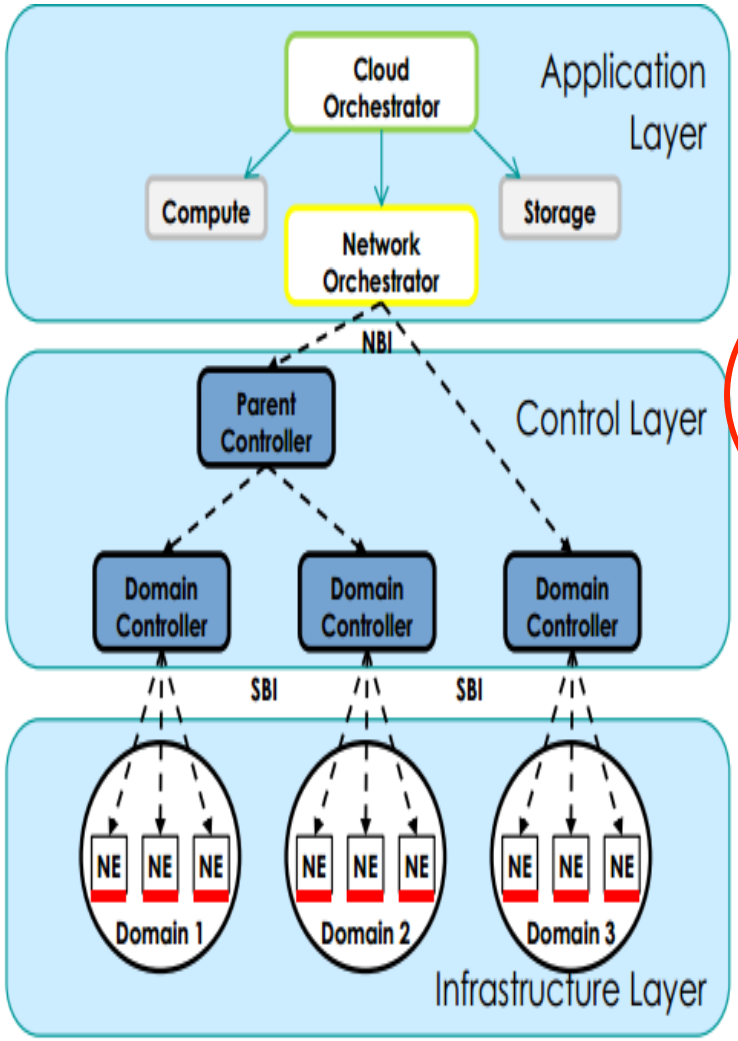
Pairings
27 Intra-lab
16 Inter-lab

2016 OIF-ONF Transport API Interop Demo



2014 OIF-ONF Transport SDN Interop Demo





OIF OPTICAL INTERNETWORKING FORUM

TAPI Precursor

REST APIs

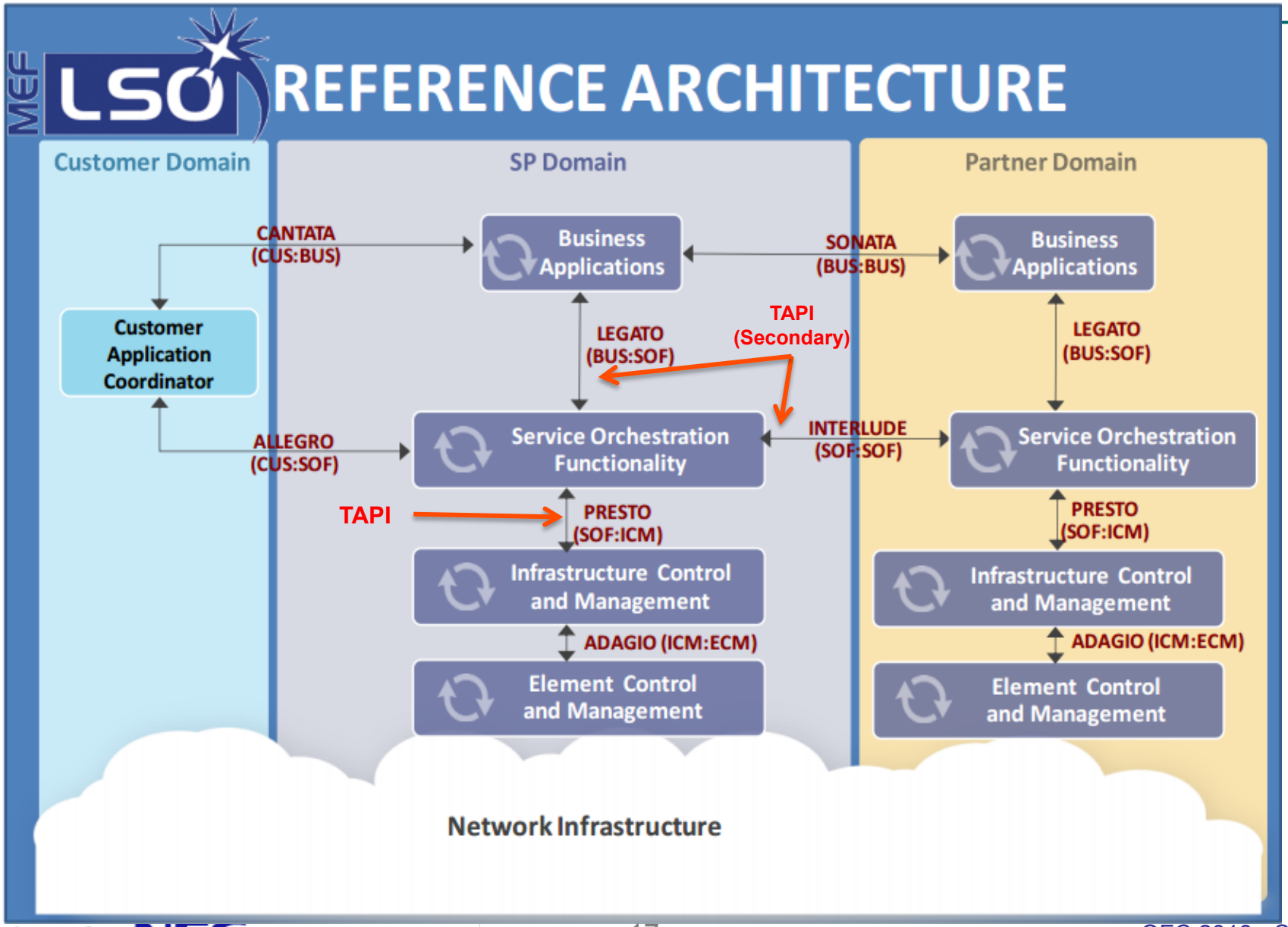
Service Request
Topology

OpenFlow

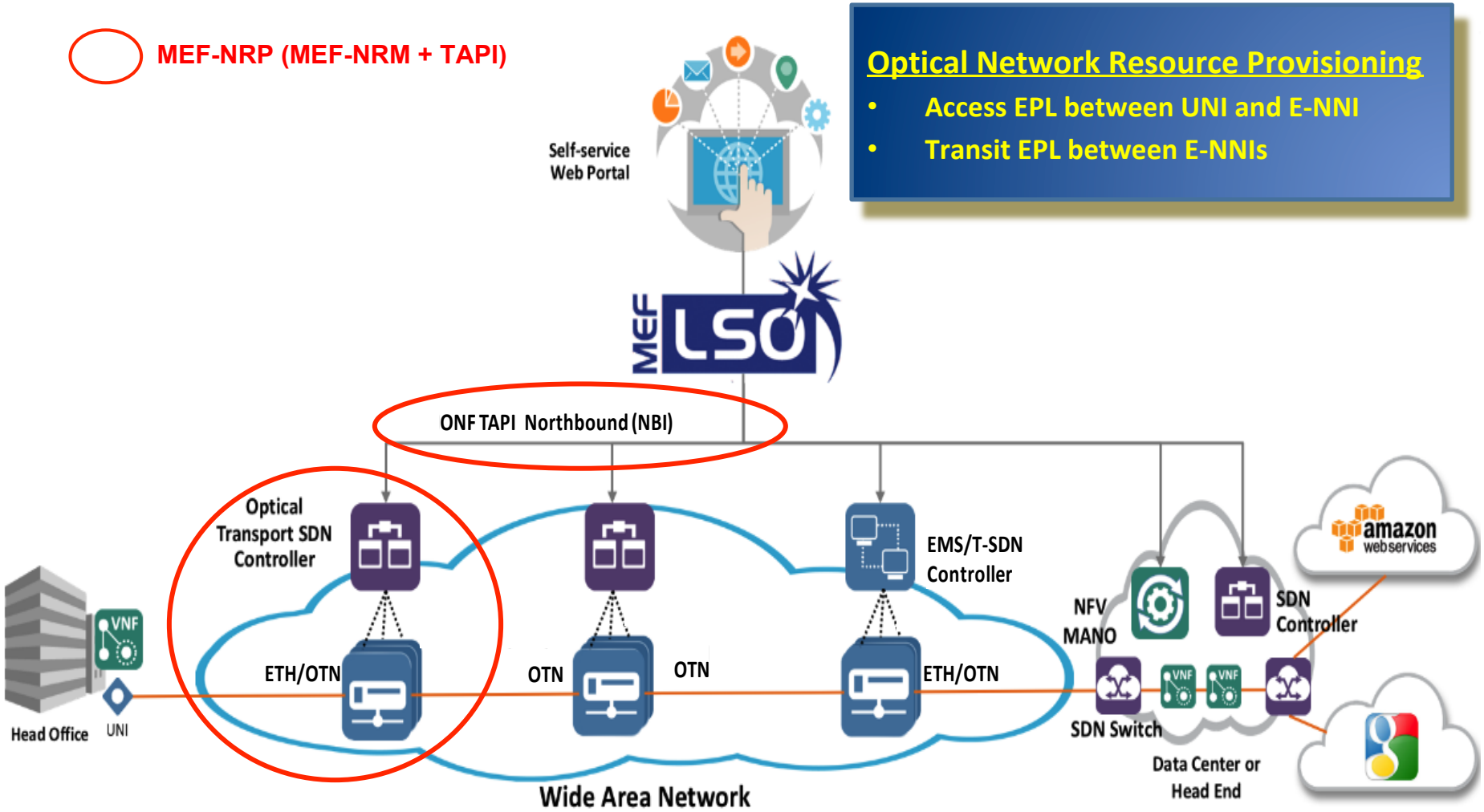
Control to Dataplane
Control to Virtual Network

CTTC NEC Orchestrating a brighter world

Where does TAPI fit in? MEF LSO Architecture



MEF 3.0 Optical Transport Use case

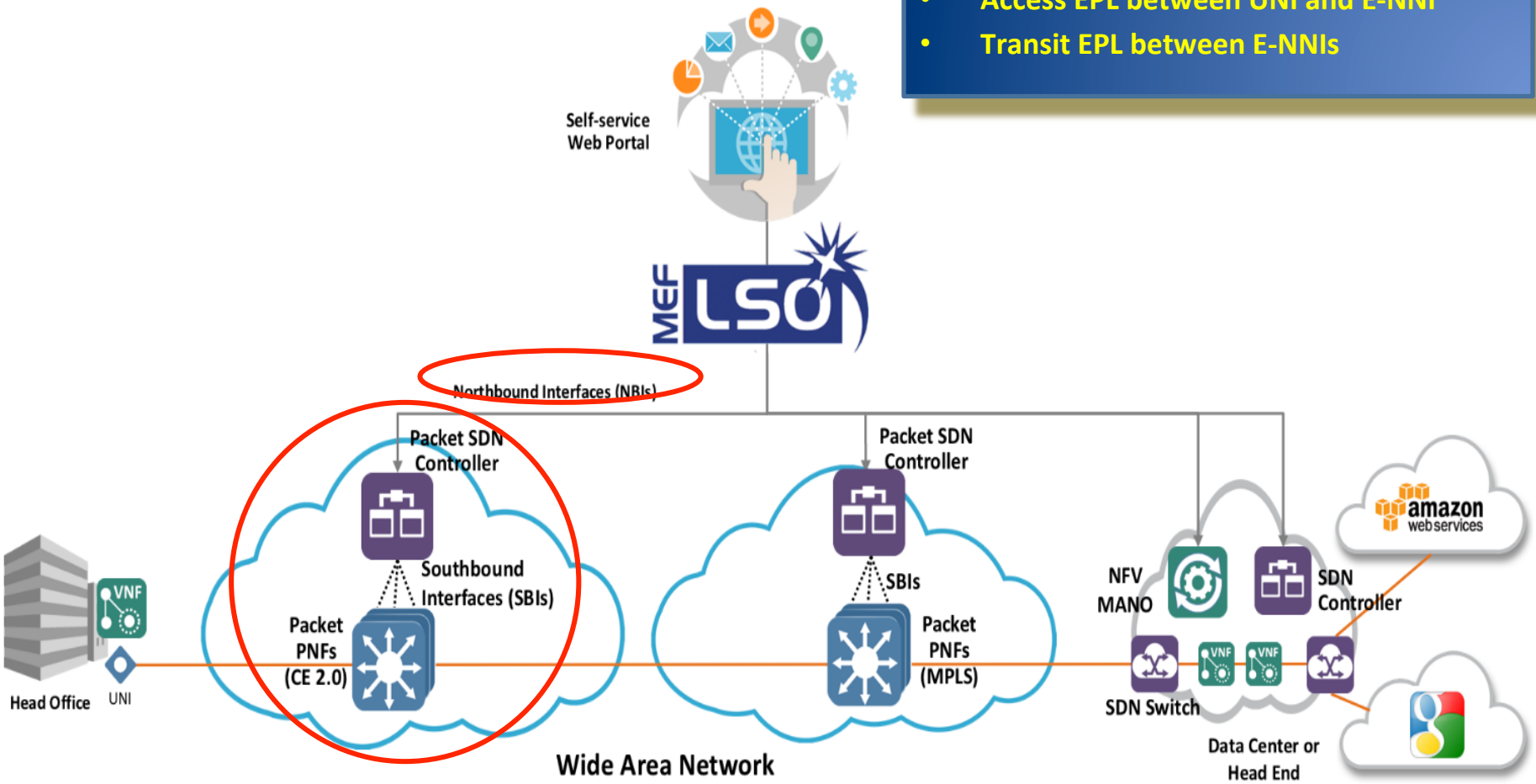


MEF 3.0 Packet WAN Use case

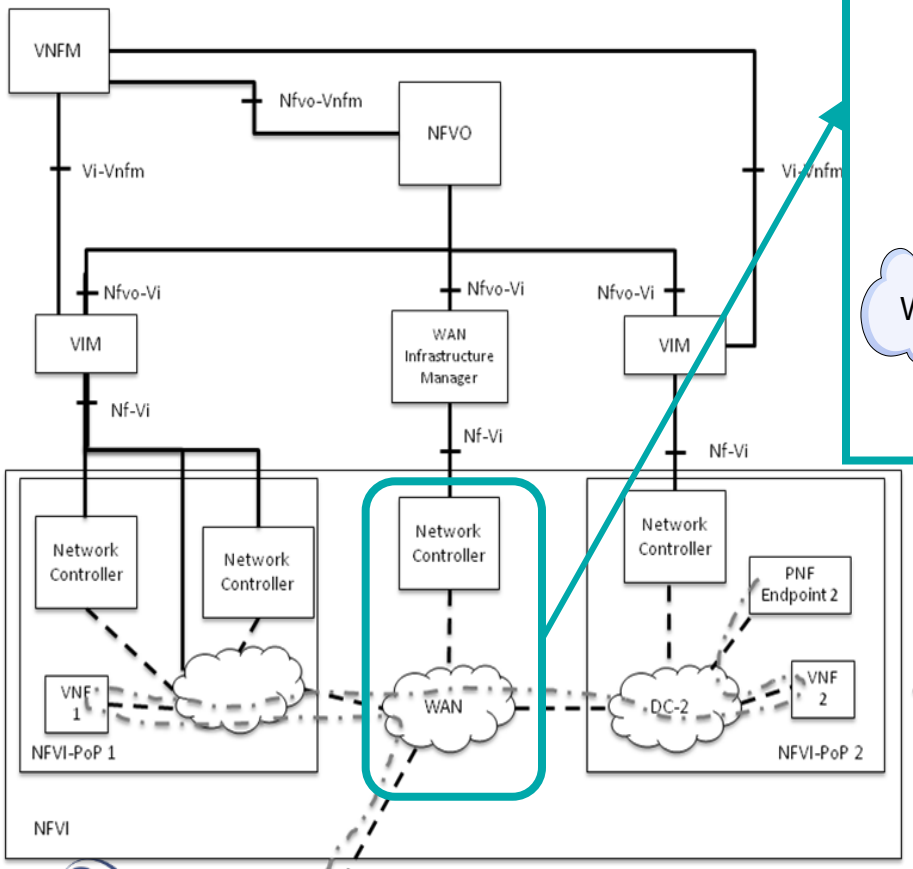
MEF-NRP (MEF-NRM + TAPI)

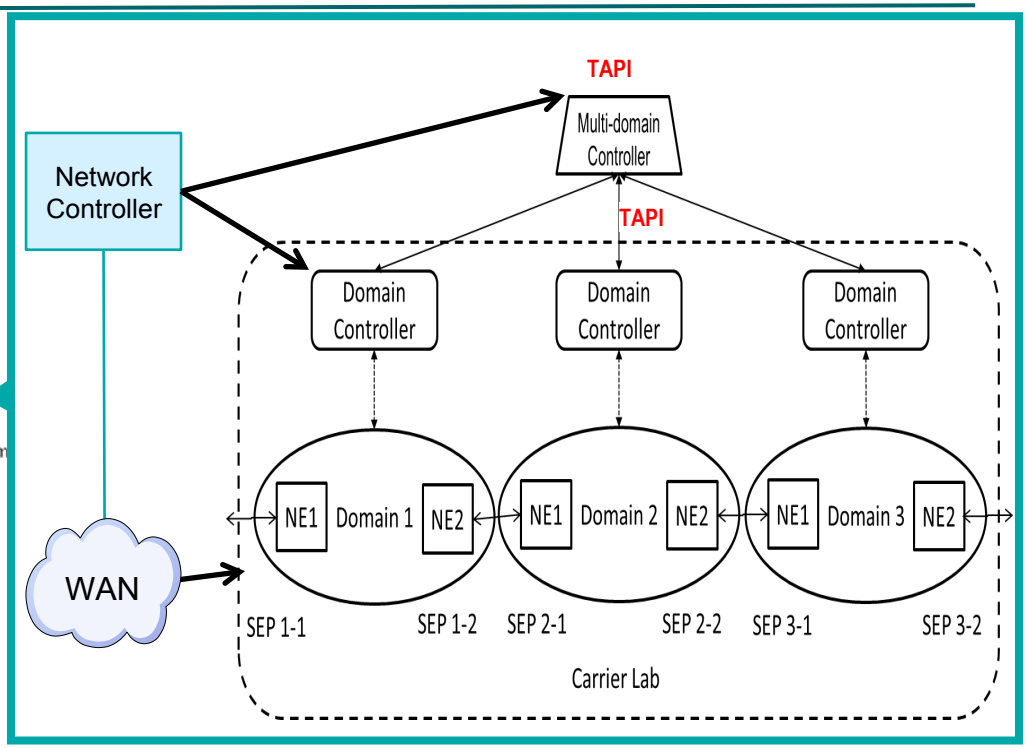
Packet Network Resource Provisioning

- Access EPL between UNI and E-NNI
- Transit EPL between E-NNIs



TAPI in ETSI-NFV WIM PoC (based on 2016 OIF-ONF Interop)





VNF Forwarding Graph

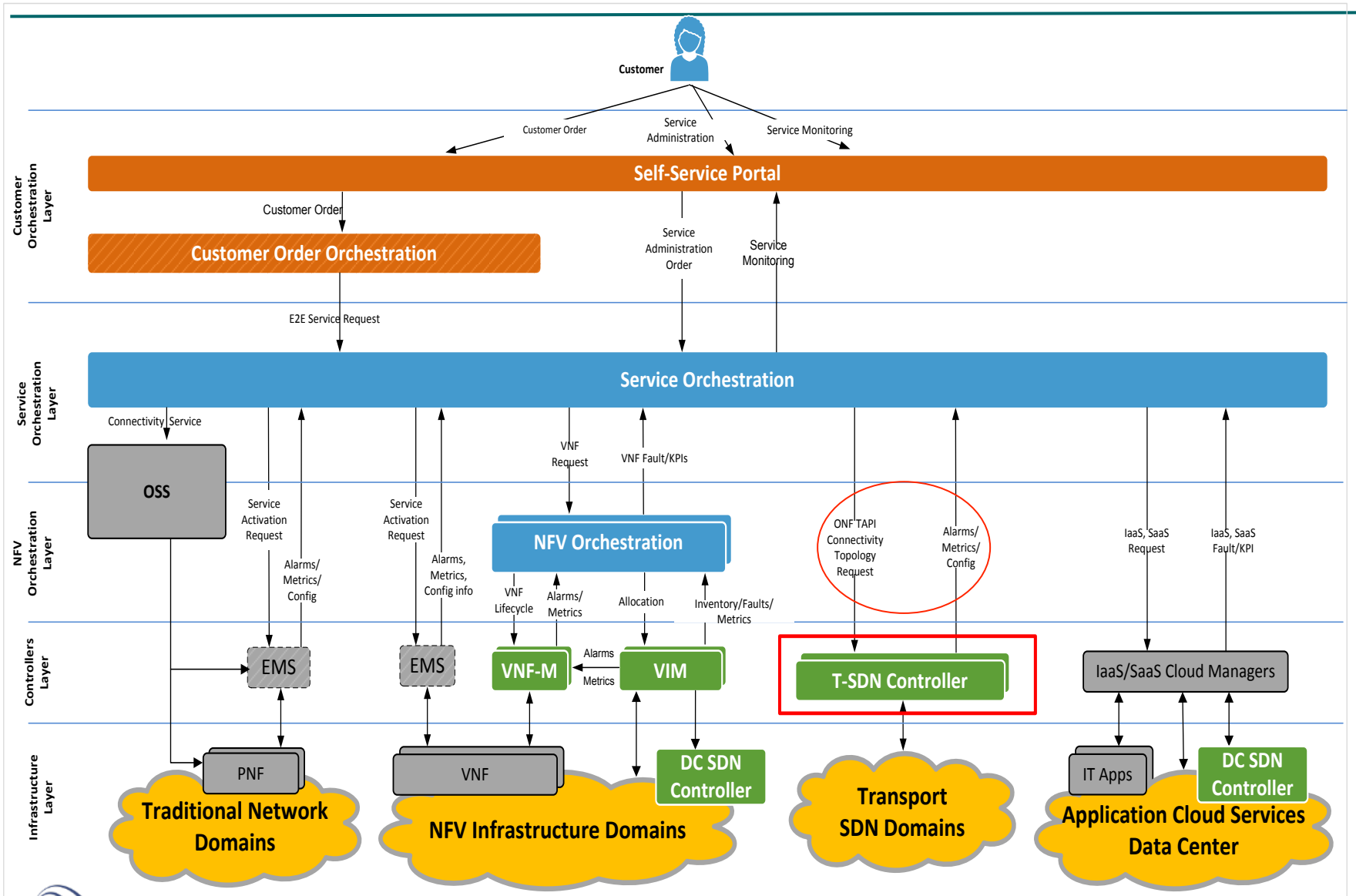
Reference point out of scope

Reference Point in scope

Multi-Site NFVI-PoP Interconnect

- WAN Infrastructure Manager interfaces
- Connectivity Lifecycle Management
- OSS/BSS, NFVO, WIM/VIM, and Network Controllers interactions

Where does T-API fit in? The Big picture…



TAPI Feature Set

TAPI SDK 1.0 (Q4 2016)

- **Topology Service**
  - Retrieve Topology, Node, Link & Edge-Point details (Across al layers)
- **Connectivity Service**
  - Retrieve & Request P2P, P2MP, MP2MP connectivity (Across all layers)
- **Path Computation Service**
  - Request for Computation & Optimization of paths
- **Virtual Network Service**
  - Create, Update, Delete Virtual Network topologies
- **Notification Framework**
  - Subscription and filtering
  - Autonomous/Push mechanism

TAPI SDK 2.0 (Q4 2017)

- **Node Constraints**
  - Ability to specify connectivity/blocking constraints
- **Protection**
  - Multi-layer, Multi-Domain
  - Based on use cases under discussion
- **OAM/Monitoring**
  - Consistent Multi-layer abstraction and model – L0-L2
- **Alarms/TCAs/Counters**
  - Multi-Point, Multi-Layer, Multi-Domain use case enhancements
- **Multi-Technology**
  - ODU
  - OTSi
  - Ethernet

Transport API Summary

General Benefits

- Provides functions necessary for multi-domain orchestration
  - Topology view and abstraction
  - Connectivity establishment
  - Hierarchical Abstraction
  - Migration path for legacy systems
- Different types of topology abstraction
  - Abstract Node (single, edge, sliced, etc)
  - Abstract Link
  - Complete (1-to-1) internal topology
- Supports multiple transport technologies
  - Packet Transport (Ethernet, MPLS-TP)
  - Optical Transport (OTN, DWDM)
- T-API localizes interoperability to Orchestrator/Controller interface

Past Interop/PoC Findings

- Controllers abstract network in different ways
  - E.g. Unidirectional v/s Bidirectional links
- Controllers provide different capabilities
  - E.g. Connectivity restrictions
- Division of responsibility between controllers unclear
  - E.g. Multi-domain Path Computation
- Supporting RPC and REST styles is hard
  - Not all implementations supported both
- Additional use cases need to be validated
  - Path Computation Service
  - Generalized Notification Service
- Based on demo feedback, TAPI 2.0 being aligned with YANG Best Practices

HANDS-ON: USING TAPI SDK REFERENCE IMPLEMENTATION (RI)

REST API

- A RESTful application is an application that exposes its state and functionality as a set of resources that the clients can manipulate and conforms to a certain set of principles:
  - All resources are uniquely addressable, usually through URIs; other addressing can also be used, though.
  - All resources can be manipulated through a constrained set of well-known actions, usually CRUD (create, read, update, delete), represented most often through the HTTP's POST, GET, PUT and DELETE;
  - The data for all resources is transferred through any of a constrained number of well-known representations, usually HTML, XML or JSON;
  - The communication between the client and the application is performed over a stateless protocol.

CURL AS AN HTTP REST CLIENT

- curl is a command line tool which is used to transfer data over the internet.
- Example:

>> curl -X GET -H "Content-Type: application/json"
http://127.0.0.1:8080/restconf/config/context/topology/

Launch/Run TAPI Reference Implementation (RI)

- Run in a terminal:

>> cd OFC_TAPI_SC/tapi_ri/server

>> python3 tapi-server.py

- Run in a new terminal:

>> cd OFC_TAPI_SC/tapi_ri/client

>> curl -X GET -H "Content-Type: application/json" http://127.0.0.1:8080/restconf/config/context/

TAPI Context, Topology & Connectivity Overview

- All TAPI interaction between an TAPI provider (SDN Controller) and an TAPI Client (Application, Orchestrator or parent SDN Controller) occur within a shared "*Context*"
- TAPI *Context* is defined by a set of *ServiceInterfacePoints* (and some policy)
  - *ServiceInterfacePoints* enable TAPI Client to request TAPI Services between them.
- A TAPI provider may expose 1 or more abstract *Topology* within shared *Context*
  - These topologies may or may-not map 1-to-1 to a provider's internal topology.
- A *Topology* is expressed in terms of *Nodes* and *Links*.
  - *Nodes* aggregate *NodeEdgePoints, Links* connect 2 Nodes & terminate on *NodeEdgePoints*
  - *NodeEdgePoints* may be mapped to 1 or more *ServiceInterfacePoints* at edge of Network
- TAPI Client requests *ConnectivityService* between 2 or more *ServiceInterfacePoints*
- TAPI Provider creates 1 or more *Connections* in response to *ConnectivityService*
  - *ConnectionEndPoints* encapsulate information related to a *Connection* at the ingress/egress points of every *Node* that the *Connection* traverses in a *Topology*
  - Every *ConnectionEndPoint* is supported by a specific "parent" *NodeEdgePoint*
  - Thus with reference to *ConnectivityServices*, a *ServiceInterfacePoint* conceptually represents a pool of "potential" *ConnectionEndPoints* at the edge of the Network

T-API RI: Retrieve Context

- GET Context Details

curl -X GET -H "Content-Type: application/json" http://127.0.0.1:8080/restconf/config/context/

Response:

```
{   “uuid” : “ctx-ref”,
    “service-interface-point” : [
        {……},
        ……
    ],
    “topology” : [
        {……},
        ……
    ],
    “connectivity-service” : [
        {……},
        ……
    ],
    “connection” : [
        {……},
        ……
    ]
}
```

Proper TAPI implementations should use UUID format. An example below: f81d4fae-7edc-11d0-a765-00a0c91e6bf6

TAPI Context is a Container for all ServiceInterfacePoints, Topologies, ConnectivityServices, Connections, etc data.

T-API RI: Retrieve List of Service Interface Points

- GET List of Service Interface Points

curl -X GET -H "Content-Type: application/json"
http://127.0.0.1:8080/restconf/config/context/service-interface-point/

Response:

```
{
    [
        “/restconf/config/context/service-interface-point/sip-pe1-uni1/”,
        “/restconf/config/context/service-interface-point/sip-pe1-uni2/”,
        “/restconf/config/context/service-interface-point/sip-pe2-uni1/”,
        “/restconf/config/context/service-interface-point/sip-pe2-uni2/”,
        “/restconf/config/context/service-interface-point/sip-pe3-uni1/”,
        “/restconf/config/context/service-interface-point/sip-pe3-uni2/”
    ]
}
```

Can use the returned URI to make additional retrievals

T-API RI: Retrieve Service Interface Point Details

- GET Service Interface Point Details

curl -X GET -H "Content-Type: application/json"
http://127.0.0.1:8080/restconf/config/context/service-interface-point/sip-pe1-uni1/

Response:

```
{   "uuid" : "sip-pe1-uni1",
    "name": [ … ],
    "layer-protocol-name": [ "ETH", "ODU" ],
    "administrative-state": "UNLOCKED",
    "operational-state": "ENABLED",
    "lifecycle-state": "INSTALLED"
    "total-potential-capacity": {
        "total-size": {"value": "10", "unit": "GBPS"},
        "bandwidth-profile": {……}
    }
    "available-capacity": {
        "total-size": {"value": "10", "unit": "GBPS"},
        "bandwidth-profile": {……}
    }
    ……
}
```

Most TAPI objects have layer & state attributes

ServiceInterfacePoint conveys the capabilities of the logical interface point

T-API RI: Retrieve List of Topologies

- GET List of Topologies

curl -X GET -H "Content-Type: application/json"
http://127.0.0.1:8080/restconf/config/context/topology/

Response:

```
{
    [
      “/restconf/config/context/topology/topo-nwk/“,
      “/restconf/config/context/topology/topo-pe1/“,
      “/restconf/config/context/topology/topo-pe2/“,
      “/restconf/config/context/topology/topo-pe3/“
    ]
}
```

Can use the returned URI to make additional retrievals

T-API RI: Retrieve Topology Details

- GET Topology Details

curl -X GET -H "Content-Type: application/json"
http://127.0.0.1:8080/restconf/config/context/topology/topo-nwk/

Response:

```
{   "uuid" : "topo-nwk",
    "name": [
        { "value-name": "name",
          "value": "NETWORK_TOPOLOGY"
        }
        ……
    ],
    "node" : [
        {……},
        ……
    ],
    "link" : [
        {……},
        ……
    ]
}
```

Every TAPI object has a name attribute that is defined as a list of name-value pairs.

Topology contains Nodes & Links (by value).

