ONAP
OPEN NETWORK AUTOMATION PLATFORM

ONAP R4+ Architecture Update

Architecture Subcommittee (ARC) Presentation

Parviz Yegani – Tiger Team Report

October 9, 2018

R4+ Architecture Update

1. **Generic NF Controller Architecture (GNFC)**
   - **Mapping of interfaces for Dublin and beyond (R4+): work in progress.**
   - **Agreed to remove all references to VF-C for now.**
   - **Contributions needed to address GNFC and VF-C alignment, targeted for Dublin and beyond.**
2. **SDN-R (SDN-C) path to GNFC**
   - **R4+ Architecture update to support SDN-R - work in progress.**
3. **ONAP SDK-Driven Sub-System Approach – SDK Libraries**
   - **Work in progress.**
4. **Recursive Service Orchestration (Gil, AT&T)**
   - **Defined all internal and external interfaces plus API Mapping (OIs/OEs).**
   - **Details are captured in subsequent slides.**
5. **Domain Orchestrator (Abinash, Netcracker)**
   - **See next few slides for details.**
6. **ONAP Modularization (Functional Decomposition) – work in progress**
   - **Functional decomposition based on domain capabilities at different layers – This might be required as a long term plan but we also need to address how we can expose well-defined APIs (standard APIs if possible).**
   - **Plan is to prepare a draft architecture contribution for the upcoming ARC F2F meeting in Montreal**

Topic #4: Recursive Service Orchestration (1/6)

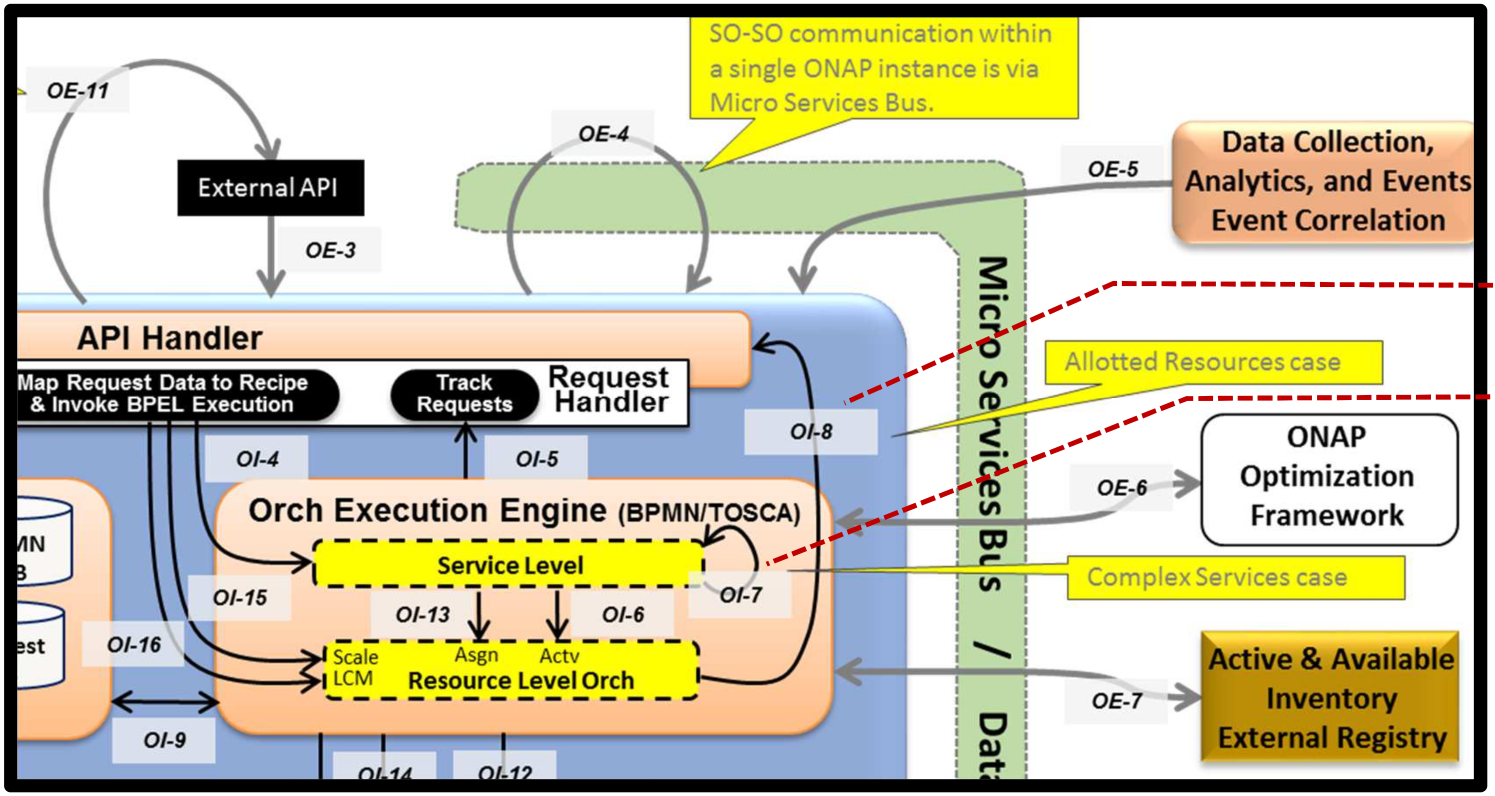
Recursive Service Orchestration in ONAP (Gil)

- **First part of the Gil's original deck (slides 1-43):**
  - **Background materials on "ONAP Orchestrator Functions - internal structure, services & resources, NFs, interfaces/APIs, service level SLOs, etc"**
  - **SDC Modeling Tool for Service Designer (service & resource level actors)**
  - **Network Functions (VNFs/PNFs) as a Service Versus Allotment**
  - **Decomposition/Homing/Instantiation (sequence diagrams)**
  - **Two examples were considered: Simple Service (example 1) and Complex/Nested Service (example 2)**
  - **Modeling approaches - for each example service scenario two modeling approaches have been proposed:**
    - Modeling Approach A – direct reference from higher-order service to lower-order service

      (the resource controller in this case cannot make "network assignments" for the lower-order service in the context of the higher-order service),
    - Modeling Approach B – indirect reference from higher-order service to lower-order service through a "Façade" object,

      (Facade makes lower-order service appear as a resource to the higher-order service. This includes the presence of an SDNC to perform assignments for the Façade resource.)
  - **The design details for nested services approach A (service level flows for decomposition/homing/instantiation and service policy considerations) was presented last month.**
- **Second part of the deck (slides 44-60) focused on Façade resources using approach B where services have resources only.**

Orchestrator Functional Internal View (2/6)



SO-SO communication within a single ONAP instance is via Micro Services Bus.

