ONAP

OPEN NETWORK AUTOMATION PLATFORM

ONAP Microservices - Approaches

Manoj Nair
NetCracker

Reference : Modular ONAP Implementation and Integration to External DMS (Alex and Ramesh)

ONAP Modular Implementation

This presentation's focus

- Improvements to Model Driven Approach proposed in Alex's presentation, with reference to some best practices
- How Modularity can be achieved (Reference to Alex's proposal earlier) : Best Practices from TMF
- : Approaches and Views
- Cloud Native Micro Services

References : https://wiki.onap.org/display/DW/Microservices+Industry+Perspectives

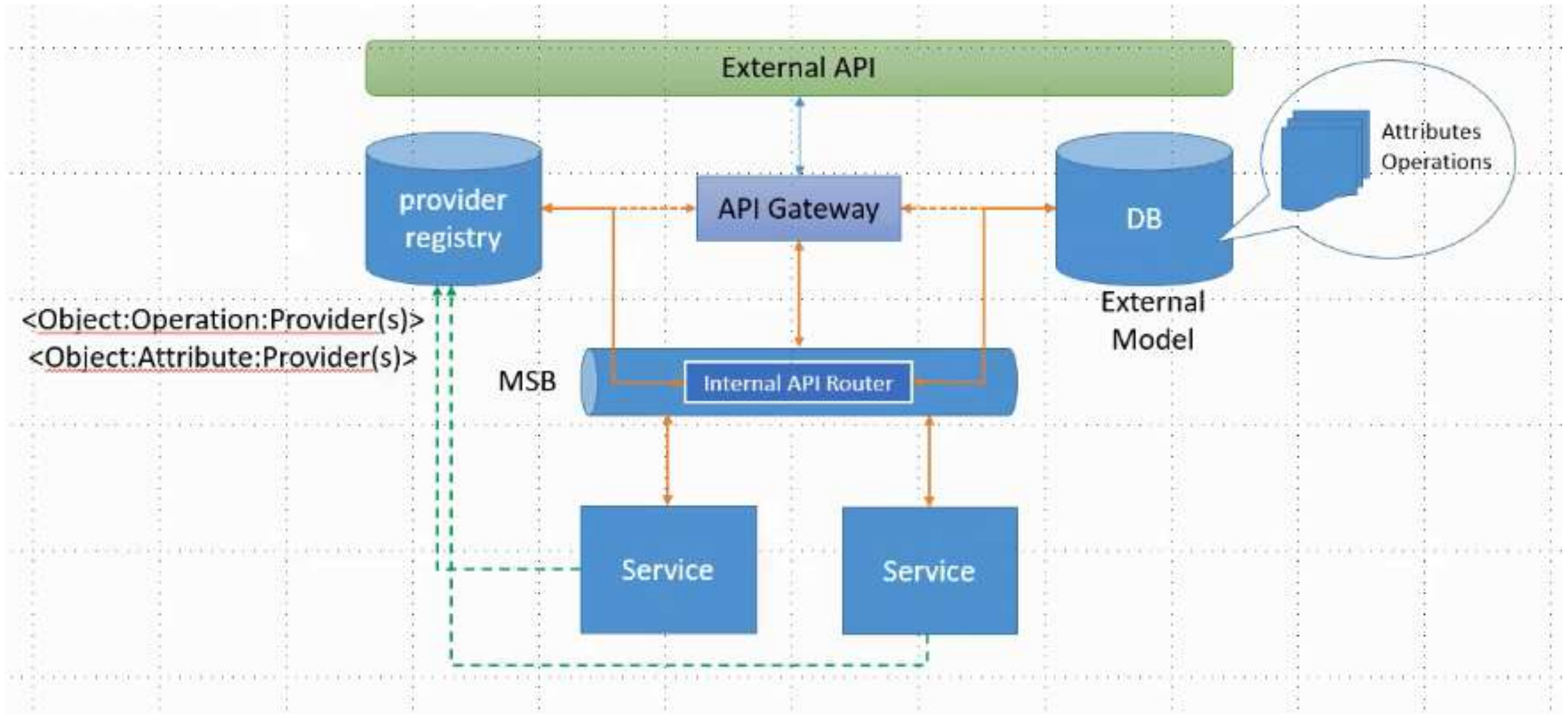
What is proposed here ?

- Three approaches to refine Microservice Architecture
- Microservice Architecture : Deals with how microservices are defined, realized and how they interact.
- Approach 1: Model Driven Microservice – Approach based on DDD
  - Other Open source implementations – ODL, OSM etc, Model driven Microservices by Alex, Reference to OSM implementation.
- Approach 2: Modular Microservices - Leverage TMForum IG1118 Management and Control Continuum and Future Mode of Operation concepts
  - Based on proposal by Alex and Ramesh on DMS, but enhancing it further with reference to a standard architecture.
  - Primarily focused on structuring microservices and components in a hierarchical "Lego blocks" like units which can be mixed and matched to create management and control functions.
- Approach 3: Cloud Native Micro Service Design (Followed by MSB)
  - Based on the Service Mesh technique used in the cloud native world.
  - Mainly an implementation approach than a consistent microservice design methodology.

Above approaches are complementary to each other and not exclusive

Approach 1: Model Driven Microservices

Reference : Model Driven Micro Services (Alex)



Domain Driven Design Concept

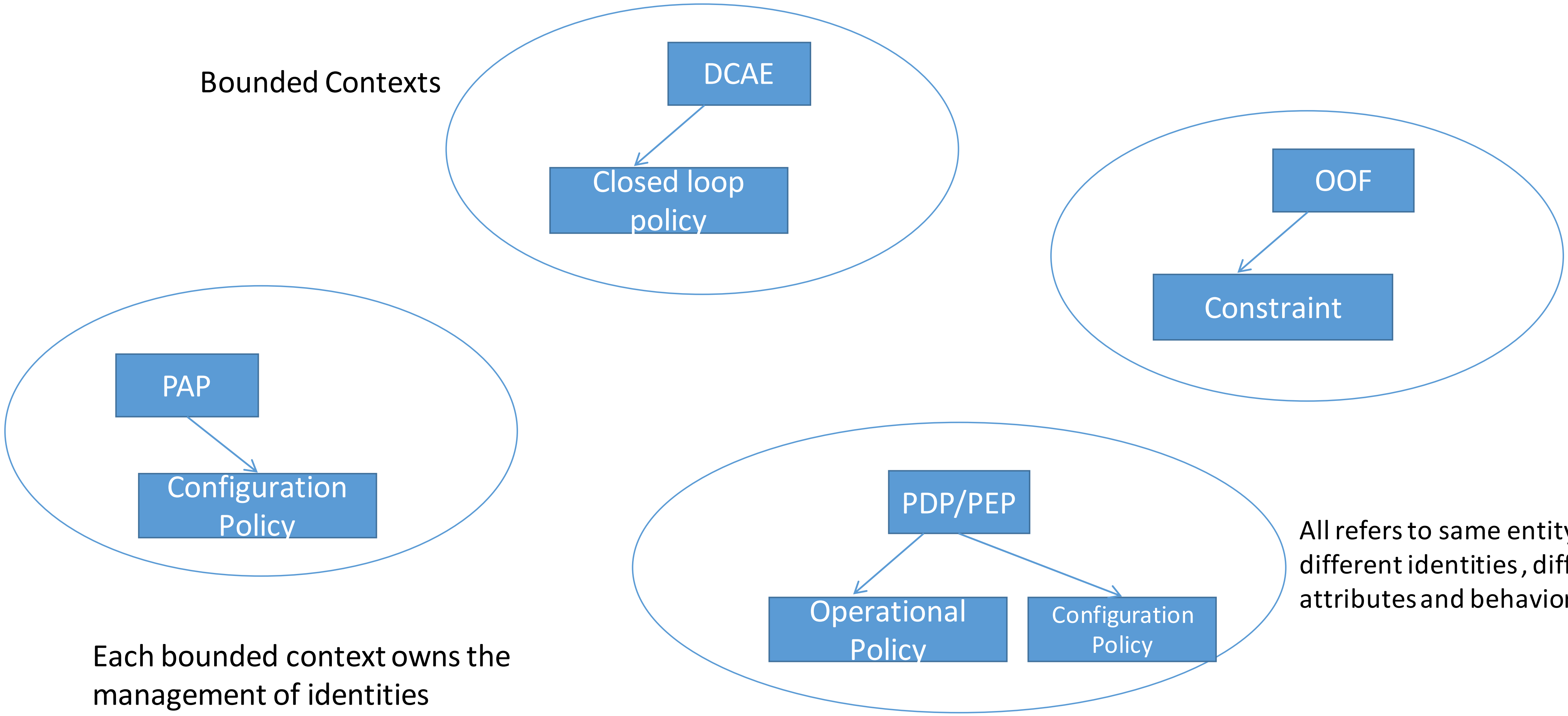
- Top down approach of designing Microservices
- Domain-driven design (DDD) advocates modeling based on the domain of business
- In the ONAP context Domain may be - Hybrid Telecom Service Orchestration and Management
- In the context of building applications, DDD talks about problems as domains
- It describes independent problem areas as Bounded Contexts (each Bounded Context correlates to a microservice)
- Bounded contexts own its own domain data, logic and behavior
- In DDD to define the domain model for each Bounded Context,
  - identify and define the entities (and other patterns if required)
  - create a boundary around things that need cohesion
  - build and refine domain model contained within the boundary that defines the context
  - Each bounded context can be modelled as a microservice

Domain Entity Pattern

- Entities represent domain objects and are primarily defined by their identity and not only by the attributes that comprise them
- An entity's identity can cross multiple microservices or Bounded Contexts.
  - For example Policy in one microservice might be TCA Configuration in another microservice or Constraint in yet another microservice
- The same entity can be modeled across multiple Bounded Contexts or microservices as different identities.
- However, that does not imply that the same entity, with the same attributes and logic would be implemented in multiple Bounded Contexts.
- Instead, entities in each Bounded Context limit their attributes and behaviors to those required in that Bounded Context's domain.