T-API RI: Retrieve Node Details - I

- GET Node Details

curl -X GET -H "Content-Type: application/json"
http://127.0.0.1:8080/restconf/config/context/topology/topo-nwk/node/node-mul-pe-1/

Response:

```
{   "uuid" : "node-mul-pe-1",
    "name": [ … ],
    "layer-protocol-name": [ "ETH", "ODU" ],
    "administrative-state": "UNLOCKED",
    "operational-state": "ENABLED",
    "lifecycle-state": "INSTALLED"
    "encap-topology": "/restconf/config/context/topology/topo-pe1/",
    "owned-node-edge-point": [ ],
    "aggregated-node-edge-point" : [
        "
/
restconf/config/context/topology/topo-pe1/node/node-eth-pe-1/owned-node-
edge-point/nep-pe1-eth-uni1/",
        ……
    ],
    ……
}
```

Node can be single or multi layer

Abstract Node is an abstraction of a Topology

Node can also constrain forwarding across its aggregated NodeEdgePoints (not shown here)

Node represents the potential to forward data between its aggregated NodeEdgePoints

T-API RI: Retrieve Node Details - 2

- GET Node Details

curl -X GET -H "Content-Type: application/json"
http://127.0.0.1:8080/restconf/config/context/topology/topo-pe1/node/node-eth-pe-1/

Response:

```
{   "uuid" : "node-eth-pe-1",
    "name": [ … ],
    "layer-protocol-name": ["ETH" ],
    "administrative-state": "UNLOCKED",
    "operational-state": "ENABLED",
    "lifecycle-state": "INSTALLED"
    "encap-topology": "",
    "owned-node-edge-point": [
       {……},
       ……
    ],
    "aggregated-node-edge-point" : [
        "
/
restconf/config/context/topology/topo-pe1/node/node-eth-pe-1/owned-node-
edge-point/nep-pe1-eth-uni1/",
        ……
]
}
```

Switch Node is typically single layer

Switch Node contains/owns a list of NodeEdgePoints

T-API RI: Retrieve NodeEdgePoint Details

- NodeEdgePoint

curl -X GET -H "Content-Type: application/json"
http://127.0.0.1:8080/restconf/config/context/topology/topo-pe1/node/node-eth-pe-1/owned-node-edge-point/nep-eth-pe1-uni1/

```
{   "uuid" : "nep-pe1-eth-uni1",
    "name": [ … ],
    "layer-protocol-name": "ETH",
    "administrative-state": "UNLOCKED",
    "operational-state": "ENABLED",
    "lifecycle-state": "INSTALLED"
    "termination-state": "LP_CAN_NEVER_TERMINATE",
    "termination-direction": "BIDIRECTIONAL",
    "link-port-direction": "BIDIRECTIONAL",
    "link-port-role": "SYMMETRIC",
    "mapped-service-interface-point" : [
        "/restconf/config/context/service-interface-point/sip-pe1-uni1/",
        ……
    ]
}
```

NodeEdgePoint is single layer

NodeEdgePoint can be mapped to (1 or more) ServiceInterfacePoint to function as a network interface. This attribute is empty for "internal" NodeEdgePoints

T-API RI: Retrieve Link Details - I

- GET Link Details

curl -X GET -H "Content-Type: application/json"
http://127.0.0.1:8080/restconf/config/context/topology/topo-nwk/link/link-pe1-odu4-nni1-pi4-odu4-nni2/

```
{   “uuid” : “link-pe1-odu4-nn1-pi4-odu4-nni1”,
    “name”: [ … ],
    “layer-protocol-name”: [“ODU” ],
    “direction”: “BIDIRECTIONAL”,
    “resilience-type”: {……},
    “total-potential-capacity”: {……},
    “available-capacity”: {……},
    “cost-characteristic”: {……},
    “latency-characteristic”: {……},
    ……
    ……
    “node-edge-point” : [
        “
/
restconf/config/context/topology/topo-pe1/node/node-odu-pe-1/owned-node-
edge-point/nep-pe1-odu4-nni1/”,
        “
/
restconf/config/context/topology/topo-nwk/node/node-odu-pi-4/owned-node-
edge-point/nep-pi4-odu4-nni2/”
```

Link conveys “transfer-characteristic” information

Link represents adjacency information between 2 NodeEdgePoints

T-API RI: Retrieve Link Details - 2

- GET Link Details

curl -X GET -H "Content-Type: application/json"
http://127.0.0.1:8080/restconf/config/context/topology/topo-pe1/link/link-pe1-eth-pool-pe1-odu2-pool/

```
{   “uuid” : “link-pe1-eth-pool-pe1-odu2-pool”,
    “name”: [ … ],
    “layer-protocol-name”: [“ETH”, “ODU”],
    “direction”: “BIDIRECTIONAL”,
    “resilience-type”: {……},
    “total-potential-capacity”: {……},
    “available-capacity”: {……},
    “cost-characteristic”: {……},
    “latency-characteristic”: {……},
    ……
    ……
    “node-edge-point” : [
        “
/restconf/config/context/topology/topo-pe1/node/node-eth-pe-1/owned-node-edge-point/nep-pe1-eth-pool/”,
        “
/restconf/config/context/topology/topo-pe1/node/node-odu-pe-1/owned-node-edge-point/nep-pe1-odu2-pool/”
```

“Transitional” Link connects NodeEdgePoints from different layers and conveys the layer-transition information

Hands-on: Writing a TAPI Topology client

- Objective:
  - Retrieve and draw Network Topology using T-API
- Steps:
  - Run TAPI-RI
  - Load topological information
  - Start coding using the following libraries:
    - NetworkX
    - Requests
    - Json

Solution: T-API App

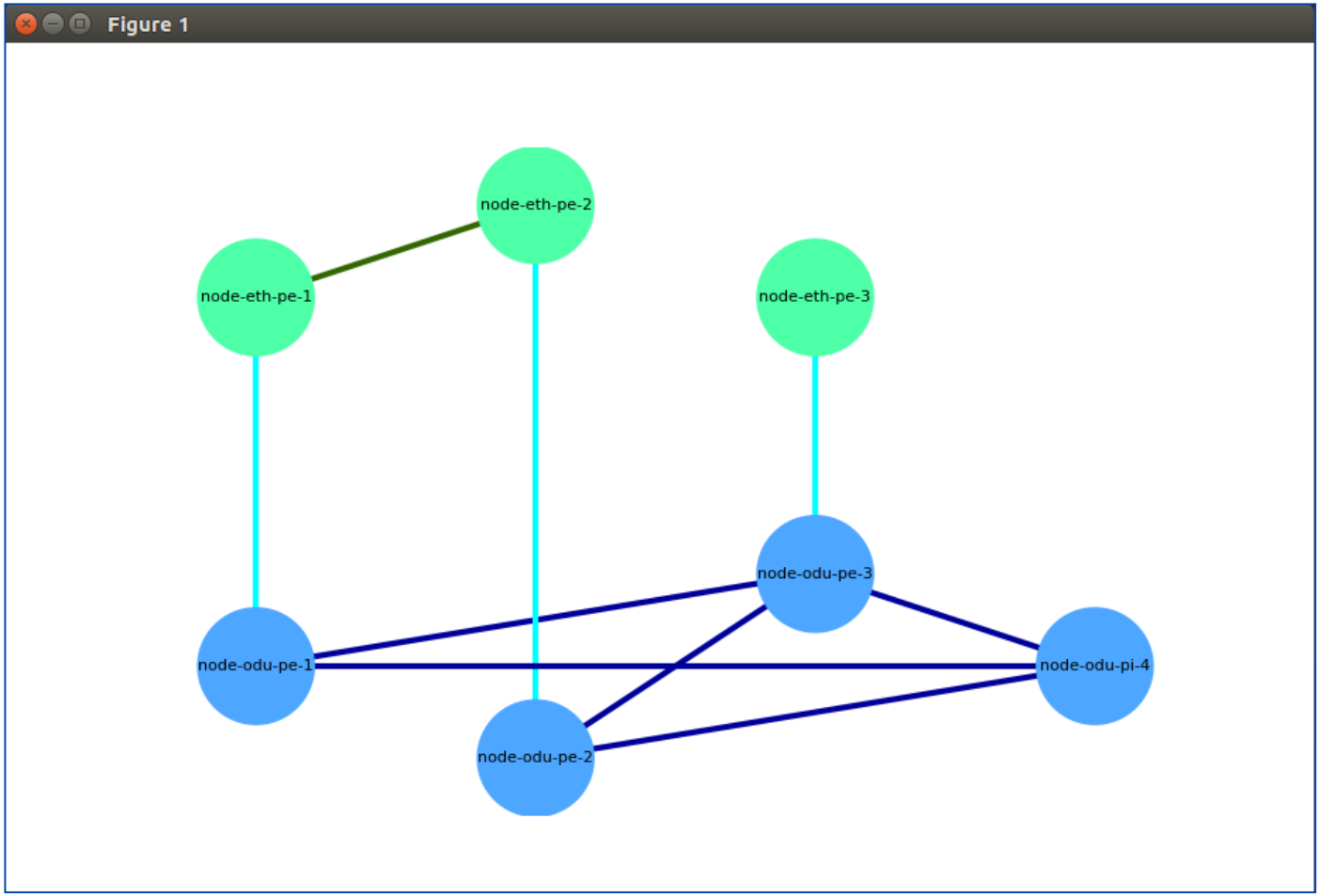
```python
#!/usr/bin/python
# -*- coding: utf-8 -*-


import requests
from requests.auth import HTTPBasicAuth
import json
import matplotlib.pyplot as plt
import networkx as nx

IP='127.0.0.1'
PORT='8080'

def retrieveTopology(ip, port, user='', password=''):
    http_json = 'http://' + ip + ':' + port + '/restconf/config/context/topology/top0'
    response = requests.get(http_json, auth=HTTPBasicAuth(user, password))
    topology = response.json()
    return topology

def load_topology ( topology) :
    G=nx.Graph()
    for link in topology['link']:
      node_src = link['node-edge-point'][0].split('restconf/config/context/topology/top0/node/')[1].split('/')[0]
      node_dst = link['node-edge-point'][1].split('restconf/config/context/topology/top0/node/')[1].split('/')[0]
      G.add_edge( node_src, node_dst )
      print 'Link: ' + node_src + ' ' + node_dst
    nx.draw(G)
    plt.show()

if __name__ == "__main__":
    print "Reading network-topology"
    topo = retrieveTopology(IP, PORT)
    print json.dumps(topo, indent=4, sort_keys=True)
    load_topology(topo)

```

Run TAPI Application Client

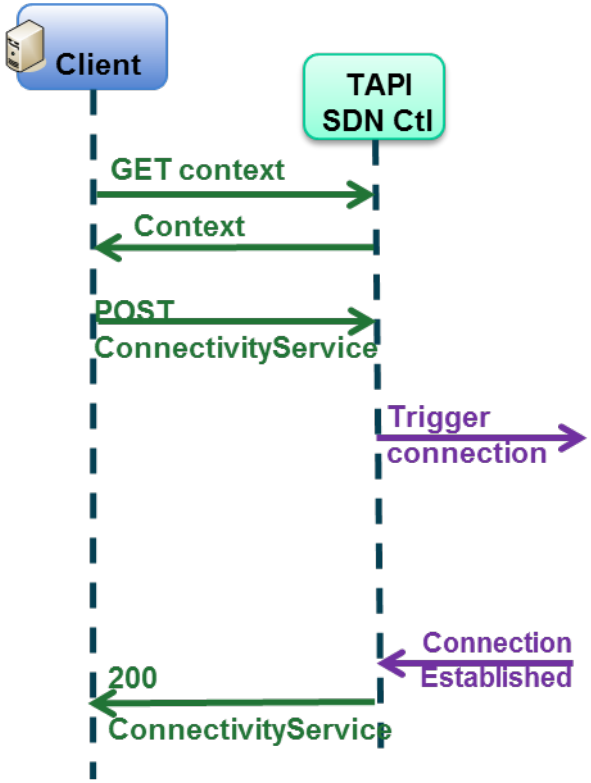
- Run in a terminal:

>> cd OFC_TAPI_SC/tapi_app/

>> python3 tapi-app.py



T-API RI: Connectivity Service workflow



T-API RI: Establish Connectivity Service

- curl -X POST -H "Content-Type: application/json" http://127.0.0.1:8080/restconf/config/context/connectivity-service/cs1/ -d @cs1.json
- cs1.json:

```
{ "uuid" : "conn-service-1",
  "service-type":"POINT_TO_POINT_CONNECTIVITY",
  "requested-capacity": {"total-size": { "value": "1", "unit": "GBPS" }},
  "end-point":[
    { "local-id": "csep-1",
      "layer-protocol-name": "ETH",
      "direction":"BIDIRECTIONAL",
      "role":"SYMMETRIC",
      "service-interface-point":
         "/restconf/config/context/service-interface-point/sip-pe1-uni1"},
    { "local-id": "csep-2",
     "layer-protocol-name": "ETH",
     "direction":"BIDIRECTIONAL",
     "role":"SYMMETRIC",
     "service-interface-point":
        "/restconf/config/context/service-interface-point/sip-pe2-uni1"}
  ]
}
```

ConnectivityService endpoint information has to specify the ServiceInterfacePoint

Created Connection

- GET Connection Details:
- curl -X GET -H "Content-Type: application/json" http://127.0.0.1:8080/restconf/config/context/connection/cs1/

```
{
    "uuid" : "cs1",
    "connection-end-point": [
       "/restconf/config/topology/top0/node/node1/owned-node-edge-point/
nep11/cep-list/cep11",
        "/restconf/config/topology/top0/node/node1/owned-node-edge-point/
nep12/cep-list/cep11“
  ]
}
```

ConnectivityService has triggered the establishment of a Connection

Node Edge Point is augmented with a list of Connection End Points

HANDS-ON : USING TAPI SDK RI TOOLING

Objective

- Generate an Python Server & Client

- Use as base TAPI Open-API Spec
- Create stub code using Swagger code generator
- Hands-on: Write application logic and execute the application

Swagger and OpenAPI specs

- Q: How can we define an standardized REST API?

- Open API (formerly known as Swagger) is a popular compact and easy to parse data schema format to describe REST APIs
  - Open API Schemas can be described in two popular web encoding languages – YAML or JSON
- The generated T-API Open API specifications provide a mapping from the Yang data schema into Open API JSON format, which can then be used to generate Python and/or Java code for implementation of the API in RestConf
- https://www.openapis.org/
- https://swagger.io

Key Features of TAPI SDK

- Technology-agnostic API Framework
  - Standardizes a single core technology-agnostic specification that abstracts common transport network functions
- Modular & Extensible
  - Functional features are packaged into small self-contained largely-independent modules
  - TAPI Core Spec is designed to be fully extensible
    - *Extensions* can be Technology, SDO, Operator or Vendor specific
- Industry-wide Interoperability Objective – developed within
  - Open Source SDN SNOWMASS project under Apache 2 license
- SDK components generated using tools for agile prototyping
  - YANG schema generated from UML using guidelines developed in an multi-SDO initiative (IISOMI)
  - SwaggerOpen APIs generated from YANG following RESTConf specification

TAPI SDK: Organization and Modularity

- ONF Transport API Functional Requirements – TR-527, June 2016
  - ONF Open Transport WG Project
  - Input to the TAPI SDK
- Software-wise, TAPI SDK is packaged as 4 eclipse sub-projects
  - Papyrus-UML Information Model
    - A pruned/refactored version of ONF Core IM
    - Is a technology-agnostic generic framework + technology specific extensions (OTN, ETH)
  - YANG Data Schema
    - auto-generated from UML using OSSDN Eagle Tools
  - Open API (Swagger) RESTConf API
    - auto-generated from YANG using OSSDN Eagle tools
  - Reference Implementation (RI) in Python
- Feature-wise, TAPI SDK is organized as a module-set of {UML-Model, YANG-Module, Open-API} per TAPI functional interface. These include
  - Common, Topology, Connectivity, OAM, Path Computation, Virtual Network, Notification
  - ODU, OTSi, ETH technology specification models

Swagger Code generator

>> cd OFC_TAPI_SC/tapi_ri

>> wget http://central.maven.org/maven2/io/swagger/swagger-codegen-cli/2.2.1/swagger-codegen-cli-2.2.1.jar -O swagger-codegen-cli.jar

>> java -jar swagger-codegen-cli.jar generate -i tapi-connectivity.swagger -l python-flask -o server/

- Inspect server
  - App.py

    >> app.app.config['JSON_SORT_KEYS']=False

Write backend I

- Create a database object, where we can store and access a context json object

  database.context={}

- Modify default controller behaviour

  def create_context_by_id(context)

  def retrieve_context()

- Use curl as client:
  - curl -X POST -H "Content-Type: application/json" http://127.0.0.1:8080/restconf/config/context/ -d @context.json
  - curl -X GET -H "Content-Type: application/json" http://127.0.0.1:8080/restconf/config/context/

Write backend II

- Write Topology basic retrievals:
  - retrieve_context_topology_topology
  - retrieve_context_topology_topology_by_id
  - retrieve_context_topology_node_node
  - retrieve_context_topology_node_node_by_id
  - retrieve_context_topology_link_link
  - retrieve_context_topology_link_link_by_id

Write backend II

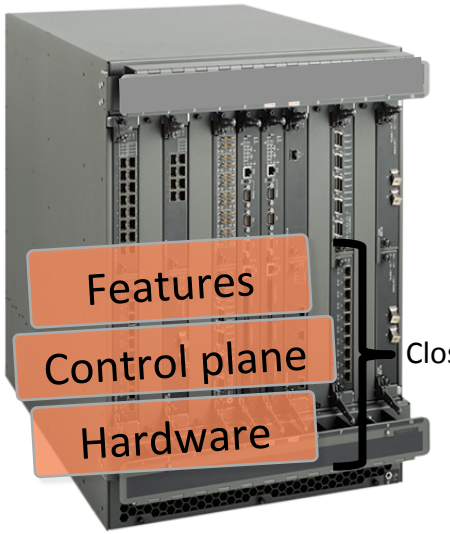
- Write Connection basic retrievals:
  - retrieve_context_connection_connection
  - retrieve_context_connection_connection_by_id
  - retrieve_context_connectivity_service_connectivity_service
  - retrieve_context_connectivity_service_connectivity_service_by_id
  - retrieve_context_service_interface_point_service_interface_point
  - retrieve_context_service_interface_point_service_interface_point_by_id

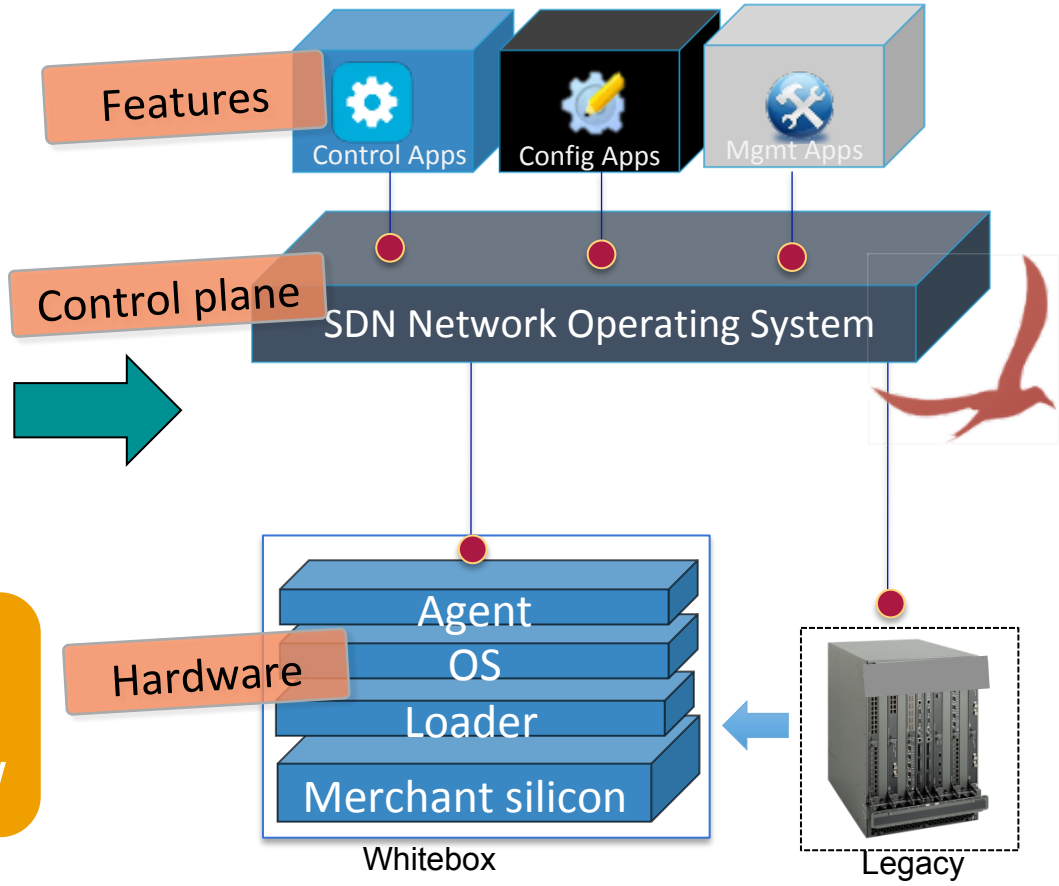
Write backend III

- Write Connectivity service create and delete:
  - create_context_connectivity_service_connectivity_service_by_id
    - Use example connectivity-service cs1.json
    - Connectivity Service triggers connection creation
  - delete_context_connectivity_service_connectivity_service_by_id
    - Remove connection
    - Remove connectivity service

HANDS-ON: USING ONOS WITH TAPI

SDN in a nutshell





Disaggregating features which does not necessarily be coupled with H/W

ONOS architecture

Applications

Bandwidth on-demand, calendaring, optical restoration
Power balancing, fault management & correlation

Northbound Abstractions

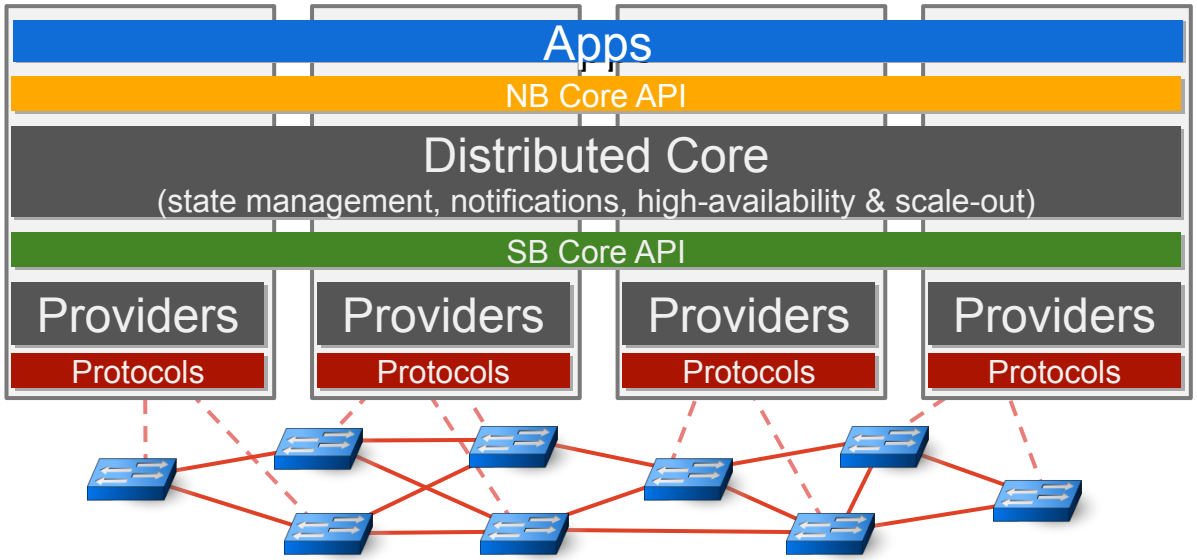
Intent framework
Converged topology graph

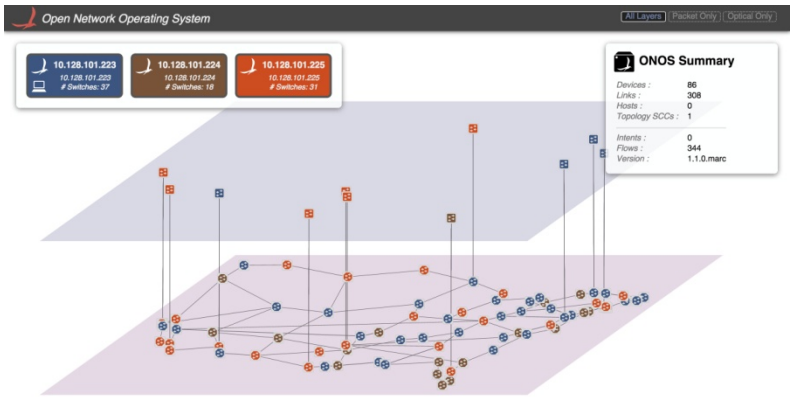
ONOS Core: Scale & HA

Modular PCE
Optical information model
Resource manager

Southbound Drivers

OpenFlow, NETCONF,
TL1, PCEP, SNMP, REST
P4Runtime





ONOS NBI

- Prepare ONOS:

>> wget http://repo1.maven.org/maven2/org/onosproject/onos-releases/onos-1.12.0/onos-1.12.0.tar.gz

>> tar –xvf onos-1.12.0.tar.gz

- Run ONOS:

>> cd onos-1.12.0/apache-karaf-3.0.8/bin

>> ./karaf clean

$$ app activate org.onosproject.openflow ←Command to run in ONOS CLI

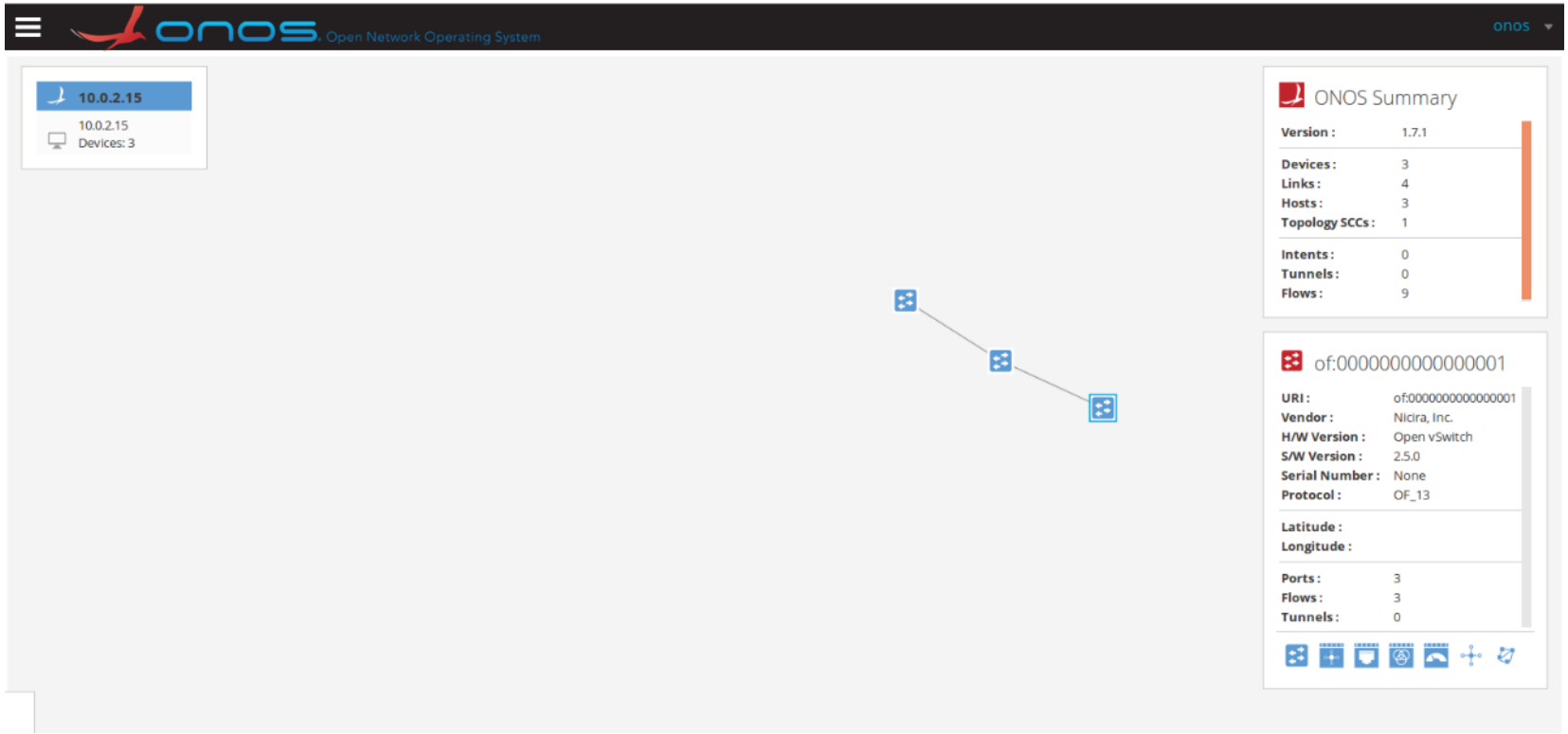
- Open Firefox:

http://127.0.0.1:8181/onos/ui/index.html

When asked for user/password use onos/rocks

RUN mininet

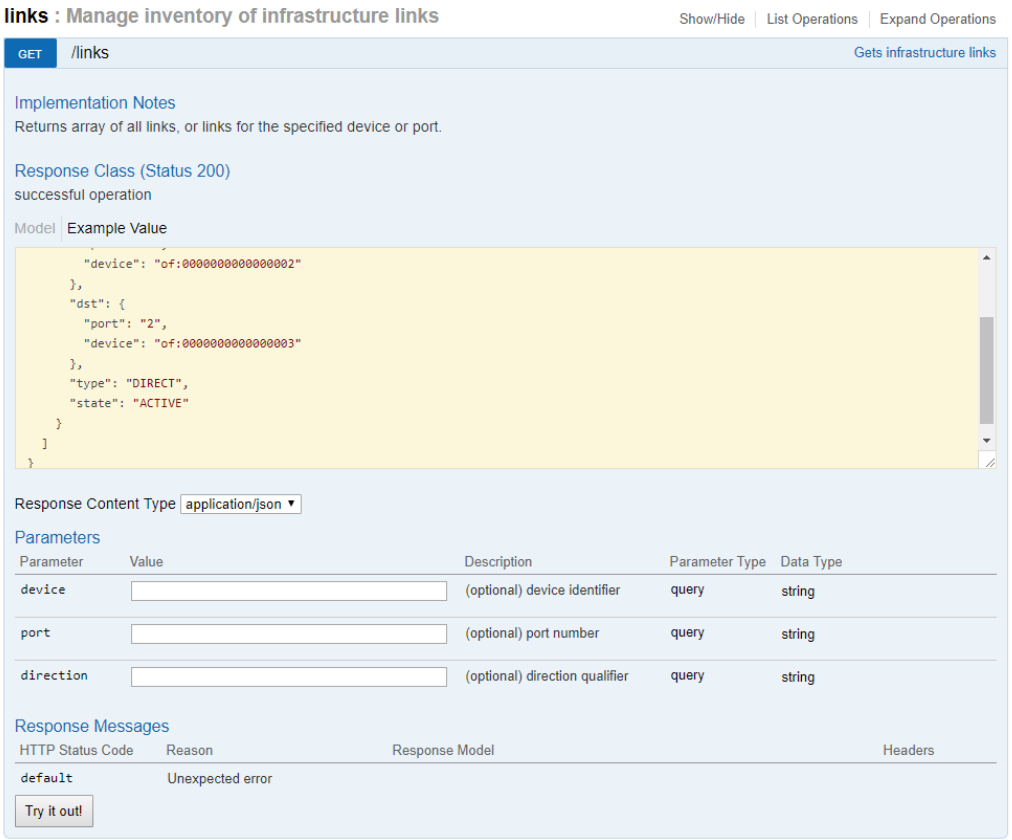
- sudo mn --topo linear,3 --mac --controller=remote,ip=127.0.0.1,port=6653 --switch ovs,protocols=OpenFlow13



ONOS LINKS REST API

- http://localhost:8181/onos/v1/docs/
- curl -X GET -u onos:rocks --header 'Accept: application/json' 'http://localhost:8181/onos/v1/links' | python –m json.tool


```
{ "links": [
{ "src": { "port": "3", "device": "of:0000000000000002" }, "dst": { "port": "2", "device": "of:0000000000000003" }, "type": "DIRECT", "state": "ACTIVE" },
{ "src": { "port": "2", "device": "of:0000000000000002" }, "dst": { "port": "2", "device": "of:0000000000000001" }, "type": "DIRECT", "state": "ACTIVE" },
{ "src": { "port": "2", "device": "of:0000000000000003" }, "dst": { "port": "3", "device": "of:0000000000000002" }, "type": "DIRECT", "state": "ACTIVE" },
{ "src": { "port": "2", "device": "of:0000000000000001" }, "dst": { "port": "2", "device": "of:0000000000000002" }, "type": "DIRECT", "state": "ACTIVE" }
] }
```

Example using ONOS TOPOLOGY REST API in Python

- OFC_TAPI_SC/onos_api/onos_topology.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import requests
from requests.auth import HTTPBasicAuth
import json

IP='127.0.0.1'
PORT='8181'
USER='onos'
PASSWORD='rocks'

def retrieveTopology(ip, port, user, password):
    http_json = 'http://' + ip + ':' + port + '/onos/v1/links'
    response = requests.get(http_json, auth=HTTPBasicAuth(user, password))
    topology = response.json()
    return topology

if __name__ == "__main__":

    print "Reading network-topology"
    topo = retrieveTopology(IP, PORT, USER, PASSWORD)
    print json.dumps(topo, indent=4, sort_keys=True)

```

Calling ONOS FLOW REST API with curl

- http://localhost:8181/onos/v1/docs/

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \
  "flows": [ \
   { \
    "priority": 40000, \
    "timeout": 0, \
    "isPermanent": true, \
    "deviceId": "of:0000000000000001", \
    "treatment": { \
     "instructions": [ \
      { \
       "type": "OUTPUT", \
       "port": "CONTROLLER" \
      } \
     ] \
    }, \
    "selector": { \
     "criteria": [ \
      { \
       "type": "ETH_TYPE", \
       "ethType": "0x88cc" \
      } \
     ] \
    } \
   } \
  ] \
}' 'http://10.1.7.17:8181/onos/v1/flows?appId=tapi0'
```

1. when device of:000…1

3. output the packet to controller

2. encounter a packet with EthType 0x88cc (=LLDP)

Example using ONOS FLOW REST API in Python

OFC_TAPI_SC/onos_api/onos_flows.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import requests
from requests.auth import HTTPBasicAuth
import json

IP='localhost'
PORT='8181'
USER='onos'
PASSWORD='rocks'