Allotted Resources case

Complex Services case

OE-11
External API
OE-3
OE-4
OE-5
Data Collection, Analytics, and Events Event Correlation
Micro Services Bus / Data
API Handler
Map Request Data to Recipe & Invoke BPEL Execution
Track Requests
Request Handler
OI-8
OI-4
OI-5
Orch Execution Engine (BPMN/TOSCA)
Service Level
OI-7
OI-15
OI-13
OI-6
OI-16
Scale LCM
Asgn
Actv
Resource Level Orch
OI-9
OI-14
OI-12
OE-6
ONAP Optimization Framework
OE-7
Active & Available Inventory External Registry

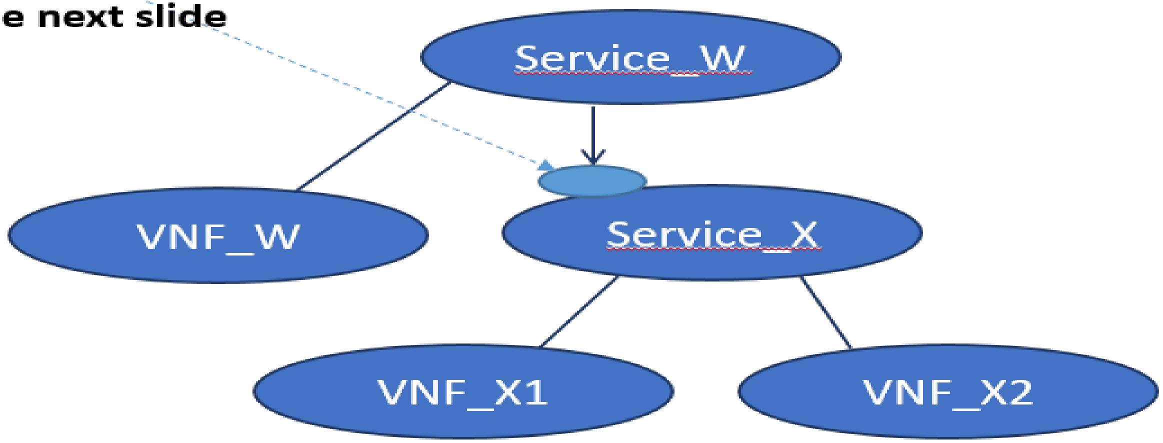
Use of the **Façade Resource approach** would eliminate OI-7 in lieu of using OI-8 for both Allotted Resources and Complex Services

Service Orchestration (3/6)

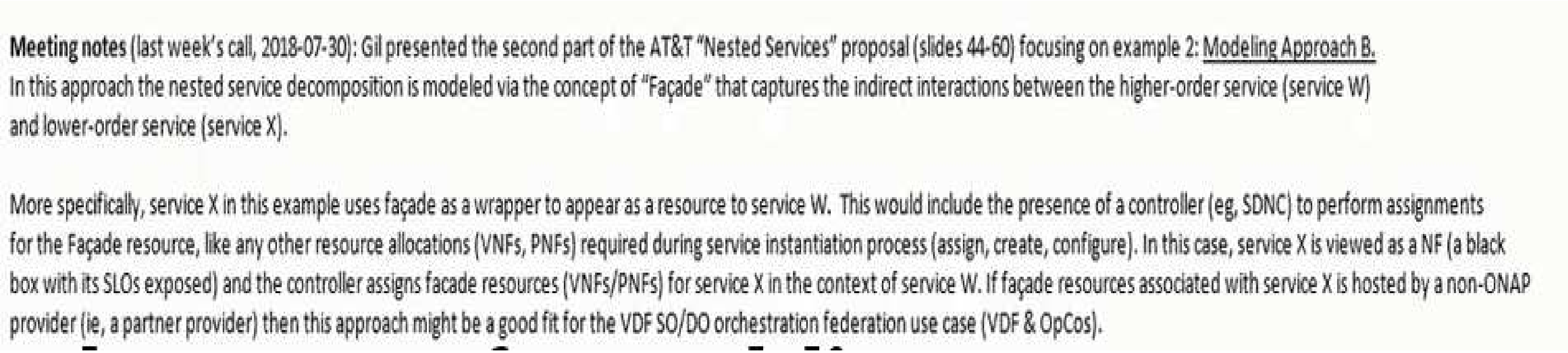
- Example 2: Modeling Approach B
- "Services Have Resources" (Only)

- **Indirect Reference from Higher-Order Service to Lower-Order Service Through a "Façade" Object, Which Makes Service_X Appear as a Resource to the Higher-Order Service W.**
- **This includes the presence of an SDNC to perform assignments for the Façade Resource.**

See next slide



Nested Services w/ Façade Resources (4/6)



Meeting notes (last week's call, 2018-07-30): Gil presented the second part of the AT&T "Nested Services" proposal (slides 44-60) focusing on example 2: Modeling Approach B. In this approach the nested service decomposition is modeled via the concept of "Façade" that captures the indirect interactions between the higher-order service (service W) and lower-order service (service X).

More specifically, service X in this example uses façade as a wrapper to appear as a resource to service W. This would include the presence of a controller (eg, SDNC) to perform assignments for the Façade resource, like any other resource allocations (VNFs, PNFs) required during service instantiation process (assign, create, configure). In this case, service X is viewed as a NF (a black box with its SLOs exposed) and the controller assigns facade resources (VNFs/PNFs) for service X in the context of service W. If façade resources associated with service X is hosted by a non-ONAP provider (ie, a partner provider) then this approach might be a good fit for the VDF SO/DO orchestration federation use case (VDF & OpCos).

The scope for Dublin:

How to implement recursive orchestration for 5G. Agreed to consider the following:

- **The façade resource**
- **Modeling implications**
- **What level of flexibility we should allow for orchestration**
- **Need to define some interfaces for proper interactions between different layers of the hierarchy of nested services**

Modular Orchestration & Homing (5/6)

- **Modular Orchestration and Homing of Complex Services and Allotted Resources**
  - Illustrated via Service Instantiation Examples Using a Separation of Concerns Approach

- **ONAP runtime support of a "Network Service" that has been onboarded into SDC and invoked for instantiation via a SOL005 API**. Work in progress
  - ONAP should provide two Service Provider options with respect to application level configuration on a per-NF/per-Service basis:
  - Option 1: ONAP supports application level configuration of the NF in the context of the Service
  - Option 2: An external OSS/BSS supports this application level configuration
  - To minimize variability to vendors, ONAP should support onboarding of SOL001 VNF Descriptors. For Service Providers who choose to do so, ONAP should also support onboarding of SOL001 Network Service Descriptors
  - ONAP runtime support of an onboarded Network Service Descriptor should minimize changes to a Service Provider's OSS/BSS infrastructure that had been supporting the corresponding "end to end service"
  - ONAP runtime support should allow Service Provider the option to either "plug in" a VNFM or not. The slides on the wiki page provide descriptions of this proposal examples for both cases.
  - To accomplish these needs, the ONAP runtime/internal model need not support a separate "object type" called "Network Service". Rather, the onboarding model of Network Service would be mapped to a standard ONAP "Service" in the internal model. Depending on whether Option 1 or Option 2 is desired, this ONAP "Service" would either be enriched to include application level configuration support, or not.