An example

Bounded Contexts



All refers to same entity with different identities , different attributes and behavior

Each bounded context owns the management of identities

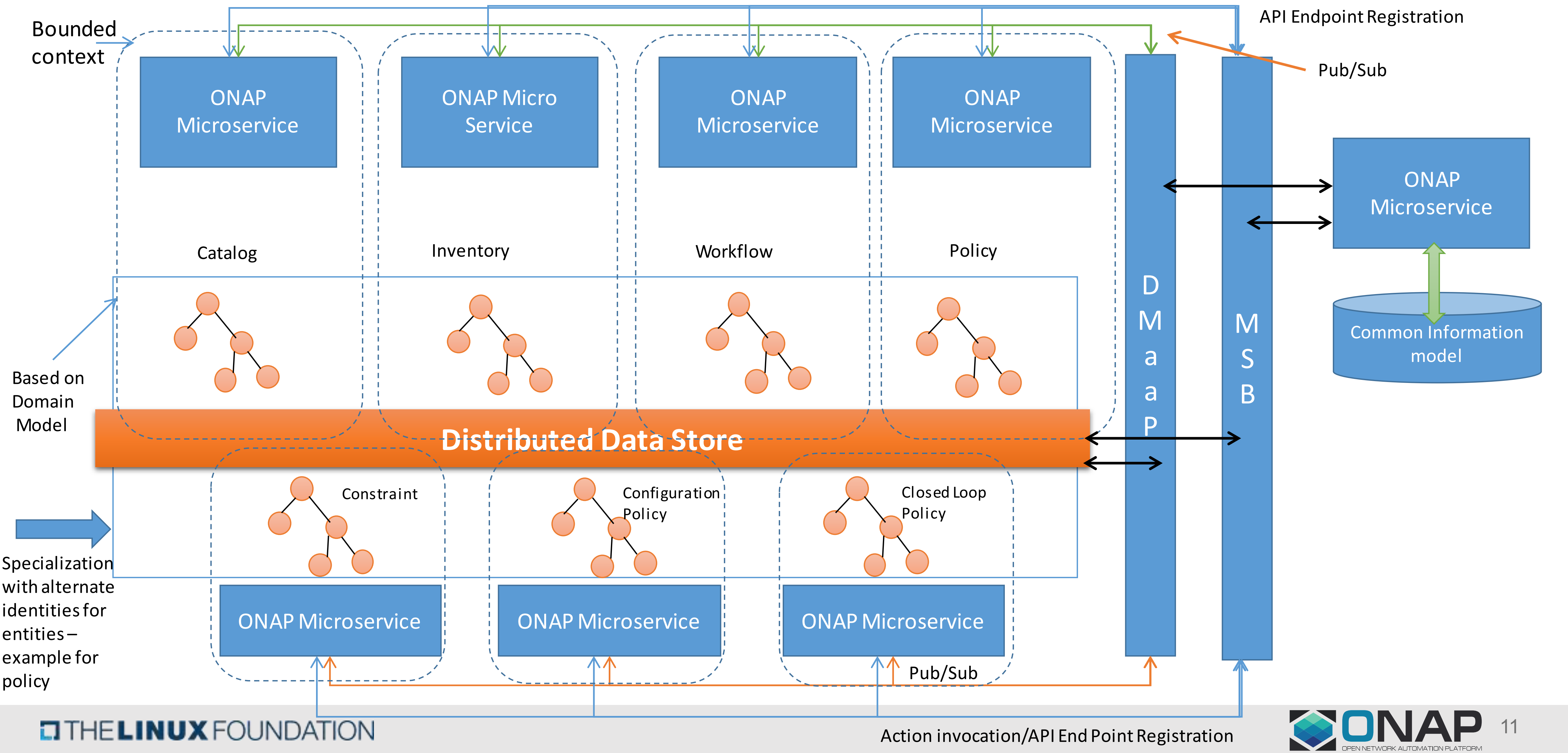
Views on ONAP Current Microservice Architecture

- ONAP's current MS architecture is defined per component. So the domain as per DDD has to be looked at a per component level
  - ONAP seem to limit domain scope to each component OR sometimes seem to refer to use cases in the context of domains.
  - Microservices are bound to components than end to end business context (not loosely defined based on domain entities but bound within each component) or use cases.
  - To realize business use case it has to be disintegrated at component level and then MS level.
  - Multiple components and associated micro services need to be augmented to support end to end capability.
  - In ONAP each component bounds the entities, domain data and associated identities. This creates a dependency and violates the independent development practices put forth by DDD through bounded context.
  - Kubernetes seems to map microservice to Service and not Pods, but currently each component in ONAP is mapped to Kubernetes Service with associated Pods representing docker containers fulfilling the micro functionality. Reference : link , link

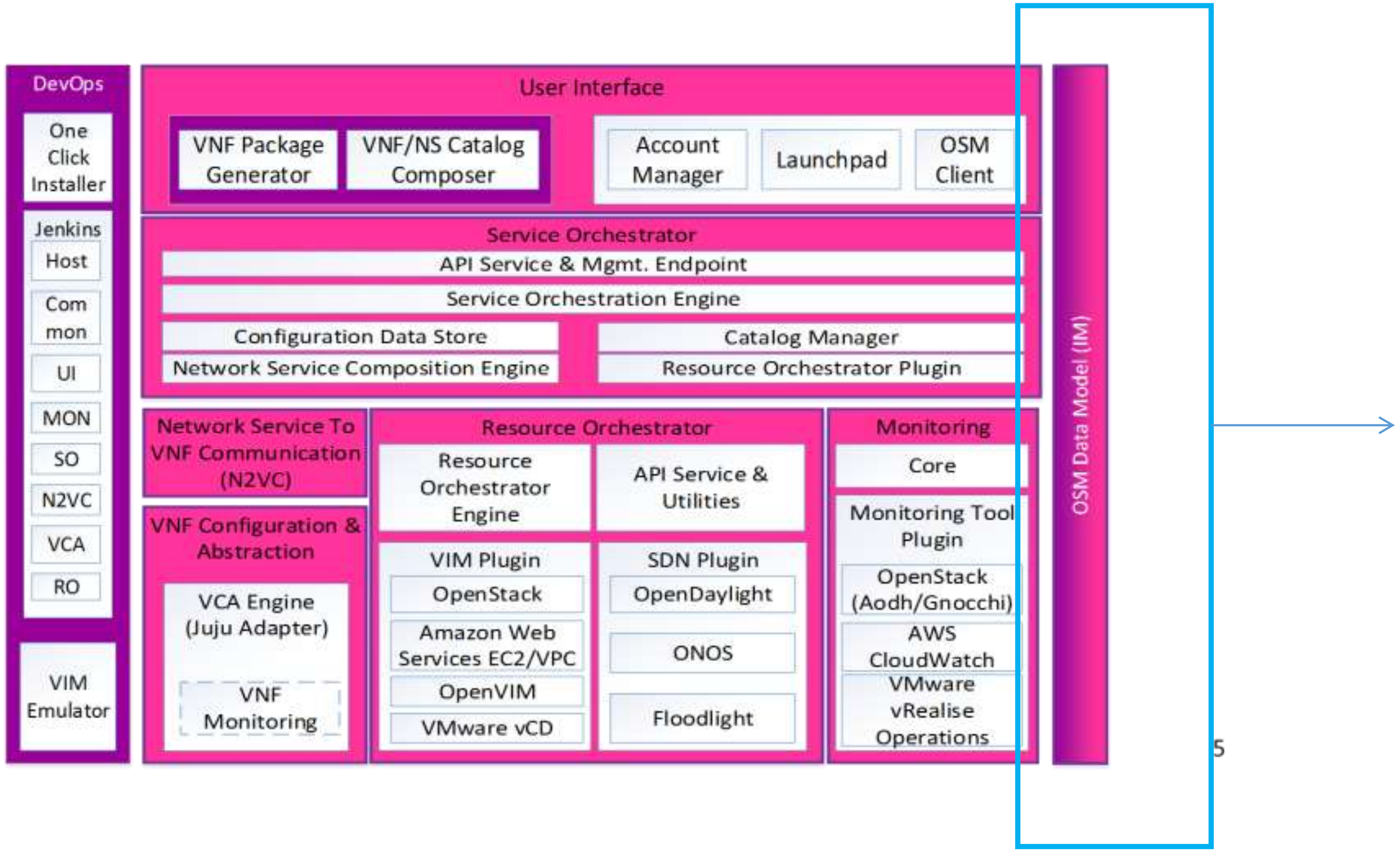
How DDD enables Model Driven Design

- Encouraged to create domain entities and bounded contexts (Quite similar to what modelling team is doing with CIM, but might require further identification of bounded contexts)
- Each bounded context has own domain model and ubiquitous language relevant in own boundary
- A context map is formed representing interaction between bounded contexts
- Microservice boundaries are defined based on bounded context