URL = 'http://' + IP + ':' + PORT + '/onos/v1/flows/'

def insertFlow( nodeId, priority, inport, outport ):

    flow='{ "priority": '+priority+', "timeout": 0, "isPermanent": true, "deviceId": "'+nodeId+
          '", "treatment": { "instructions": [ { "type": "OUTPUT", "port": "'+outport+
          '" } ] }, "selector": { "criteria": [ { "type": "IN_PORT", "port": "'+inport+'" } ] } }'


    print "Flow: " + flow
    url = URL + nodeId + '?appId=tuto'
    headers = {'content-type': 'application/json'}
    print url
    response = requests.post(url, data=flow,
                        headers=headers, auth=HTTPBasicAuth(USER,
                        PASSWORD))
    print response
    return { 'status':response.status_code, 'content': response.content}

def deleteFlow(nodeId, flow_id):

    url = URL + '' + nodeId + '/' + flow_id
    response = requests.delete(url, auth=HTTPBasicAuth(USER, PASSWORD))
    return {'flow_id':flow_id, 'status':response.status_code, 'content': response.content}


if __name__ == "__main__":

    print "Setting flow"

    res = insertFlow(nodeId="of:0000000000000001", priority="40001", inport="1", outport="2")
    print json.dumps(res, indent=4, sort_keys=True)
```

Integration of ONOS with TAPI_RI

- Load topology from ONOS
  - Use:

    ONOS REST DEVICE AND PORTS

    ONOS REST LINKS

- **Optional**: Create necessary flows from requested connectivity service
  - Use:

    ONOS REST FLOWS

Example: Parse ONOS topology

OFC_TAPI_SC/tapi_ri_onos/server/orchestrator/network_manager.py

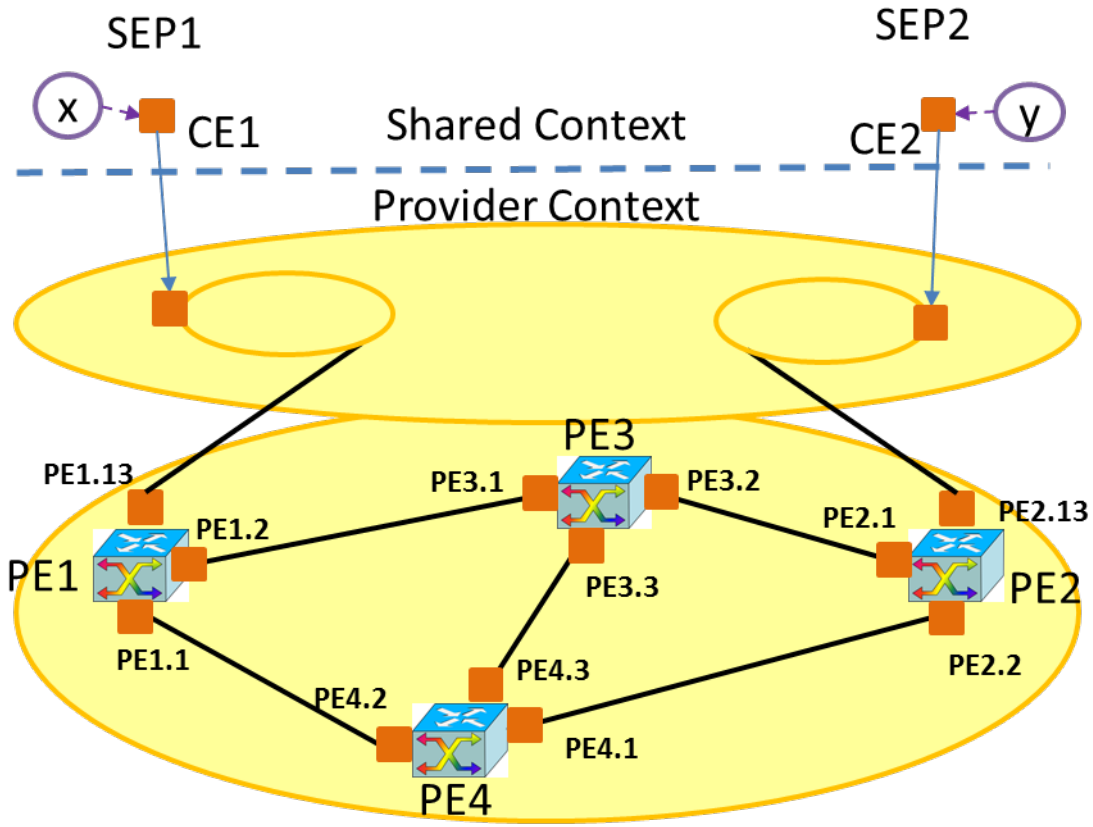
```
    def load_onos_topology(self, ctl_address, ctl_port, ctl_user, ctl_password):
        logging.info("load_onos_topology %s %s %s %s", ctl_address, ctl_port, ctl_user, ctl_password)
        node_array = []
        link_array = []
        #nodes
        http_json = 'http://' + ctl_address + ':' + ctl_port + '/onos/v1/devices'
        response = requests.get(http_json, auth=HTTPBasicAuth(ctl_user, ctl_password))
        onos_nodes = response.json()
        logging.info("Retrieved onos nodes:\n%s", json.dumps(onos_nodes, indent=4, sort_keys=True) )
        for node in onos_nodes["devices"]:

            node_edge_point = []
            http_json = 'http://' + ctl_address + ':' + ctl_port + '/onos/v1/devices/' + node['id'] + '/ports'
            response = requests.get(http_json, auth=HTTPBasicAuth(ctl_user, ctl_password))
            node_ports = response.json()
            logging.info("Retrieved node %s node_ports:\n%s", node['id'], json.dumps(node_ports, indent=4, sort_keys=True) )

            for port in node_ports["ports"]:
                if port["isEnabled"] == True :

                    nep = collections.OrderedDict()
                    nep['uuid'] = "nep" + self.getNodeId(node['id']) + port["port"]
                    node_edge_point.append( nep )

            node_json = collections.OrderedDict()
            node_json['uuid'] = "node" + self.getNodeId(node['id'])
            node_json['owned-node-edge-point'] = node_edge_point

            node_array.append( node_json )

        #links
        http_json = 'http://' + ctl_address + ':' + ctl_port + '/onos/v1/links'
        response = requests.get(http_json, auth=HTTPBasicAuth(ctl_user, ctl_password))
        onos_links = response.json()
        logging.info("Retrieved onos links:\n%s", json.dumps(onos_links, indent=4, sort_keys=True) )
        for link in onos_links['links']:
            link_json = {}
            link_id= "link" + self.getNodeId(link['src']['device'])+ self.getNodeId(link['dst']['device'])
            link_json['uuid'] = link_id
            link_json['node-edge-point'] = []
            nep_src = "restconf/config/context/topology/top0/node/node" + self.getNodeId(link['src']['device']) + "/owned-node-edge-point/nep" + self.getNodeId(link['src']['device']) + link['src']['port'] + "/"
            nep_dst = "restconf/config/context/topology/top0/node/node" + self.getNodeId(link['dst']['device']) + "/owned-node-edge-point/nep" + self.getNodeId(link['dst']['device']) + link['dst']['port'] + "/"
            link_json['node-edge-point'].append(nep_src)
            link_json['node-edge-point'].append(nep_dst)
            link_array.append( link_json )


        #topology
        topo = collections.OrderedDict()
        topo['uuid'] = "top0"
        topo['name'] = [{"value-name":"topo0","value":"0"}]
        topo['node'] = node_array
        topo['link'] = link_array
        logging.debug("Topo: %s", topo)
        return topo
```

Solution

- Example files in tapi_ri_onos
- Basic:
  - orchestrator/network_manager.py
    - Loads topology and translates to TAPI (with assumptions)
    - Receives Connectivity Service and translates to flows (with assumptions)

CREATE AN EXAMPLE MININET SCENARIO

T-API RI: Reference Network



Described in:
https://github.com/OpenNetworkingFoundation/Snowmass-ONFOpenTransport/blob/develop/TAPI_RI/server_backend_state.json

Create a custom mininet topology

```
#!/usr/bin/python
###################################
#  README
#
#  THIS FILE NEEDS TO BE COPIED IN mininet/custom FOLDER
#
###################################
import sys

from mininet.net import Mininet
from mininet.node import Controller, RemoteController
from mininet.log import setLogLevel, info, warn
from mininet.cli import CLI


setLogLevel( 'info' )
net = Mininet()

info( '*** Adding controller\n' )
c0 = RemoteController( 'c0', ip='0.0.0.0', port=6633)
net.addController( c0 )

#info( '*** Adding hosts\n' )
#h1 = net.addHost( 'h1', ip='10.0.0.1' )

# creating switches
info( '*** Adding switch\n' )
s1 = net.addSwitch( 's1' )
s2 = net.addSwitch( 's2' )
s3 = net.addSwitch( 's3' )
s4 = net.addSwitch( 's4' )