Orchestrator Issues (6/6)

Other issues raised so far:

- Recursive orchestration – it's scope is bigger than Gil's example discussions with VDF
- Platform enhancements via example use cases
- Globally scalable/deployable
- What SPs need is to deploy ONAP more widely
  - Modularity is a longer term activity
  - Deployability – short term focus
- End user group is being formed (VDF/Vz leading)
- User community need to help
  - Requirements for deployability
  - Sustainability of use cases
  - List of gaps
- Platform enhancements: demonstrated by a given use case
- Use case should expand the feature (business driven) + prioritize the list of features
- Use case has to be additive (no overlapping scope) – need example that can show the gaps

Topic #5: Domain Orchestrator

Scope of Domain Orchestration (DO) work

The DO concept is under discussion (led by Abinash) – areas of focus:

- ONAP Mapping to Domain Orchestrator Concepts:
  - Challenges,
  - Transformation and
  - APIs alignments to Standards
- Potential options for overcoming ONAP Deployment Challenges.
- How DO can be applied to ONAP?
- Identify operators' requirements - Abinash is asking several tier-1 operators about their specific requirements.
- There are different views/thoughts on DO in the community. These views need to be harmonized/consolidated.
- It is good to assess what everyone is trying to achieve first.
- A set of resources to be orchestrated within a region/country or even a city. VDF views this as a more generic set of resources.
- Requirements drive architecture work
- External APIs (external to both ONAP as a whole or ONAP components/projects)

Topic #6: ONAP Modularization - Straw Proposal

Action plan: Continue to use the modularization weekly call to drive the evolution of the strawman proposal. We're hoping to make the proposal ready for the Architecture F2F in Montreal later this month (Oct. 29-31).

The owner(s) of each item (identified below) should have their drafts ready for group's review on a timely basis!!

- **Modularization working assumptions – (Nigel/Dave)**
- **Define managed objects/Model – follow the CLI project approach (Andy/Manoj/Alex)**
- **Define ONAP functionality into functional components (Manoj/Kevin/Steve)**
- **Define a set of operational work flows across ONAP (eg, use Orange slide deck presented in Beijing) (Kevin/Ramki/Alex/Margaret)**
- **Identify gaps between ext/int interfaces & models (Andy)**
- **Refactor based on gaps (all+PTLs). ARC F2F in Montreal**
- **Functional decomposition/modularization strawman proposal**

(target deadline: Oct. 26th , 2018)

Modularity - What do we want to address? (1/5)

General (high-level) Comments

- Instead of using some bullet lists as a guide for the modularity discussions we need to be very specific about the technical details as to how each software component once modified (upgraded to a new version) is going to impact the entire system.
- "Modularity" is a way overloaded term today. There are two (perhaps distinctive) views wrt what modularity means as far as ONAP is concerned:
  - Option A) views ONAP as a common infrastructure using common components such as DB, TOSCA parser, Data Management System, etc.
  - Option B) views ONAP as a set of independent plug & play functional components. (We may want to rephrase option B to focus more on integration of legacy components vs "rip & replace" such components. Integrating an existing component into the platform seems to be a more practical approach that can add tangible values). Of course, "rip & replace" of a target component could remain as a choice but should be considered as the last resort (having a much lower priority than other options).
- Given the above two different views we must first agree to use common terminologies to properly reflect the goals we are trying to achieve. Ie, are we going to use a common TOSCA parser to achieve modularity (option A) or alternatively can we accomplish modularity by unplugging an ONAP component (eg, SO, SDNC, A&AI, ..) and plugging in an orchestrator/SDNC/etc of operator's liking (option B)? This is very critical! Whether these two views are mutually exclusive or not is FFS.

Modularity - What do we want to address? (2/5)

Technical Comments

- Need to avoid duplicated effort to solve the same problem (eg, having multiple TOSCA parsers in ONAP adds unnecessary complexity).
  - Can't we redesign the ONAP system having a single (common) parser? If so, what would be the implications of this change on other ONAP components, the ones that rely on parser's functionality?
  - Using a single (common) TOSCA parser would help eliminate any incompatibility issues. This may have to do more with ONAP commonality than modularity.
  - Rather than analyzing various parser implementation options for a much wider deployment scenarios we may want to identify the low-hanging fruit (at least as part of our short-term focus for the next release or so).
- Pending question: What are the key ONAP components (from operators' point of view) that need to be modularized? Tentatively agreed to start with SO (led by Seshu).
- Modularization of other components will happen over time once there is enough interest from community and proper resources are committed from participating projects/PTLs.

Modularity - What do we want to address? (3/5)

Technical Comments (cont'd)

- Different versions of the same imported component
  - An example imported component could be DB but we should standardize many common components within the framework.
  - What version (or a range of versions) of a software package we want to use and have a clear understanding of what impact of an upgraded version is going to have on the entire system.
  - Eg, if we're going to upgrade the same tosca parser to the next version and there are multiple components already using it (as a common service). How is this new version going to impact the functionality of the other components? This has to be coordinated carefully across all affected components so that we can eliminate any impact on the entire ONAP system.
  - This is true for the DB as well. OOM is using DBaaS which will follow a similar logic wrt version upgrade. Each component uses its own DB where it is well-coordinated with other components through DBaaS as a common service.
  - If tosca parser is upgraded to a new version and impacts multiple components, are we going to force all these components to be upgraded in one release, multiple releases? How? This is FFS.
- The work on interface versioning is already in progress in ONAP. The discussion we're having here on versioning of software components would complement that work.

Modularity - What do we want to address? (4/5)

Technical Comments (cont’d)

- Enabling differentiators (operator, vendors)
  - This item seems to be more related to the two views (option A vs option B) mentioned earlier. However, we should focus more on the scope than the granularity per se.
  - Identifying (over time) the interface that are demarcation lines.
    - Maybe internal to projects as well.
- Platform extensibility (innovative idea)
  - We may want to address the platform extensibility by thinking of a new concept - dealing with two types of functionally-equivalent Microservices implementations (MS): a **model-driven MS** vs a **custom-built MS**.
    - You can design a service via SDC using a well-defined model and the model-driven MS would do a good job to perform the desired function.
    - Due to the limitations of any model there could be a situation where an operator encounters a new service type and would want to replace this model-driven MS with a functionally-equivalent but custom-built MS which complies with the APIs. How this option should be supported/implemented in ONAP is for further study.

Modularity - What do we want to address? (5/5)

FFS

- **Service creation modelling tool unification:**
  - Are we going to use a single parser (based on TOSCA) or we will need to continue to use other DM modeling tools like YANG, HEAT (eg, for modeling the NSD template)?
  - Common look and feel, design rules, role-based, etc. across all the modules.
  - Also ensure that the entire service design lifecycle works (design time / run time).
- **Domain driven orchestration/controller that will drive unified data models per domain to reduce integration costs.**
  - General domain specific down to vendor specific.
  - Still model-driven.
- **Integration ease:** Reduce the complexity of the pairwise integration
- **Other topics that came up during the discussion:**
  - Consistency on the external APIs and fragmented exposure

TT Recommendations for Dublin Architecture

Tiger Team recommendations for the Montreal F2F meeting (OCT 29-31):

- Focus on actionable recommendations that
  - are achievable in Dublin and
  - enjoy or are likely to enjoy widespread support in the community,
- Add/amend any agreements from the community
- Suggest any features/requirements that should be dropped because
  - they're too vague,
  - they're too ambitious for Dublin, or
  - will create significant pushback.