An Abstract View of Realizing DDD

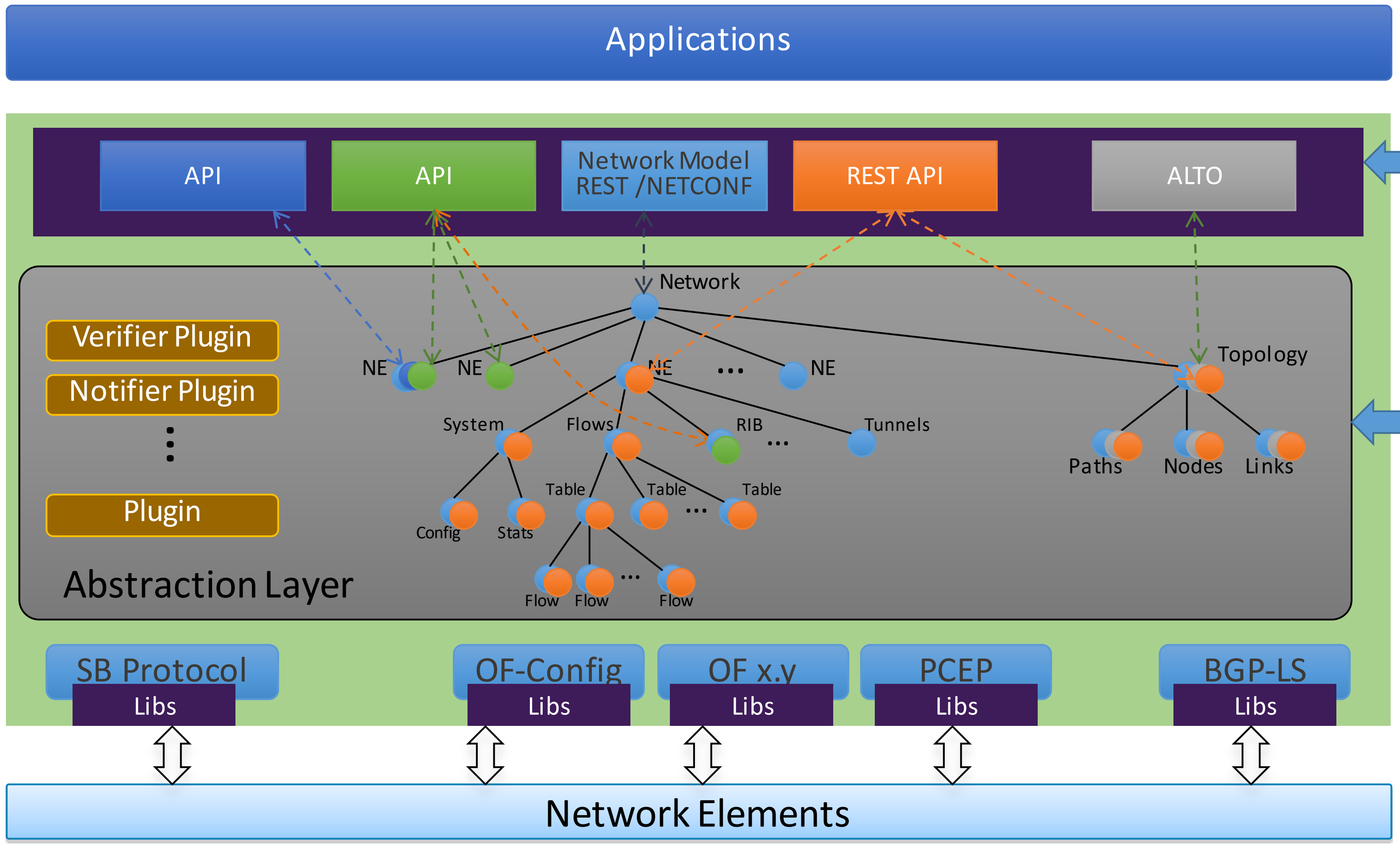


Example – Model driven approach by OSM



- Models are centrally managed
- Yang models are compiled to generate base classes used by all the components.
- IM-NBI uses pyang tool set to generate python code, json schema and json message templates to be used by modules in OSM
- <u>Centralized management of data models helps in having a unified view across OSM and extensions are managed centrally</u>

Example – Model driven and Event Driven MS in ODL



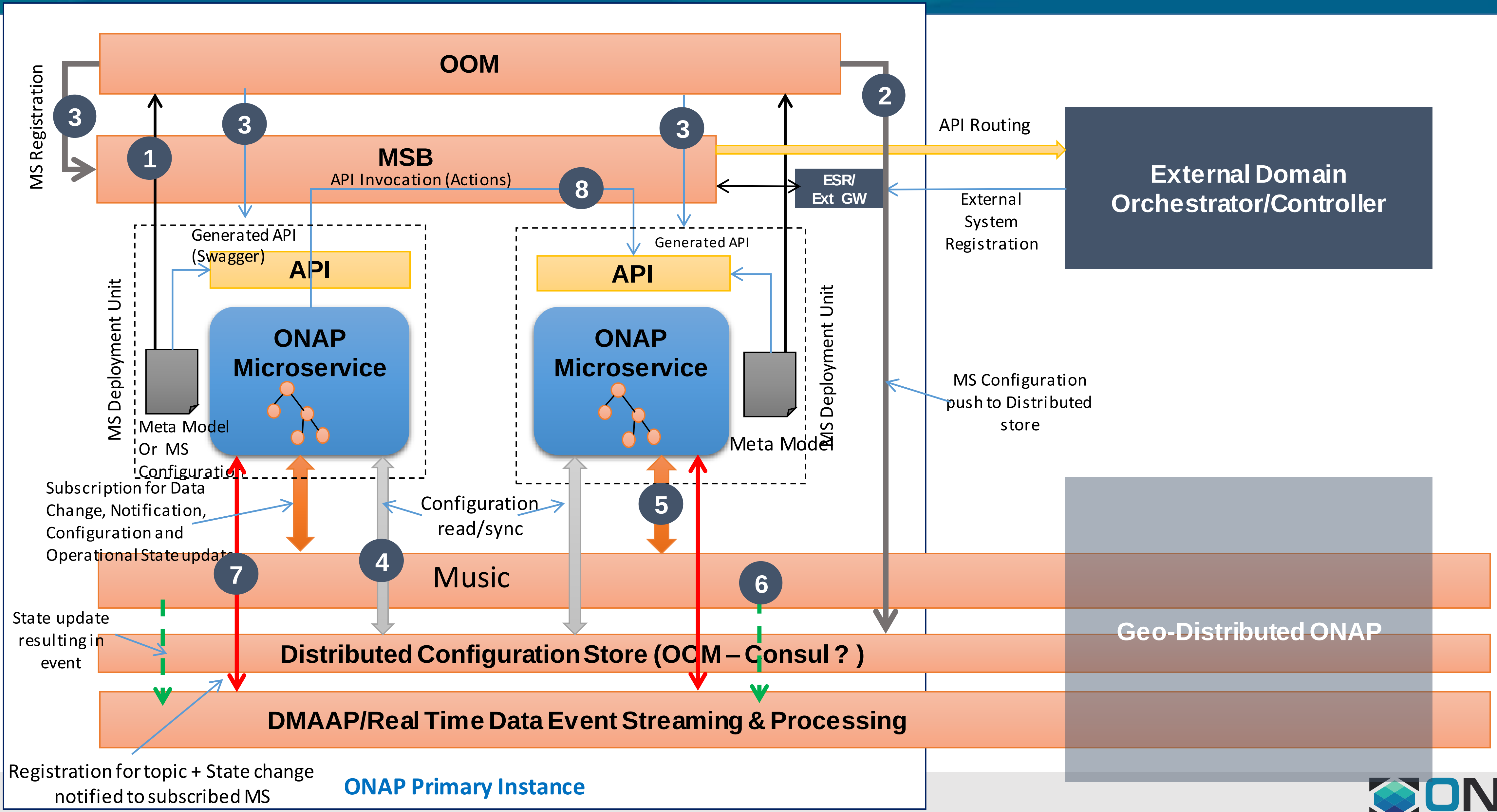
Micro services register with MD-SAL platform – about capabilities, interfaces, entities etc. Each MS is modelled using YANG and compiled to create skeleton code

OpenDaylight MD-SAL – Framework stores Config and Operational data and enables interaction between MS.

MD-SAL takes care of data storage , cluster data consistency, sharding, interaction between micro services, set of tools for project skeleton . Any change in distributed config/operational tree is notified to subscribed MS

Reference: Opendaylight MD-SAL Architecture

Model Driven Microservices – One possible implementation view



OOM
MS Registration
3
3
3
2
1
MSB
API Invocation (Actions)
8
API Routing
ESR/ Ext GW
External Domain Orchestrator/Controller
External System Registration
Generated API (Swagger)
API
Generated API
API
MS Deployment Unit
ONAP Microservice
ONAP Microservice
MS Deployment Unit
Meta Model Or MS Configuration
Meta Model
MS Configuration push to Distributed store
Subscription for Data Change, Notification, Configuration and Operational State update
Configuration read/sync
5
4
7
6
Music
Geo-Distributed ONAP
State update resulting in event
Distributed Configuration Store (OOM – Consul ?)
DMAAP/Real Time Data Event Streaming & Processing
Registration for topic + State change notified to subscribed MS
ONAP Primary Instance

Key Concepts 1/2

- Microservice deployment unit consisting of the core MS software along with metafile which contains configuration and deployment descriptor (Can be modelled using SDC ?)
- MUSIC for distributed , highly available storage for domain entities
- Distributed Configuration Store for maintaining runtime configuration data of MS
- Microservice Bus which act like a MS Endpoint Registry and API call routing GW
- DMaaP or Real time data event streaming bus (based on RESP) which routes messages and file transfers across distributed MSs
- OOM which reads the Metafile of MS and carries out deployment, registration.

Key Concepts 2/2

1. Micro services are packaged as deployment units – Packaging here is just logical in nature. Physically this can be an ONAP micro service component with a well-known (XML. YAML etc) configuration, deployment descriptors which acts like a metadata for MS. In practical sense the Metadata may be split to two files – deployment descriptor and configuration descriptor. Deployment descriptor is used as input for deployment by Kubernetes or Docker Engine. Configuration Descriptor captures following details
   - MS configuration like DB connectivity details
   - MSB registration including the capabilities
   - DMaaP registration including the topics published and subscribed
   - CHAP related configuration
   - Domain data bound by MS
   - Swagger for generating APIs for MS (which can be consumed by MSB)
2. During the microservice deployment (using deployment descriptor), metadata is read from the microservice predefined directory structure and pushed to the configuration data store. Rest of the configuration changes can be applied on the configuration store
3. OOM deploys MS and carries out registration of MS on MSB based on the registration data read from configuration store . Similar registration mechanism can be carried out for DMaaP as well.
4. MS carries out self-configuration or OOM initiates configuration of MS by reading the configuration data store
5. During the lifecycle of the MS the state is updated on the CHAP distributed DB which triggers notification on DMaaP
6. Notification forwarded to consumer by DMaaP
7. Consumer act upon state change : This is optional – indicating the event driven microservice behavior
8. <u>Consumer potentially invokes action on Provider through an API call.</u> : This is optional – indicating the event driven microservice behavior