# creating links
info( '*** Creating links\n' )
net.addLink( s1, s3 )
net.addLink( s3, s1 )
net.addLink( s1, s4 )
net.addLink( s4, s1 )
net.addLink( s2, s3 )
net.addLink( s3, s2 )
net.addLink( s2, s4 )
net.addLink( s4, s2 )
```

CONCLUSION

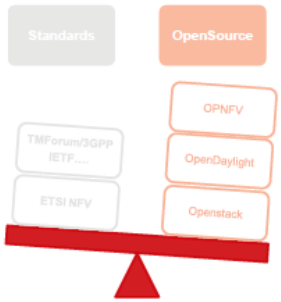
We are ready for writing Transport APIs

- TAPI Overview and Architecture
  - Relevance and position of TAPI in SDN Architecture
- Hands-on : Using TAPI SDK Reference Implementation
  - Hands-on: TAPI Concepts – REST API, Context, Topology, Connectivity, etc
  - Hands-on: Writing and running a simple TAPI Topology & Connectivity client
  - Hands-on: Tooling: Generate an Python Server & Client from TAPI Open-API Spec
  - Hands-on: Write application logic and execute the application
- Hands-On : Using ONOS with TAPI
  - Introduction to ONOS northbound REST API, Mininet config
  - Hands-on: Write ONOS client (topology & flows)
  - Hands-on: Write TAPI RI interacting with ONOS
  - Hands-on: Mininet configuration Scenario

Standards vs Open Source

Standards Outcome/Benefits

- Reference architectures
- Functional requirements
- Interface requirements
- Information models
- Data Models
- Protocols



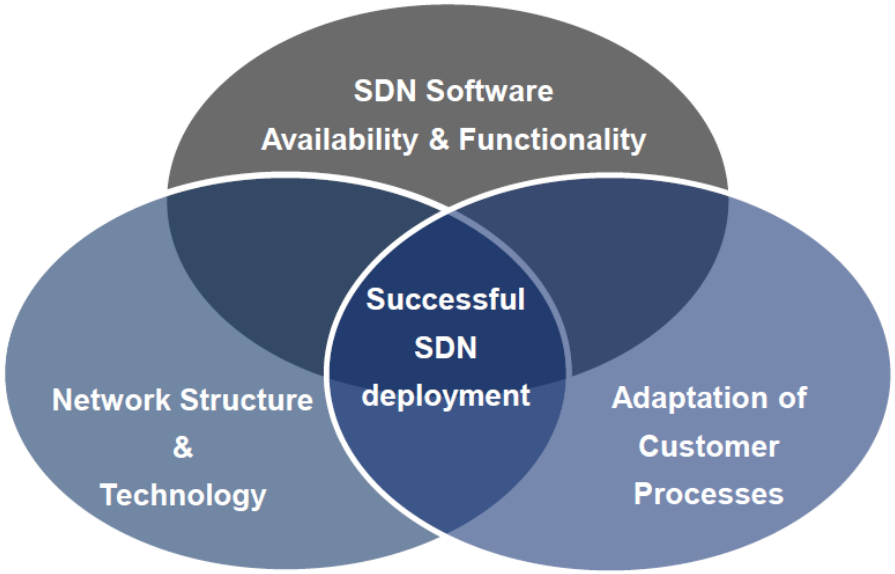
Open Source Outcome/Benefit

- Source code
- Reference implementation
- De-Facto API's
- Inter-operability testing
- Explore new features
- It's all about the community
- Upstream vs Integration

PoC Outcome/Benefits

- Technology Exploration
- Feasibility Check
- Create knowhow
- PR

Transport SDN Benefits and Challenges



- **Benefit**: Completely automated, programmable, integrated and flexible network – leveraging the installed base in an optimized manner.
- **Technical Challenges**:
  - agree on standardized architectures and abstraction/ virtualization models
  - performance of centralized systems & OF
- **Commercialization Challenges**:
  - Open Source business models
  - New business models leveraging SDN
- **Organizational Challenges**:
  - Adapt deep rooted processes across traditional silos & boundaries to leverage SDN flexibility
- **Deployment Challenges**:
  - Carrier grade SDN systems for field deployments
  - Maturity of SDN network technologies for green field deployments as well as integration of legacy networks

References

- ONF SDN Architecture 1.1 - https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-521_SDN_Architecture_issue_1.1.pdf
- Transport API (TAPI) 2.0 Overview, August 13, 2017 https://www.opennetworking.org/wp-content/uploads/2017/08/TAPI-2-WP_DRAFT.pdf
- TAPI SDK 2.0 (SNOWMASS) - https://github.com/OpenNetworkingFoundation/Snowmass-ONFOpenTransport
- UML Tools (EAGLE) - https://github.com/OpenNetworkingFoundation/EAGLE-Open-Model-Profile-and-Tools
- TAPI 1.0 SDK Overview ONF MWD, Sept 7, 2016 https://github.com/OpenNetworkingFoundation/Snowmass-ONFOpenTransport/raw/develop/DOCS/presentations/onf2016.307_TAPI_SDK.01.pptx
- TAPI Functional Requirements 1.0 - https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-527_TAPI_Functional_Requirements.pdf

ricard.vilalta@cttc.es
karthik.sethuraman@necam.com
y-higuchi@opennetworking.org

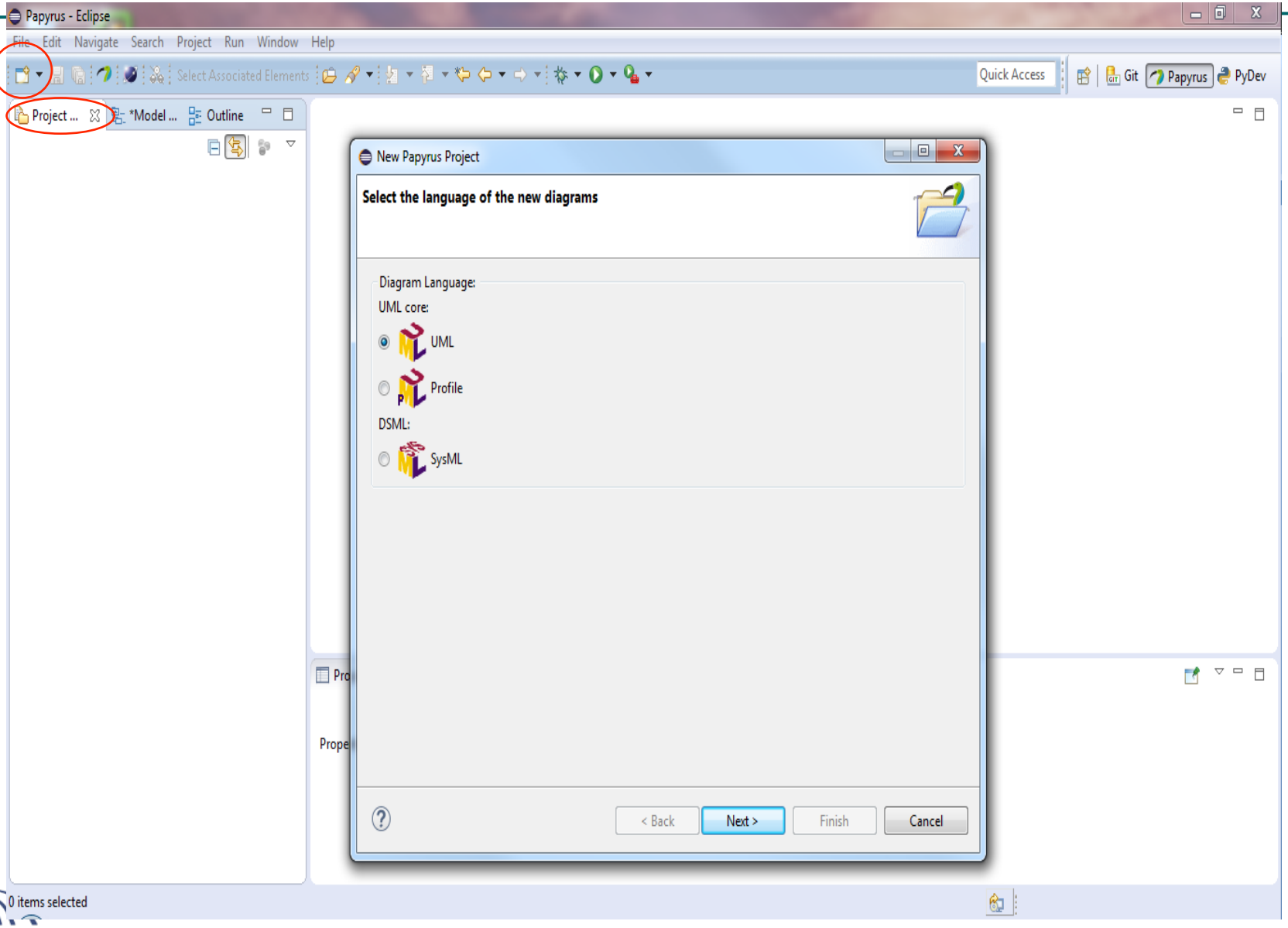
CTTC NEC \Orchestrating a brighter world

EXTRAS:HANDS-ON DEVELOPING TAPI EXTENSIONS: FROM UML TO CODE EXAMPLE

TAPI 2.0 Extensions Approach Overview

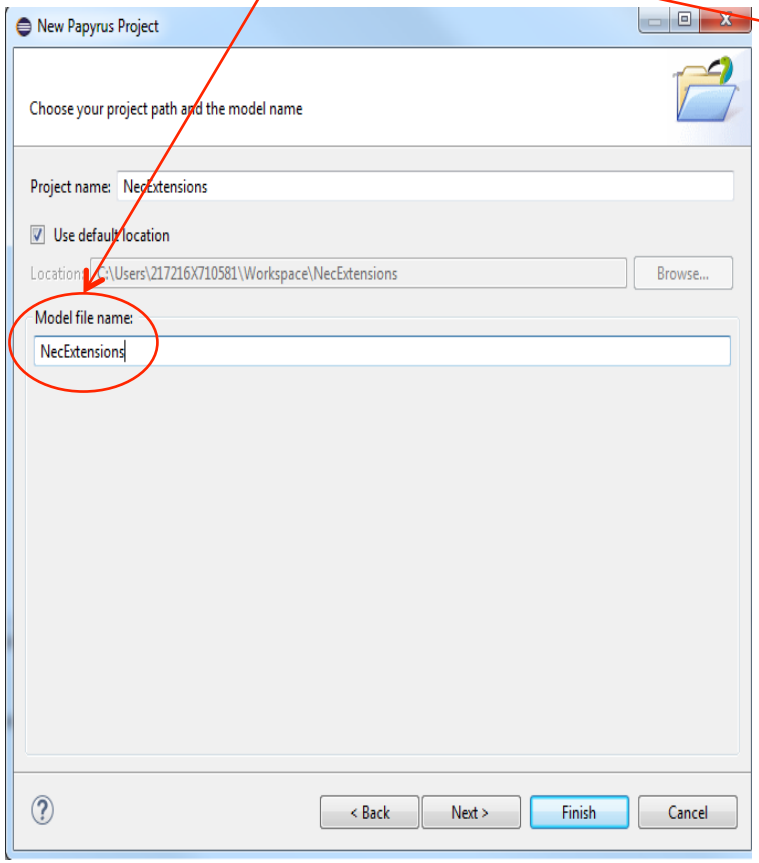
- TAPI follows an "model-driven" approach to generating interfaces
- Entire TAPI model is specified in UML using Eclipse Papyrus IDE
- The YANG data-schema and RESTConf Open API (Swagger) are then generated using scripting-tools rather than by hand
  - These require installation of Node.js and Pyang/Python and corresponding generation-tool plugin-software from the ONF Eagle project
- It is highly recommended that the same approach be employed to generate extensions to TAPI
  - While it is possible to manually extend the generated TAPI Yang or RESTful Open API files directly, the future maintenance and alignment could be problem as TAPI evolves and is updated
- Finally TAPI 2.0 allows any TAPI class to be augmented/extended (unlike TAPI 1.x which had defined specific extension points)
  - This includes even enumeration values and data-types

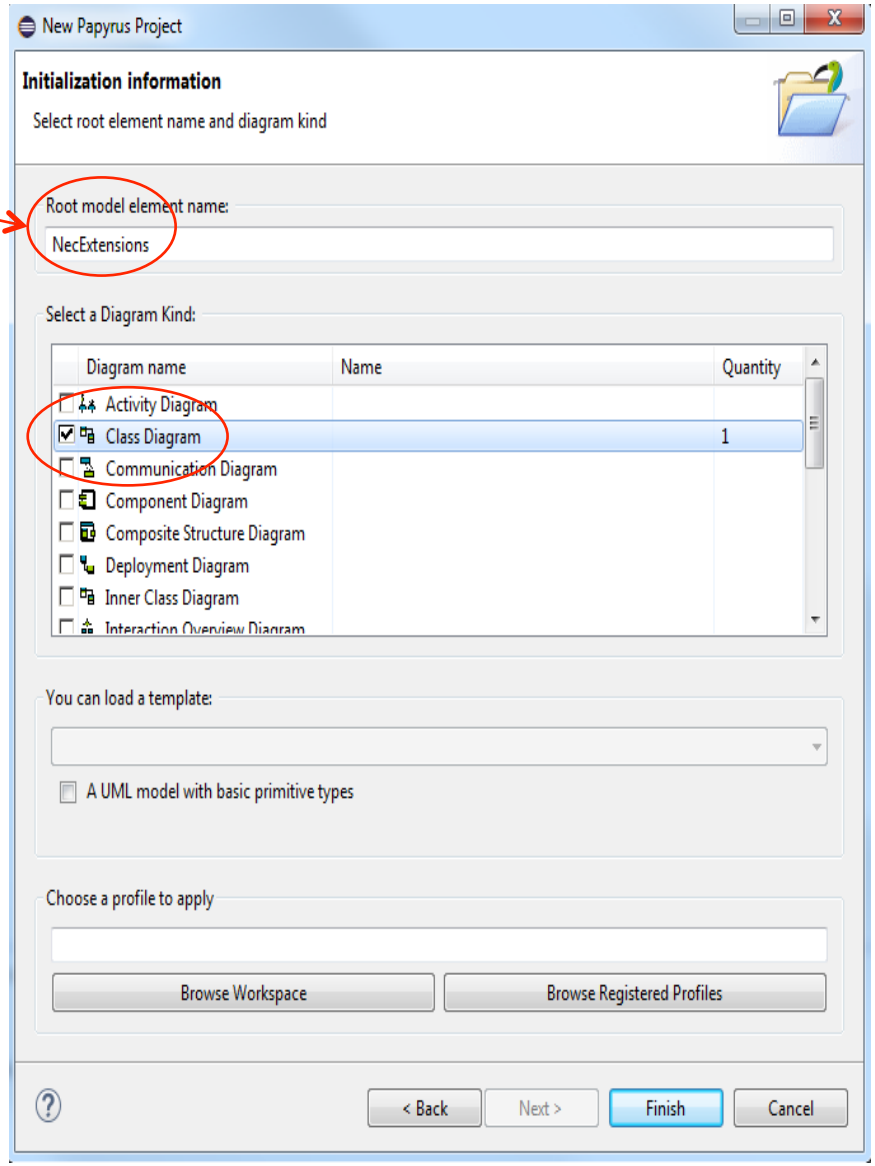
1: Create a Papyrus Uml Project



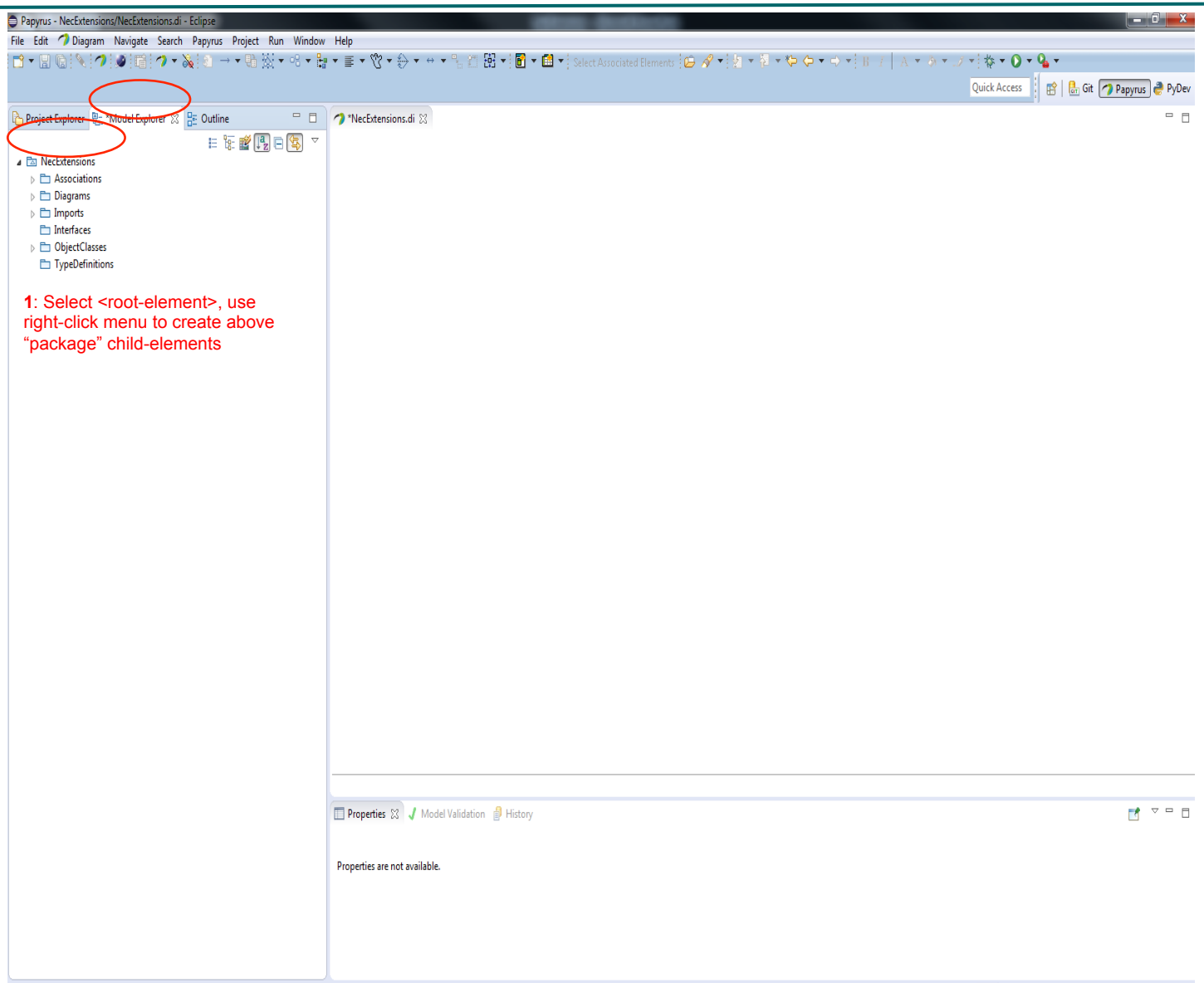
1: Create a Papyrus Uml Project

Important to choose proper “Model” name

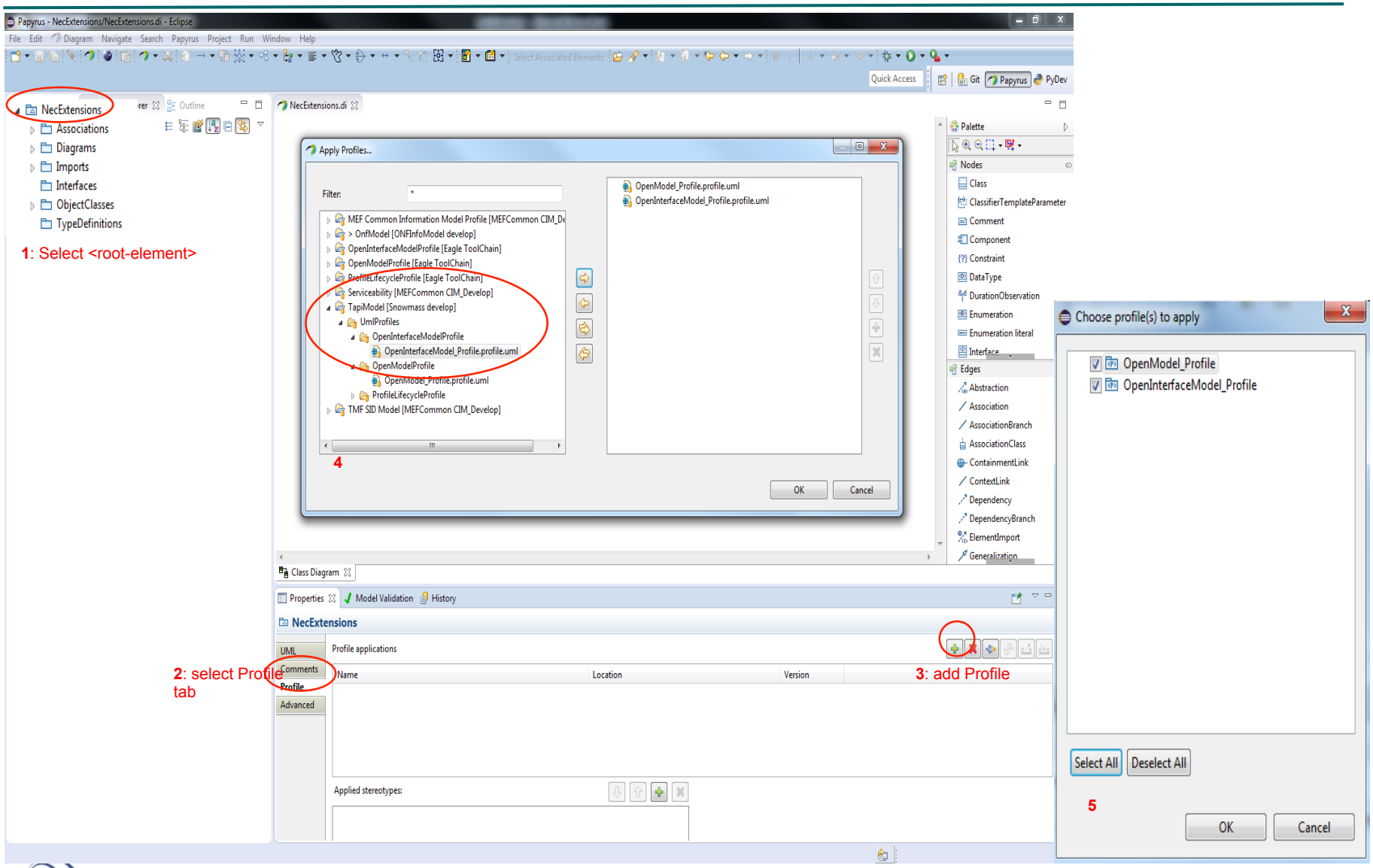




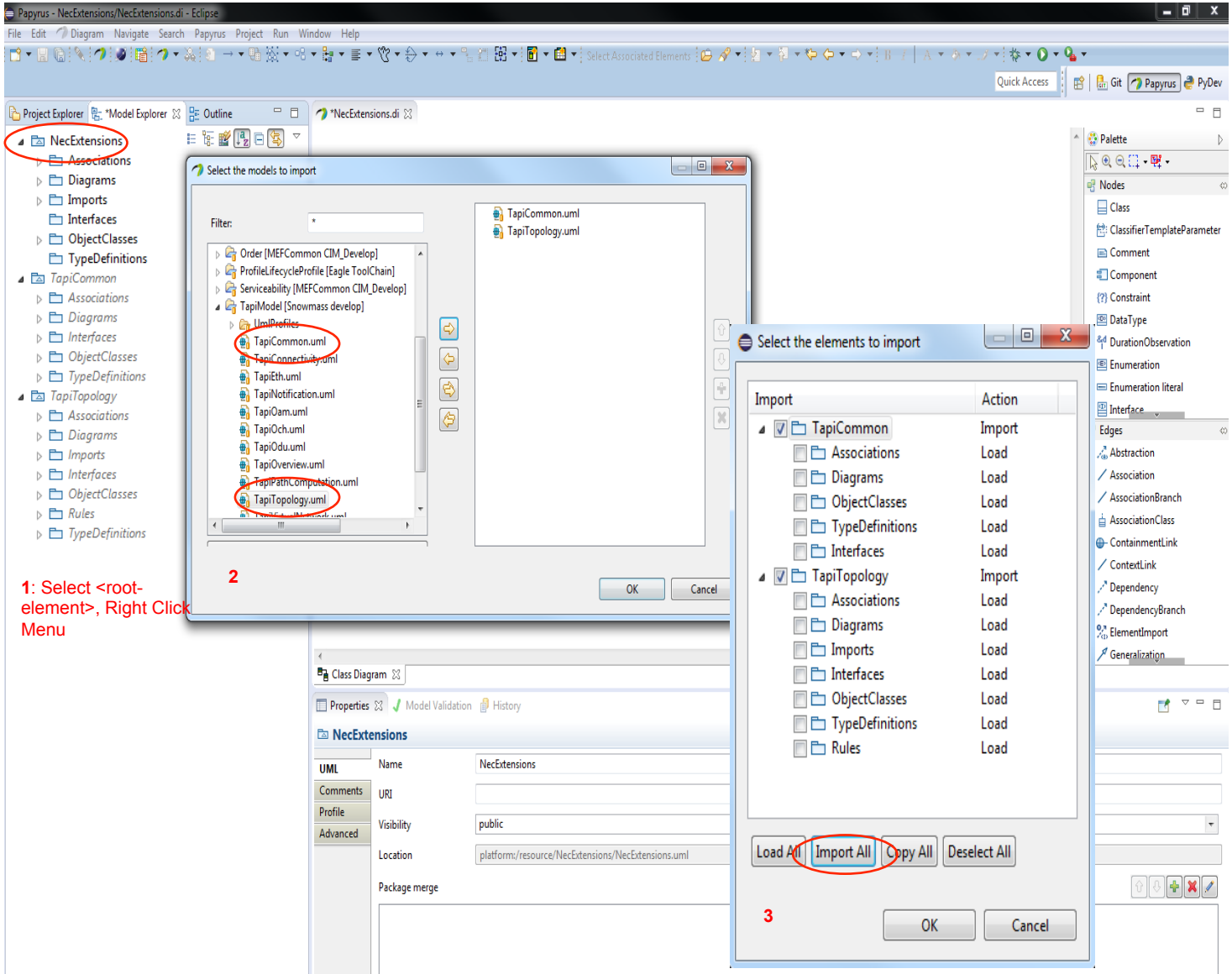
2: Create the appropriate package structure



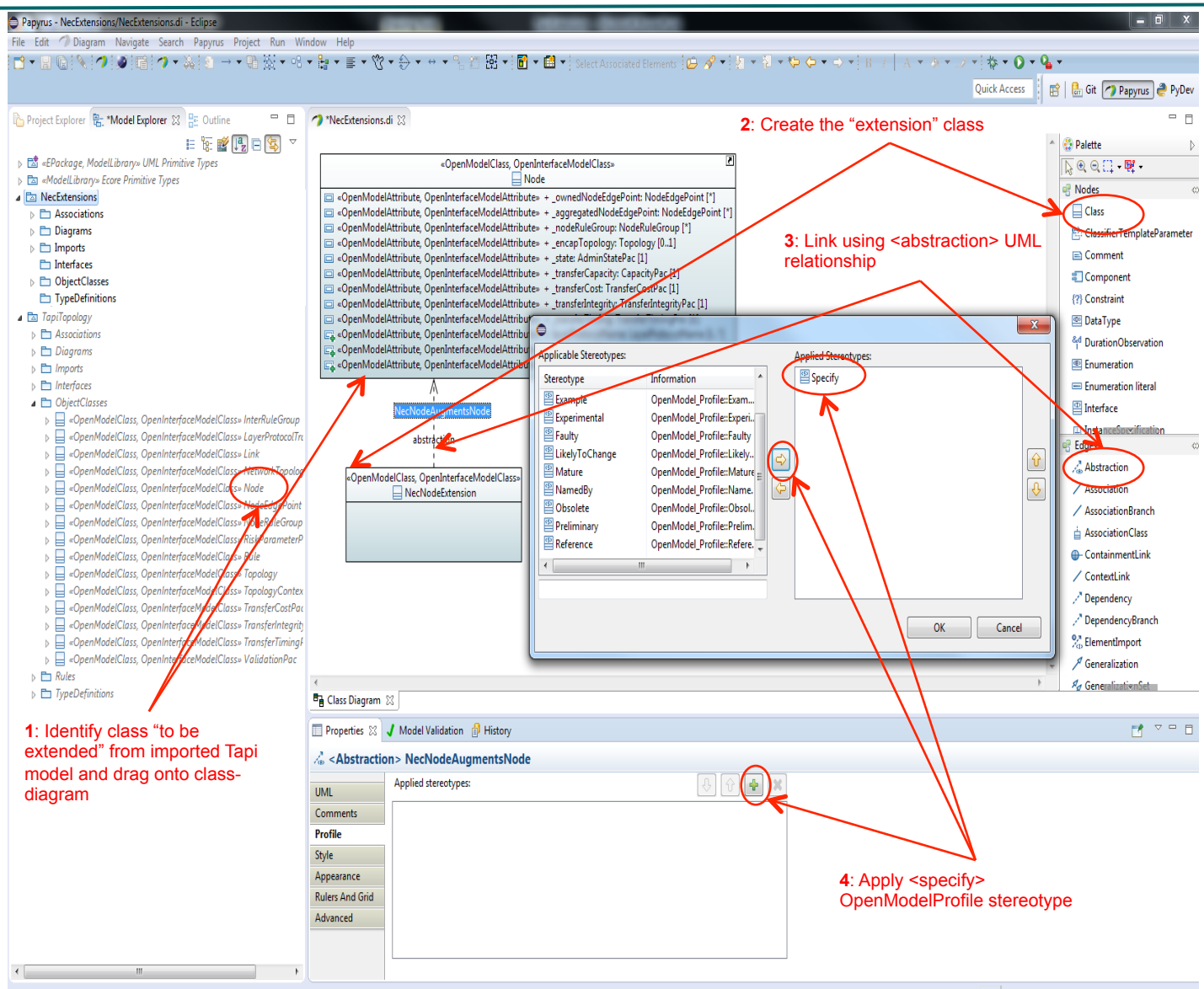
3: Apply OpenModel & OpenInterfaceModel Profiles



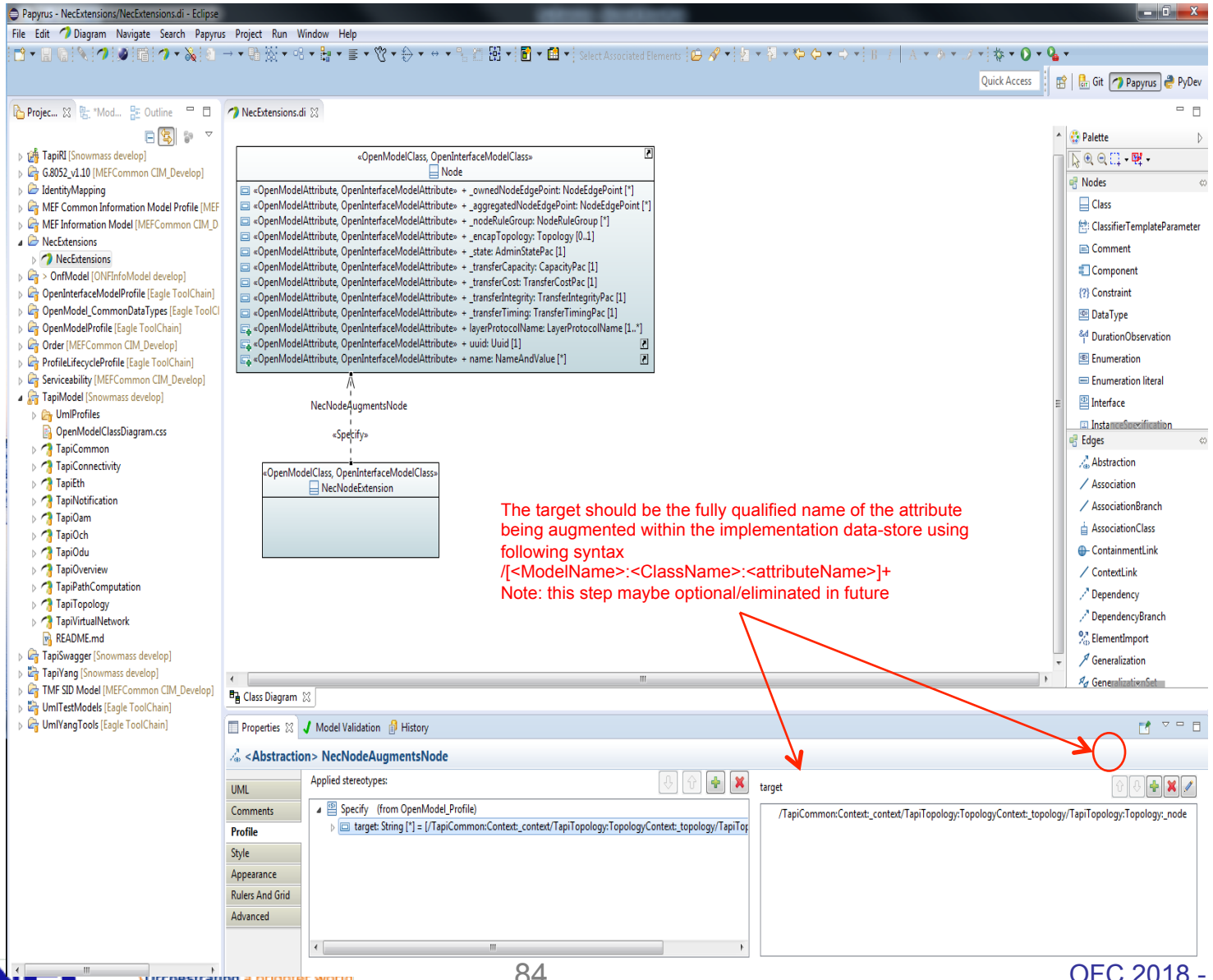
4: Import Necessary TAPI Models



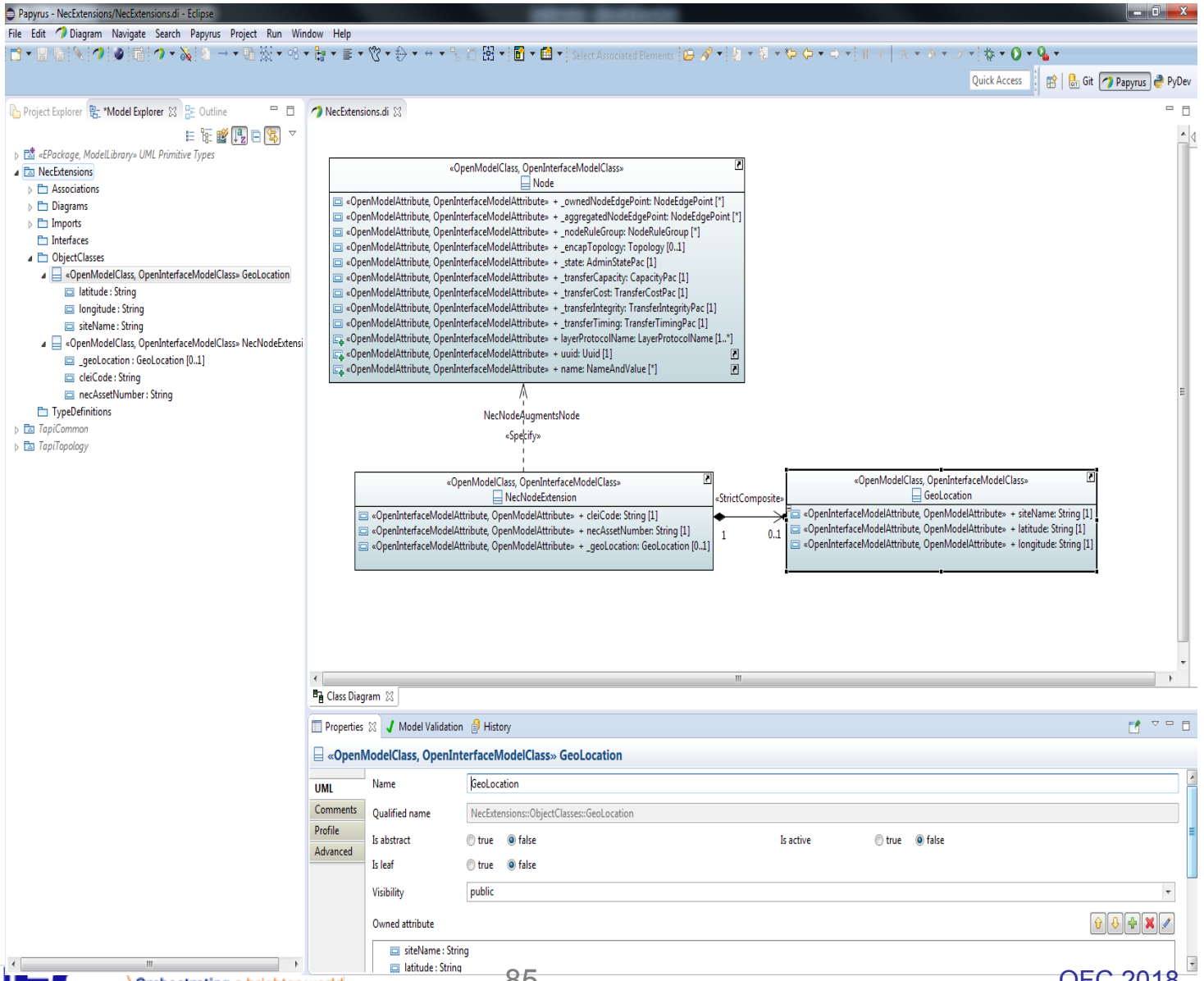
4: Extend the TAPI Class in class diagram



4: Specify the "target" location in the instance tree



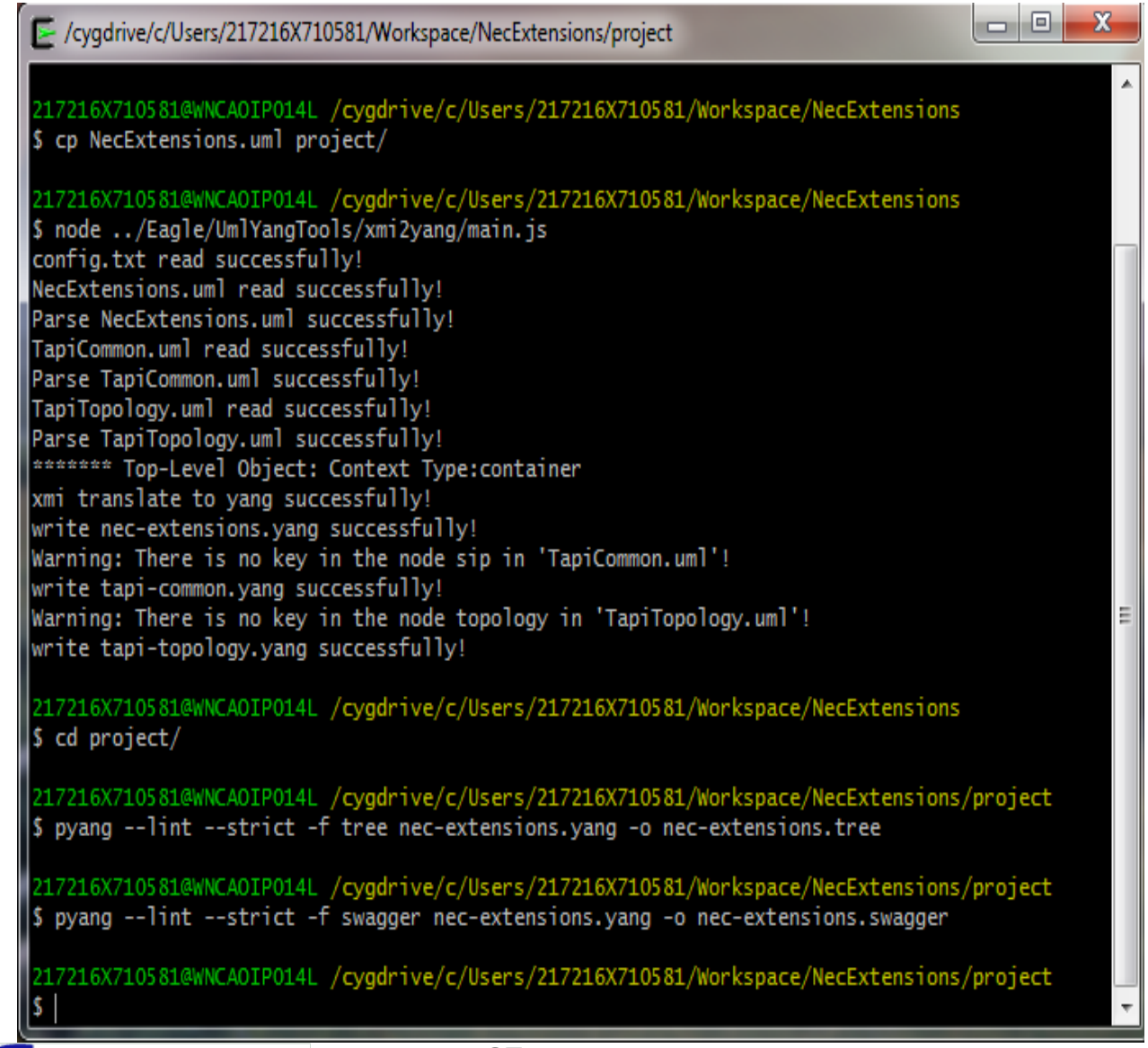
5: Add the additional/proprietary attributes



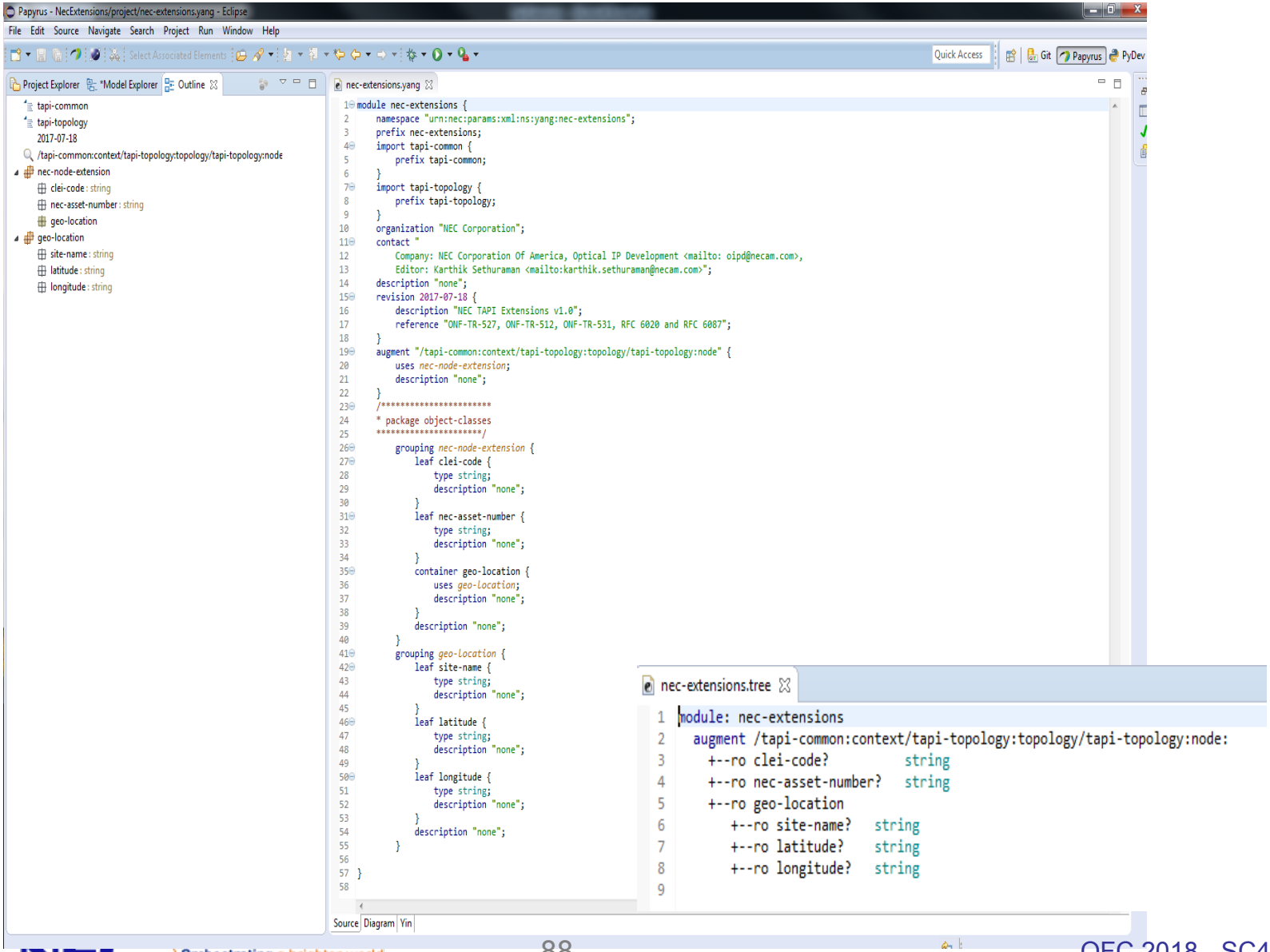
6: Run the Xmi2Yang Generator Tool

- Create “project” folder and the config file
  - use TAPI config as template
- Copy necessary .uml files (including imported TAPI UML files) into “project” folder
- Run the Xmi2Yang tool (need node.js) from the parent folder
  - node <path-to-xmi2yang-folder>/main.js
- Run the pyang tool (with Eagle plugin) from the “project” folder
  - pyang --lint --strict -f tree <yang-file-name> -o <tree-file-name>
  - pyang --lint --strict -f swagger <yang-file-name> -o <swagger-file-name>

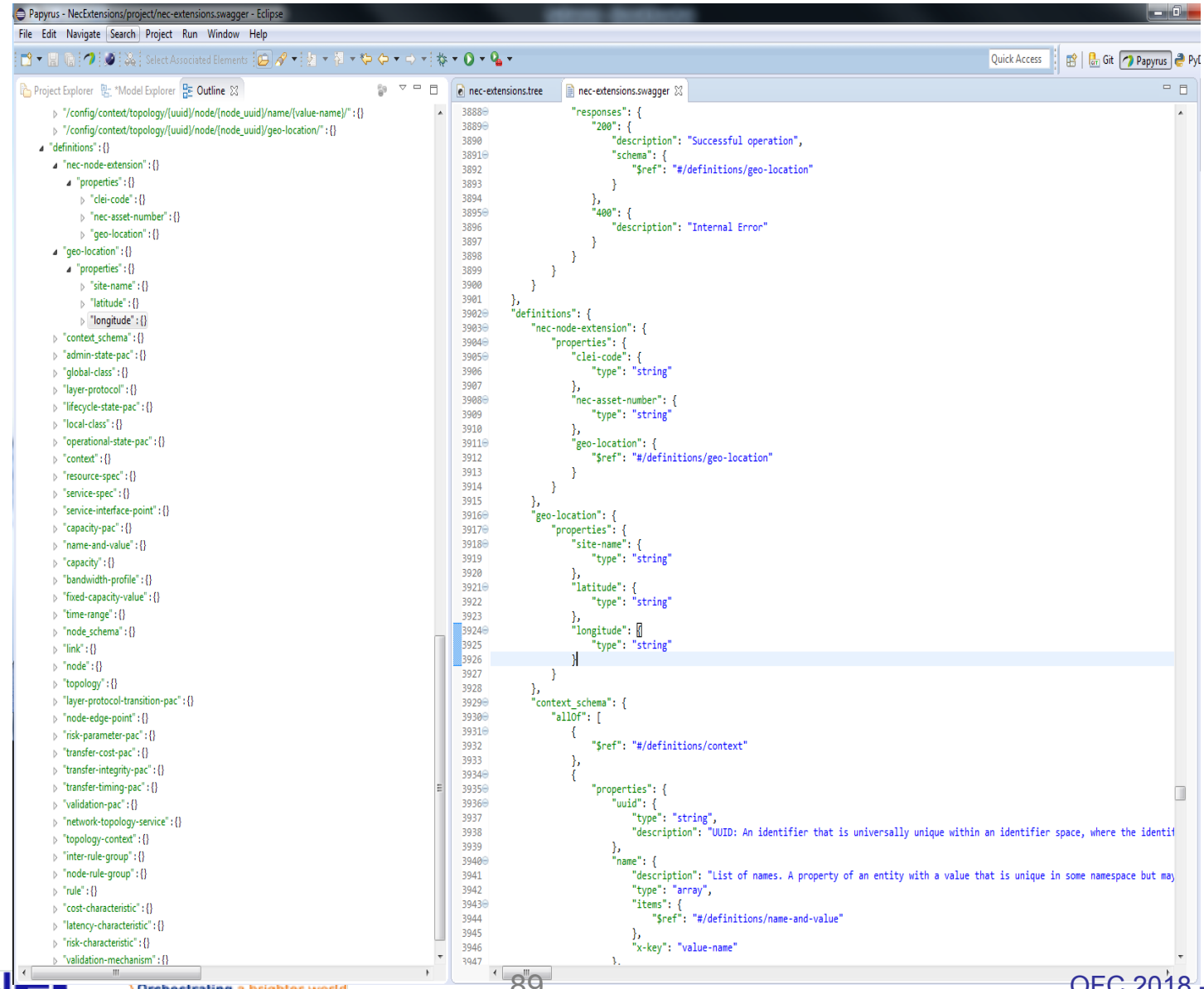
6: Console and Commands



7: Generated Extensions Yang Model & Tree file



8. Generated Open API/Swagger Snapshot



EXTRAS: TAPI SDK & TOOLING OVERVIEW

Key Features of TAPI SDK

- Technology-agnostic API Framework
  - Standardizes a single core technology-agnostic specification that abstracts common transport network functions
- Modular & Extensible
  - Functional features are packaged into small self-contained largely-independent modules
  - TAPI Core Spec is designed to be fully extensible
    - *Extensions* can be Technology, SDO, Operator or Vendor specific
- Industry-wide Interoperability Objective – developed within
  - Open Source SDN SNOWMASS project under Apache 2 license
- SDK components generated using tools for agile prototyping
  - YANG schema generated from UML using guidelines developed in an multi-SDO initiative (IISOMI)
  - SwaggerOpen APIs generated from YANG following RESTConf specification

TAPI SDK: Organization and Modularity

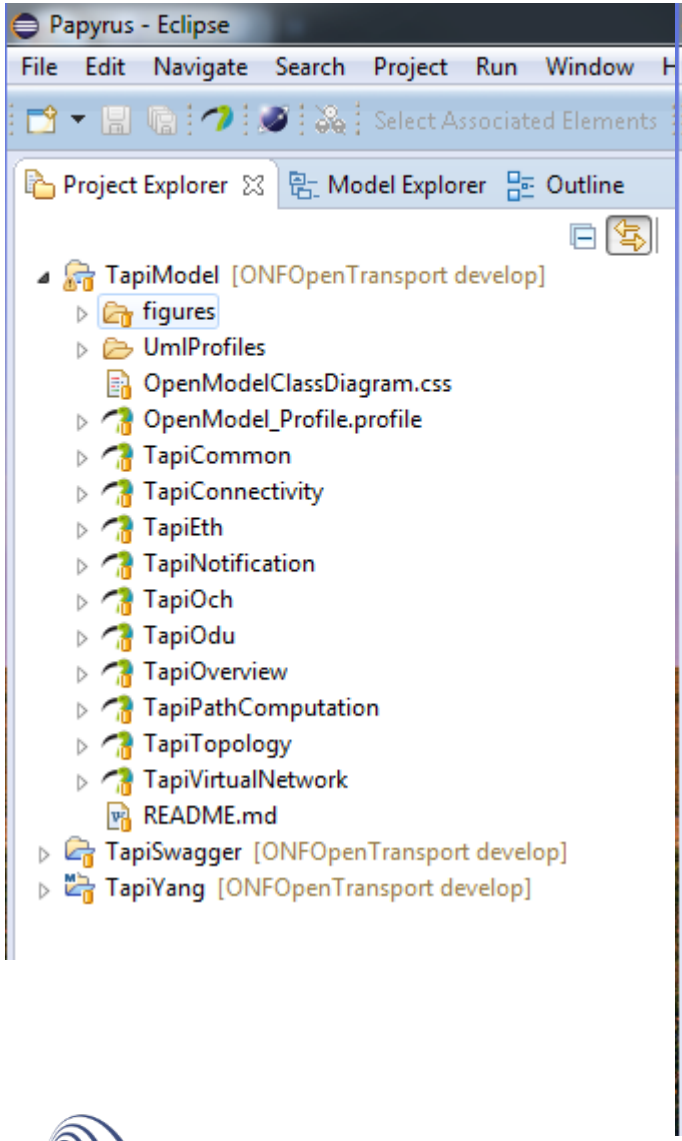
- ONF Transport API Functional Requirements – TR-527, June 2016
  - ONF Open Transport WG Project
  - Input to the TAPI SDK
- Software-wise, TAPI SDK is packaged as 4 eclipse sub-projects
  - Papyrus-UML Information Model
    - A pruned/refactored version of ONF Core IM
    - Is a technology-agnostic generic framework + technology specific extensions (OTN, ETH)
  - YANG Data Schema
    - auto-generated from UML using OSSDN Eagle Tools
  - Open API (Swagger) RESTConf API
    - auto-generated from YANG using OSSDN Eagle tools
  - Reference Implementation (RI) in Python
- Feature-wise, TAPI SDK is organized as a module-set of {UML-Model, YANG-Module, Open-API} per TAPI functional interface. These include
  - Common, Topology, Connectivity, OAM, Path Computation, Virtual Network, Notification
  - ODU, OTSi, ETH technology specification models

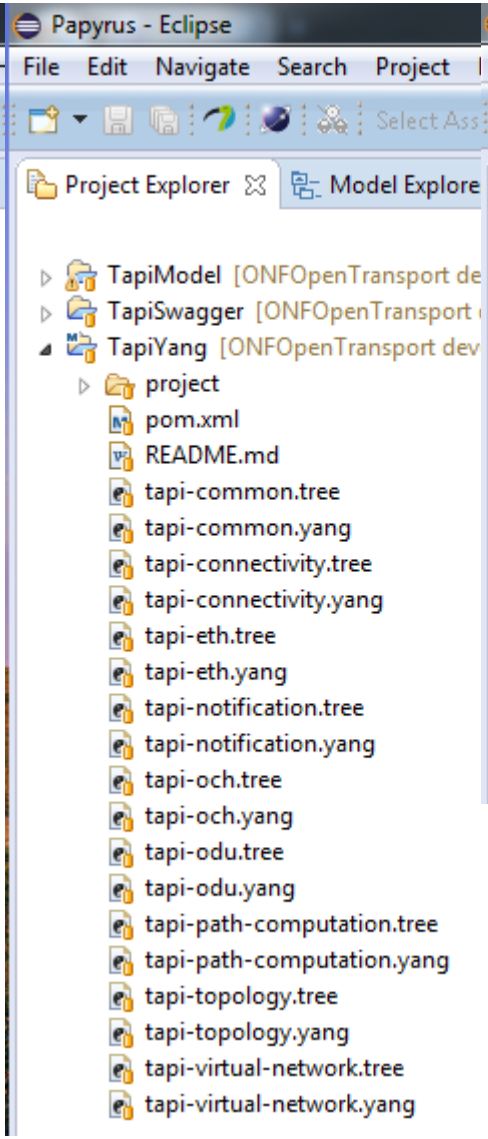
ONF Open Source Projects

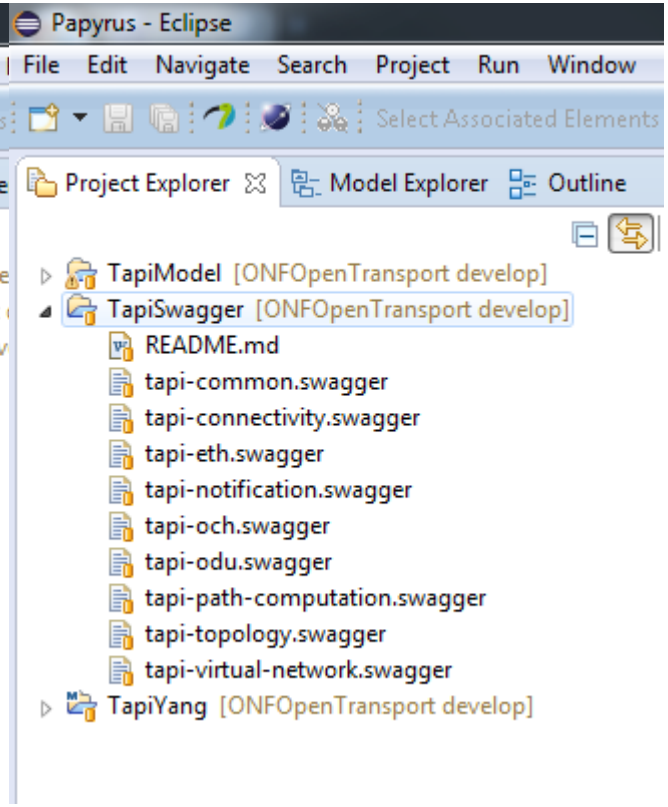
- Project SNOWMASS: OpenSourceSDN Repository for the ONF Transport API Project (TAPI) SDK. Includes
  - TAPI Information Model (UML)
  - TAPI YANG Data Schema auto-generated using OSSDN EAGLE project tools
  - TAPI Swagger/Open API specifications also using OSSDN EAGLE tools
  - TAPI Reference Implementation in Python
  - Supporting documentation
  - Allows vendors to incorporate easily into their models and products.
- https://github.com/OpenNetworkingFoundation/Snowmass-ONFOpenTransport

- Project EAGLE: OpenSourceSDN repository for the IISOMI (Informal Inter-SDO Open Model Initiative). Includes..
  - OpenModelProfile and InterfaceModelProfile
    - UML profiles used in TAPI and code generation tools
  - Open source code to auto-generate YANG schema from the papyrus UML models
  - Also includes code to generate Swagger/Open API schema from the YANG models as well as Python code stubs
- https://github.com/OpenNetworkingFoundation/EAGLE-Open-Model-Profile-and-Tools

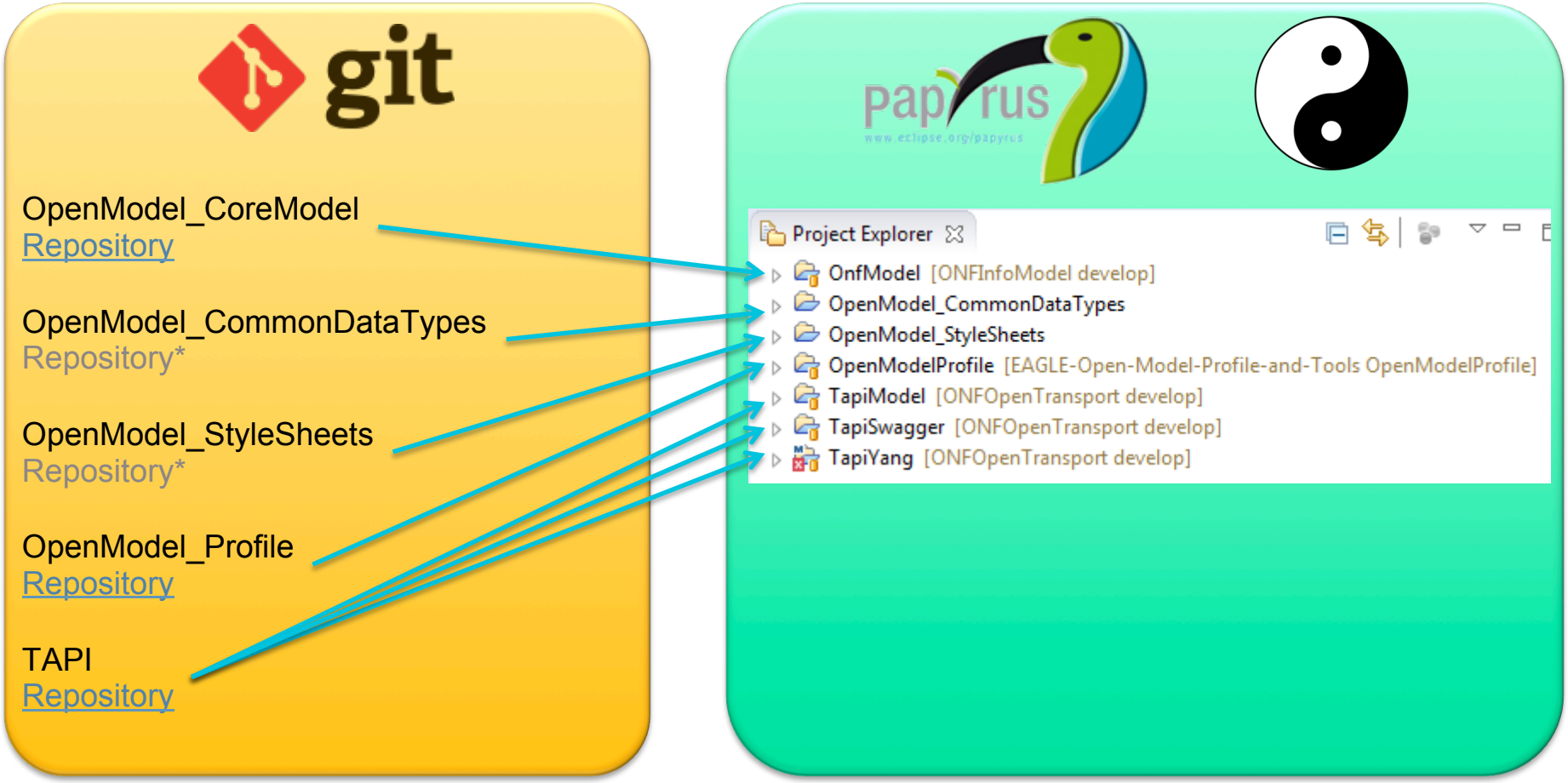
TAPI Eclipse Project Organization





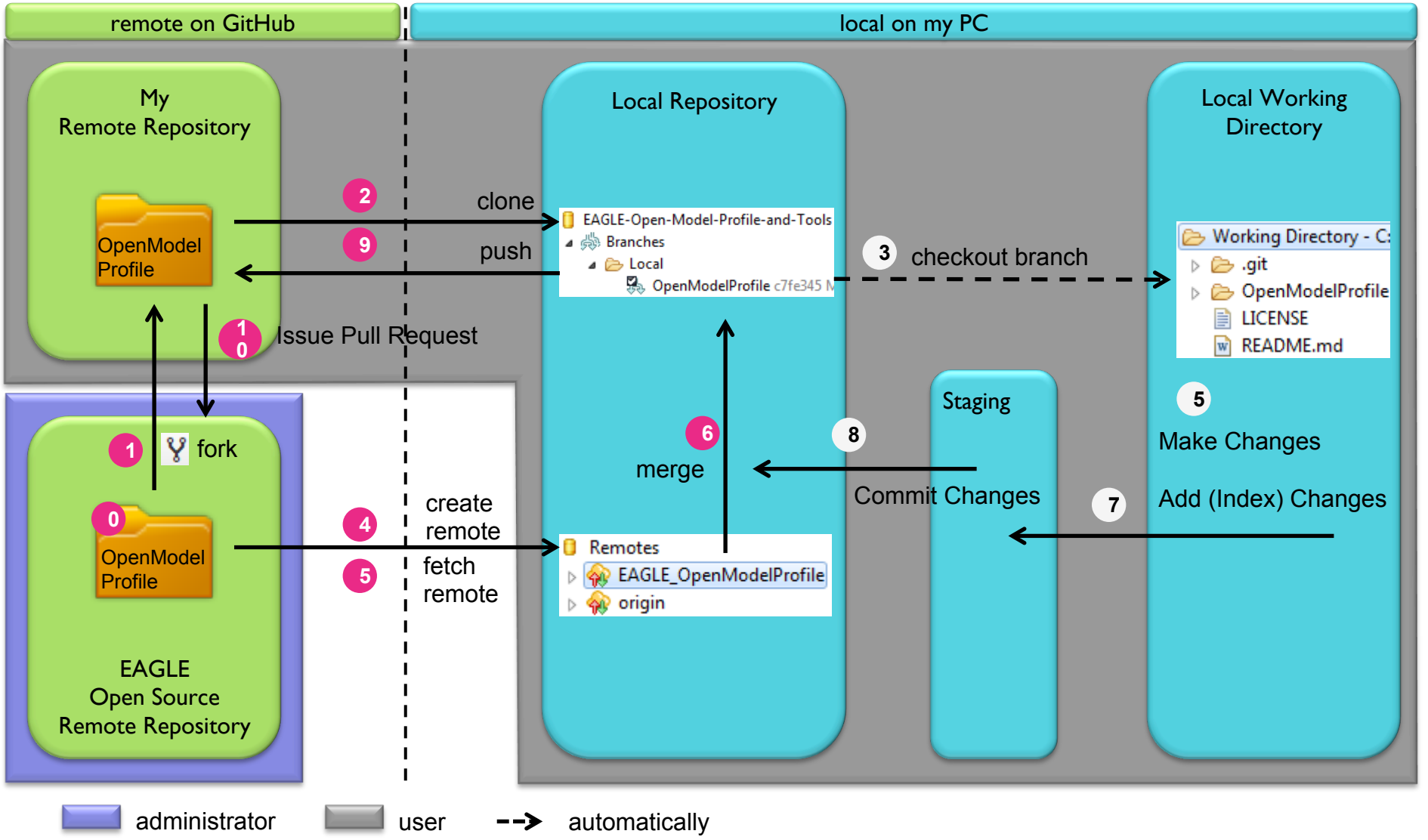


TAPI Resource Dependencies



* not yet created → cloned

GIT Workflow



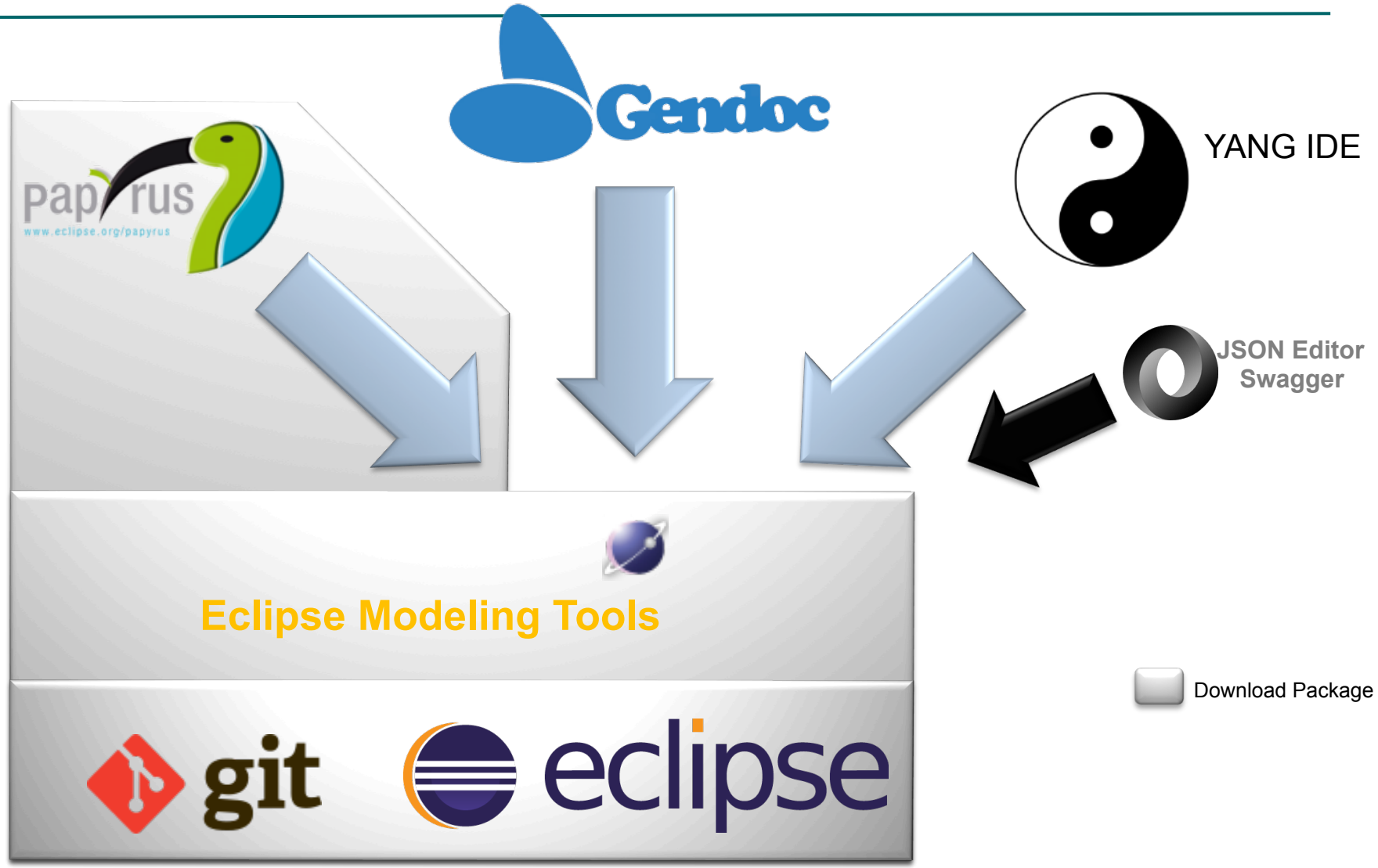
Unified Modeling Language

- Union of all Modeling Languages - Pictorial language to describe software artifacts, their behavior & development cycle.
- Very big, but a nice standard that has been embraced by the industry.
  - Use case diagrams
  - Class diagrams
  - Object diagrams
  - Sequence diagrams
  - Collaboration diagrams
  - State diagrams
  - Activity diagrams
  - Component diagrams
  - Deployment diagrams
  - ……

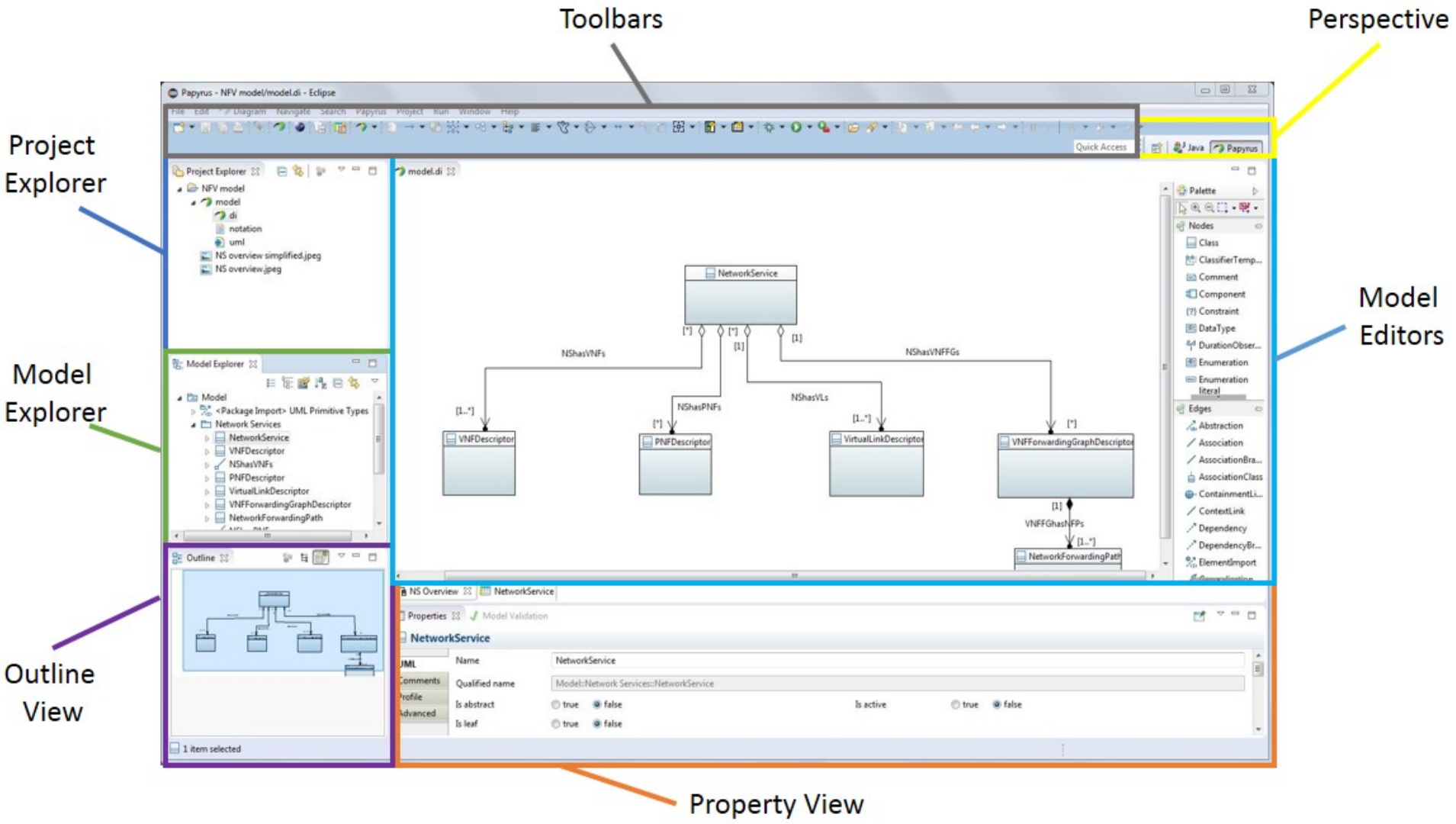
TAPI UML Model

- TAPI (and ONF Core IM) primarily uses the UML Class Diagrams to describe the information that is exchanged over TAPI interface
- Key UML Constructs used are
  - Class
    - Properties, Association-ends
  - Interface
    - Operations
  - Relationships
    - Association (Aggregation, Composition)
    - Dependency (Abstraction, Realization)
    - Generalization
    - Constraint
- TAPI uses open-source papyrus modeling tool for UML modeling
  - https://eclipse.org/papyrus/

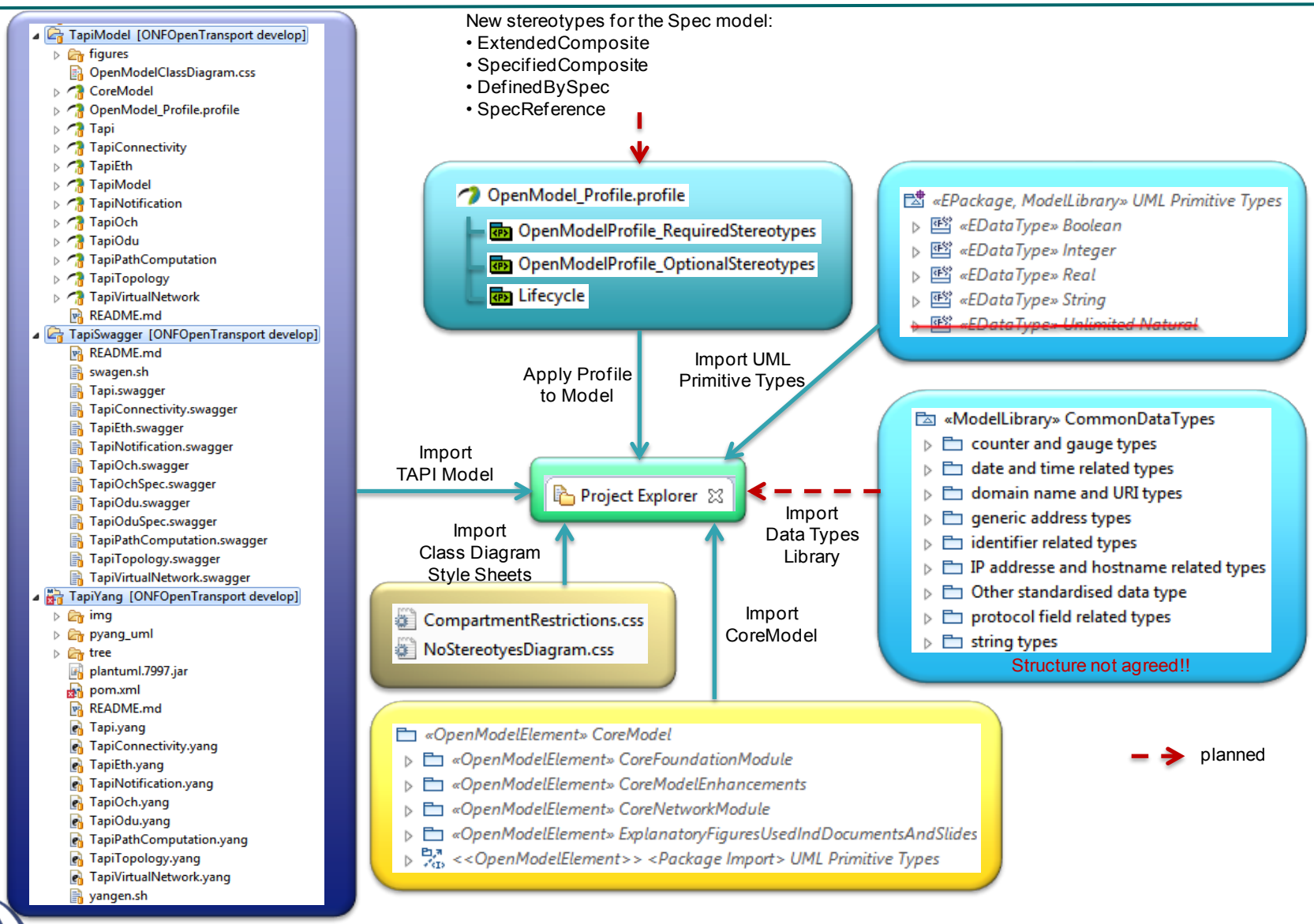
Papyrus UML


Gendoc
YANG IDE
papyrus
www.eclipse.org/papyrus
JSON Editor
Swagger
Eclipse Modeling Tools
git
eclipse
Download Package

Outline of Papyrus Perspective



Constructing TAPI Modeling Environment



YANG Data Schema

- YANG is a modeling language designed to create data models for the NETCONF protocol and more recently to RESTCONF. YANG is capable to define configuration and operational state data, remote procedure calls (RPCs) and Notifications.

- YANG includes description of:
  - Config
  - State
  - RPCs
  - Notifications

YANG Language Overview

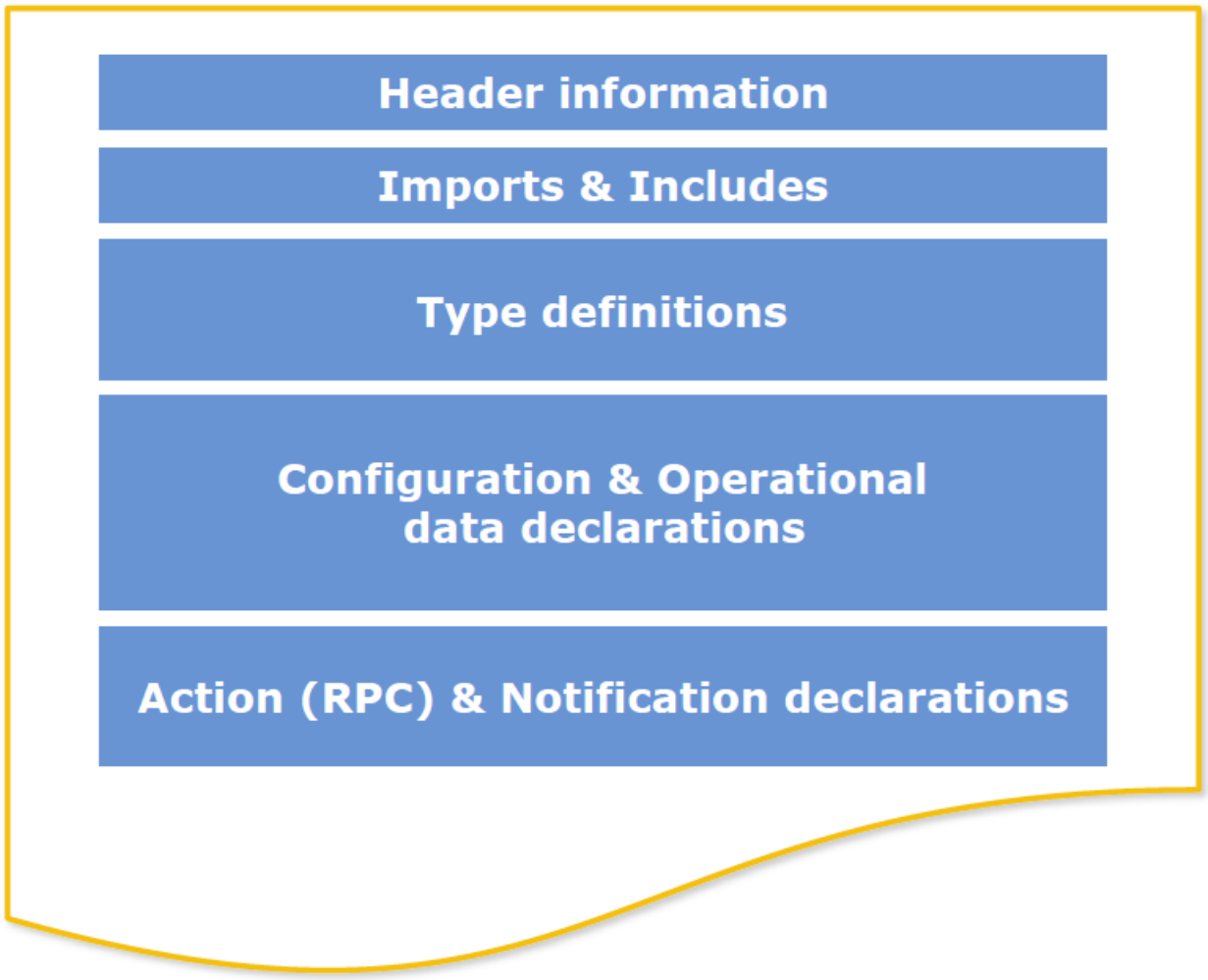
- YANG has... A reasonable self-contained specification
- A focus on readers and reviewers
  - Text-based - Email, patch, and RFC friendly
- Limited Scope, but extensible
- Ability to model config data, state data, RPCs, and notifications
- Experience gained by existing implementations
  - Design is based on four proprietary modeling languages
- Support for YANG
- Running code (multiple implementations)
- People outside the YANG team using it in other WGs in the IETF
- Backing of several NETCONF implementers
- Growing group of fans and admirers

Yang: Easy to read, easy to learn