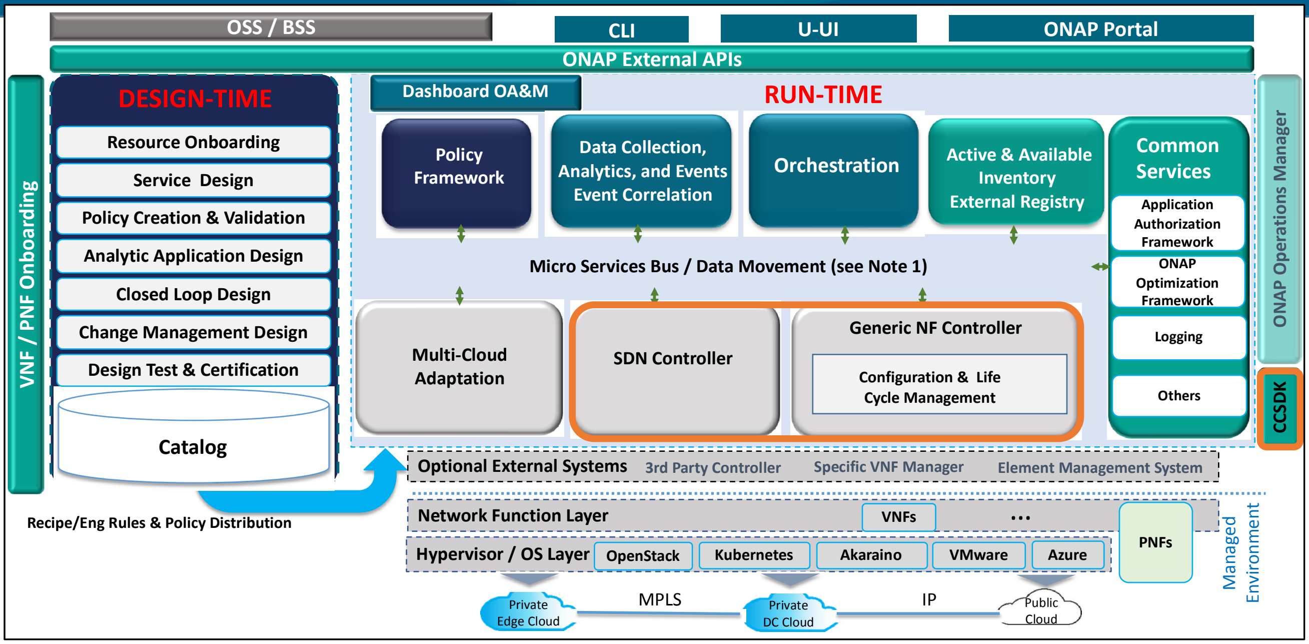
ONAP

OPEN NETWORK AUTOMATION PLATFORM

R4+ Architecture Slides

ONAP Target Architecture

(High-Level Functional View)



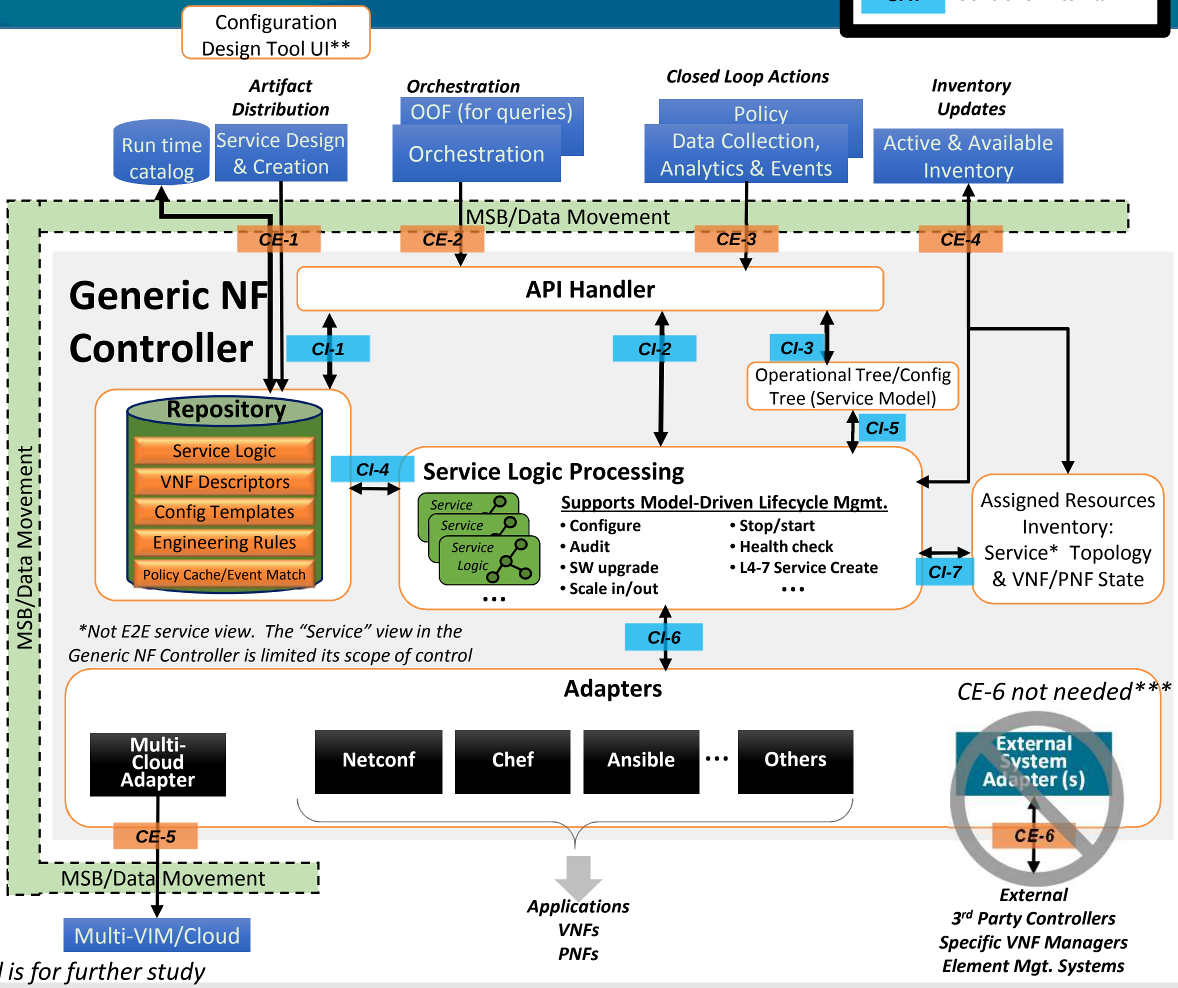
Key Takeaways:

- Today, ONAP Architecture has the Controller Layer with 2 controllers – SDNC and GNFC. We need to consolidate into a single generic NF controller – namely GNFC.
- SDNC, and APPC ~~and VF-C~~ should evolve to GNFC
- SDNC, and APPC ~~and VF-C~~ should consolidate code into CCSDK as a superset controller library for creating different controller persona instances by Operators