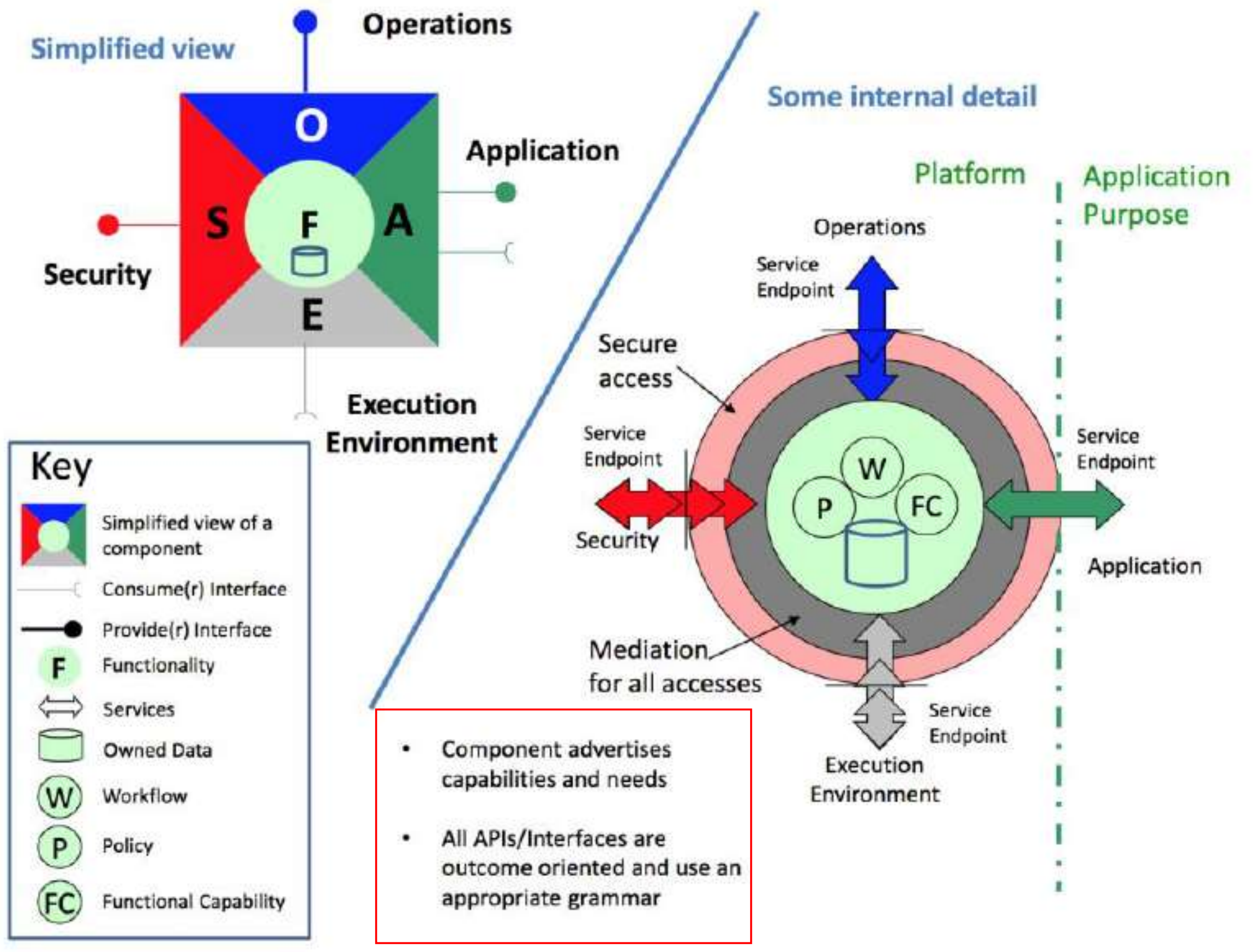
Approach 1: Observations

- The concepts described in this approach focus on the boundary, structure and interaction patterns of microservices
- Microservices can be model driven following the domain driven design principle with bounded context mapped to a microservice. Each bounded context maintaining own domain model.
- The reusability of the microservices and aggregation of microservices to form a system or a solution with a well defined grouping pattern is not covered well.
- Require effort in refactoring the existing microservices to define context map, domain models etc.
- Will enforce top down modelling of microservices with alignment to domain rather than independent projects or components.

ONAP
OPEN NETWORK AUTOMATION PLATFORM

Approach 2: Modular Microservices Design

TMForum IG1118 : FMO Component (Concept)



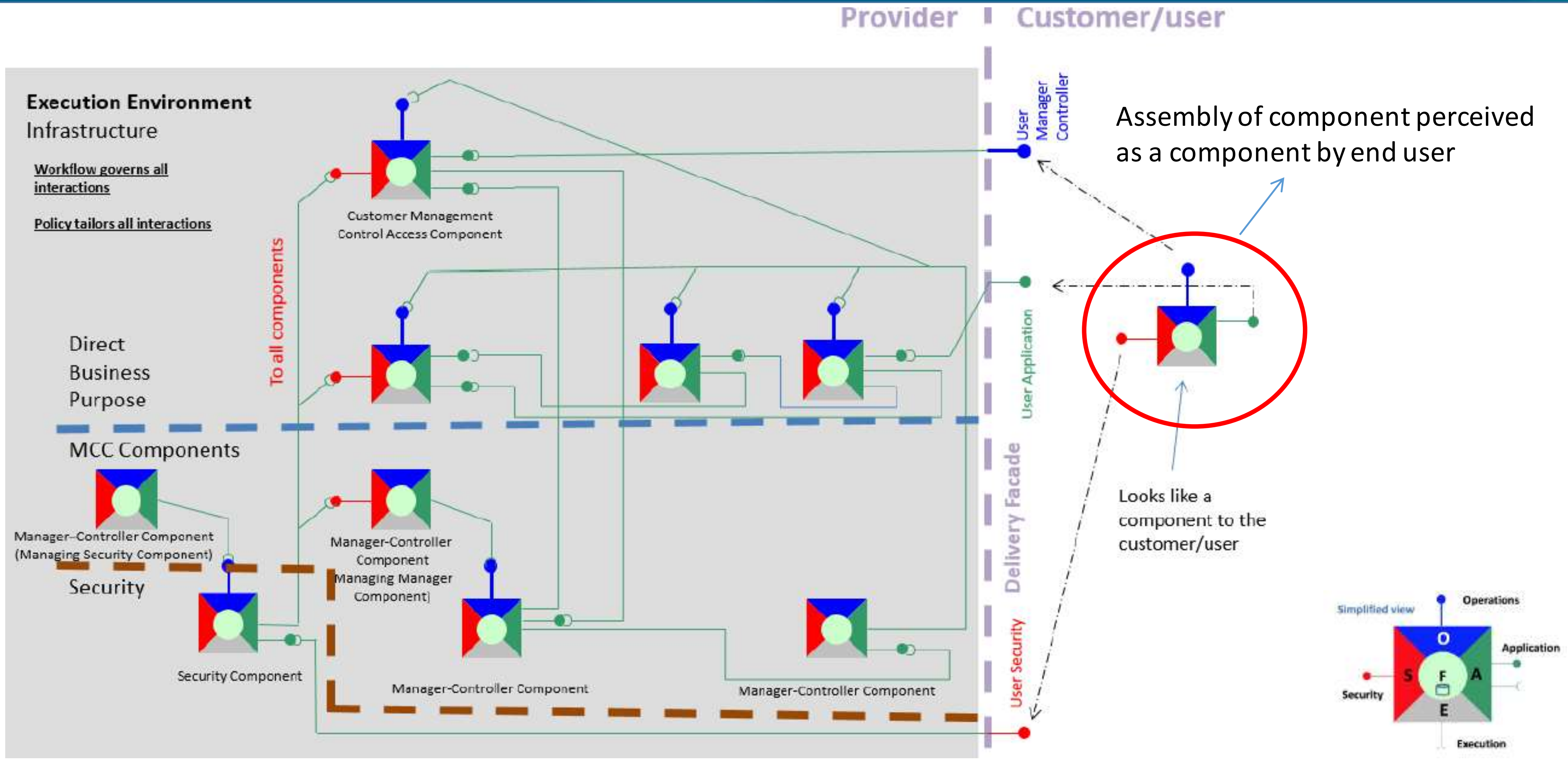
- Component advertises capabilities and needs
- All APIs/Interfaces are outcome oriented and use an appropriate grammar

- **Consumer Interface**: Interface used by component to fulfil needs for tasks to be carried out related to its purpose. The tasks will include the evaluation of information and the providing of results.
- **Provider Interface**: Interface through which a component offers a capability to carry out specific tasks related to its purpose.
- **Application Function**: Exposed through Application API and provides access to all the related to the primary business purpose of the component
- **Operations Function**: Exposed through operations API and implements the support for managing the lifecycle process for the component
- **Security Function**: Exposed through security API and and provides security features for the component
- **Environment Function**: Accessed through the execution environment APIs and used by the component to access the infrastructure resources that it needs to support its functionality or operations.

Management and Control Continuum

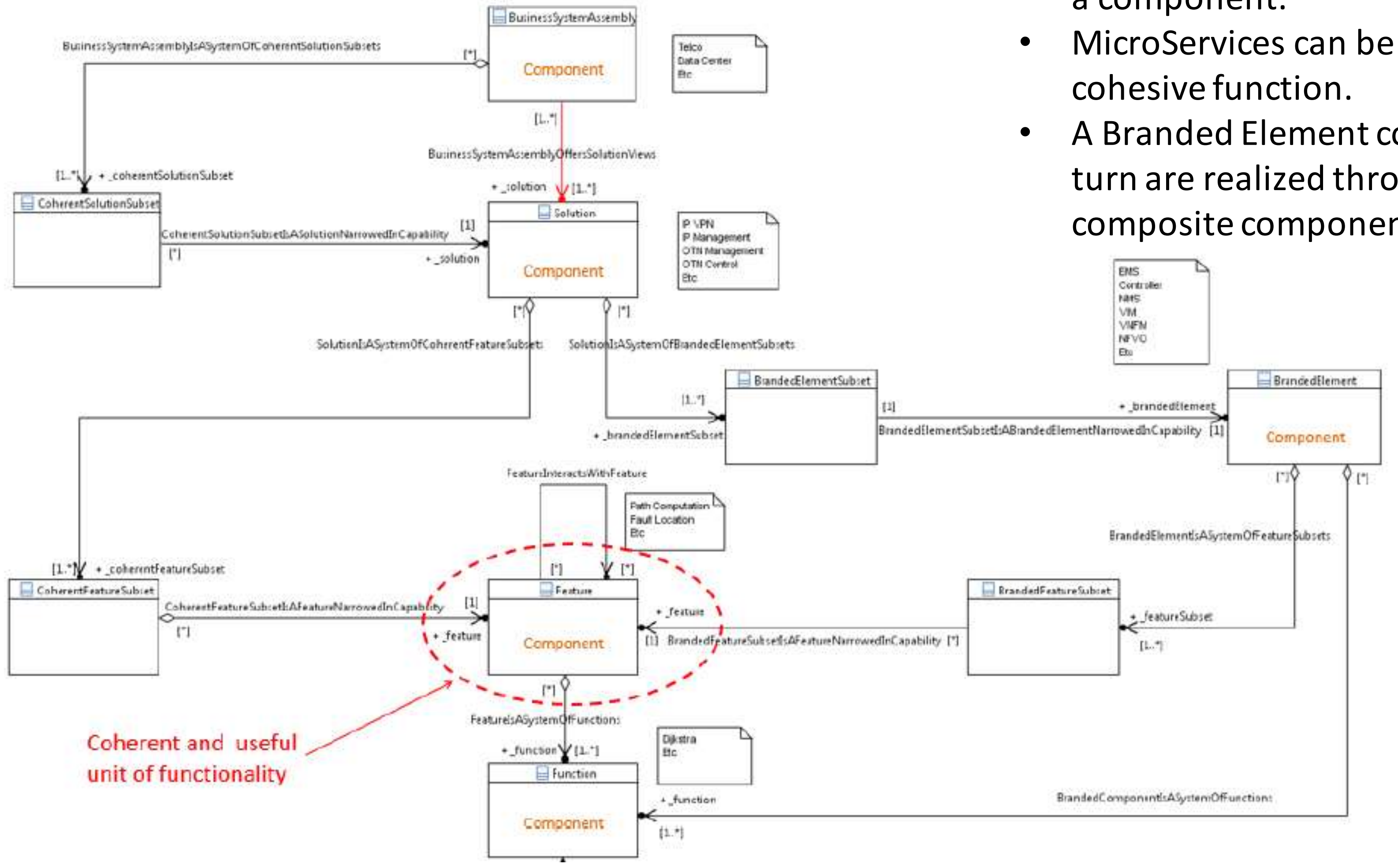
- A part of FMO Concept which focus on operational aspects of services in the target architecture built on NFV and SDN.
- Deals with component that have management-control functions as their primary functions.
- In traditional systems functions like EMS/NMS, Control Plane etc are considered as MCC functions.
- **In the MCC architecture MCC components can be assembled to form MCC Systems. An assembly of MCC components can be given a certain name/branding.**
- **A collection of functional components can be packaged together and delivered as a solution to a particular problem.**
- **Such an assembly in itself can be considered as a component.**
- Each component in an assembly is contributing its application function to the capabilities of the overall branded component
- The same MCC component can be offered as part of different branded components.