```
container system {
    leaf host-name {
        type inet:host;
       description "Name of this host";
    }
    container services {
        container ssh {
            presence "Enables SSH";
            description "SSH service specific
configuration";
             // more leafs, containers and stuff here...
        }
    }
}
```

```
<system>
  <host-name>fred</host-name>
  <services>
     <ssh/>
  </services>
</system>
```

YANG module contents



RESTconf

- RESTCONF is a REST-like protocol that provides a HTTP-based API to access the data, modeled by YANG. The REST-like operations are used to access the hierarchical data within a data store. The information modeled in YANG is structured in the following tree:
  - /restconf/config : "Data (configuration/operational) accessible from the client but not editable"
  - /restconf/modules : "Set of YANG models supported by the RESTCONF server"
  - /restconf/operations : "Set of operations (YANG-defined RPCs) supported by the server"
  - /restconf/streams: "Set of notifications supported by the server". They are implemented in a websocket

- With YANGs + RESTconf ➔
  - We have the complete Transport REST NBI

RESTCONF (Cont.)

- RESTCONF
  - Almost RFC
  - RESTful protocol to access YANG defined data
  - Representational State Transfer, i.e. server maintains no session state
  - URIs reflect data hierarchy in a Netconf datastore
  - HTTP as transport
  - Data encoded with either XML or JSON
  - Operations :

RESTCONF	Netconf
GET	<get-config>, <get>
POST	<edit-config> (“create”)
PUT	<edit-config> (“replace”)
PATCH	<edit-config> (“merge”)
DELETE	<edit-config> (“delete”)
OPTIONS	(discover supported operations)
HEAD	(get without body)

RESTCONF HTTP tree

- RESTCONF is a REST-like protocol that provides a HTTP-based API to access the data, modeled by YANG. The REST-like operations are used to access the hierarchical data within a datastore. The information modeled in YANG is structured in the following tree:
  - /restconf/data : "Data (configuration/operational) accessible from the client but not editable"
  - /restconf/modules : "Set of YANG models supported by the RESTCONF server"
  - /restconf/operations : "Set of operations (**YANG-defined RPCs**) supported by the server"
  - /restconf/streams: "Set of notifications supported by the server"

YANG mapping to JSON vs XML

JSON – 214 octets*

```
{
   "ietf-interfaces:interfaces": {
      "interface": [
          {
              "name": "eth0",
              "type": "ethernetCsmacd",
              "location": "0",
              "enabled": true,
              "if-index": 2
          },
          {
              "name": "eth1",
              "type": "ethernetCsmacd",
              "location": "1",
              "enabled": false,
              "if-index": 2
          }
      ]
   }
}
```

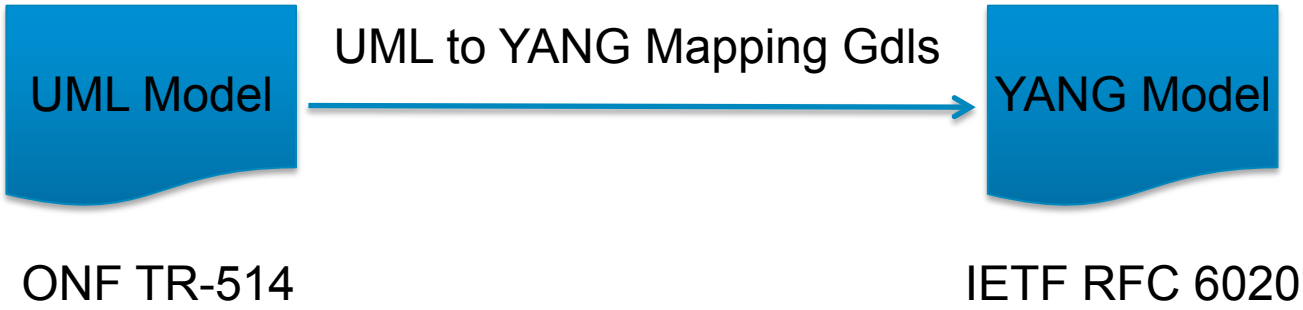
XML – 347 octets*