Generic NF Controller Architecture

Key	
CE-x	Controller External API
CI-x	Controller Internal API

- **Generic NF Controller configures and maintains the health of VNFs/PNFs/services* (L1-7) throughout their lifecycle.**
  - **The Lifecycle Management Functions are a normalization of the controller aspects of VF-C and APP-C functions into a common, extensible library**
- **Programmable network application management platform**
  - **Behavior patterns programmed via models and policies**
  - **Standards based models & protocols for multi-vendor implementation**
  - **Extensible SB adapter set including vendor specific VNF-Managers**
  - **Operational control, version management, software updates, etc.**
- **Manages the health of VNFs/PNFs within its scope**
  - **Policy-based optimization to meet SLAs**
  - **Event-based control loop automation to solve local issues near real-time**
- **Local source of truth**
  - **Manages inventory within its scope**
  - **All stages/states of lifecycle**
  - **Configuration audits**
- **Key Attributes of Generic NF Controllers**
  - **Intimate with network protocols**
  - **Manages the state of services**
  - **Provide Deployment Flexibility to meet user scalability / resilience needs**



Configuration Design Tool UI**

Artifact Distribution

Run time catalog

Service Design & Creation

Orchestration

OOF (for queries)

Orchestration

Closed Loop Actions

Policy

Data Collection, Analytics & Events

Inventory Updates

Active & Available Inventory

MSB/Data Movement

CE-1 CE-2 CE-3 CE-4

Generic NF Controller

API Handler

CI-1 CI-2 CI-3

Operational Tree/Config Tree (Service Model)

CI-5

Repository

Service Logic

VNF Descriptors

Config Templates

Engineering Rules

Policy Cache/Event Match

CI-4

Service Logic Processing

Service

Service

Service Logic

…

Supports Model-Driven Lifecycle Mgmt.

- Configure
- Audit
- SW upgrade
- Scale in/out
- Stop/start
- Health check
- L4-7 Service Create
- …

CI-7

Assigned Resources Inventory: Service* Topology & VNF/PNF State

*Not E2E service view. The "Service" view in the Generic NF Controller is limited its scope of control

CI-6

Adapters

CE-6 not needed***

Multi-Cloud Adapter

Netconf

Chef

Ansible

…

Others

External System Adapter (s)

CE-5

CE-6

MSB/Data Movement

Multi-VIM/Cloud

Applications

VNFs

PNFs

External

3rd Party Controllers

Specific VNF Managers

Element Mgt. Systems

*How the services are to be handled is for further study

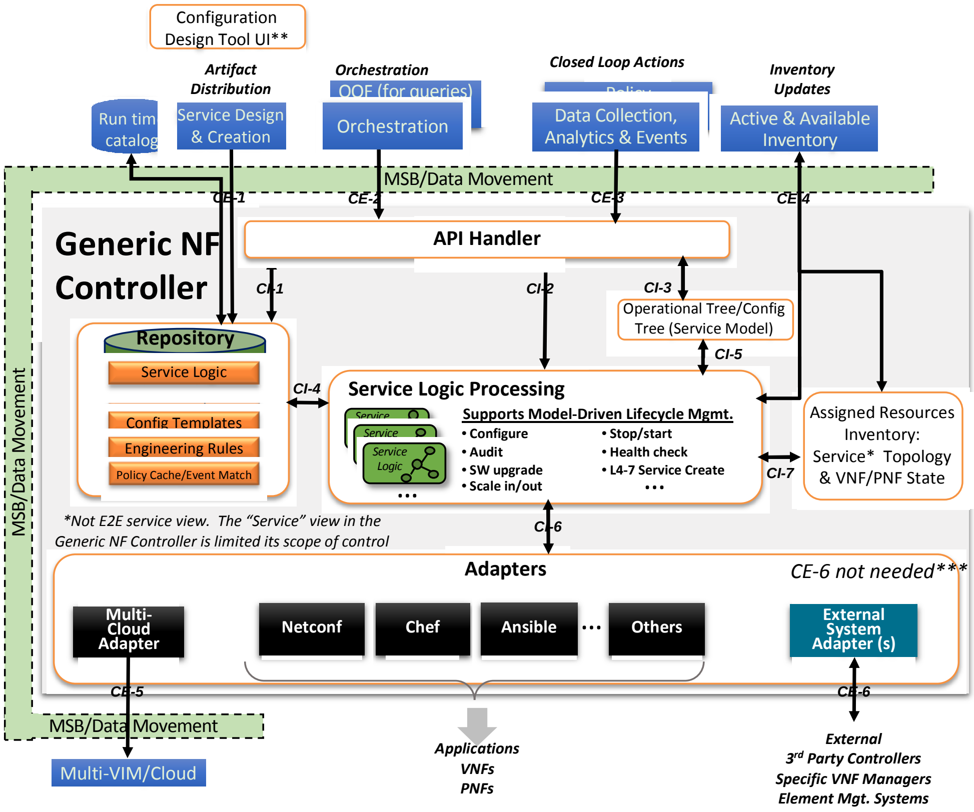
** Configuration Design Tool (CDT) to be integrated into SDC

***CE-6 not needed - see External Controller materials

Generic NF Controller – External/Internal Interface Definitions

Key

CE-x	*Controller External API*
CI-x	*Controller Internal API*



	Interface Definitions
CE-1	Distribution of artifacts from Service Design and Creation – artifacts distributed to Run Time Catalog, GNFC receives notification and pulls from Run Time Catalog *Note: Configuration Design Tool UI to be integrated into Service Design & Creation*
CE-2	Service requests from Orchestration ONAP Optimization Framework (OOF) queries for VNF state and available capacity
CE-3	Closed Loop action requests from Data Collection, Analytics & Events/Policy
CE-4	Inventory retrieval from Active & Available Inventory by Service Logic Processing engine Inventory updates to Active & Available Inventory by Assigned Resources Inv
CE-5	Lifecycle management requests to Multi-Cloud (e.g., stop/start VM)
~~CE-6~~	~~Lifecycle management requests to an external controller or system that has responsibility of the target VNF~~
CI-1	API Handler looks up or retrieves the corresponding Service Logic instance that maps to NB service request (service/network yang)
CI-2	API Handler calls Service Control Processing to perform the Service Logic on the target service or network
CI-3	Prior to CI-2, API Handler might query the (in-memory) Operational/Config Trees for the network or service details (if already existing)
CI-4	Service Control Processing retrieves the Service Logic, Config Templates, Engineering rules, and Policies as part of processing the requested action
CI-5	Service Control Processing queries and/or updates Operational/Config Trees as part of making changes to the network (VNFs/PNFs)
CI-6	Service Control Processing requests adapter layer to update/configure VNF/PNF update using the appropriate adapter for the VNF/PNF
CI-7	Service Control Processing queries and/or updates local Assigned Resources Store/Inventory as part of making changes to the network (VNFs/PNFs)

GNFC – External Interface Details

	Interface Definitions	Beijing Rel.	Casablanca Rel.	Protocol /Service	Comments
CE-1	Distribution of artifacts from Service Design and Creation	SDC➔[no GNFC]	SDC ➔ GNFC (trigger) GNFC ➔ Run Time Catalog (pull)	DMaaP	
CE-2	Service requests from Orchestration Queries from ONAP Optimization Framework (OOF) for VNF state and available capacity	SO, Portal ➔[no GNFC] OOF ➔ [no GNFC]	SO, Portal ➔ GNFC OOF queries – not in scope?	REST	Generic Request API. See next slide for orchestration requests for LCM actions.
CE-3	Closed Loop action requests from Data Collection, Analytics & Events & Policy	DCAE ➔ [no GNFC] Policy – not in scope	DCAE ➔ GNFC Policy – not in scope	DMaaP	
CE-4	Inventory retrieval from Active & Available Inventory by Service Logic Processing engine Inventory updates to Active & Available Inventory by Assigned Resources Inventory	A&AI ⬄ [no GNFC]	A&AI ⬄ GNFC	REST	
CE-5	Configuration requests for cloud infrastructure networking Lifecycle management requests to Multi-Cloud (e.g., stop/start VM)	Multi-Cloud – not in scope	GNFC ➔ M-Cloud	REST	