FMO Architecture: Composition of Components


Provider
Customer/user
Execution Environment
Infrastructure
Workflow governs all interactions
Policy tailors all interactions
To all components
Customer Management Control Access Component
Direct Business Purpose
MCC Components
Manager-Controller Component (Managing Security Component)
Security
Security Component
Manager-Controller Component (Managing Manager Component)
Manager-Controller Component
Manager-Controller Component
User Manager Controller
User Application
Delivery Facade
User Security
Assembly of component perceived as a component by end user
Looks like a component to the customer/user
Simplified view
Operations
Application
Security
Execution
O
S
F
A
E

Reference: TMForum IG1118

FMO Component : Positioning of a Component in a hierarchy of modular constructs


BusinessSystemAssembly
Component
Telco
Data Center
Etc
BusinessSystemAssemblyIsASystemOfCoherentSolutionSubsets
[*]
[1..*]
BusinessSystemAssemblyOffersSolutionViews
[1..*]
+ _coherentSolutionSubset
+ _solution
[1..*]
CoherentSolutionSubset
CoherentSolutionSubsetIsASolutionNarrowedInCapability
[1]
[*]
+ _solution
Solution
Component
IP VPN
IP Management
OTN Management
OTN Control
Etc
[*]
[*]
SolutionIsASystemOfCoherentFeatureSubsets
SolutionIsASystemOfBrandedElementSubsets
EMS
Controller
NMS
VIM
VNFM
NFVO
Etc
BrandedElementSubset
[1..*]
+ _brandedElementSubset
[1]
BrandedElementSubsetIsABrandedElementNarrowedInCapability
[1]
+ _brandedElement
BrandedElement
Component
[*]
[*]
FeatureInteractsWithFeature
Path Computation
Fault Location
Etc
BrandedElementIsASystemOfFeatureSubsets
[1..*]
+ _coherentFeatureSubset
CoherentFeatureSubset
CoherentFeatureSubsetIsAFeatureNarrowedInCapability
[*]
[1]
+ _feature
[*]
[*]
Feature
Component
+ _feature
[1]
BrandedFeatureSubsetIsAFeatureNarrowedInCapability
[*]
BrandedFeatureSubset
+ _featureSubset
[1..*]
[*]
FeatureIsASystemOfFunctions
Coherent and useful unit of functionality
+ _function
[1..*]
Dijkstra
Etc
Function
Component
+ _function
[1..*]
BrandedComponentIsASystemOfFunctions

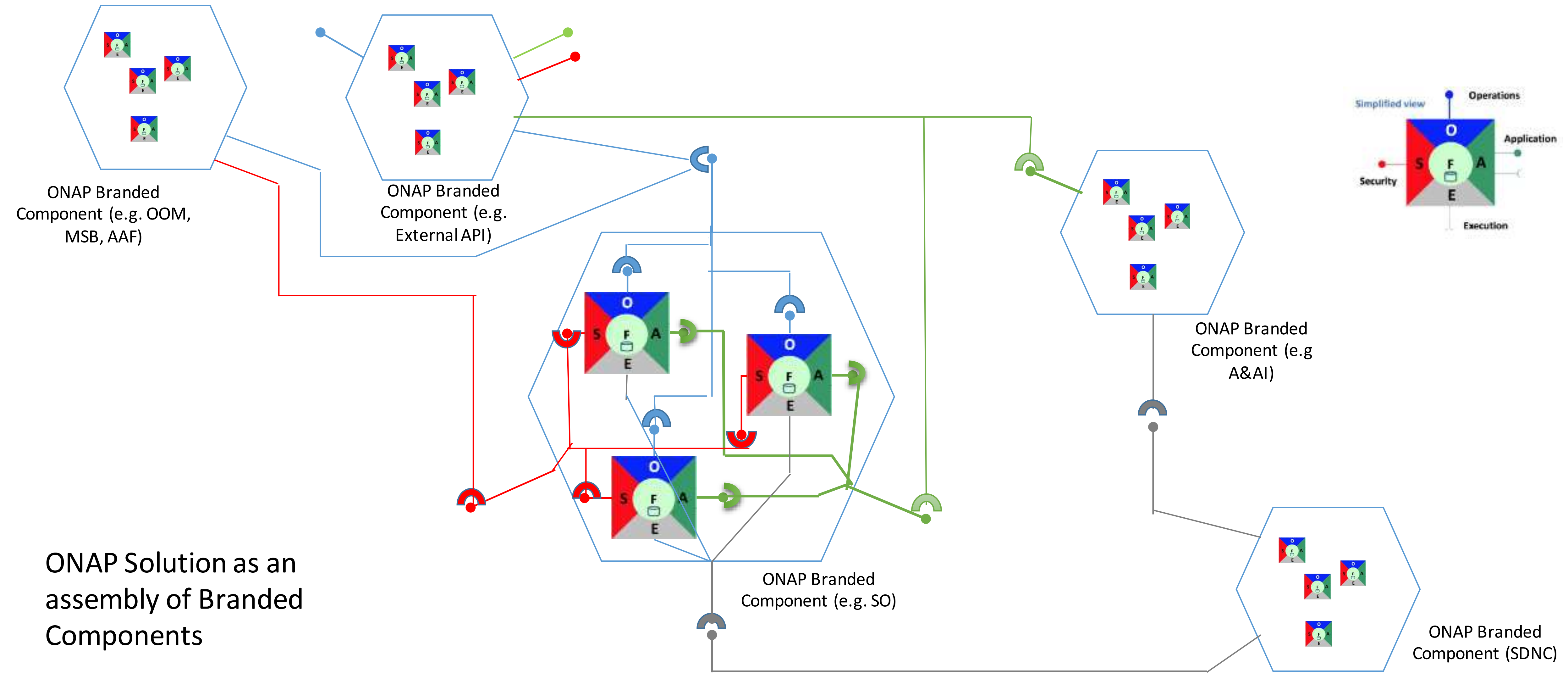
- A component-system pattern is fractal, split a component and you get more components, combine components and you get a component.
- MicroServices can be perceived as a component fulfilling a cohesive function.
- A Branded Element constitutes of multiple features which in turn are realized through set of functions – all levels can be a composite component.

Reference: TMForum IG1118

FMO Component : Key characteristics

- **FMO component is a good way of visualizing the Microservices with well defined boundaries expressed through – Operations, Application, Security and Environment Functions**
- FMO Component supports a provider consumer pattern and interfaces for offering or consuming capabilities that can be expressed through APIs.
- FMO Component may have multiple contexts – information context (e.g: software, image, descriptors), delivery context (packaging, modules), deployment context (deployable unit – as an assembly of functional components in a deployment unit), catalogue context (represented in a catalogue as a functional component along with delivery and deployment context), runtime context (component artifacts turned into runtime entities consuming resources and exposing APIs)
- **For ONAP, MCC Component can be interpreted as the basic unit of functionality that can be represented in a microservice and can be assembled to form a system or a branded component (for example DCAE).**