```
<interfaces xmlns:="urn:ietfparams:xml:ns:yang:ietf-interfaces">
    <interface>
        <name>eth0</name>
        <type>ethernetCsmacd</type>
        <location>0</location>
        <enabled>true</enabled>
        <if-index>2</if-index>
    </interface>
    <interface>
        <name>eth1</name>
        <type>ethernetCsmacd</type>
        <location>1</location>
        <enabled>false</enabled>
        <if-index>7</if-index>
    </interface>
</interfaces>
```

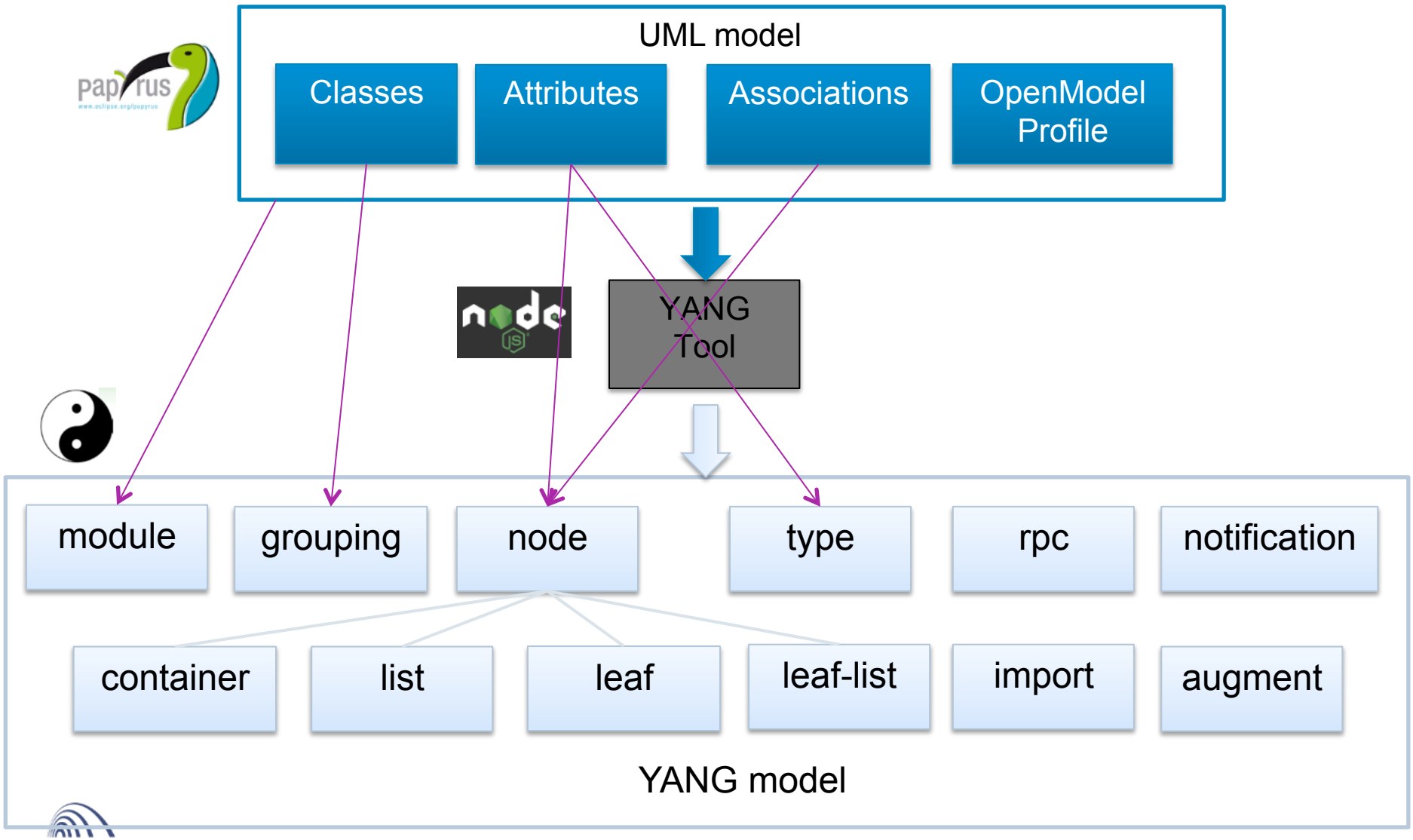
*all white space removed

Project EAGLE : Uml2Yang Generation Tool

- **Objective**
  - Translates a UML (Unified Modeling Language) model to a YANG model defined in RFC6020
  - Uses Node.JS platform
- **Usage of the generated YANG output**
  - Fed to controllers, Apps for interface generation
  - Output is also used by eagle swagger tool to generate RESTful/RESTconf APIs
  - https://github.com/OpenNetworkingFoundation/EAGLE-Open-Model-Profile-and-Tools/tree/ToolChain/UmlYangTools

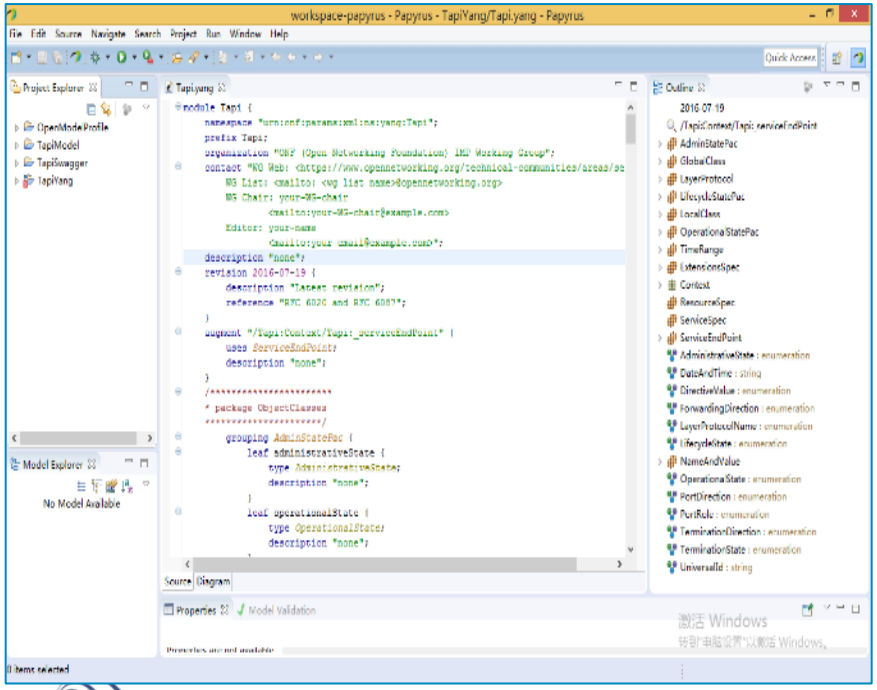


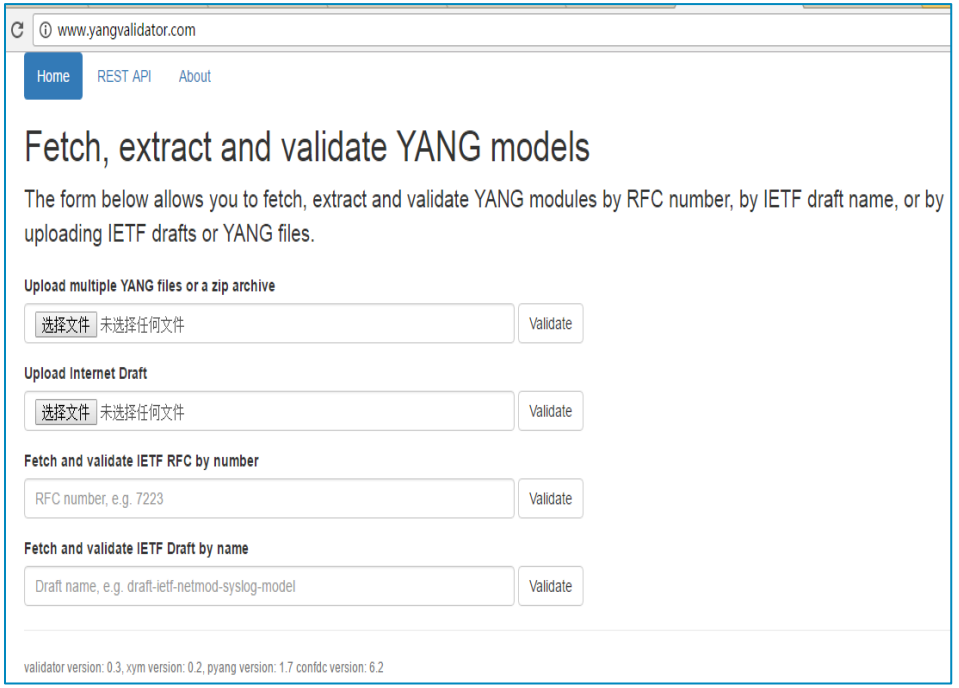
UML to YANG Tool



Validate/View YANG files

- Eclipse Yang IDE: RFC 6020 validation, GUI view
  - https://marketplace.eclipse.org/content/yang-ide
- YANG validator: RFC 6020/6087 validation, online tool
  - http://www.yangvalidator.com/
- pyang: RFC 6020/6087 validation, command line





T-API Open API (Swagger) specifications

- Open API (formerly known as Swagger) is a popular compact and easy to parse data schema format to describe REST APIs
  - Open API Schemas can be described in two popular web encoding languages – YAML or JSON
- The generated T-API Open API specifications provide a mapping from the Yang data schema into Open API JSOn format, which can then be used to generate Python and/or Java code for implementation of the API in RestConf
- https://www.openapis.org/
- https://swagger.io

YANG Mapping to Open API (Swagger)

- YANG data schema can be translated to Open API schema
- Translation driven by YANG data model (must be known in advance)
- YANG datatype information is used to translate leaf values to the most appropriate Open API representation
- Slightly more compact (irrelevant with compression)
- Increased human readability (less noise)

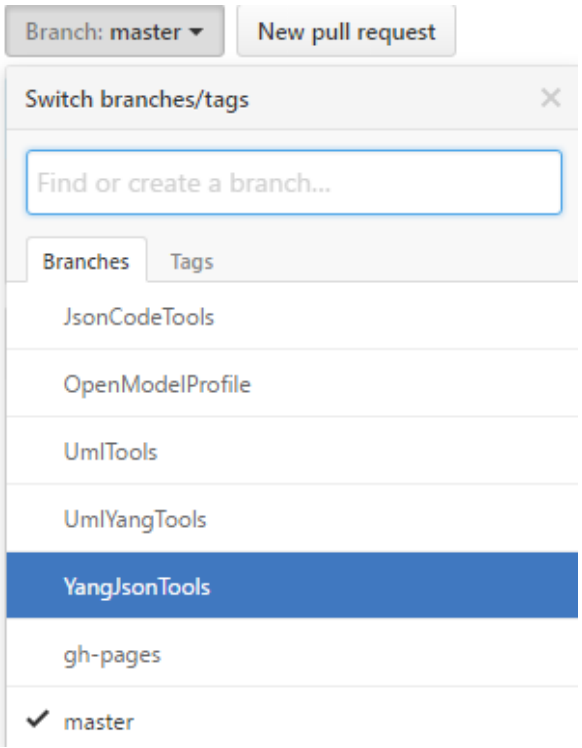
T-API Swagger: From YANG to Swagger Specs

Project Eagle YangJsonTools
https://github.com/OpenNetworkingFoundation/EAGLE-Open-Model-Profile-and-Tools/tree/ToolChain/YangJsonTools

Includes a plugin for Pyang
https://github.com/mbj4668/pyang

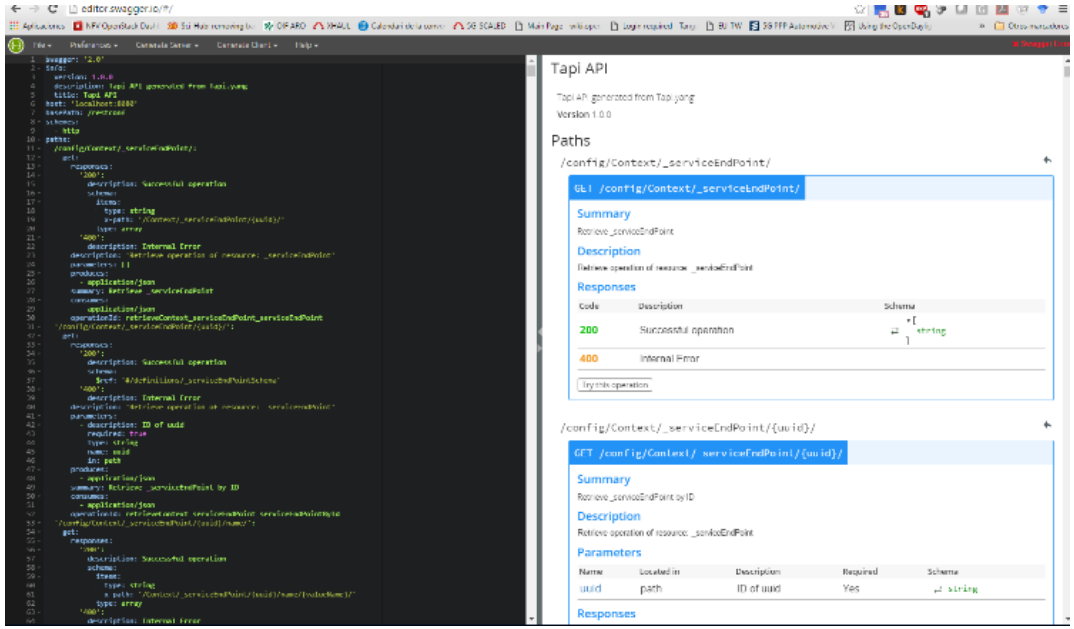
Pyang is a YANG validator, transformator and code generator, written in python. It can be used to validate YANG modules for correctness, to transform YANG modules into other formats, and to generate code from the modules.

The project EAGLE YangJsonTools plugin processes the YANG files and describes the associated RESTconf interface using Swagger REST definitions.



T-API Swagger: Editing Swagger Specs

- Using Eclipse JSON editor
  - Described in T-API Developer Guide
- Using Swagger on-line editor
  - http://editor.swagger.io/
  - https://swaggerhub.com/



T-API Swagger: Understanding Swagger specs

- Paths
  - Each path may include CRUD (POST, GET, PUT, DELETE) if config
  - Only GET is allow for State data
  - Each CRUD includes the following details:
    - Summary
    - Parameters (in path or in body)
    - Responses
    - Produces/consumes


```
"/config/Context/_connectivityService/{uuid}/": {
    "put": {
    },
    "post": {
        "responses": {
            "200": {
                "description": "Successful operation"
            },
            "400": {
                "description": "Internal Error"
            }
        },
        "description": "Create operation of resource: _connectivityService",
        "parameters": [
            {
                "description": "ID of uuid",
                "required": true,
                "type": "string",
                "name": "uuid",
                "in": "path"
            },
            {
                "required": true,
                "description": "_connectivityServicebody object",
                "schema": {
                    "$ref": "#/definitions/ServiceSpec"
                },
                "name": "_connectivityService",
                "in": "body"
            }
        ],
        "produces": [
            "application/json"
        ],
        "summary": "Create _connectivityService by ID",
        "consumes": [
            "application/json"
        ],
        "operationId": "createContext_connectivityService_connectivityServiceById"
    },
    "delete": {
    },
    "get": {
    }
},
```

T-API Swagger: Understanding Swagger specs

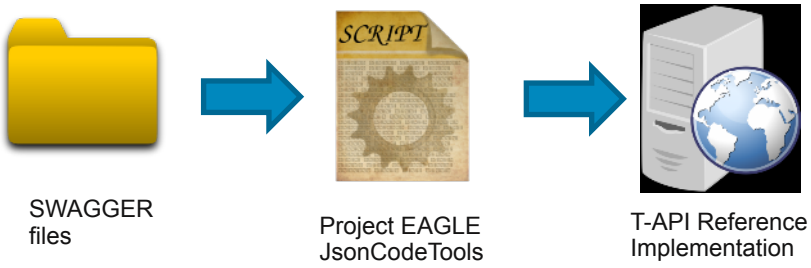
- Definitions
  - They allow inheritance (allOf)
  - Items are described in properties
  - Other descriptions might be referenced (e.g., connectivityService)