- Controllers are to be Model-Driven – APIs in Dev, Design, Run-Time catalogs
- Payloads: parameter values defined in the platform Data Dictionary (model/meta-data driven)
- CE-6 interface (to external controllers) is not needed and has been deleted. External controller will be interfacing to the whole ONAP platform – via CE-1 thru CE-4
- Beijing Release does not have an implementation of GNFC
- For Casablanca it is recommended that VF-C and APPC begin to transition toward GNFC

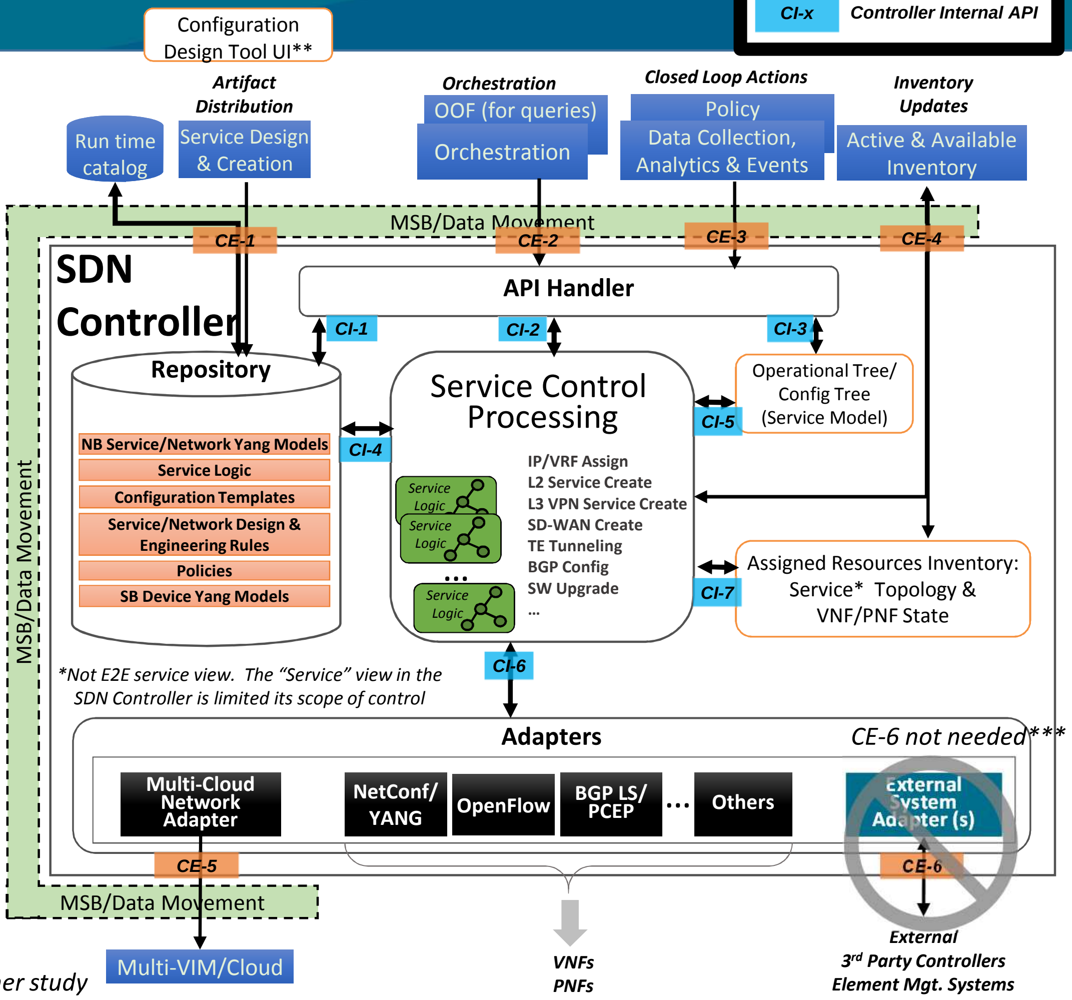
Note: Beyond Casablanca, SDNC functionality and software will begin to merge into GNFC

SDN-Controller Architecture

Key

CE-x	Controller External API
CI-x	Controller Internal API

- **SDN Controller configures and maintains the health of VNFs/PNFs for cloud networking (underlay/overlay) and WAN transport services* throughout their lifecycle**
- **Programmable network application management platform**
  - **Behavior patterns programmed via models and policies**
  - **Standards based models & protocols for multi-vendor implementation**
  - **Extensible SB adapter set supporting various network config protocols, including 3rd party controllers**
  - **Operational control, coordinated state changes across devices, source of telemetry/events, etc.**
- **Manages the health of VNFs/PNFs/transport services in its scope**
  - **Policy-based optimization to meet SLAs**
  - **Event-based control loop automation to solve local issues near real-time**
  - **Action executor for outer control loop automation**
- **Local source of truth**
  - **Manages inventory within its scope**
  - **All stages/states of lifecycle**
  - **Configuration audits**
- **Key Attributes of Controllers**
  - *Intimate with network protocols*
  - *Manages the state of services*
  - *Single service/network domain scope per instance*



*Not E2E service view. The "Service" view in the SDN Controller is limited its scope of control

*How the services are to be handled is for further study

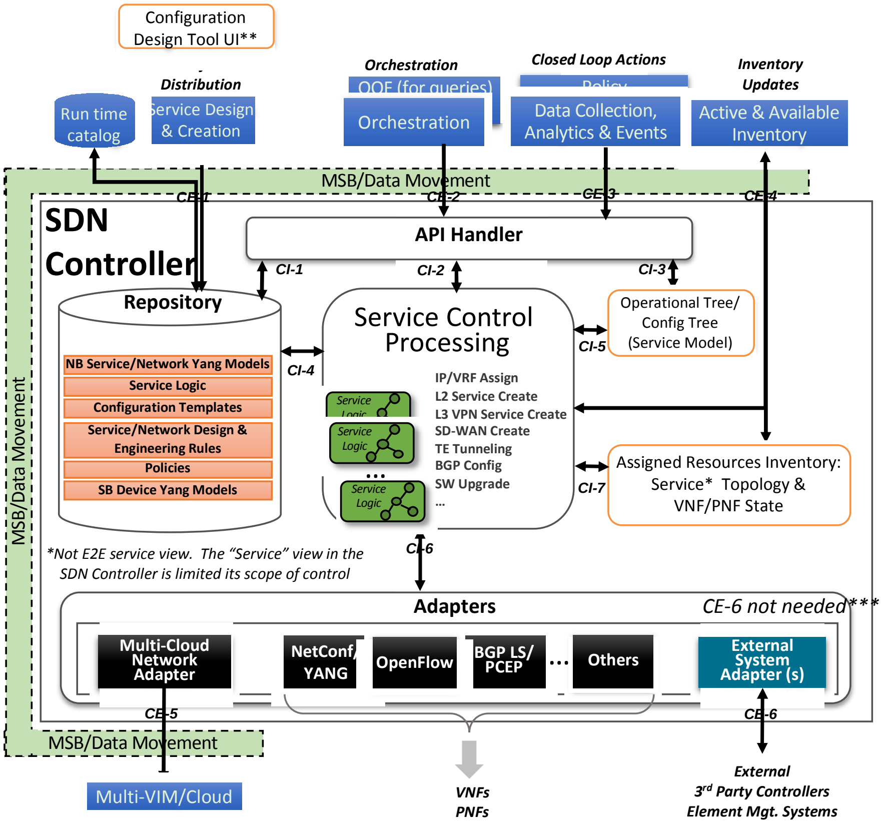
** Configuration Design Tool (CDT) to be integrated into SDC