Relevance in ONAP – Assembly of Branded FMO/MCC Component

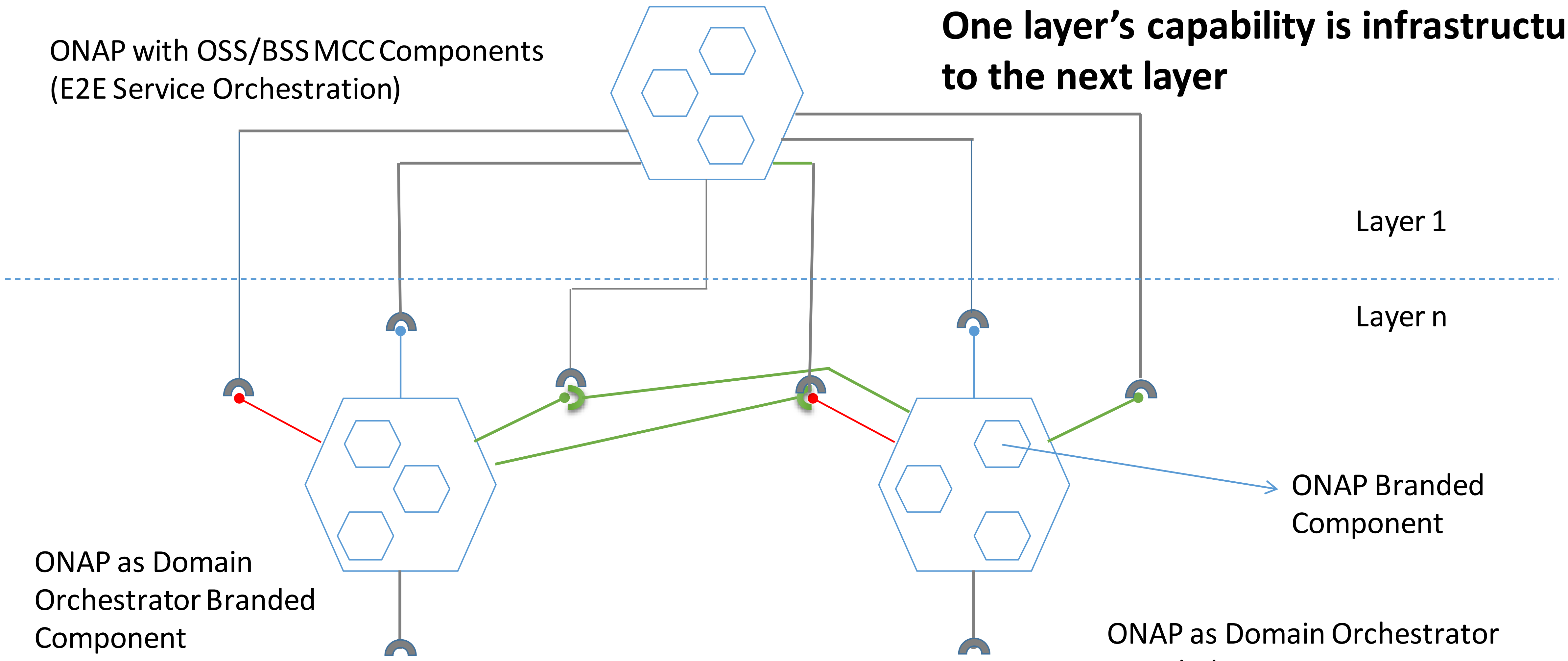


ONAP Branded Component (e.g. OOM, MSB, AAF)
ONAP Branded Component (e.g. External API)
Simplified view
Operations
Application
Security
Execution
O
S
F
A
E
ONAP Branded Component (e.g A&AI)
ONAP Branded Component (e.g. SO)
ONAP Branded Component (SDNC)

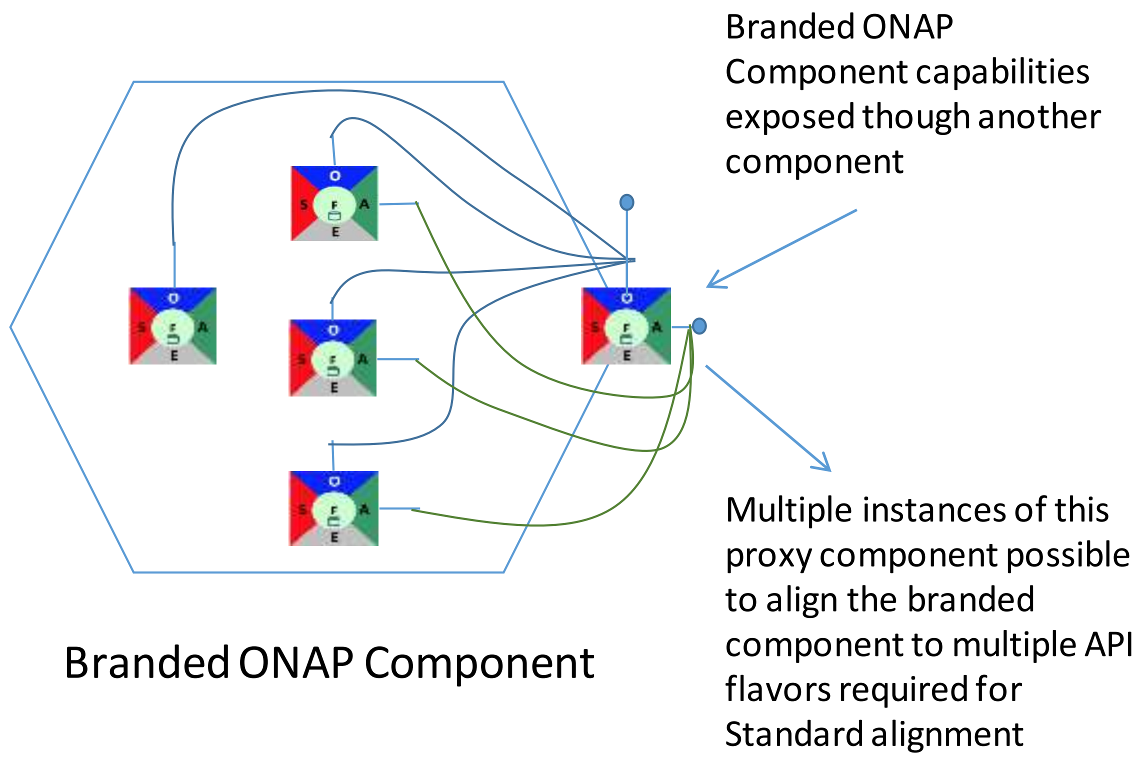
ONAP Solution as an assembly of Branded Components

ONAP as an assembly of Solution subsets

One layer's capability is infrastructure to the next layer

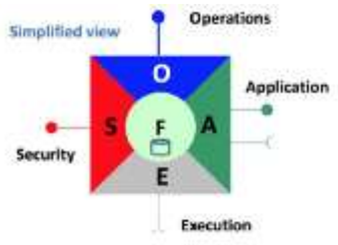

ONAP with OSS/BSS MCC Components
(E2E Service Orchestration)
Layer 1
Layer n
ONAP Branded Component
ONAP as Domain Orchestrator Branded Component
ONAP as Domain Orchestrator Branded Component

ONAP Modularity and Standard Alignment through Proxy Component



Branded ONAP Component capabilities exposed though another component
Multiple instances of this proxy component possible to align the branded component to multiple API flavors required for Standard alignment

Branded ONAP Component



- A branded ONAP component (such as SO or DCAE) can be represented through the interfaces consolidated/exposed by a single component which acts like a proxy.
- This proxy can be realized using ONAP MSB or can be dedicated microservice within ONAP Branded Component (e.g. side car exposing the component to the service mesh)
- This approach can be used to enable standard alignment. For example if two SDO interfaces need to be supported based on different deployment models, two proxy components can be created with different component interfaces.
- Modularity is ensured by the FMO component structure and the APIs exposed by micro/macro levels of components which are consolidated through proxy component

Suggested changes in ONAP

- Refactoring of APIs across components/Microservices in the Security, Operation, Application and Environment categories (Anatoly from NOKIA: This is not as easy as we think, also there may be some ambiguity which need to be cleared – Operation API for one MS may be Application for another, there is no clear guideline for classifying the API to different categories)

- Have basic minimum set of entities to be supported in microservices to enable manageability and control.

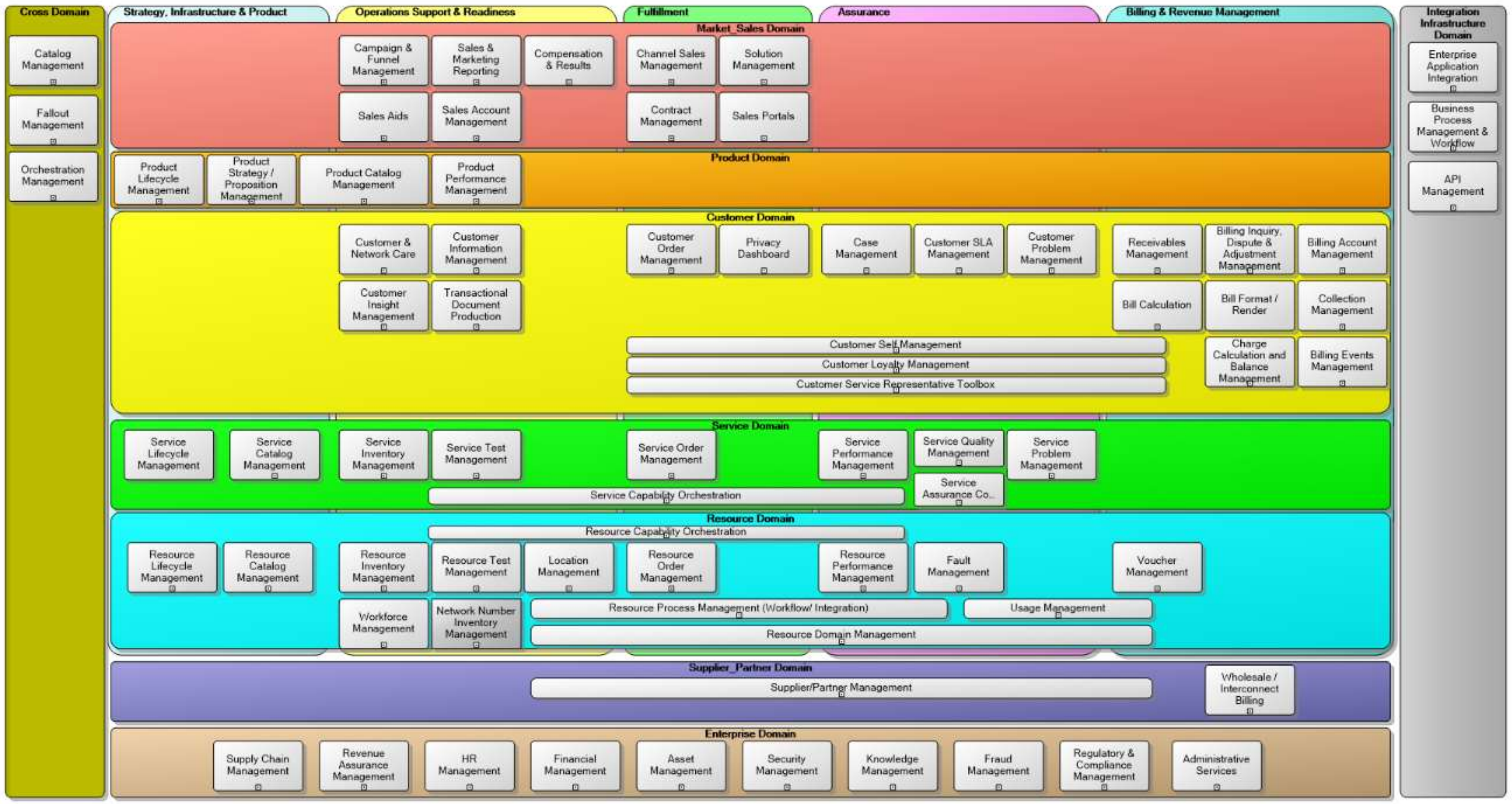
- A means for the components/Microservices to express capabilities in a catalogue to enable appropriate assembly (MSB capability limited to end point registration, not capability registration)

- Governance of interaction between components through policy (Already available through Configuration Policy ? – Policy Driven Orchestration ?)

