FUTURE OSS

WHITE PAPER

ISSUE: REV1.0 DATE 05/02/2017

orange™

Future OSS

Providing the AGILITY to support digital operations transformation of hybrid networks

HUAWEI

CONTENTS

ISSUE: REV1.0 DATE 05/02/2017

orange

HUAWEI

orange

HUAWEI

EXECUTIVE SUMMARY

"Providing the AGILITY to support digital operations transformation of hybrid networks"

Advances in technology and competition from agile Over-The-Top (OTT) players means that Communication Service Providers (CSPs) face significant threats and challenges to their current business models. CSPs must digitally transform their operations to head off this OTT threat.

In 2015, Orange unveiled a five-year strategic plan, "Essentials 2020" to address this digital challenge. That plan identified a major underlying goal, namely ***Agility***, in order to be always responsive and in touch with what is essential for the customer. This covers all dimensions (marketing, sales, shops, customer care…) including the evolution of operations to the Future Mode of Operation (FMO).

The collaboration between Orange and Huawei in this area started in 2015 with a TMF Catalyst Project. This proof-of-concept project – Model-driven Service Orchestration via FMO Architecture - won three consecutive TMF Awards: "**Best Adoption of Frameworx"** in 2015**,** "**Greatest Contribution to the Evolution of TM Forum Assets**" and "**Best Contribution to TMF Assets**" in 2016. Orange and Huawei decided to leverage this Catalyst success to develop a shared architectural vision that details how to design and build a Future OSS (Operations Support System) to provide agility.

The Future OSS will demonstrate the opportunity to implement a unifying architectural framework for all the complementary paradigms: Physical, SDN, NFV, Autonomic Management and Control, E2E Orchestration of Services and Resources, Assurance platforms, and Big Data analytics for network management & control, which have so far been developed in silos.

The Future OSS is a new and holistic approach to enable agility and create an operating model for CSPs like Orange working with suppliers such as Huawei. This is a top-down approach that defines the management of hybrid networks (virtual and legacy) and has a more complete vision compared to pure virtualized networks taken by other blueprints in the industry.

In the context of this white paper, the Future OSS **is *not*** a monolithic system; rather, it is a dynamic set of applications that are combined and extended as needed to address the needs of the business. The Future OSS ***cannot*** simply be a rebadged, virtualized onto cloud infrastructure version of what CSPs may already have. The Future OSS must instead be truly cloud-native, resilient and designed for the unpredictable failure modes of the cloud.

The Future OSS must have the following capabilities to support digital transformation and enable agility:

- Operating models with Continuous Integration/Continuous Delivery (CI/CD), Joint Agile Delivery (JAD), Site Reliability Engineering (SRE), DevOps incorporating DesignOps (e.g. Visual Design Studio) and technology best practice throughout the cycle of Think, Build and Run. DesignOps manages the communication and translation of requirements between design and DevOps teams
- Intelligent and progressive operational automation, self-management, and self-configuration
- Usage of microservices, metadata and open APIs

The Future OSS defines an architecture framework as shown below:

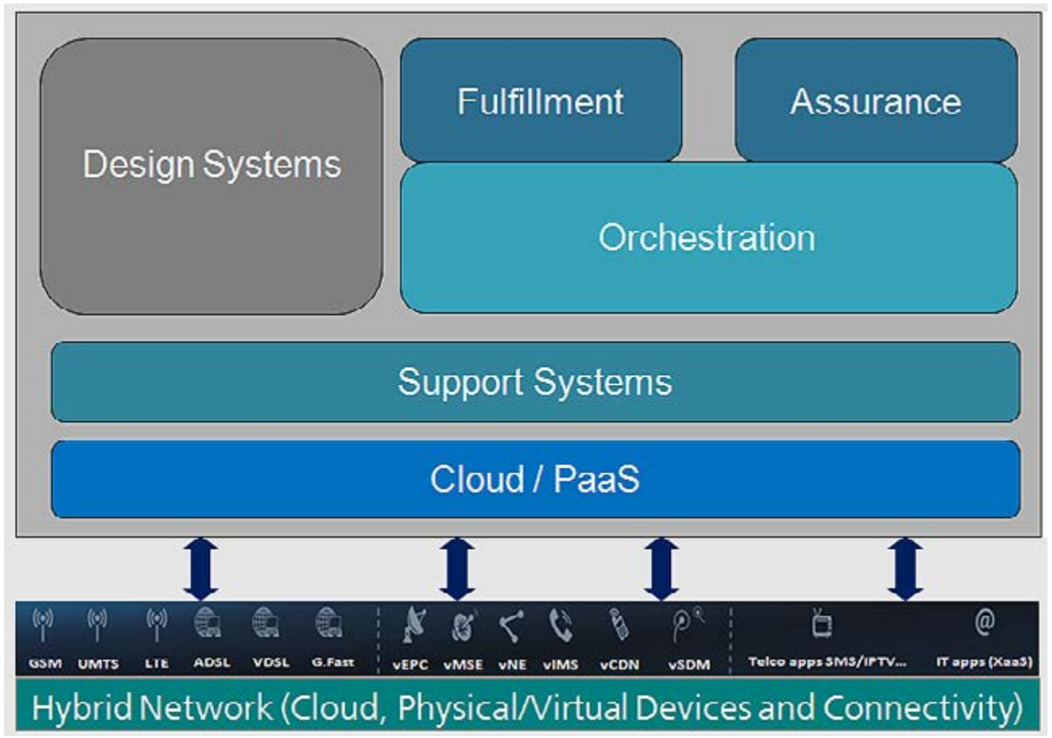


Figure 1: Future OSS Functional Architecture

The Future OSS architecture is designed to provide a common digital platform for the management of hybrid networks and services, including the following domains:

- **Cloud/PaaS** is the cloudified-based platform of the Future OSS
- **Design Systems** enables DevOps, DesignOps and model-driven design
- **Support Systems** bridges the Future OSS Design-Time and Run-Time via Dynamic Inventory, Catalogue (all information including policy, used for designing and automating services and resources), Policy Management, and API Management
- **Orchestration** automates and accelerates all actions necessary (both cloud and legacy) to provide end-to-end service management for cost effective hybrid fulfillment and assurance. Data Collection and Analytics modules within Orchestration extend this support to allow intelligent closed loop operation
- **Fulfillment** provides the link with existing OSS/BSS (Business Support System) implementations to transform business requests into solutions
- **Assurance** ensures service quality via dynamic operations and automation of manual, time consuming tasks

The Future OSS is to be used for on-boarding Virtual Network Functions (VNFs) and for working on top of existing legacy OSS, with progressive legacy decommissioning. It is important to have a harmonized approach in the industry, starting with a common understanding – the purpose of this white paper.

The Future OSS end-to-end holistic approach offers an architecture that can be adopted by the industry as a reference in moving towards comprehensive digital operations, thereby allowing CSPs to be agile and suppliers to focus on new functions or technologies rather than integrations to many specific environments. The Future OSS forms a framework within which to view initiatives like the TMF APIs, and open source contributions such as the Open Network Automation Platform (ONAP) and Open Daylight (ODL).

INTRODUCTION

The Need for a Future OSS

Many Communication Service Providers (CSPs) see an urgent need to overhaul their back-office systems and processes to prosper in a new digital services future. Most initiatives in the Telco domain today mainly focus on virtualized networks, whereas the industry challenges are about hybrid networks and convergence of Communication Technology (CT) and Information Technology (IT).

During the last three years, Orange and Huawei have established a strong collaboration in the OSS area, including the advances and lessons learned in the TM Forum catalysts [1]. These catalysts focussed on hybrid network management using policy-driven orchestration.

Moreover, both Companies identified the need to break the current silos and harmonize on standards efforts in the complementary paradigms of SDN, NFV, Autonomic Management and Control, E2E Orchestration of Services and Resources, Assurance platforms, and Big Data analytics for network management & control, through a non-formal unified Multi-SDO Industry initiative [2].

Scope and Content of the White Paper

This White Paper focuses on the key principles which are required to deliver agility. It defines an architecture framework for the core domains, functional modules, and their supporting concepts and interactions. The structure of the White Paper is shown in the figure below:

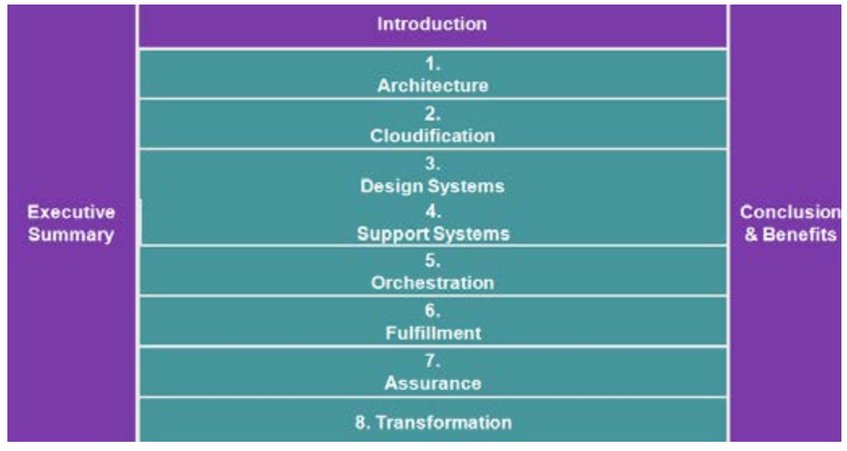


Figure 2: Future OSS White Paper Document Map

Architecture provides an overview of the key principles of the Future OSS to enable agility, which includes a common information model, closed-loop control and autonomic operations, data federation, and API Management functional modules.

Cloudification looks at the impacts of cloud technologies on OSS, both from the point of view of the Future OSS platform itself and also the delivery of services and operating models that support it.

Design Systems covers all OSS functions of service, resource, policy, and analytics design, and proposes a Visual Design Studio to support their development.

Support Systems define how the Catalogue, Policy Management, Dynamic Inventory, and Adaptation modules (in particular the API Management and Common Event Bus components) act as a bridge between the Think, Build and Run of the dynamic services required.

Orchestration leverages the models, policies, and metadata required to support end-to-end closed loop fulfillment and assurance. It defines how the Data Collection and Analytics modules provide context information to determine the optimal, cost effective course of action.

Fulfillment covers the feasibility of accepting an order so that it can be delivered while meeting and exceeding the required SLAs. It focuses on reservation of the resources required to fulfill orders, order lifecycle management, and order status tracking within the Future OSS.

Assurance defines the application of intelligence to manual operations, to automate in the event of fall out from Orchestration. It investigates the dynamic applications, Visualisation and Operations required to ensure service quality levels are maintained.

Transformation examines the fundamental changes and impacts that CSPs will embrace to migrate from legacy to Future OSS environments.

1. ARCHITECTURE

"Designing a Common Digital Platform for Hybrid Networks and Services"

1.1 Introduction

The issues that influence today's major costs, burdens and latencies in OSS include:

- Many interlinked, but disparate and non-standardized systems, each requiring code, configuration, API, and/or metadata updates, all with long delivery cycles
- Specialized software for similar functions, limited by application silo, layer, and/or geography
- Different definitions of standards and "openness"
- Traditional systems have no way of interpreting high level requirements and translating them into resource and service offerings, and therefore tend to solve all problems by requiring more low level complexity – this burdens both human operators and developers, often requiring repetitive and manual work with similar functions developed several times

Overall, the present Mode of Operation can be characterized by weak capabilities in agility. This manifests itself in increased Time to Market (TTM) and higher Total Cost of Ownership (TCO).

To address these problems and achieve the agility needs new thinking and transformation in the characteristics and architecture of OSS.

Goals	Architectural and Process Requirements
Quick Time-To-Market	Model-driven processes - DevOps – DesignOps
Open Ecosystem	Open APIs [3] - Federation of systems - Everything-as-a-Service (XaaS), e.g. PM-as-a-Service, FM-as-a-Service, etc.
Process Automation and Autonomics	OODA Control Loop - Analytics - Policy-based decision making – Knowledge Plane
Adaptable Technology	Common Information Model - Cloud-based computing platform – Event and microservice-based - Future Mode of Operation (FMO) architecture

Table 1: Future OSS Architecture Goals and Requirements

Orange has devoted research resources on the above topics especially on the topic of a global orchestration domain [4].

1.2 Future OSS Architecture Concepts

Common Information Model

An extensible, flexible and structured information model must underpin the capabilities offered by the Future OSS. This is essential for providing comprehensive, common and reusable functionality, and also to respond in an agile manner to change. The model should also have the following basic properties:

- Re-uses and adopts Industry Best Practices, Information Frameworks, and Modelling Patterns, with the goals of interoperability, extensibility, and configurability through metadata. This enables application simplification and the ability to dynamically create new functionality without having to recompile and redeploy
- Allows for loose-coupling between components. This means that the components adopt a common information model definition, to minimise adaptation effort when synchronizing changes
- Permits the inter-relationship of data and metadata, governed by policies, such that all relevant objects may reuse similar “blueprints”
- Enables multiple data models, each targeted to a particular type of operation, to be used while maintaining data coherency and synchronicity of object definitions