***CE-6 not needed - see External Controller materials

SDN-Controller – External/Internal Interface Definitions

Key

CE-x	*Controller External API*
CI-x	*Controller Internal API*



*Not E2E service view. The "Service" view in the SDN Controller is limited its scope of control

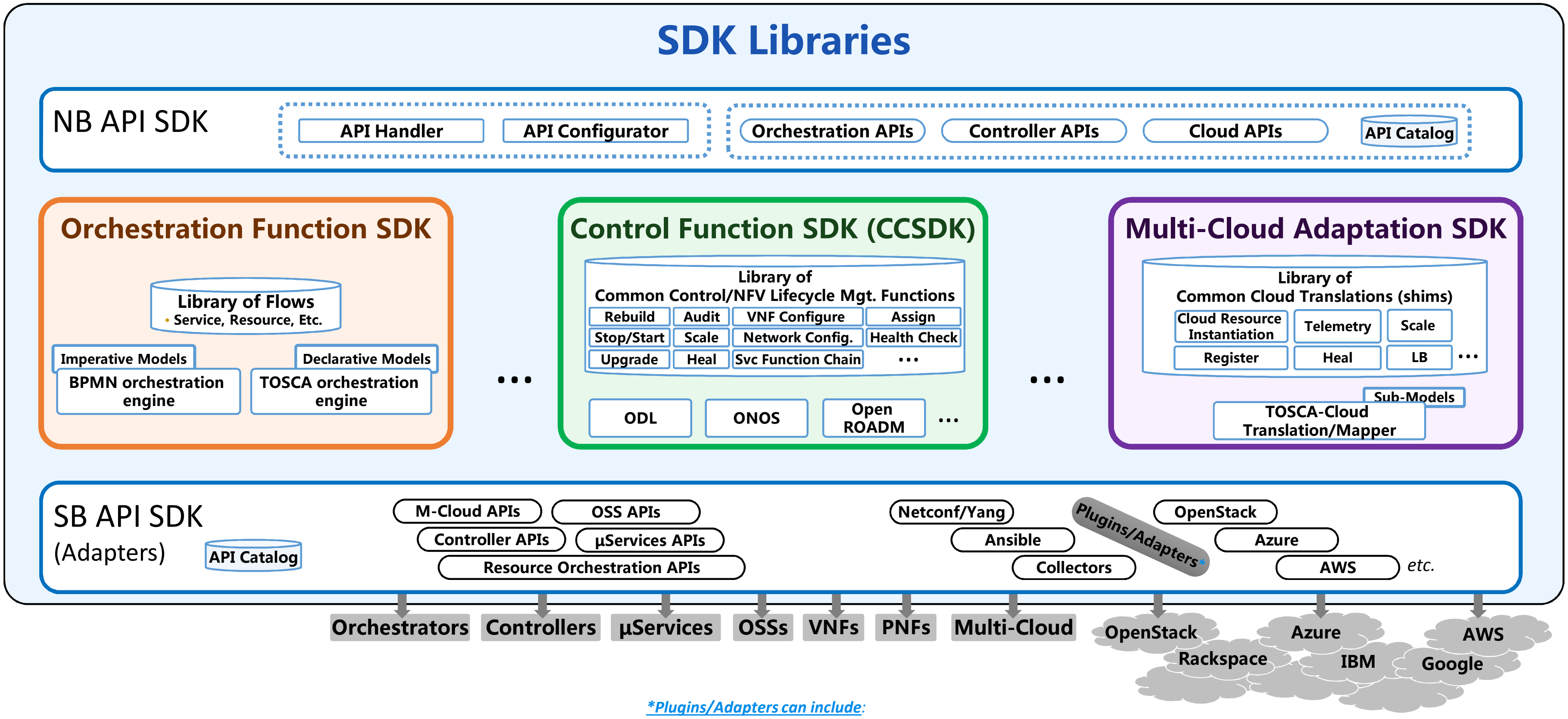
	Interface Definitions
CE-1	Distribution of artifacts from Service Design and Creation – artifacts distributed to Run Time Catalog, SDNC receives notification and pulls from Run Time Catalog *Note: Configuration Design Tool UI to be integrated into Service Design & Creation*
CE-2	Service requests from Orchestration Queries from ONAP Optimization Framework (OOF) for VNF state and available capacity
CE-3	Closed Loop action requests from Data Collection, Analytics & Events/Policy
CE-4	Inventory retrieval from Active & Available Inventory by Service Control Processing engine Inventory updates to Active & Available Inventory by Assigned Resources Inventory
CE-5	Configuration requests for cloud infrastructure networking Lifecycle management requests to Multi-Cloud (e.g., stop/start VM)
~~CE-6~~	~~Lifecycle management or configuration requests to an external controller or system that has responsibility of the target VNF~~
CI-1	API Handler looks up or retrieves the corresponding Service Logic instance that maps to NB service request (service/network yang)
CI-2	API Handler calls Service Control Processing to perform the Service Logic on the target service or network
CI-3	Prior to CI-2, API Handler might query the (in-memory) Operational/Config Trees for the network or service details (if already existing)
CI-4	Service Control Processing retrieves the Service Logic, Config Templates, Engineering rules, and Policies as part of processing the requested action
CI-5	Service Control Processing queries and/or updates Operational/Config Trees as part of making changes to the network (VNFs/PNFs)
CI-6	Service Control Processing requests adapter layer to update/configure VNF/PNF update using the appropriate adapter for the VNF/PNF
CI-7	Service Control Processing updates the local Assigned Resources Store/Inventory once network updates are made successfully

SDK-Driven Sub-System – Libraries

(including CCSDK)

Benefits

- Improve agility
- Reduce SW footprint
- Reusable framework
- Enable technology swap
- Consistent NB/SB APIs
- Flexible platform extensions



SDK Libraries

NB API SDK

API Handler | API Configurator | Orchestration APIs | Controller APIs | Cloud APIs | API Catalog

Orchestration Function SDK

Library of Flows
- Service, Resource, Etc.