- To have capabilities built in to components/microservices to achieve lifecycle compatibility (API/Interfaces change and upgrade support) with relevant mediation functions or adaptors

- A mechanism to express different context of Components/Microservices – Delivery context, Deployment Context, Information Context, Catalogue context, Runtime Context etc.

Modularity : How Microservice Composition can be Represented? Learnings from TMForum TAM



Cross Domain
Strategy, Infrastructure & Product
Operations Support & Readiness
Fulfillment
Assurance
Billing & Revenue Management
Integration Infrastructure Domain
Catalog Management
Fallout Management
Orchestration Management
Market_Sales Domain
Campaign & Funnel Management
Sales & Marketing Reporting
Compensation & Results
Channel Sales Management
Solution Management
Sales Aids
Sales Account Management
Contract Management
Sales Portals
Product Domain
Product Lifecycle Management
Product Strategy / Proposition Management
Product Catalog Management
Product Performance Management
Customer Domain
Customer & Network Care
Customer Information Management
Customer Order Management
Privacy Dashboard
Case Management
Customer SLA Management
Customer Problem Management
Receivables Management
Billing Inquiry, Dispute & Adjustment Management
Billing Account Management
Customer Insight Management
Transactional Document Production
Bill Calculation
Bill Format / Render
Collection Management
Customer Self Management
Customer Loyalty Management
Customer Service Representative Toolbox
Charge Calculation and Balance Management
Billing Events Management
Service Domain
Service Lifecycle Management
Service Catalog Management
Service Inventory Management
Service Test Management
Service Order Management
Service Performance Management
Service Quality Management
Service Problem Management
Service Assurance Co...
Service Capability Orchestration
Resource Domain
Resource Capability Orchestration
Resource Lifecycle Management
Resource Catalog Management
Resource Inventory Management
Resource Test Management
Location Management
Resource Order Management
Resource Performance Management
Fault Management
Voucher Management
Workforce Management
Network Number Inventory Management
Resource Process Management (Workflow/ Integration)
Usage Management
Resource Domain Management
Supplier_Partner Domain
Supplier/Partner Management
Wholesale / Interconnect Billing
Enterprise Domain
Supply Chain Management
Revenue Assurance Management
HR Management
Financial Management
Asset Management
Security Management
Knowledge Management
Fraud Management
Regulatory & Compliance Management
Administrative Services
Enterprise Application Integration
Business Process Management & Workflow
API Management

- Define Microservice Levels (Level 1, Level 2 etc) that compose domain functions
- Define well defined APIs for each level of Microservice- roughly based on IG1118 categorization.

- What problem this solves :
  - Reduce ambiguity in defining microservice boundary, ensure consistent view for MS.
  - Well defined APIs at each level makes it modular and ensures core properties of MS – Easily replaceable, Can be independently developed and upgraded

Approach 3: Cloud Native Micro Services

Cloud native microservices

- Microservices designed with cloud native principles in mind
- 12 factor principles – link (need not be cloud native, but general MS principles)
- CNCF currently is tool focused that work with COEs and dockerized containers. Heavy Kubernetes focused
- ONAP already uses some of these tools sets (Kubernetes in OOM, Tracing tools in Logging)
- What is missing ? A framework which integrates all these toolsets that can be readily used for implementing microservices (i.e consistently across microservices)
- CNCF so far has not given MS design guidelines , structure or modelling for MS other than what COE provides.
- Interfaces like gRPC/GPB seem to be quite efficient compared to REST APIs. Is there any need to migrate from REST API to gRPC?

How Cloud Native is ONAP ?

- Not all Micro services in ONAP are designed with an objective to be cloud-native. Some important characteristics that should be considered are
  - Leverage the cloud characteristics like scaling, healing and migration (handling lifecycle management operations) (Some components support this – e.g. DCAE)
  - Version and Update/Upgrade handling without impacting the functionality (?)
  - Portable across cloud infrastructure (No dependency on Cloud Infrastructure – Intel vs AMD, AWS vs OS vs GC vs Azure vs VMWare vs COE)
  - Inbuilt state management , consistency checking mechanisms (Project specific, How each project mitigates CAP- Consistency, Availability, Partition Tolerance)
  - Inbuilt health check mechanisms (Supported by OOM ?)
  - In built fault tolerance mechanisms (Supported by OOM ?)
  - Fault isolation capability (OOM/MSB?)
  - Secure interactions – internal /external (aaf?)
  - Tracing of flows across microservices (Planned to be supported by Logging?)
- Not all microservices endpoints plugged into MSB
- Each project use own model of state management and consistency mechanism across distributed instances of MS
- Scaling/Healing and Migration of MS are not consistent and in the early stages.
- No consistent deployment mechanism - Mix of blueprint, heat templates, helm charts, dockerfiles for microservice onboarding
- No consistent configuration formats – Consul , YAML templates, Helm, XML, JSON, HOT
- Version Portability (Amsterdam to Beijing) ? Lossless Software Update ?

CNCF Toolsets that can be leveraged in ONAP – Reference CNCF, IETF

Service Mesh: Istio, Linkerd, Envoy

gNMI/gRPC/ProtoBuf as enhancement for VES Collector

Prometheus for MS/VNF telemetry collection , monitoring

TUF for Secure SW update

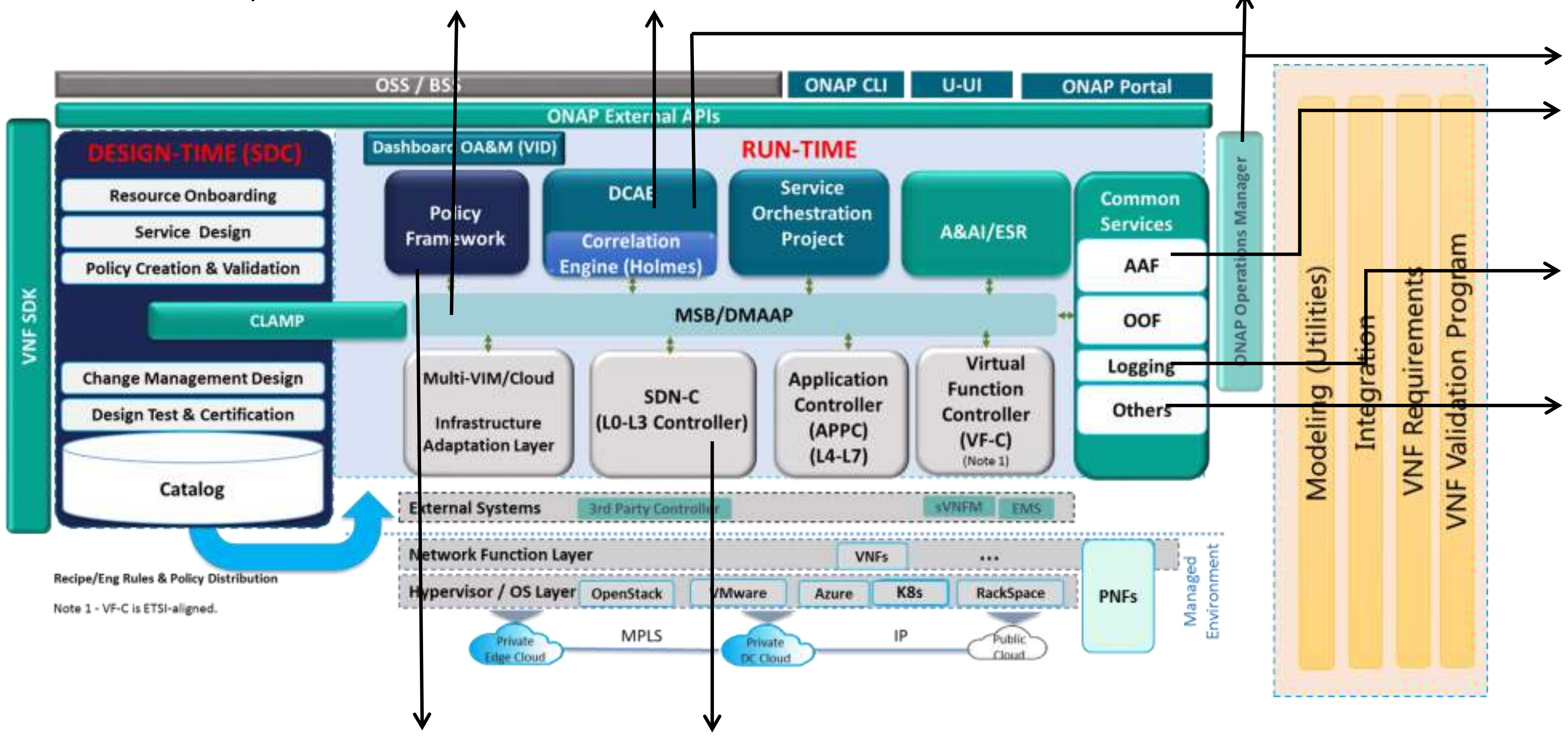
Notary, SPIFFE, SPIRE

Fluentd, Jaeger, OpenTracing

Rook for Cloud native distributed storage , which can be leveraged in MUSIC

Open Policy Agent

gNMI/gRPC, QUIC for NE/VNF Management



CNCF Projects Maturity Levels (Subjective Assessment) 1/2

Project	Relevant application in ONAP	Maturity level	Backing companies	Recommended ?
Kubernetes (COE)	OOM, DCAE	High	Google + Many others	Yes . But troubleshooting, monitoring, deployment seem to be complex.
Prometheus (Monitoring)	DCAE, OOM	Medium	Robust Perceptions, Container Solutions , Independent Contractors	Can be used for container monitoring at least for time being. Roadmap is not clear.
Fluentd (Log data collector) , Jaeger (Distributed trace analysis), OpenTracing (Distributed trace collector)	Logging	Medium (currently at 1.0 Release)	Jaeger from Uber, Opentracing – Lightstep. Github contributors – Fluentd(135), Jaeger(45)	Distributed tracing using Jaeger or OpenTracing might be useful for ONAP. (Requires instrumentation in code)
Envoy (edge proxy, Service Mesh Data plane)	MSB , ONAP Projects, OOM , AAF	High	Strong contributor base (161), Originally built by Lyft	MS proxy might be relevant while adapting service mesh. But need to debate if service mesh is good enough for ONAP. Istio Service Mesh CP works well with envoy.
Linkerd (Edge proxy, Service Mesh Data Plane)	MSB, ONAP Projects, OOM , AAF	High	Healthy contribution base (70), Buoyant	As per this comparison Linkerd seems to be feature rich , but consumes more CPU and memory compared to Envoy. Supports HTTP/1.1, Thrift, ThriftMux, HTTP/2 (experimental) and gRPC (experimental). MSB team to comment suitability of Linkerd.
Istio – Service Mesh Control plane	MSB, OOM ?	High	Google, IBM, Lyft.	Preferred if there is a plan to adapt Service Mesh in ONAP. How this will be integrated ? As a separate project ? As part of MSB/OOM ?