```
"ContextSchema": {
    "description": "The Network Control Domain (NCD) object cl
    "allOf": [
        {
            "$ref": "#/definitions/GlobalClass"
        },
        {
            "properties": {
                "_connection": {
                },
                "_notification": {
                },
                "_vnwService": {
                },
                "_topology": {
                },
                "_notifSubscription": {
                },
                "_serviceEndPoint": {
                },
                "_pathCompService": {
                },
                "_nwTopologyService": {
                },
                "_path": {
                },
                "_connectivityService": {
                    "items": {
                        "$ref": "#/definitions/ServiceSpec"
                    },
                    "type": "array",
                    "description": "none",
                    "x-key": "uuid"
                }
            }
        }
]
```

Creating a T-API Reference Implementation

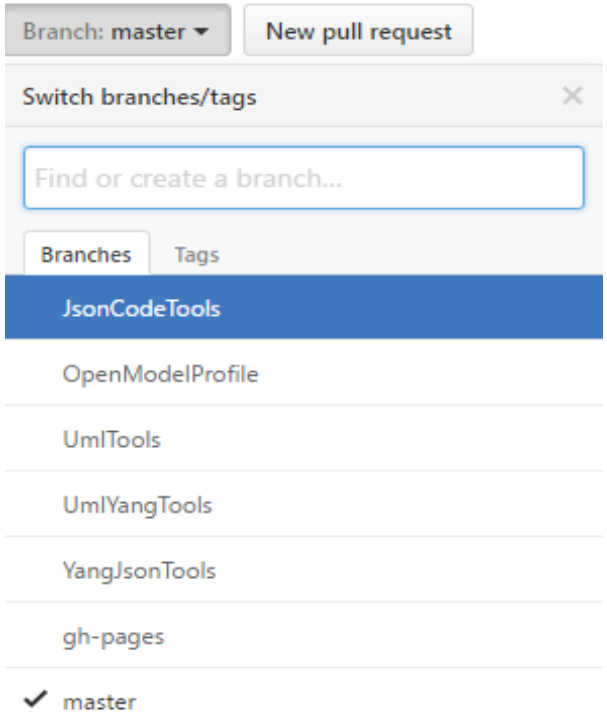
Project EAGLE JsonCodeTools

https://github.com/OpenNetworkingFoundation/EAGLE-Open-Model-Profile-and-Tools/tree/ToolChain/JsonCodeTools



Server Generator for Python

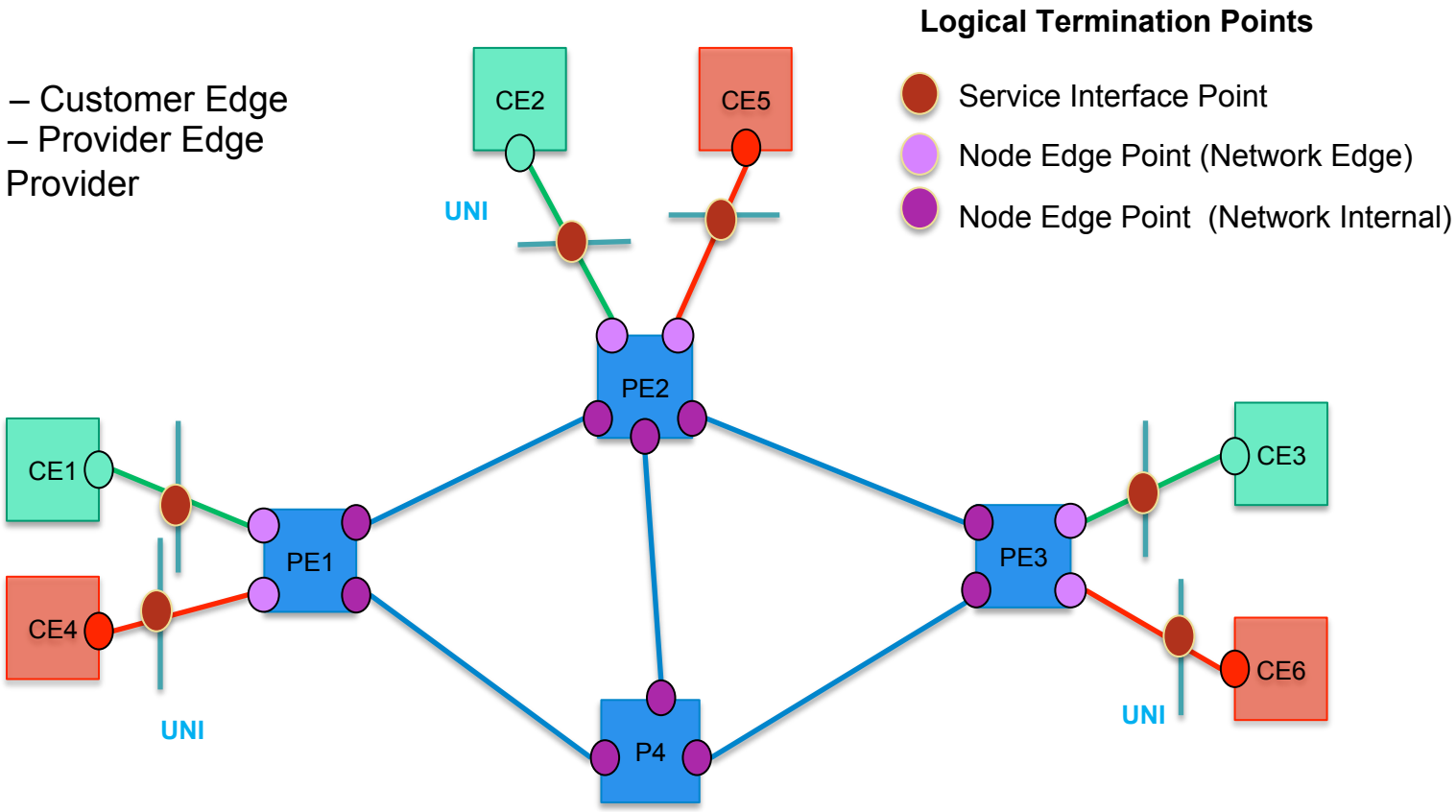
Python code that uses Swagger json outputs from **YangJsonTools** to generate a **RESTful and websocket server** and the defined swagger objects, paths and notifications.



EXTRAS: TAPI CONCEPTS & EXAMPLE

Simple Physical Network Example to illustrate T-API

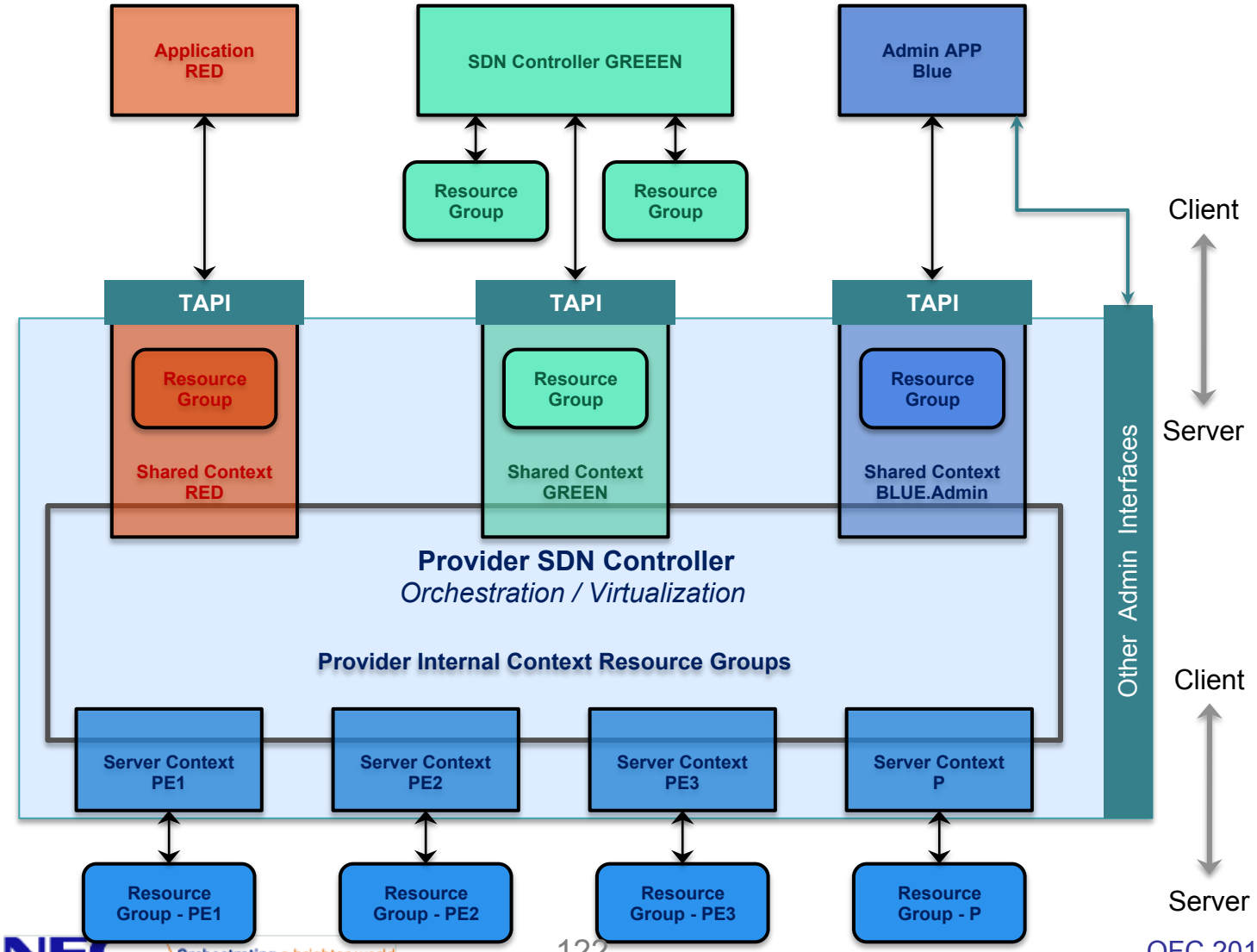
CE – Customer Edge
PE – Provider Edge
P - Provider



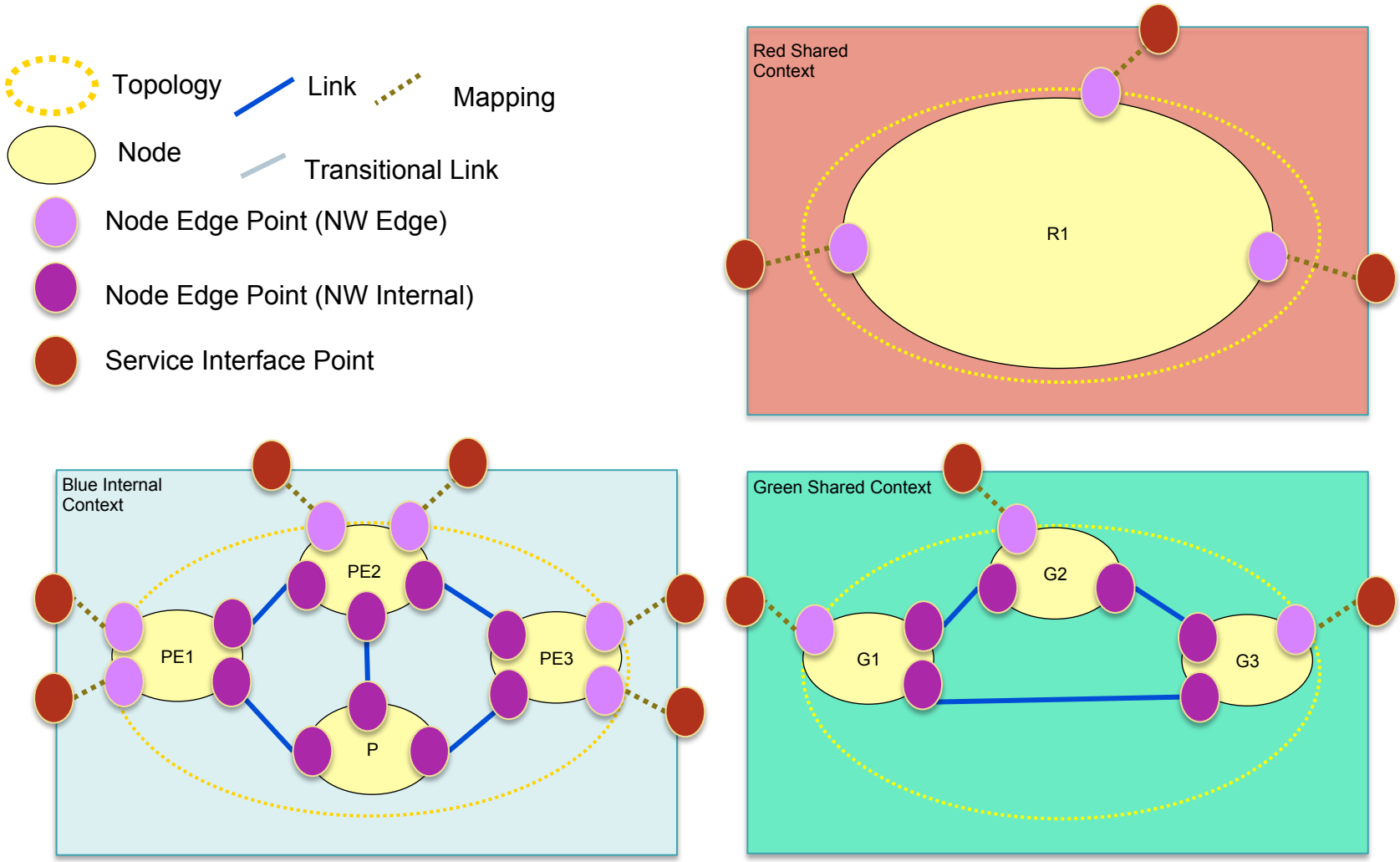
- A Network Provider (Blue) with two Customers (Red and Green)
- All UNI interfaces are ETH (e.g. 10GE), I-NNI interfaces are OTU (e.g. 100G OTN)
- All PE-NE are ODU/ETH switch capable, while P-NE is only ODU switch capable

T-API Contexts for the Simple Network Example

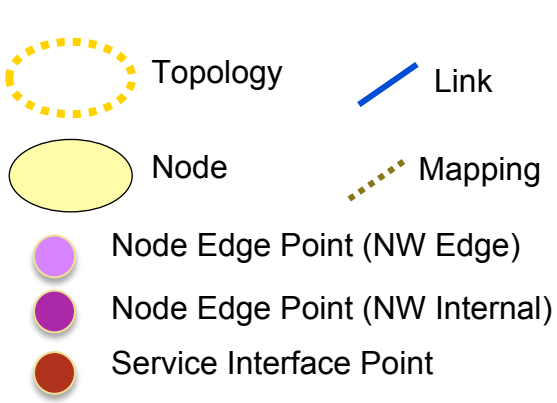
(based on ONF Architecture v1.1)

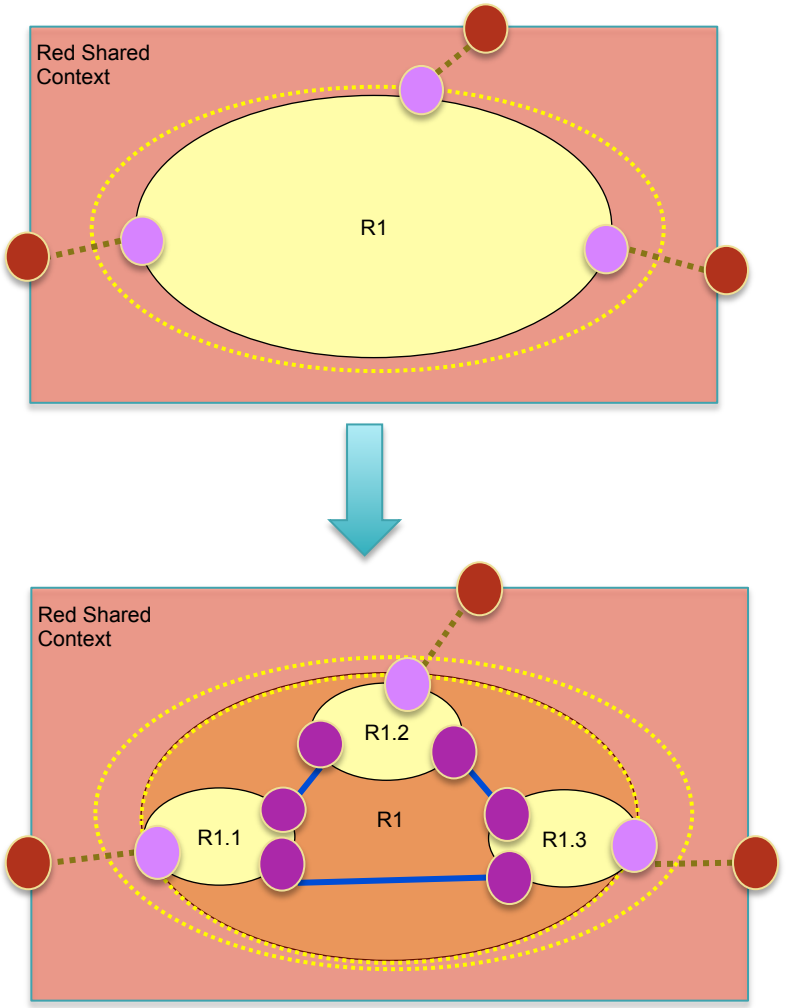


Example Topology Abstractions in a Shared Context



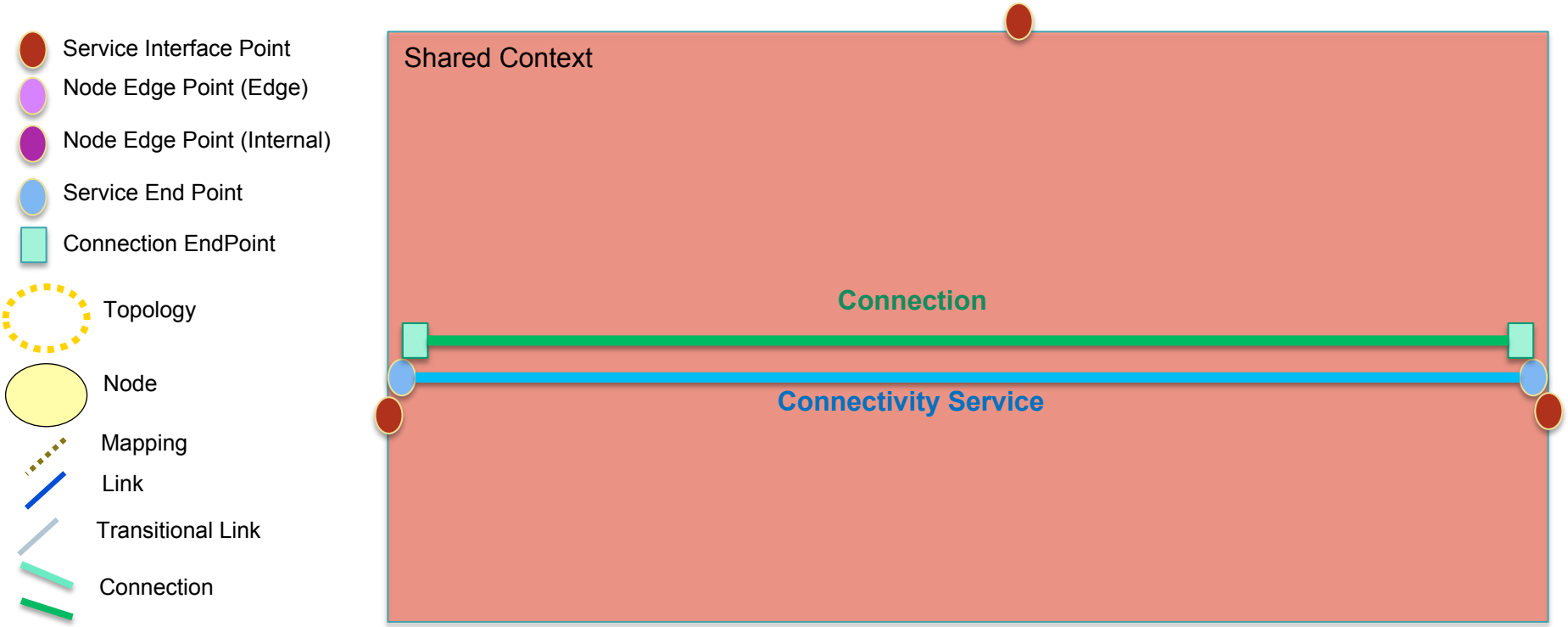
Recursive Topology Decomposition within Context





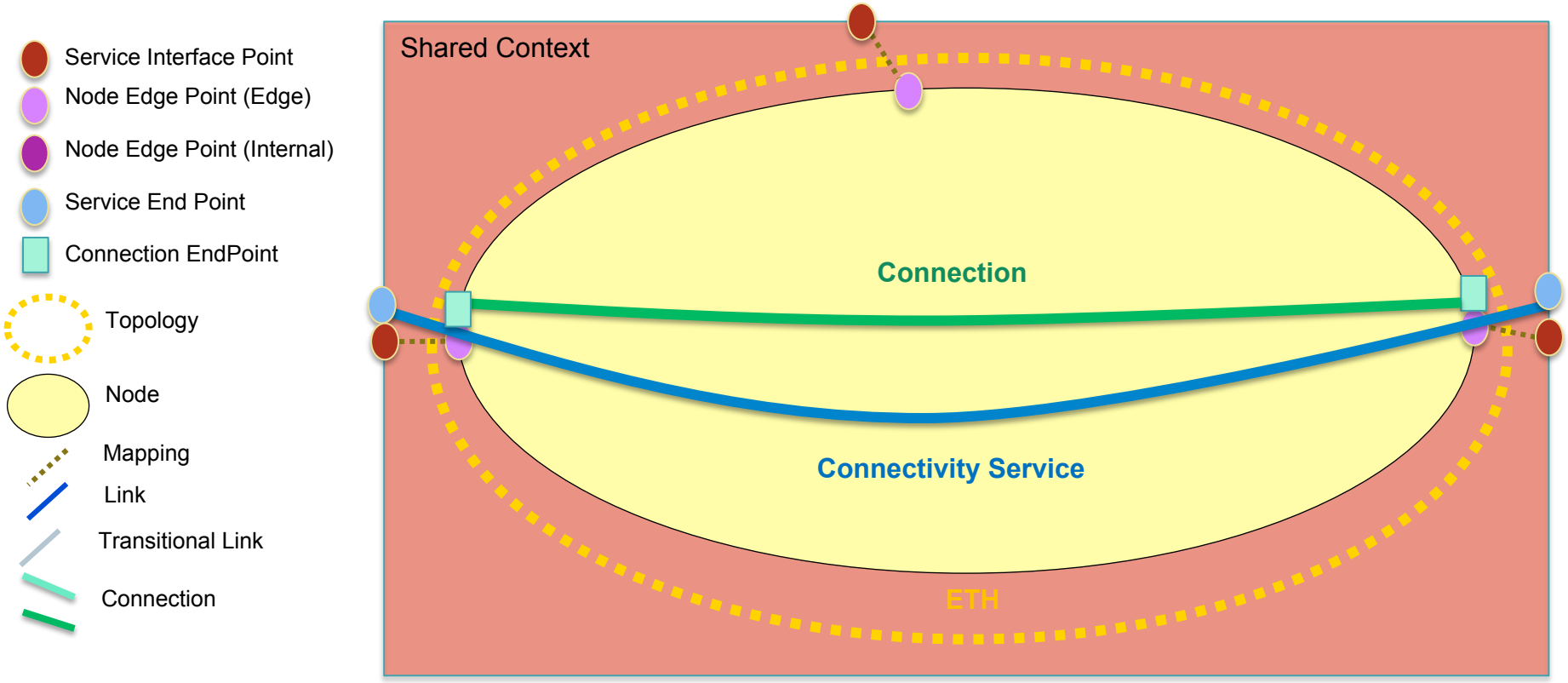
Client1 (Red) Shared Context: with empty Topology

- Shared *Context* defined by the set of *ServiceEndPoints*
- No *Topology* exposed - Retrieving topology will return empty
- *ConnectivityService* can be requested between *ServiceInterfacePoints*
- Client provides input *ConnectivityConstraints* – Capacity/BW is the only required input
- Associated *Connection* representing network resources is created/returned to Client



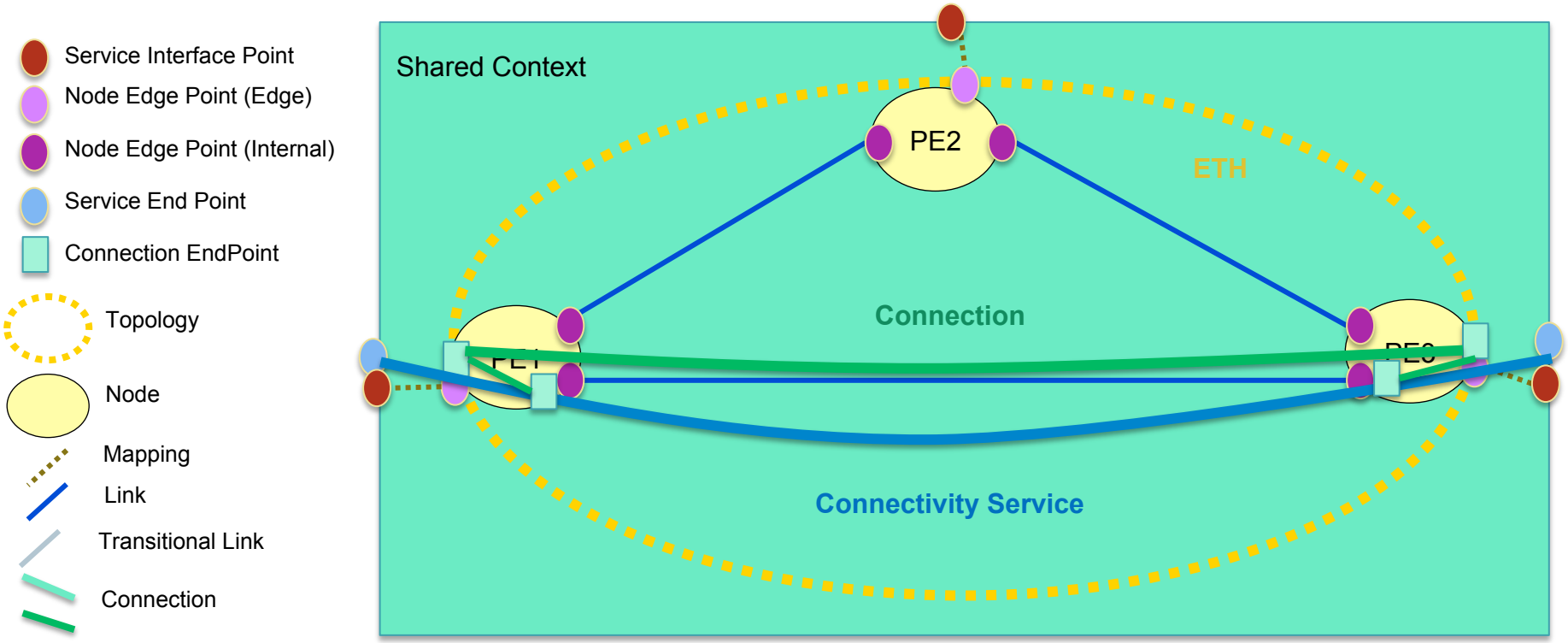
Client-1 (Red) Shared Context: Single Node Topology

- Single Node abstraction example
- *Node* and its *NodeEdgePoints* provide some approximation of the network capabilities
- *ConnectivityService* can be requested between *ServiceInterfacePoints*
- *Connections* appear as cross-connections across node, no visibility of underlying route



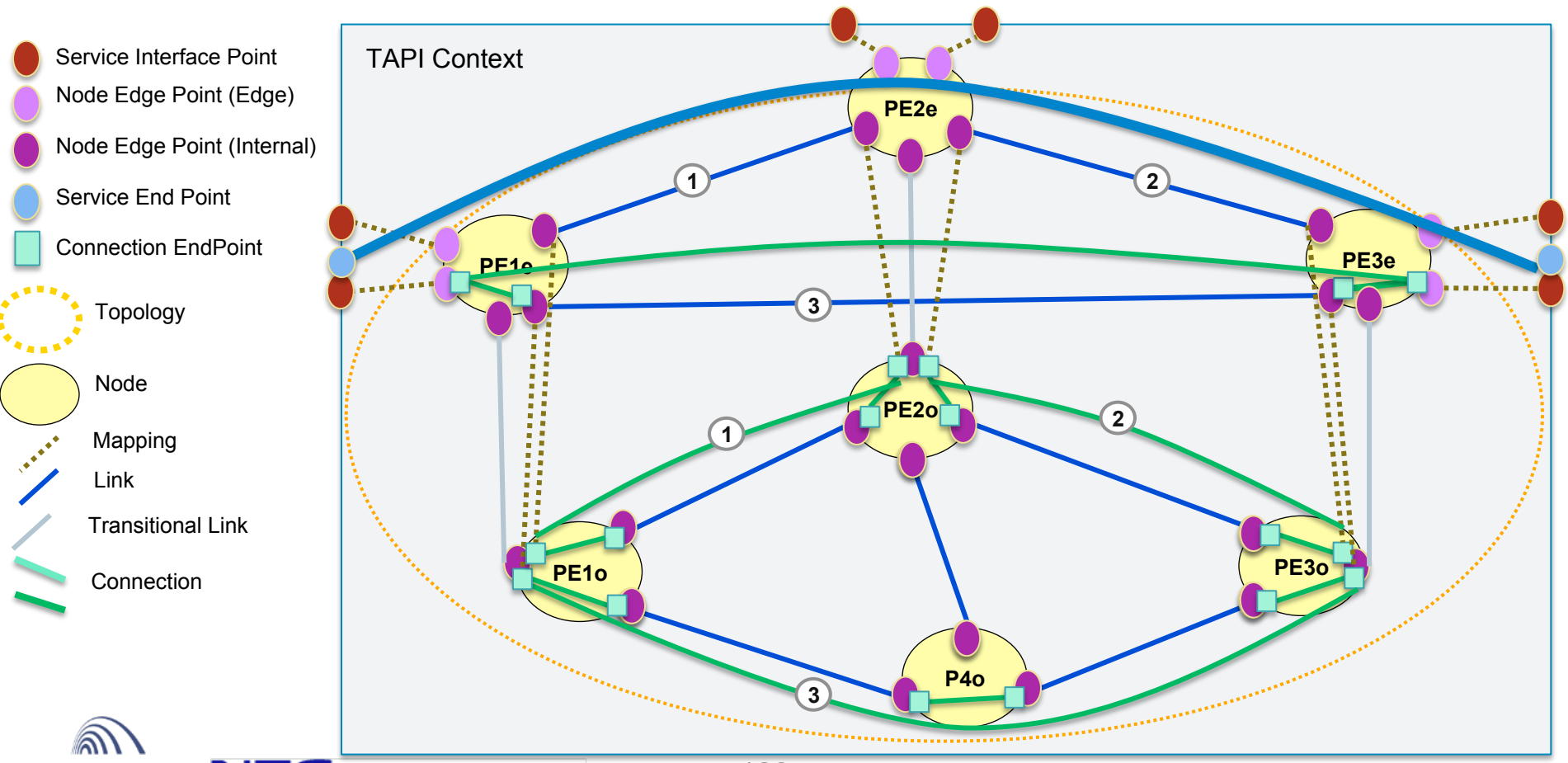
Client 2 (Green) Shared Context: Multi-Node Topology

- Multiple *Nodes* (PEs) *Topology* example
- *Node* and its *NodeEdgePoints* provide reasonable information of their capabilities
- *ConnectivityService* can be requested between *ServiceInterfacePoints*
- Top-level *Connection* is recursively decomposed into lower-level *Connections*, 1 per Node
- *Connection* route can be traced over the exposed *Topology*



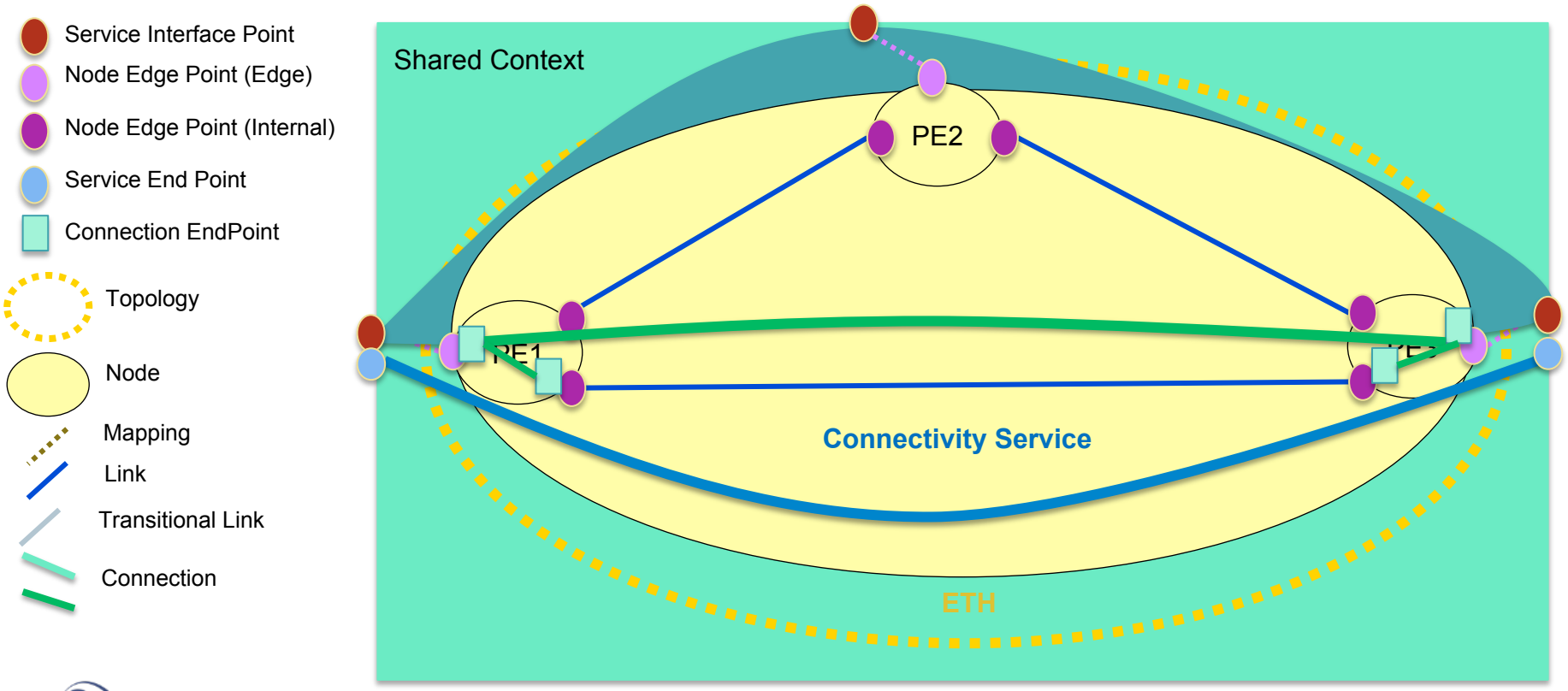
Admin (Blue) Shared Context: Multi-layer Topology

- Each physical device is represented by a separate *Node* per supported layer (ETH & ODU)
- *Node* and its *NodeEdgePoints* provide information of their capabilities at that layer
- *Transitional Links* interconnect the *NodeEdgePoints* at different layers
- Top-level *Connection* is recursively decomposed into lower-level *Connections*, 1 per Node
- Top-level *Connections* at lower (server) layer result in *Links* at upper (client) layer



Client-1 (Green) Shared Context: Virtual Network

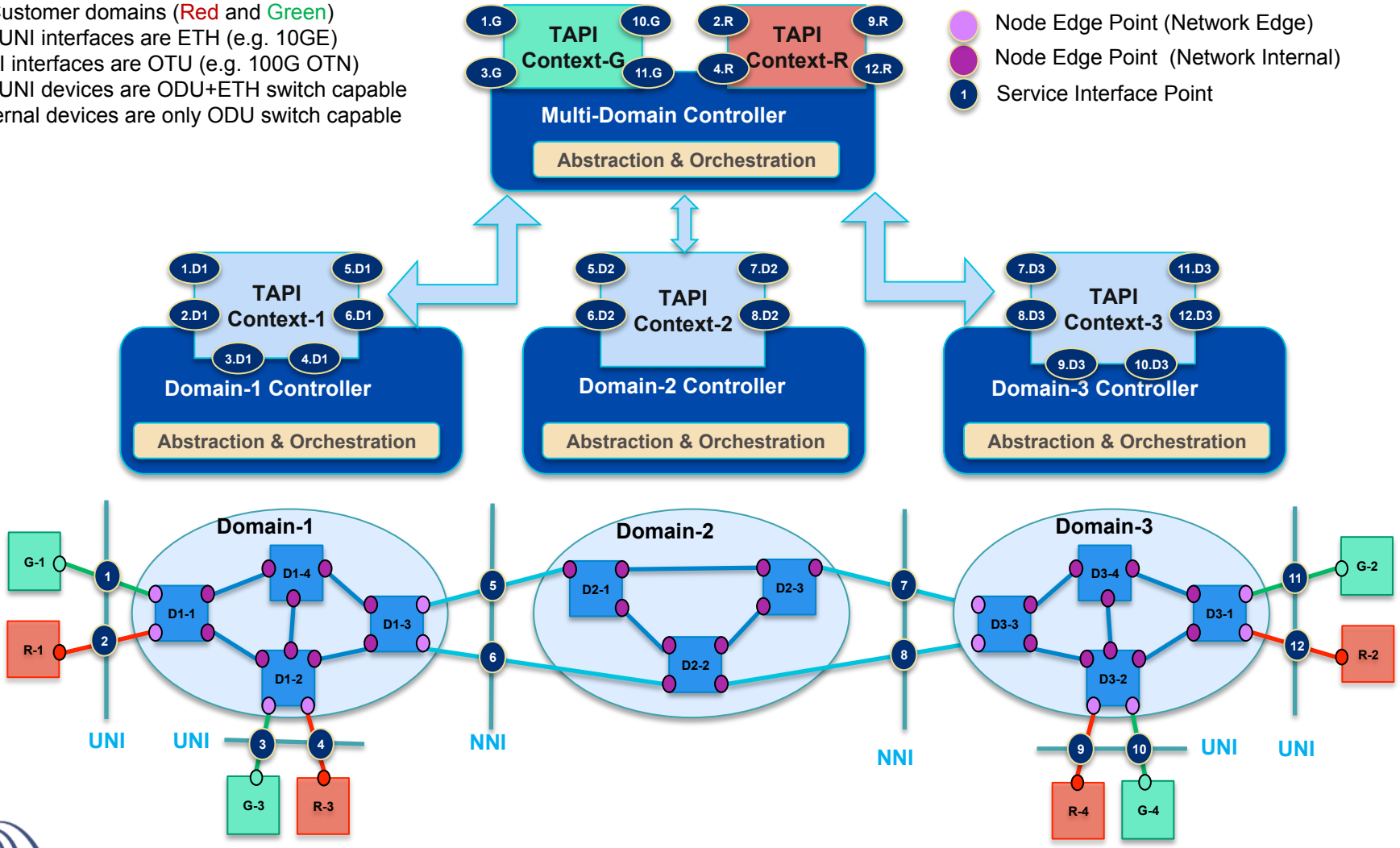
- Client requests VN *Topology* between *ServiceInterfacePoints* within its Shared *Context*
- Provider creates a *Topology* based on the input Topology constraints (e.g. traffic matrix)
- Single *Node* or Multiple *Node* abstraction of *Topology*
- *Node* and its *NodeEdgePoints* provide some approximation of the network capabilities
- *ConnectivityService* can be requested between *ServiceInterfacePoints*



EXTRA: TAPI MULTI-DOMAIN CONCEPTS & HIERARCHICAL CONTROL EXAMPLE

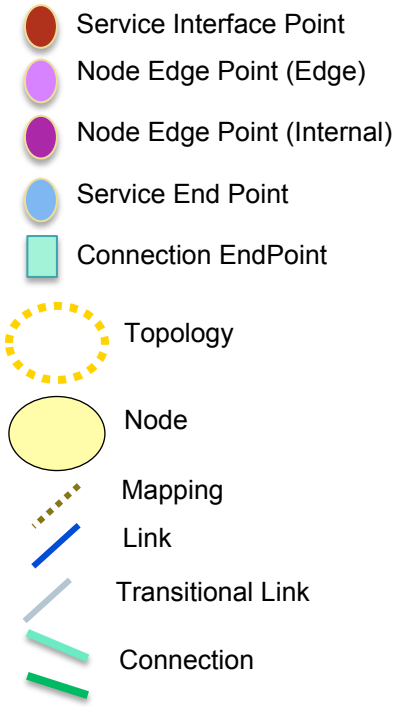
TAPI Reference Hierarchical Control Example

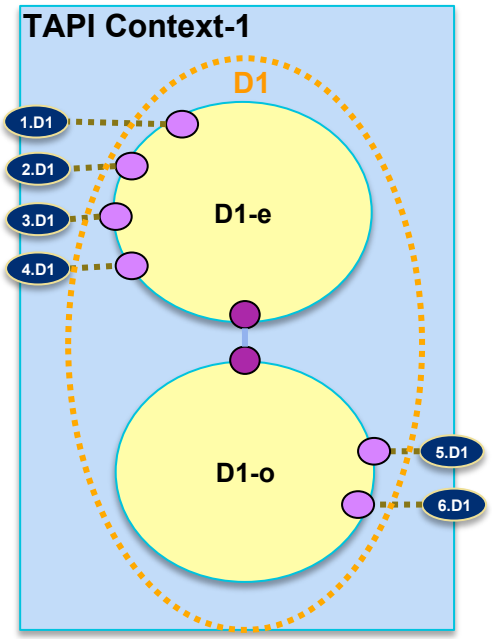
- 3 Provider admin domains (Blue)
- 2 Customer domains (Red and Green)
- All UNI interfaces are ETH (e.g. 10GE)
- NNI interfaces are OTU (e.g. 100G OTN)
- All UNI devices are ODU+ETH switch capable
- Internal devices are only ODU switch capable

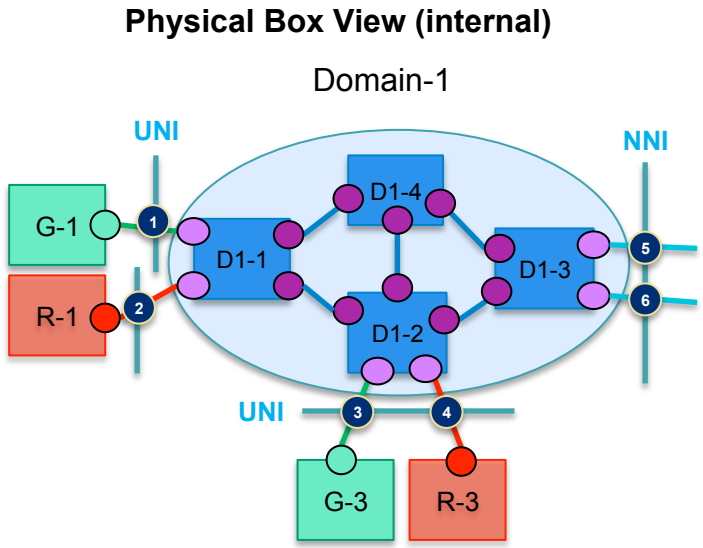


Domain-1 TAPI Context Topology

- This slide depicts the complete Topology exposed by domain-controller 1 to the multi-domain-controller*
- It is assumed that the exposed TAPI-Context Topology is a 2 *Node* abstraction (1 per layer) of domain-controller ‘s network
- It is assumed that no Connectivity has been setup in the entire domain and this is the initial Topology view



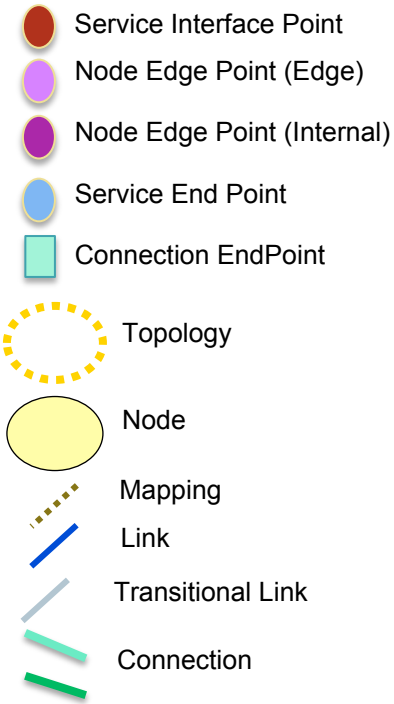


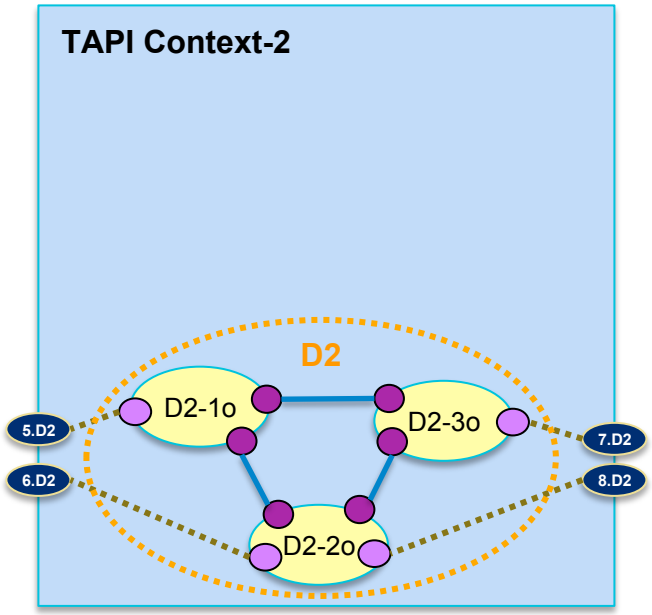


*The multi-domain controller may have to invoke more than one retrieval API operation to get this view

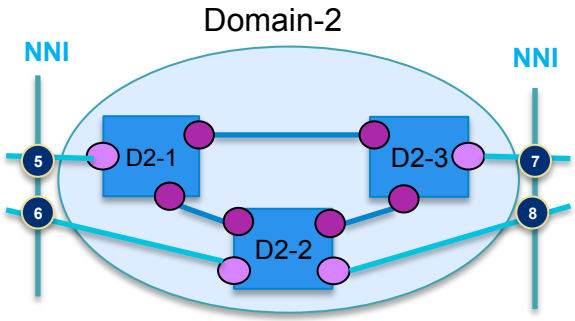
Domain-2 TAPI Context Topology

- This slide depicts the complete Topology exposed by domain-controller 2 to the multi-domain-controller*
- It is assumed that the exposed TAPI Context Topology contains one Node per device in the domain controllers' network.
- It is assumed that no Connectivity has been setup in the entire domain and this is the initial Topology view





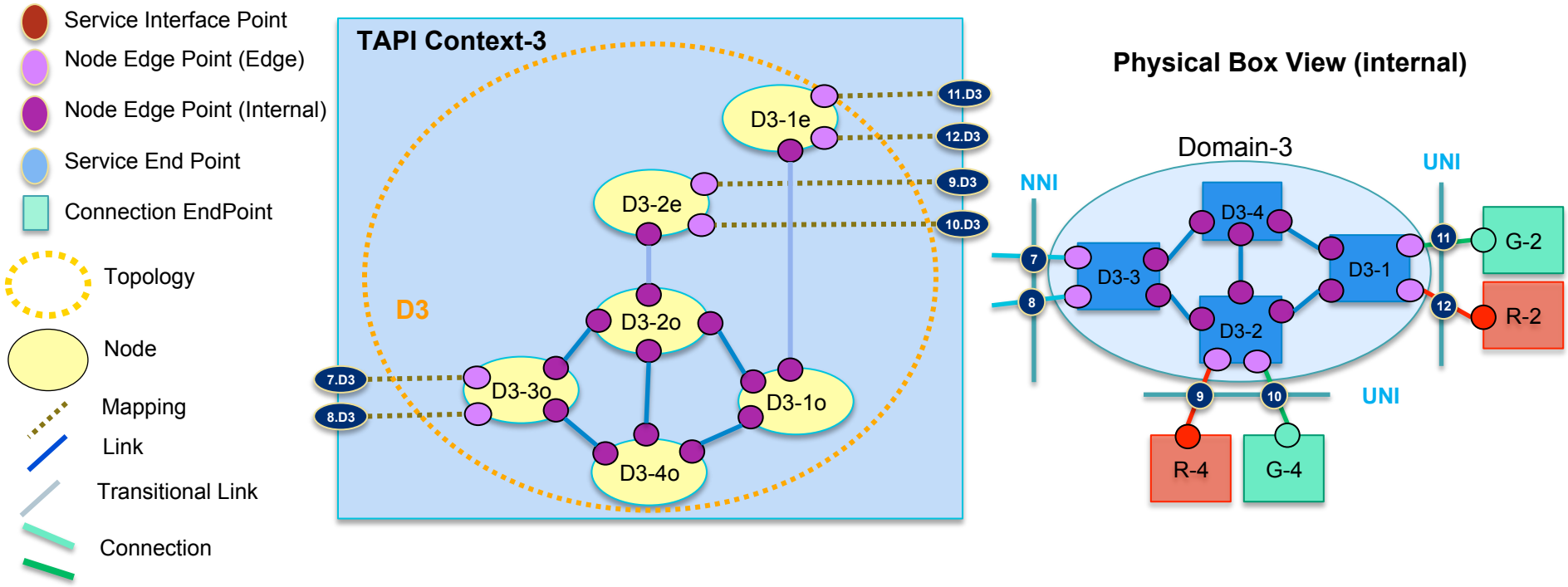
Physical Box View (internal)



*The multi-domain controller may have to invoke more than one retrieval API operation to get this view

Domain-3 TAPI Context Topology

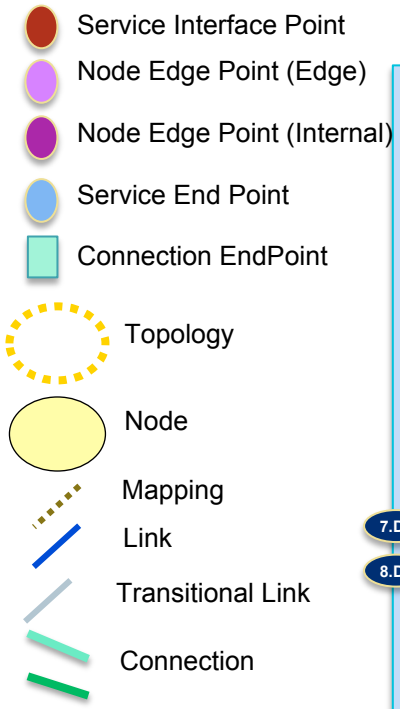
- This slide depicts the complete Topology internal to the multi-domain-controller*
- It is assumed that the exposed TAPI Context Topology contains one Node per layer per device in the domain controller's network
- It is assumed that no Connectivity has been setup in the entire domain and this is the initial Topology view

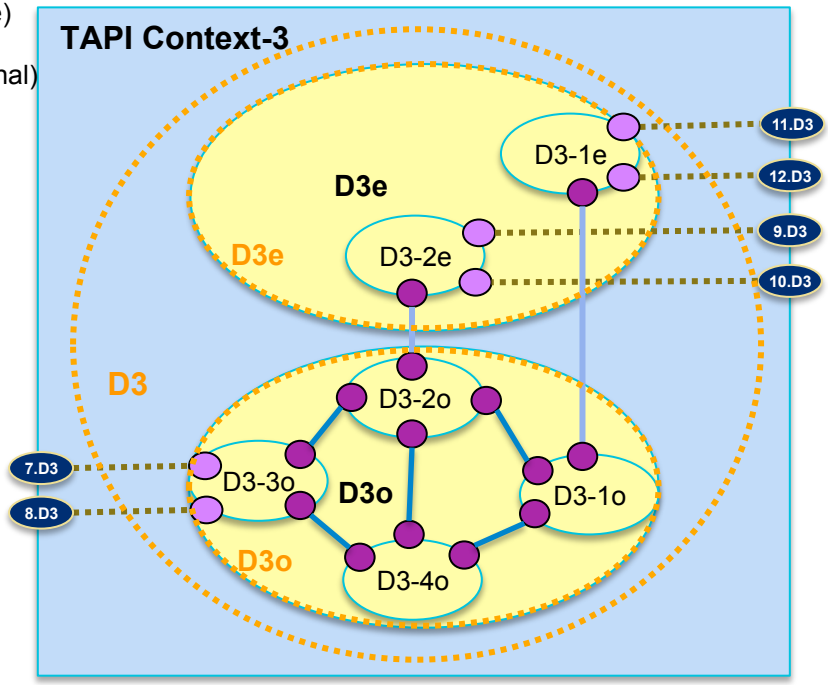


*The multi-domain controller may have to invoke more than one retrieval API operation to get this view

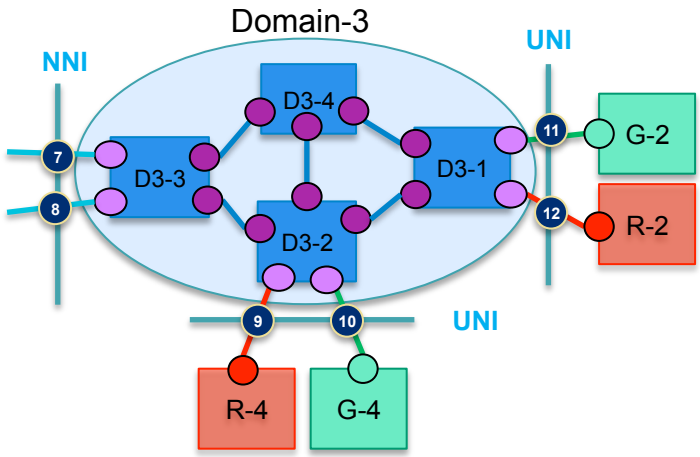
Domain-3 TAPI Context Hierarchical Topology

- This slide depicts complete Topology exposed to multi domain-controller as 2-Level hierarchy*