Imperative Models – BPMN orchestration engine

Declarative Models – TOSCA orchestration engine

…

Control Function SDK (CCSDK)

Library of Common Control/NFV Lifecycle Mgt. Functions

Rebuild | Audit | VNF Configure | Assign
Stop/Start | Scale | Network Config. | Health Check
Upgrade | Heal | Svc Function Chain | …

ODL | ONOS | Open ROADM | …

…

Multi-Cloud Adaptation SDK

Library of Common Cloud Translations (shims)

Cloud Resource Instantiation | Telemetry | Scale
Register | Heal | LB | …

Sub-Models

TOSCA-Cloud Translation/Mapper

SB API SDK (Adapters)

API Catalog

M-Cloud APIs | OSS APIs | Controller APIs | µServices APIs | Resource Orchestration APIs

Netconf/Yang | Ansible | Collectors

Plugins/Adapters*

OpenStack | Azure | AWS | etc.

Orchestrators | Controllers | µServices | OSSs | VNFs | PNFs | Multi-Cloud

OpenStack | Rackspace | Azure | IBM | AWS | Google

__Plugins/Adapters can include__:

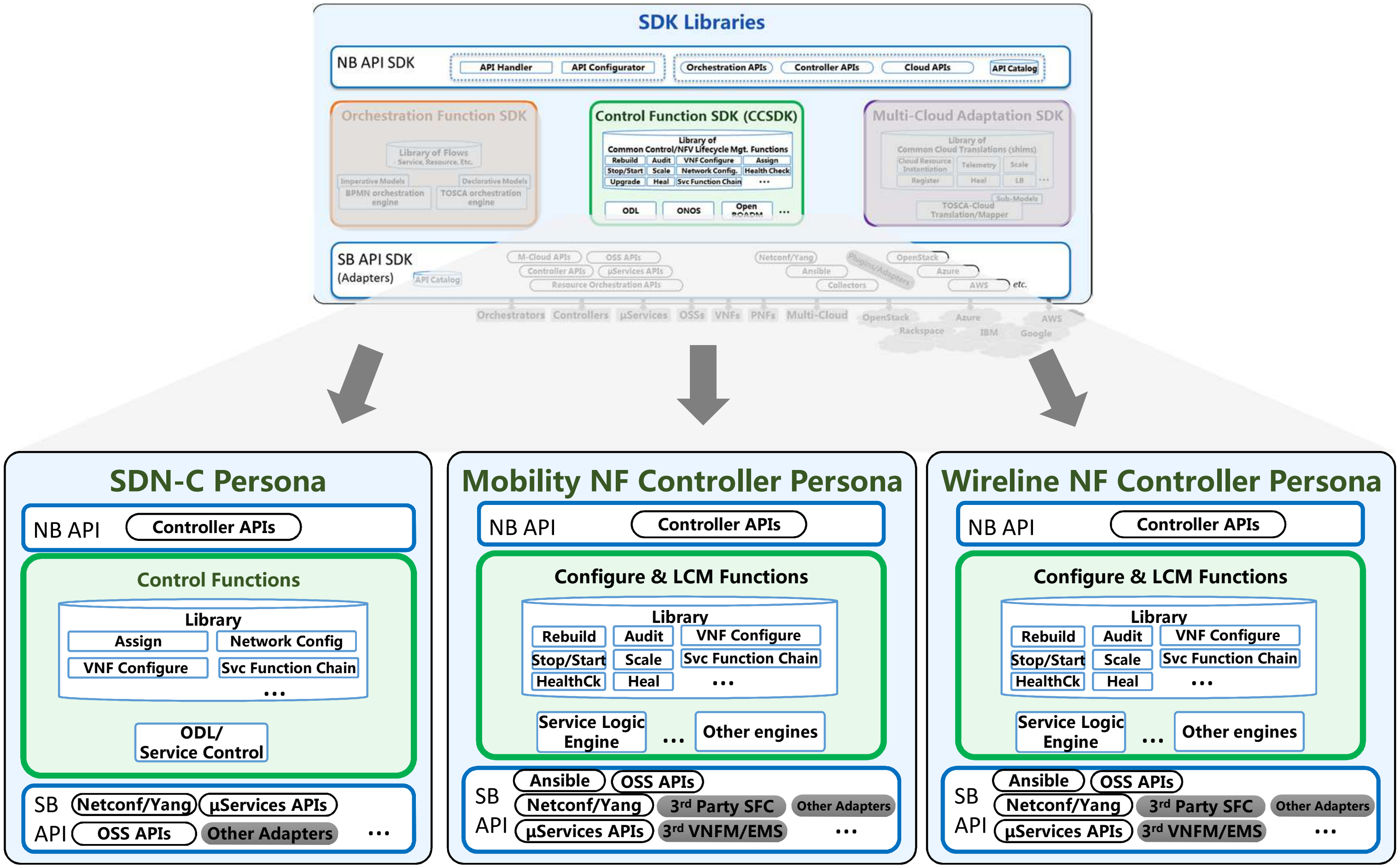
- *3rd party VNFM Drivers*
- *3rd party SFC Drivers*
- *Netconf/Yang*
- *SNMP*
- *3rd party EMS Drivers*
- *Ansible/Chef/Puppet*
- *CLI*
- *etc.*

Controller Personas Based on CCSDK Libraries

CCSDK Libraries

Controller Personas Examples

(created from CCSDK)


SDK Libraries
NB API SDK
API Handler
API Configurator
Orchestration APIs
Controller APIs
Cloud APIs
API Catalog
Orchestration Function SDK
Library of Flows
Service, Resource, Etc.
Imperative Models
Declarative Models
BPMN orchestration engine
TOSCA orchestration engine
Control Function SDK (CCSDK)
Library of
Common Control/NFV Lifecycle Mgt. Functions
Rebuild
Audit
VNF Configure
Assign
Stop/Start
Scale
Network Config.
Health Check
Upgrade
Heal
Svc Function Chain
…
ODL
ONOS
Open ROADM
…
Multi-Cloud Adaptation SDK
Library of
Common Cloud Translations (shims)
Cloud Resource Instantiation
Telemetry
Scale
Register
Heal
LB
…
Sub-Models
TOSCA-Cloud Translation/Mapper
SB API SDK
(Adapters)
API Catalog
M-Cloud APIs
OSS APIs
Controller APIs
µServices APIs
Resource Orchestration APIs
Netconf/Yang
Ansible
Collectors
Plugins/Adapters
OpenStack
Azure
AWS
etc.
Orchestrators
Controllers
µServices
OSSs
VNFs
PNFs
Multi-Cloud
OpenStack
Rackspace
Azure
IBM
AWS
Google
SDN-C Persona
NB API
Controller APIs
Control Functions
Library
Assign
Network Config
VNF Configure
Svc Function Chain
…
ODL/
Service Control
SB
API
Netconf/Yang
µServices APIs
OSS APIs
Other Adapters
…
Mobility NF Controller Persona
NB API
Controller APIs
Configure & LCM Functions
Library
Rebuild
Audit
VNF Configure
Stop/Start
Scale
Svc Function Chain
HealthCk
Heal
…
Service Logic Engine
…
Other engines
SB
API
Ansible
OSS APIs
Netconf/Yang
3rd Party SFC
Other Adapters
µServices APIs
3rd VNFM/EMS
…
Wireline NF Controller Persona
NB API
Controller APIs
Configure & LCM Functions
Library
Rebuild
Audit
VNF Configure
Stop/Start
Scale
Svc Function Chain
HealthCk
Heal
…
Service Logic Engine
…
Other engines
SB
API
Ansible
OSS APIs
Netconf/Yang
3rd Party SFC
Other Adapters
µServices APIs
3rd VNFM/EMS
…