The Future OSS uses concepts which are common to a number of Information Models without advocating any specific model, as a different level of abstraction may be needed, new models may be necessary and CSPs may have different approaches. For instance Orange is aligned closely to the TMF Information Framework (SID) [5] through its own Orange SID. However, harmonisation of information models is clearly important for inter-vendor and inter-carrier understanding and ease of communication via open APIs, promoting provisioning of services across CSPs/DSPs (Digital Services Providers). The information model may give rise to a number of specific data models for different implementations (e.g. in specific domain orchestration systems) but models used by legacy systems cannot be easily changed and may need more extensive adaptations and wrapping as necessary.

Model-driven and Intent-Based Processes

The Future OSS Common Information Model is independent of platform, language, and protocol. It provides a set of common concepts and terminologies that all components use. This extends to the modelling of processes within the Future OSS. This principle is well established, and must also guide communication through policy-based, metadata-driven, open APIs. The model must also address intent-based processes, where the requirements of a service, resource or API can be specified without needing to define how the process is

implemented, such as the need of a service to provide a certain Quality of Service (QoS) between two points. This separates the specification of a function from its implementation, and allows expressing process requirements in a more portable and interoperable form.

Model-driven principles form the basis of a number of open source implementations including ONAP [6] as presently being trialled by Orange.

Federation of Data and Data Model

The Future OSS must be able to operate in a distributed environment, for instance where a CSP has to provide a Service through another CSP’s network, but this must be made more intelligent and cooperative than before. Traditionally, data models are optimized for different repositories (e.g. relational database vs. NoSQL store vs. directory), and hence, are dependent on the platform, language, and protocol used; this allows the OSS to take advantage of the characteristics of each type of repository. The Future OSS must allow as many different repositories as needed, but maintain coherency among them. This requires the use of a single information model and a set of semi-automated translation algorithms that transform the desired subset of concepts in the information model to targeted data models and domain specific languages such as TOSCA and YANG.

The data models are federated to form a cooperative mesh of information across systems, rather than attempting to massively transform data from one system to another. Federation techniques also depend on open APIs to form a topology of relationships between catalogue, inventory, and instance data. Policies are used to ensure that these specialized repositories are updated and used by authorized users. This topology supports the dynamic and intent-based relationships that are needed to implement dynamic inventory, dynamic catalogue, and dynamic orchestration of fulfillment and assurance.

An example of using a federated model to unify provisioning over different SDN architectures has been recently demonstrated as part of an Orange, AT&T, Colt, TMF and MEF collaboration, using the MEFs Lifecycle Service Orchestration (LSO) framework and the TM Forums Open API framework [7, 8].

Hybrid Operation and Everything as a Service (XaaS)

A key part of achieving an open ecosystem is the adoption of an “Everything-as-a-Service” (XaaS) model by the Future OSS, which allows both new (e.g. VNF-based systems) and legacy (e.g. physical networks accessed by EMS) resource pools to be managed by the same Future OSS platform. This XaaS model also extends the service concept beyond traditional network and compute resource domains to include, for example, OSS components as a service. This includes, among others and at a high level, Fault Management as a Service, Performance Management as a Service and Work Force Management as a Service. This is also a way to harmonize different operations using the

same XaaS platform.

Each process, and its appropriate components, are adapted into the API, catalogue, and policy-based, metadata-driven framework, incorporated in the appropriate closed loop processes, and offered via a consistent set of northbound open APIs.

Closed Control Loops and Autonomic Operation

The process automation of the future must be part of a path towards an autonomic based system that is aware of its operating environment and able to respond dynamically to observed changes. The Future OSS adopts the Boyd cycle of decision-making, also known as the “Observe, Orient, Decide, Act” (OODA) loop. Use of OODA [9] permits a level of flexibility in adopting a variety of new levels of autonomics (inclusive or not of human involvement) [10].