  - It is assumed that the exposed TAPI Context top-level Topology is an 2 *Node* abstraction of domain-controller's internal Context
  - This view is constructed by recursively traversing/expanding the internal Topology of top-level Nodes
- It is assumed that no Connectivity has been setup in the entire domain and this is the initial Topology view





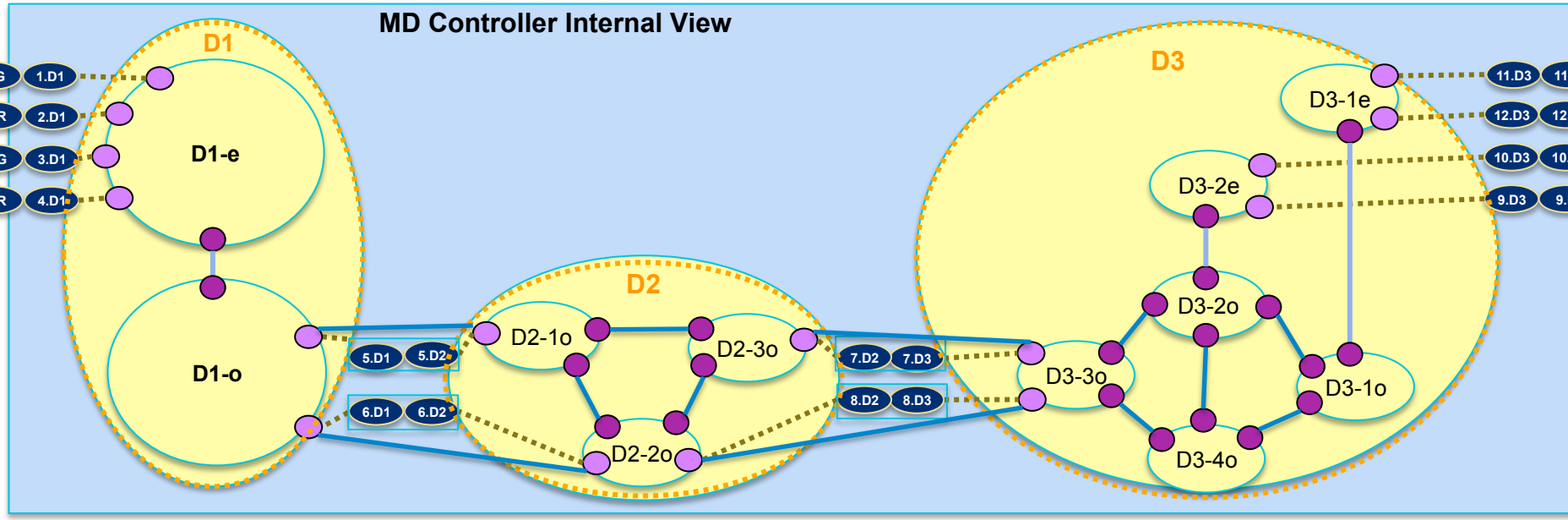
Physical Box View (internal)



*The multi-domain controller may have to invoke more than one retrieval API operation to get this view

MD Controller Local Resource Pool (Internal)

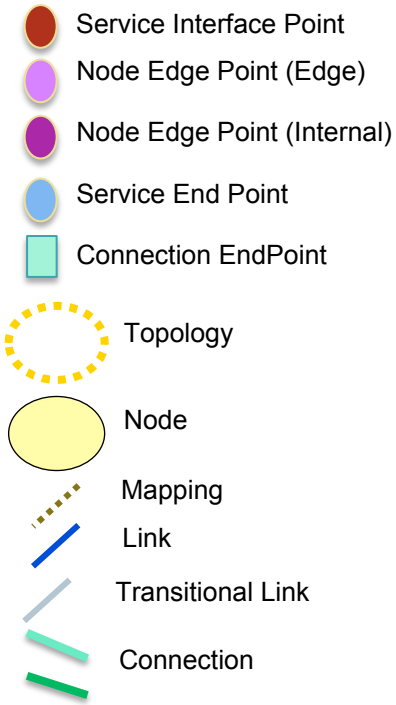
- This slide depicts the complete Topology internal to the multi-domain-controller in terms of TAPI-like constructs*
  - This view is not exposed to the applications over TAPI.
  - The abstraction mapping is maintained internally by the MD domain controller
  - The MD controller is responsible for matching-up and mapping service-interface-points of different domains
- It is assumed that no Connectivity has been setup in the entire domain and this is the initial internal Topology view

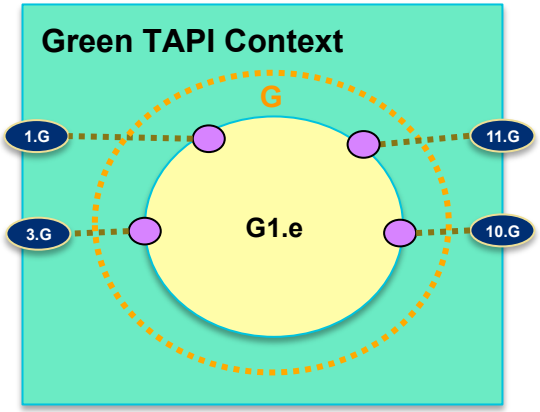


* An controller internal model may not be based on TAPI

Green TAPI Context Topology: single-layer, single Node

- This slide depicts the complete Topology exposed by the multi-domain-controller to the Green application
- It is assumed that exposed TAPI-Context Topology is a single *Node* abstraction of multi-domain-controller 's internal Context
- It is assumed that this is an ETH layer Topology
  - The *LayerProtocol* of *NodeEdgePoints* contain attributes specified by the Ethernet extension schema
- It is also possible that no layer information is exposed and layer specific information is implicitly known to client and provider
- It is assumed that no Connectivity has been setup in the entire domain and this is the initial Topology view

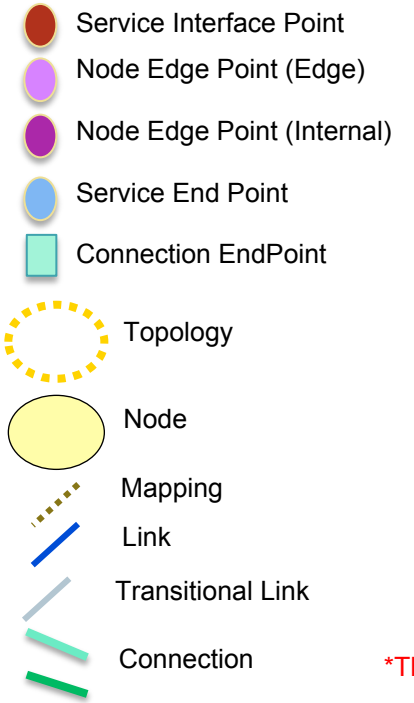


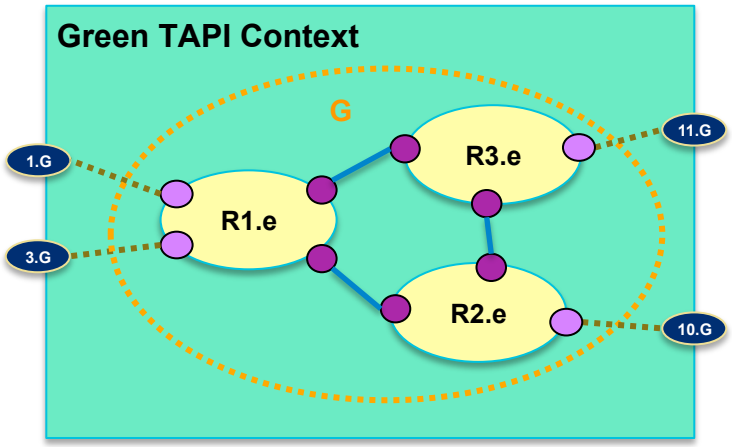


*The application may have to invoke more than one retrieval API operation to get this view

Green TAPI Context Topology: single layer, edge-Node

- This slide depicts an alternate Topology exposed by the multi-domain-controller to the Green application
- It is assumed that exposed TAPI-Context Topology is an edge-*Node* abstraction of multi-domain-controller 's internal Context
- It is assumed that this is an ETH layer Topology
  - The *LayerProtocol* of *NodeEdgePoints* contain attributes specified by the Ethernet extension schema
- It is assumed that no Connectivity has been setup in the entire domain and this is the initial Topology view
  - The ETH layer *Links* are an abstraction of the potential connectivity at this layer
  - TAPI Capacity pac is used to represent all the potential, provisioned and available capacity information

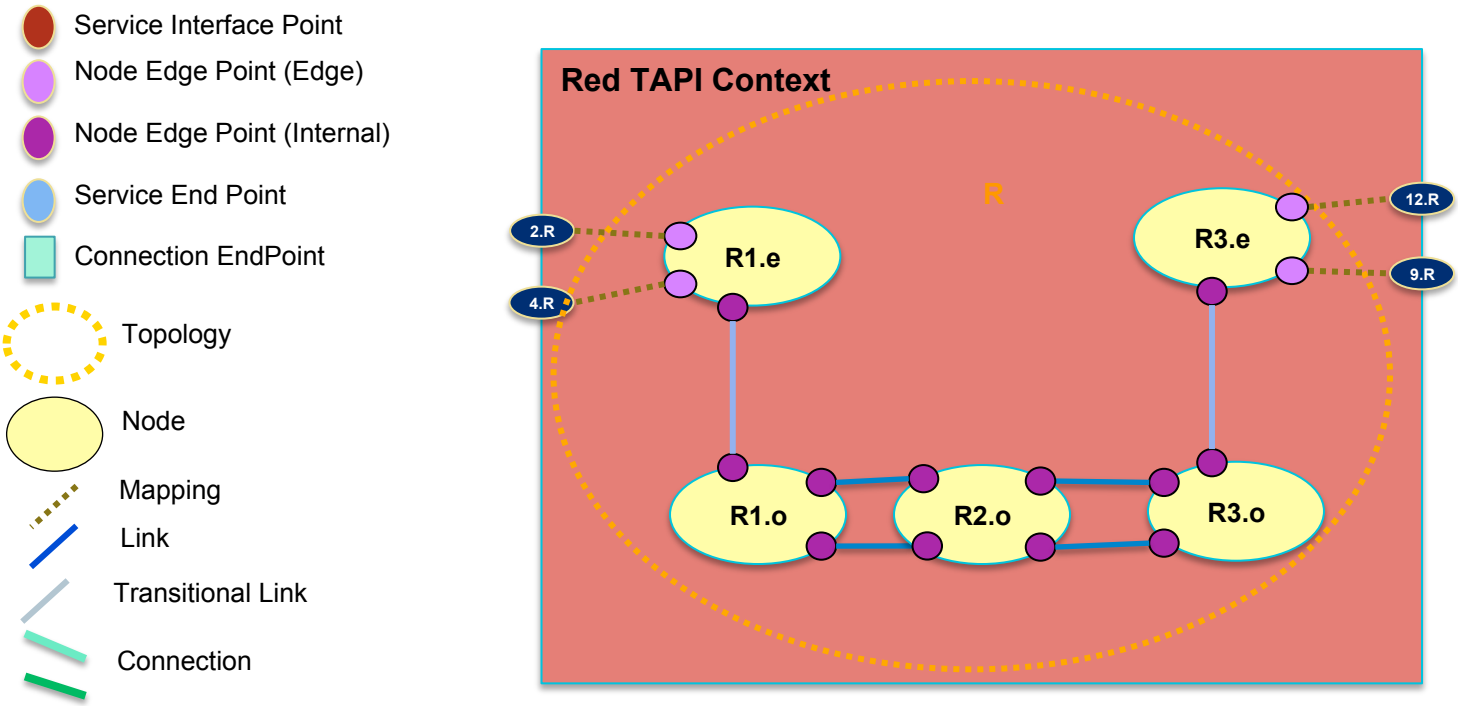




*The application may have to invoke more than one retrieval API operation to get this view

Red TAPI Context Topology: multi-layer

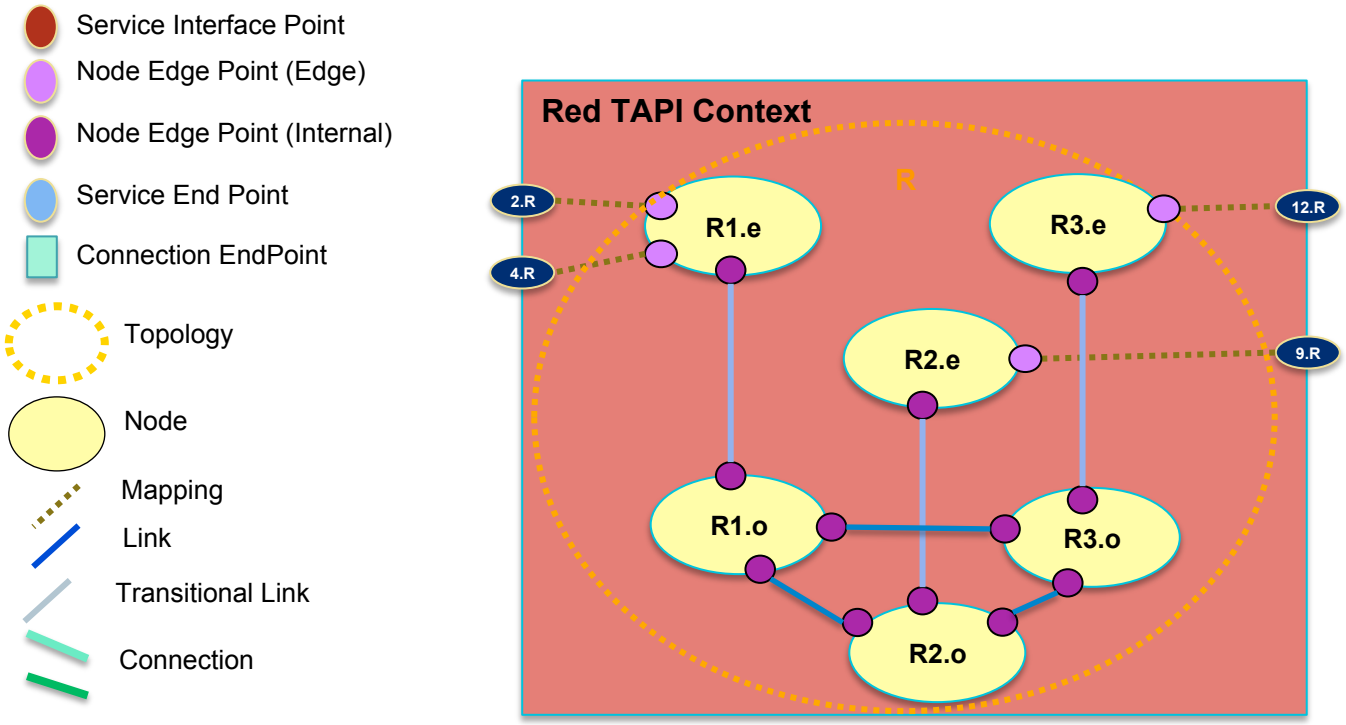
- This slide depicts a multilayer Topology exposed by the multi-domain-controller to the Red application
- It is assumed that exposed TAPI-Context Topology is per-layer-*Node* abstraction of multi-domain-controller's internal Context
- It is assumed that this is an ETH + ODU layer Topology
- It is assumed that no Connectivity has been setup in the entire domain and this is the initial Topology view
  - Initially there no *Links* in the ETH layer
  - The ODU layer *Links* are an abstraction of the potential connectivity at this layer



*The application may have to invoke more than one retrieval API operation to get this view

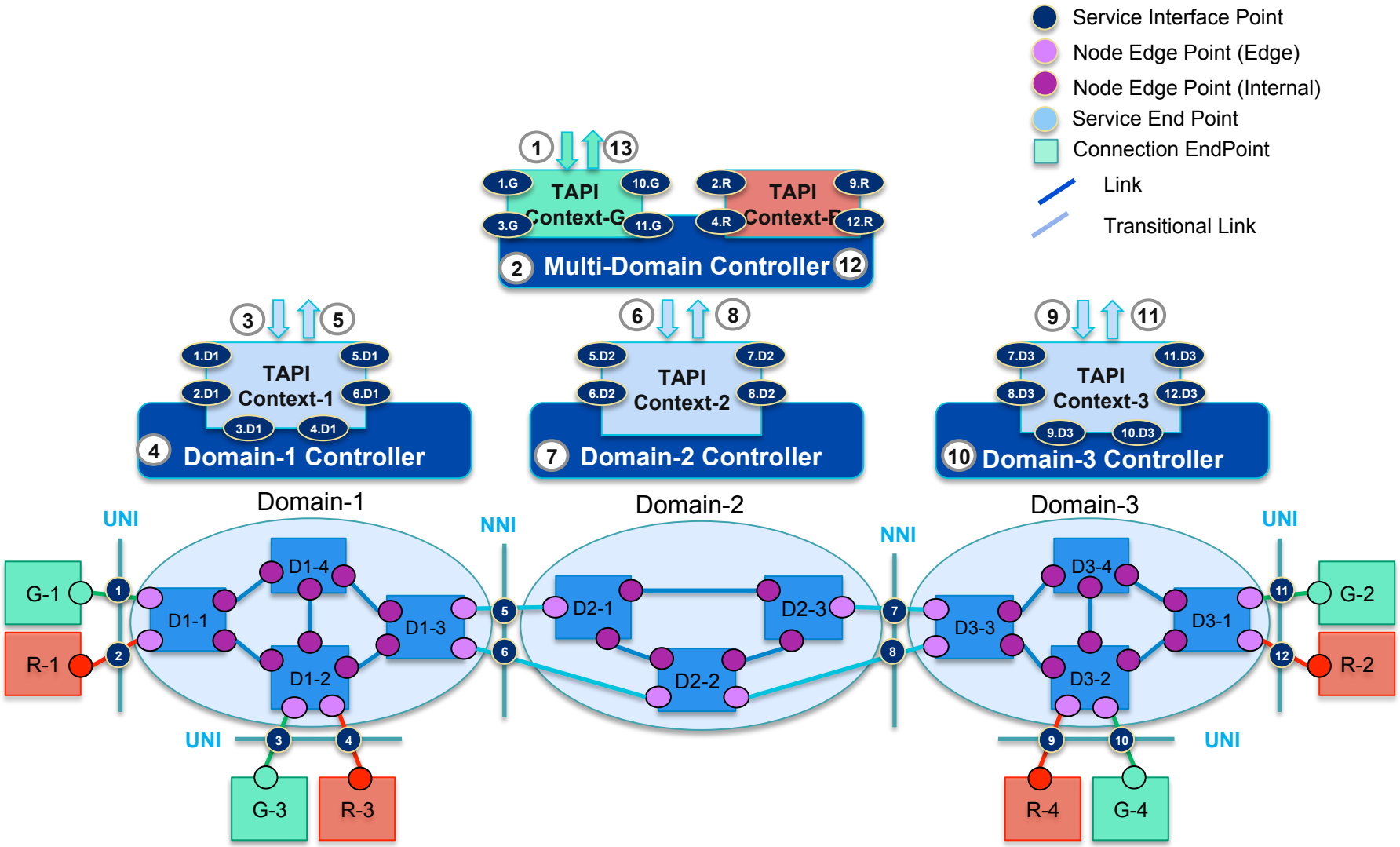
Red TAPI Context Topology: multi-layer

- This slide depicts an alternate Topology exposed by the multi-domain-controller to the Red application
- It is assumed that exposed TAPI-Context Topology is an edge-*Node* abstraction of multi-domain-controller 's internal Context
- It is assumed that this is an ETH + ODU layer Topology
- It is assumed that no Connectivity has been setup in the entire domain and this is the initial Topology view
  - Initially there no *Links* in the ETH layer
  - The ODU layer *Links* are an abstraction of the potential connectivity at this layer



*The application may have to invoke more than one retrieval API operation to get this view

Service Example 10G EPL



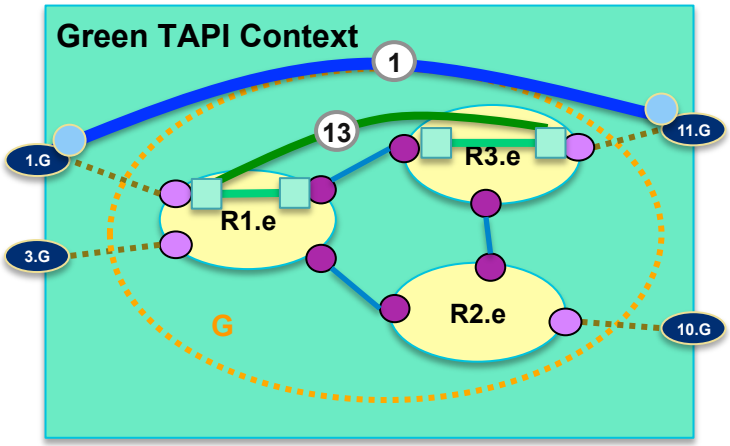
10G EPL Service Setup Sequence

parallel requests

1. Green-app requests 10G EPL *ConnectivityService* between SEPs (1.G, 11.G)
2. MD-controller computes & provisions E2E connectivity within its internal topology
3. MD-controller requests 10G *ConnectivityService* between SEPs (1.D1, 5.D1)
   - Domain1-controller computes & provisions ODU+ETH *Connections* in its domain
4. Domain1-controller returns provisioned *Connections* in terms of Context-1 topology
5. MD-controller requests 10G *ConnectivityService* between SEPs (5.D2, 7.D2)
   - Domain2-controller computes & provisions ODU *Connections* in its domain
6. Domain2-controller returns provisioned *Connections* in terms of Context-2 topology
7. MD-controller requests 10G *ConnectivityService* between SEPs (7.D3, 11.D3)
   - Domain3-controller computes & provisions ODU+ETH *Connections* in its domain
8. Domain3-controller returns provisioned *Connections* in terms of Context-3 topology
9. MD Controller updates its internal Topology and resource pool capacity/usage metrics
10. MD-controller returns provisioned *Connections* in terms of Context-G topology
    - MD-controller updates capacity/usage metrics in Context-G topology
11.
12.
13.

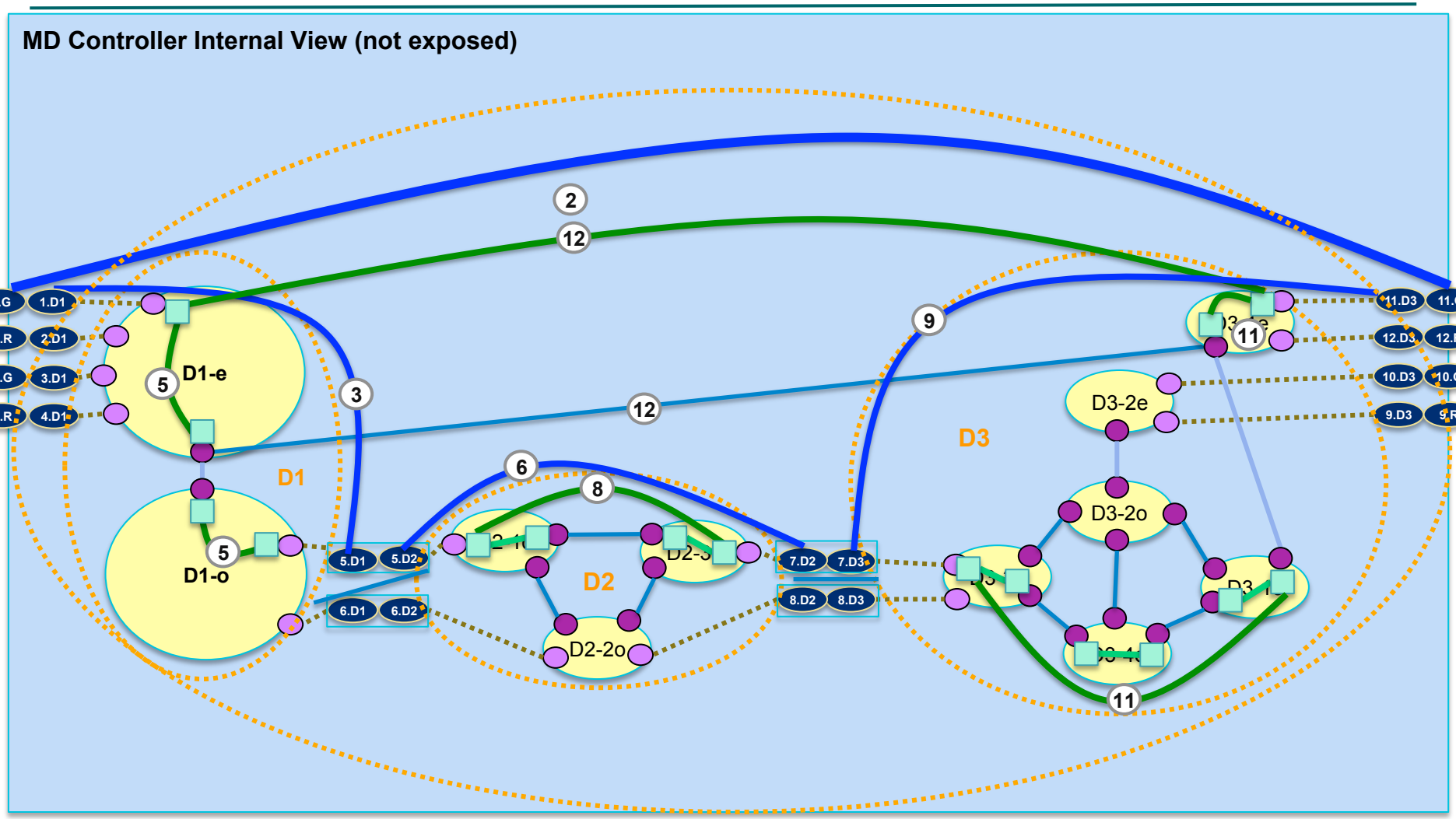
10G EPL Service setup: Green Context

- Service Interface Point
- Node Edge Point (Edge)
- Node Edge Point (Internal)
- Service End Point
- Connection EndPoint
- Topology
- Node
- Mapping
- Link
- Transitional Link
- Connection
- Connectivity Service



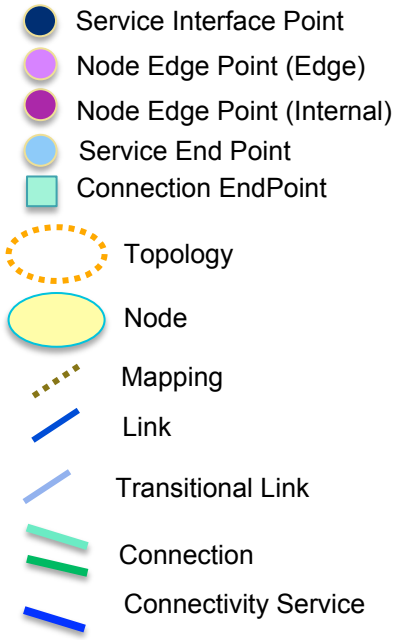
*The application may have to invoke more than one retrieval API operation to get this view

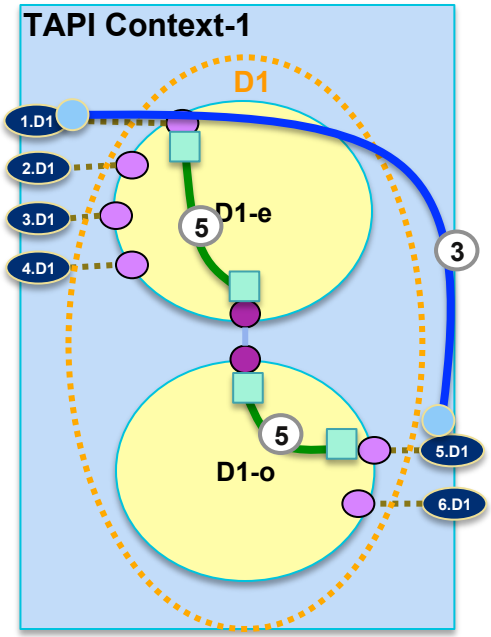
10G EPL Service setup: MD Controller Internal view

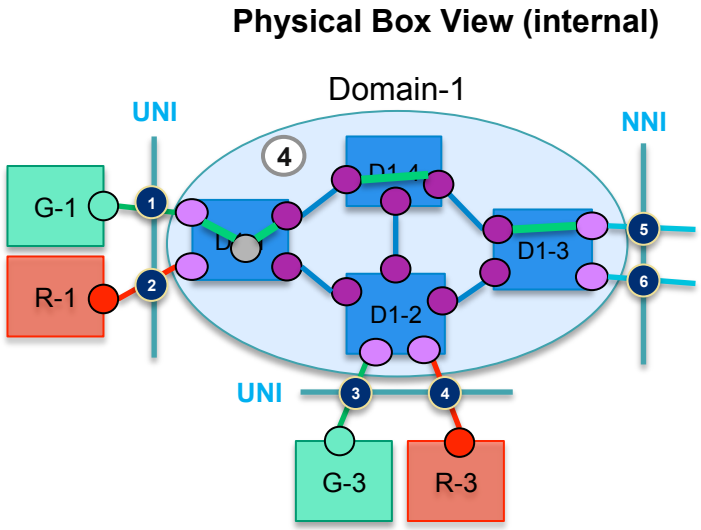


* An controller internal model may not be based on TAPI

10G EPL Service setup: DC1/Context1 view

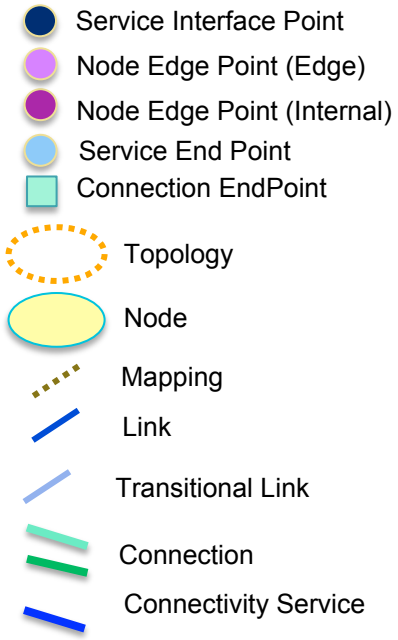


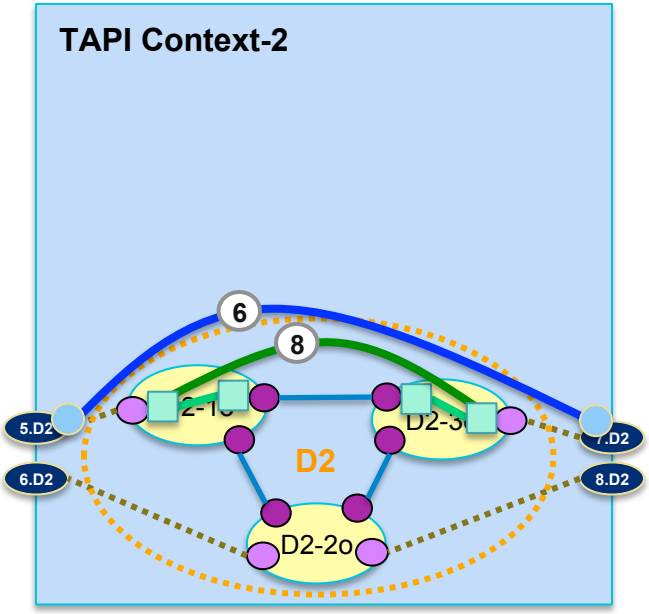




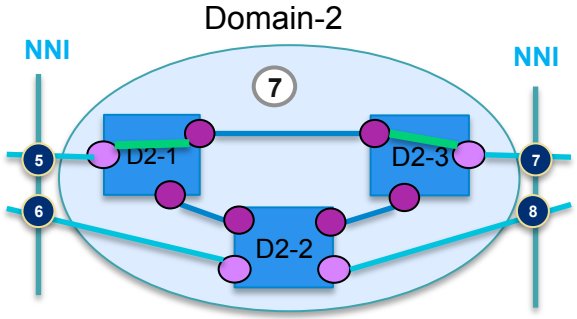
*The multi-domain controller may have to invoke more than one retrieval API operation to get this view

10G EPL Service setup: DC2/Context2 view



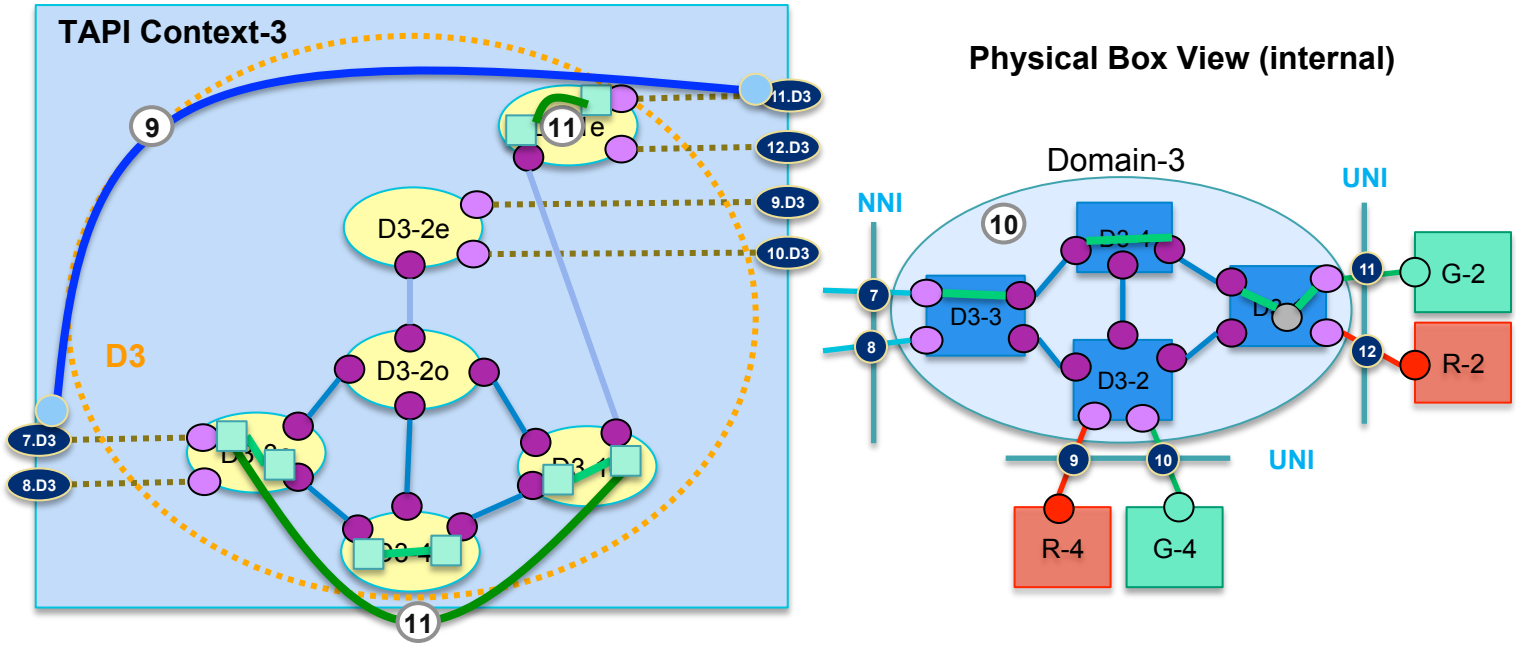


Physical Box View (internal)



*The multi-domain controller may have to invoke more than one retrieval API operation to get this view

10G EPL Service setup: DC3/Context3 view



*The multi-domain controller may have to invoke more than one retrieval API operation to get this view