CNCF Projects Maturity Levels (Subjective Assessment) 2/2

Project	Relevant application in ONAP	Maturity level	Backing companies	Recommended ?
Notary (secure content sharing) , TUF (secure software update spec)	AAF (Not sure), VNF SDK, OOM (software update – to be confirmed), SDC Package manager (to be confirmed)	Low (Not yet in a major release)	Healthy contributor base (60+)	AAF team or Security subcommittee to comment . Might be relevant for software upgrade, image upload, Marketplace, VNF SDK (all external facing)
Vitess (Clustering of MySQL, Sharding)	Many projects – that use MySQL or MariaDB	Medium	Healthy contributor base (100+)	Project teams to comment. May be relevant for distributed deployment of ONAP components, Scalability enhancements etc.
Rook (Cloud native distributed persistent store)	Music, Other projects that require distributed persistent store	Low (Not yet in major Release)	No known companies . Good contributor base (70)	Not recommended. Low maturity and no known deployment.
SPIFFE (Spec)/SPIRE (Implementation) – Service Identity Mgmnt	MSB,AAF (Service Identity Management)	Low	Scytale , Istio	No. Similar capabilities in MSB and AAF. But low maturity level. Istio-Auth seems to incorporate this. AAF and MSB to comment.
Open Policy Agent	Policy, MSB	Low	No known companies	No. This is primarily meant for controlling the MS behavior through policy with a daemon deployed along side the service . Not recommended immediately due to maturity issues. Based on maturity can be considered to position this as a PDP per component (Policy team to confirm)
NATS (NAT Server) - Messaging, Publish/Subscribe, Request Reply, Queuing	MSB	Medium	Healthy contributor base – used by Ericsson, HTC, Siemens, VMWare	Require evaluation. Similar capabilities like MSB. Supports different types of communication models. Uses a proprietary messaging format (gnatd) which is claimed to be efficient. May be useful if we want to unify communication models under single project (but will require lot of refactoring)
Core DNS (DNS functionality, Service discovery)	MSB , DNS VM	Medium	Healthy contributor base (80+). Used by Sound Cloud, MIT	Require evaluation. Already part of K8S v 1.9 onwards. Need to see if the scope is limited to K8S cluster and containers.

gRPC/gNMI/gNOI Views

Characteristic	gNOI	gNMI	gRPC
What is the focus	gRPC based Operational interfaces to network devices	gRPC based protocol for modification and retrieval of configuration from target device	Generic RPC mechainsm
What is available ?	Microservices which can be reused for enabling the interface	Client library implementing network management interface	Many implementations in different languages
Who is driving ?	Open Config	Open Config	Driven by CNCF
Contributors ?	Limited	Limited	Many
Where is it relevant in ONAP	SDNC, App-C, VF-C, DCAE (all SBI)	SDNC, DCAE, App-C, VF-C (all SBI)	Between MS , ONAP and external systems
Dependency	gRPC, protobuf	gRPC, protobuf	gRPC Stub/Service implementation , protobuf. HTTP/2
Why it is relevant ?	gRPC based, efficient communication, ready to use operational protobuf specs	gRPC based, efficient communication, ready to use operational protobuf specs	Bidirectional, Asynchronous, Can be used for notifications or RPCs, low latency, can support many communication patterns , Can also support REST HTTP1.1 (proxy) . Very low dev overhead.
Recommended for ONAP ?	Need to check how many VNF vendors support this. **May not be immediately**. SDOs do not seem to endorse this yet	Need to check how many VNF vendors support this. **May not be immediately**. SDOs do not seem to endorse this. yet	May be some wrapper rather than exposing gRPC directly to MS ? (Service Mesh ?)

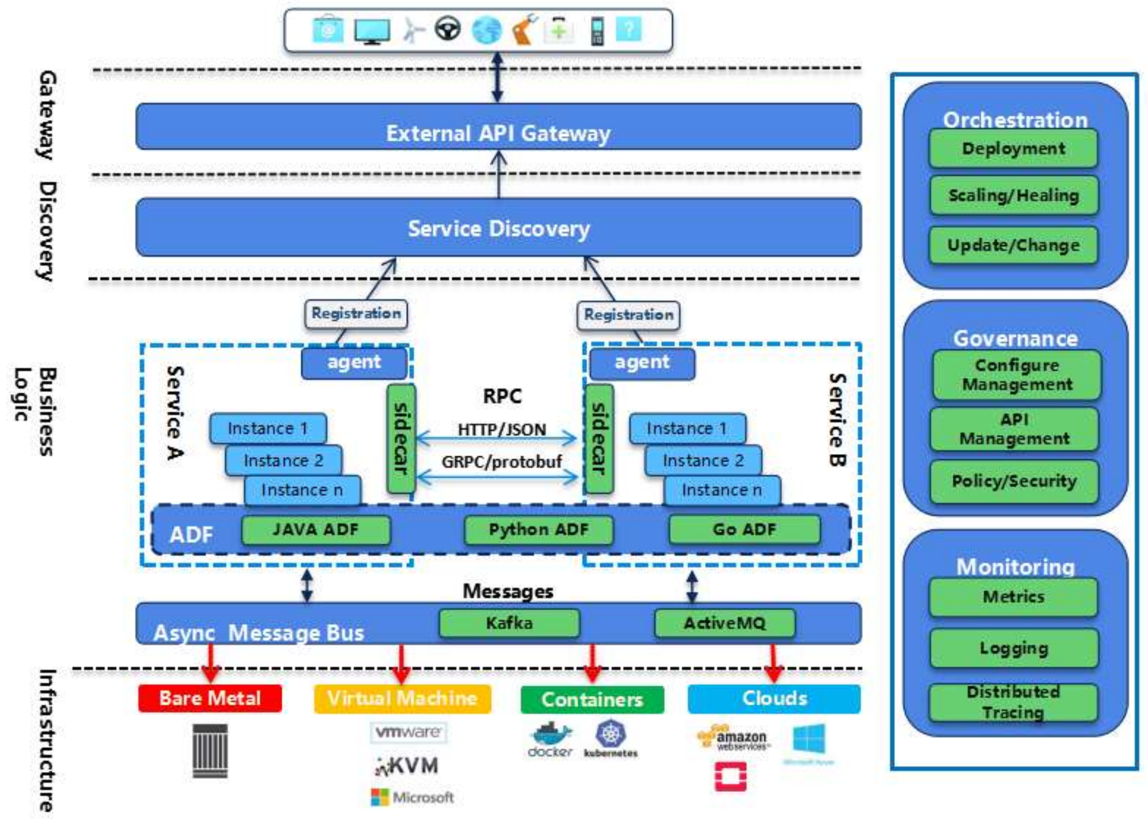
All options above may require Kubernetes

gRPC vs REST

Characteristic	gRPC	REST
Performance	Impressive	Relatively slow
Development overhead	Low (Stubs can be auto generated)	High
Limitation	Version management, Compatibility, Requirement for retooling , Stub/Service Management	Processing overhead
Compelling capabilities	Low development overhead, performance, communication patterns	Wide adoption, tool sets, Developer familiarity
Where it best fits	Internal ONAP MS communication	ONAP External communication

OMSA (By MSB team)

OMSA-ONAP Microservice Architecture



OMSA is the vision of ONAP Microservice Architecture to support carrier-grade requirements of ONAP Microservices, which includes service registration/discovery, service communication, API gateway, service orchestration, service governance and service monitoring.

Note: this diagram is a functional view of OMSA, which is not mapped to specific projects

OMSA Observations

- OMSA is primarily an implementation approach than dealing directly with the Microservices design principles –Modularity, Model Driven, Independent development, consistency concerns.
- Limited Scope – i.e covering the Micro Service discovery, Interaction . How about nature and boundary of microservices ?
- Summary :
  - Good for short term benefits
  - Scope of MS should cover more than just inter-MS interaction and discovery

Summary of views

- All approaches bring-in significant practical value
- Among the three approaches , Approach 2 seems to be more favorable considering the ease of adoption and the modularity that can be achieved
  - Approach 2 might be favorable from a component level standard API and model alignment point of view as flexible proxy components can be supported
  - Approach 2 can also leverage some of the concept of bounded context pattern described in approach 1
  - Approach 1 might require a major revamp across projects - the way Model states are maintained, propagated and acted upon. So this approach may be considered for long term.
- Approach 3 is more of an implementation approach with focus on discovery and interaction of MS. MS still need to be structured well, consistent and follow cloud native principles to leverage benefit of this approach.
- Approach 3 seem to give some immediate benefits and may be considered for short term, while other two approaches may be considered for future releases.

Recommendations for Casablanca

- Right mapping of Micro Service – Pod ? Service? Docker ? API Endpoint ? Component ? Project ? (or TMF IG1118 based Component System pattern)
- Model Driven Micro Service : (Can be for long term- based on community view)
  - Central model repository referenced by all components
  - Common library of model base classes used by components
  - Anything Else ?
- Modularity of Microservice (Can be for long term- based on community view)
  - API Refactoring – Consistency in representation of APIs across MS and components
  - Policy tailoring all interaction between microservices/components.
- Communication between Microservice
  - Service Mesh or gRPC or RESP or DMaaP or REST ?
- Testability of ONAP components based on Cloud Native Principles (enhance S3P)
- Define guidelines for creating cloud native microservices
- Microservices Meta Modelling through SDC ? (to ensure consistency)

Recommendations for Future Releases

- Best practices incorporated from Approach 1 and/or Approach 2
- How Microservices Architecture support following
  - Different deployment models – Brown field, Green field
  - Different Integration models – point to point , geo distributed
  - Different testing models – unit testing, integration testing, CI/CD
- How following requirements are supported
  - In service software update
  - ONAP release upgrade
  - Model/Policy/Workflow tuning
  - Troubleshooting – Remote, Local, ONAP specific OAM KPIs