This white paper defines the use of one or more OODA-based control loops, including nested control loops, and their integration with model-driven processes/workflow and decision-making software. The term “decision-making software” includes mechanisms such as expert systems and more generic Artificial Intelligence (AI) technologies (e.g. machine learning). A run-time view of the OODA loop illustrates the approach as shown below:

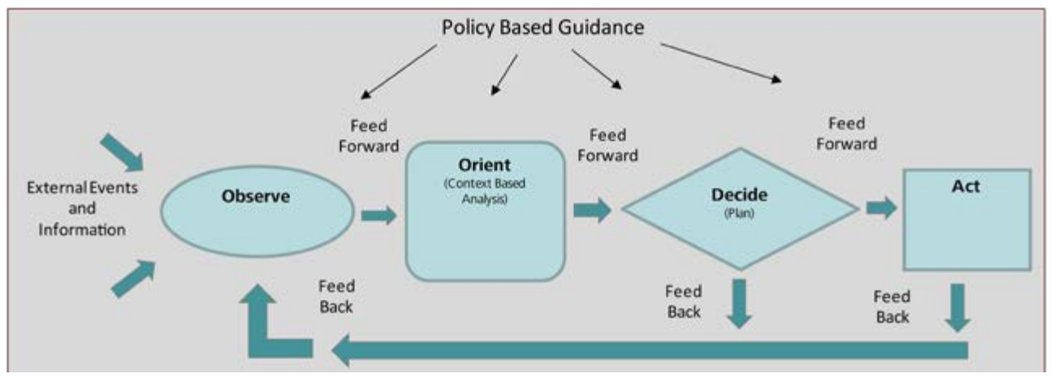


Figure 3: OODA Run-time View

The feed forward of information progresses from left to right. The Observe phase reacts to new events, and gathers data from one or more sources in the environment. The Orient phase normalises these data, and develops understanding of the situation context by using knowledge-based techniques. The Decision phase analyses one or more possible courses of action based on the situation - taking into account the likely negative or positive impacts of each path, and then feeds into the Act phase to carry this out.

A significant feature of OODA loop is its use of feedback. The feedback process is designed to change the way that the system reacts to data, and complements the

knowledge acquisition process. For example, feedback may be used in situations that require gathering additional data on a fault to more precisely determine its root cause (possibly by designating events to be collected, for instance, in a trouble shooting phase). In other cases feedback may function to suppress further events that represent the same information. It could also be used to gather additional data after acting to determine the effectiveness of the actions. As a result of this new actions may also be designed to have a better fault handling outcome.

The description above does not apply solely to externally managed applications or network. This feed forward and feedback process forms the basis for the self-monitoring, self-configuring, self-healing, self-optimising, and self-protecting features that collectively make the Future OSS more robust and agile.

In all phases, different types of policies are used to guide the processes, as dealt with in the Support Systems Domain. Within the Future OSS, operations are transformed and focussed on policy design. The Orchestration section provides more details on the mapping of the OODA steps to functional components.

1.3 Future OSS Architecture

A structured approach is needed to build an architecture that will permit the separation and cooperation between system components. The diagram below describes the Future OSS architecture:

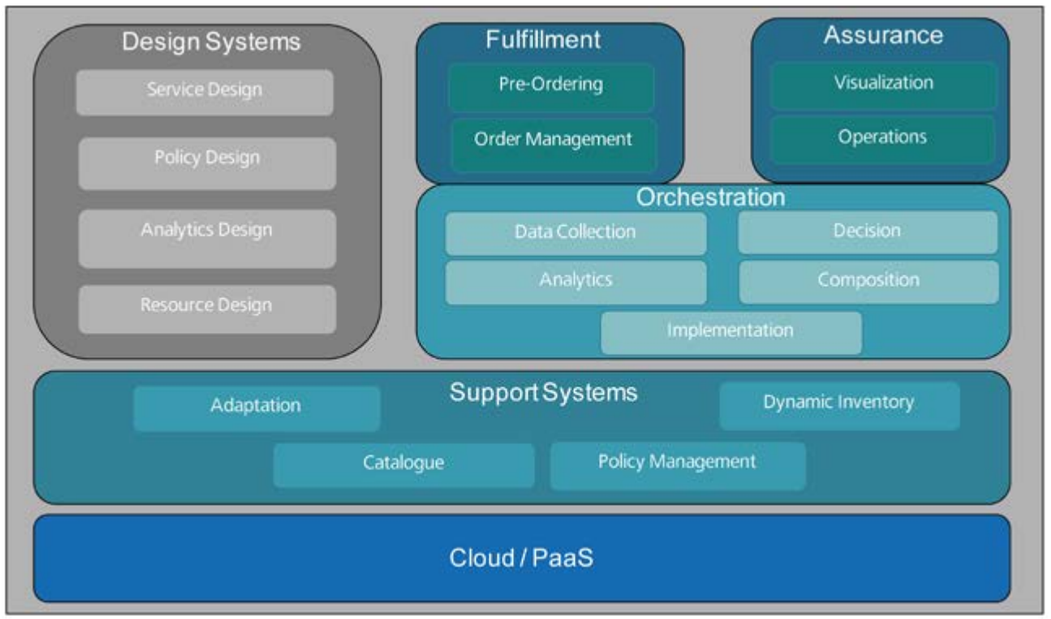


Figure 4: Future OSS Functional Domains and Modules

Design Systems

Design Systems provides an overall idea/portfolio management and on-boarding environment. This is primarily focused on the ”front-end” creation, assembly and management of new or varied services and their dependent artefacts. Design Systems is comprised of:

- **Service Design** – manages service on-boarding and definition (using the Catalogue, linking with other artefacts, and packaging of the complete service metadata)
- **Resource Design** – manages resources (reusable objects) on-boarding and definition (including interaction with legacy resources)
- **Policy Design** – manages the definition of how the system reacts to events (that need to be defined) and reasons
- **Analytics Design** – designs and manages a library of analytics models and functions used to discover and understand data as well as patterns in data

Support Systems

Support Systems act as a set of common capabilities, providing the back-end management and full consistency of the design processes as well as the run-time processes (Fulfillment, Assurance and Orchestration). Support Systems is comprised of:

- **Adaptation** – manages the library of API definitions, and the event handling and mediation process
- **Dynamic Inventory** – provides a federated cross domain and real-time view of services and resource instances
- **Catalogue** – describes and manages the lifecycle of all models including Service, Resource, Analytics and Policy metadata across domains
- **Policy Management** – manages cross domain policy validation, enforcement and verification

Orchestration

Orchestration coordinates and automates the execution of processes that complete various tasks, including configuration, monitoring, fulfillment, and assurance. However it goes beyond some generally used definitions of orchestration to include the implementation of OODA-based control loops, driven by context-based analysis of events. Orchestration uses policies and metadata to:

- Collect data, analyse a situation and decide on a response
- Arrange, sequence and implement tasks based on policies and rules/recipes to coordinate the creation, modification or removal of logical and physical resources in the managed environment
- Execute models/workflows/recipes to manage the completion of tasks necessary to create, modify, and retire services

Orchestration is comprised of:

- Data Collection – determines which data to collect how and when, normalises event data to a common format, and loads the data onto a Common Event Bus
- Analytics – receives normalised collected data, applies contextual information, and uses a variety of techniques, including rule-based expert systems, machine learning and inferencing, to extract actionable context-aware intelligence
- Decision – determines the best course of action based on the analysis using policies
- Implementation – translates the overall course of action into its component steps, coordinates the required actions (including the invocation of controllers and orchestrators) and verifies the results

Fulfillment

Fulfillment handles the interactions with external order processes and management of those orders. Fulfillment is comprised of:

- Pre-Ordering – manages order feasibility and reservations
- Order Management – manages order lifecycle and tracking. This may include non-automated actions (such as a work order for CPE installation)

Assurance

Assurance provides the management of the longer-cycle and user-facing Assurance operations. Assurance is comprised of:

- Visualization – provides dynamic views of status, configuration and ongoing analysis
- Operations – manages coordinated and collaborative Fault, Incident and Problem Management through dynamic applications, making the link with DevOps and DesignOps

Cloud/PaaS

The Cloud/PaaS domain performs virtualisation management of storage, computing, and networking that are necessary to achieve cloudification of the Future OSS. It supports the self-management and resilience needed for a dynamic and real-time OSS by providing a

platform that is distributed and can flexibly scale and heal problems.

1.4 Example Use Cases

The descriptions of these six Future OSS domains are further developed within the individual domain chapters. The basic roles of the Future OSS components can be best illustrated through the two simplified use cases below:

Use Case 1: Redesign, orchestrate and assure a new virtualized VoLTE service option for an existing VoLTE service

This use case allows a CSP to reduce CAPEX and OPEX for handling peak demands on VoLTE Services. The Future OSS helps to achieve this by:

1. **Support Systems** will enable rapid discovery of VoLTE resource definitions (including pre-designed and supplier provided) and topology accessed from the Visual Design Studio:
   - ***a.*** Discover and on-board physical and virtual resource definitions using Catalogue and Adaptation
   - ***b.*** Discover and integrate virtual and physical data and topology using Dynamic Inventory and Adaptation

2. **Design Systems** will enable rapid E2E VoLTE service design
   - ***a.*** Design and create new hybrid VoLTE service options
   - ***b.*** Design the VoLTE service analytics models (traffic, capacity, KPIs, etc.)
   - ***c.*** Design the VoLTE service policies to enable automated scale-in and scale-out, including fall out to manual operation in case of issue
   - ***d.*** Design and execute VoLTE tests using JAD (joint work between CSP and supplier)
   - ***e.*** Deploy VoLTE Service to production (Future OSS Run-Time) using JAD in cooperation with Support Systems

3. **Orchestration** will deploy the new feature (instantiating the virtualized VoLTE components, adapting the physical EPC). It will use the new VoLTE service models to rapidly scale-in and scale-out the VoLTE service, providing real-time autonomic assurance following the OODA principles

4. **Assurance** will handle manual operations in the event of repeated failures and fall out (including roll back to a previous more stable state, trouble shooting, and designing updated policies to close the loop)

Use Case 2: Design and fulfill a new virtualized vEPC network slice service

This use case allows a CSP to gather new revenues from its existing network by offering a

compartmentalized slice of its virtualized EPC network. The Future OSS helps to achieve this by:

1. **Support Systems** will enable rapid integration of the MVNO customer

2. **Design Systems** will enable the rapid EPC network slice service design

3. **Fulfillment** will enable the coordination and tracking of the order for the virtual network slice
   - ***a.*** Check feasibility in terms of network coverage, service options
   - ***b.*** Create and manage service and resource order lifecycle

4. **Orchestration** will provide the configuration, implementation and verification of the network slice service

5. **Fulfillment** will provide status update/completion notifications

6. **Orchestration** and Assurance will provide automated and manual assurance of the network slice

<u>Control and Knowledge Plane</u>

The split into multiple OSS/orchestration domains is inevitable, both between and within service providers. In Orange, technology domains for large countries may also be operationally separated. Multiple cooperating domains may together form a “master” control plane to ensure that services and resources in different domains, which use different technologies, can have their actions coordinated. The knowledge acquisition and gathering functions described above form a knowledge plane [11, 12] and ensure that changes encountered in the managed environment are understood and implemented using policies and metadata. Legacy OSS may also be adapted into the knowledge plane and control plane through the Support Systems, as shown in the Figure below:

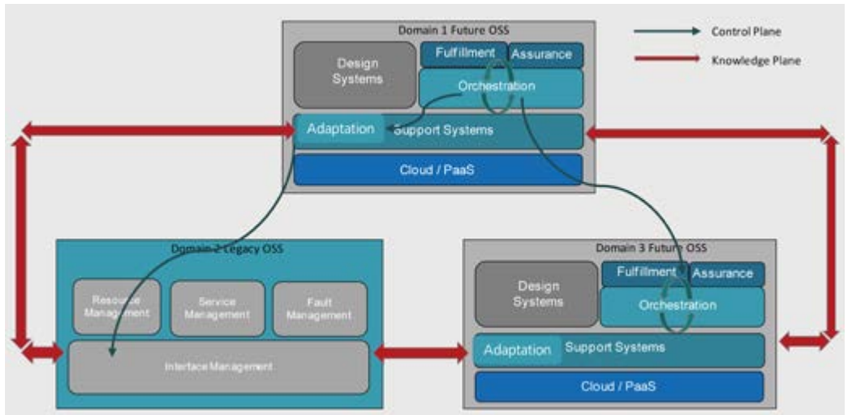


Figure 5: Knowledge and Control Planes

1.5 Mapping to ETSI MANO

The Future OSS covers functionality and vision well beyond the scope of NFV, however the ETSI MANO framework [13] is still a highly relevant mapping for the deployment of the Future OSS architecture in such environments. The TM Forum and Orange have further studied OSS/ BSS requirements for hybrid networks including NFV [14] within the ZOOM project [15, 16, 17].

The mapping (figure below) shows how the Future OSS approach encompasses equivalent functional capability in the Network Functions Virtualisation Orchestrator (NFVO), Virtualised Network Function Manager (VNFM) and Virtualised Infrastructure Manager (VIM) layers through control domains/hierarchy levels. The Future OSS architecture embraces all OSS functionalities, communicating with single or multiple instances of the Future OSS Orchestrators/controllers (such as those configured as an NFVO, VNFM or VIM), including links to legacy PNFs, and aided by Support Systems capabilities.

The Future OSS allow for several ways to split orchestration domains. In this way, MANO can be considered as a set of specialized orchestration domains within the Future OSS, and hence is able to inter-operate seamlessly with other domains of the Future OSS via specialized configuration of the Future OSS software. In the figure, the no-EMS option (present in ETSI) is represented, as the Future OSS has the capability to interact directly with a VNF (this is also a key technical option in ONAP).

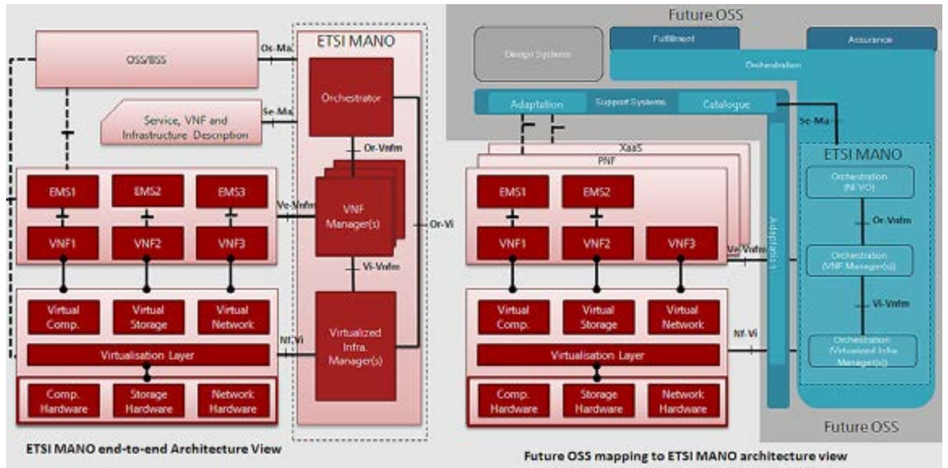


Figure 6: Future OSS and ETSI MANO Architecture Mapping

1.6 Future OSS within a Digital Operations Ecosystem

The Digital Operations Environment [18] enables an ecosystem of Business Partners, Suppliers and Customers and is supported through the use of well defined, standardised open APIs. The use of open APIs will allow each of the actors that interact with the Digital Operations Environment to utilise and benefit from the Future OSS Agility.

The open APIs provide ecosystems interoperability by exposing the Future OSS functionality. The Catalogue module of the Future OSS shares common information models (services, resources, policies, analytics, etc.) with BSS for agile product offering composition. The Adaptation module of the Future OSS allows the Common Event Bus and API Management components to enable a common communication channel between Future OSS, BSS and any other actors of the digital ecosystem.

Within the Future OSS Orchestration, the Big Data technology function is central to the execution of the analytics. By sharing this Future OSS Big Data analytics with the BSS and other Big Data platforms, the Future OSS intelligence will be enhanced, further improving policy-based decision making. Likewise the BSS will also benefit from this enhanced unified intelligence.

The Future OSS can be integrated within a Digital Operations Ecosystem as shown below:

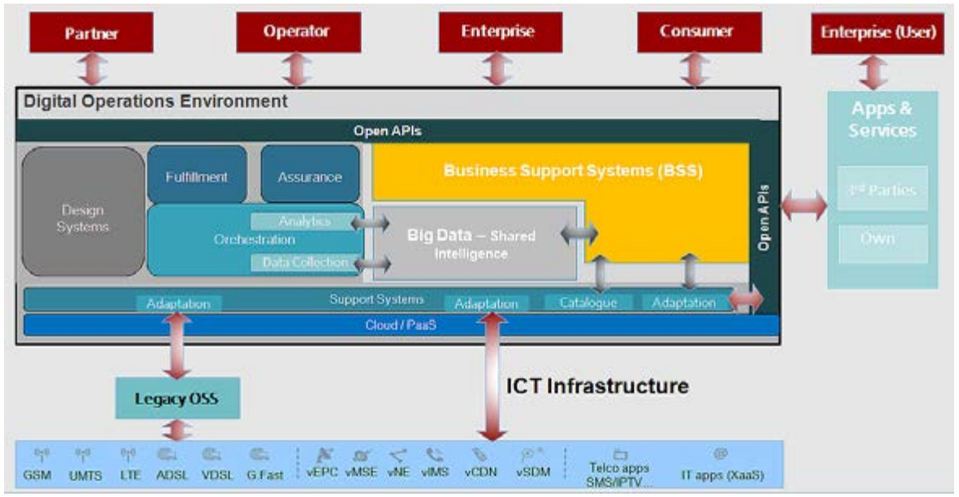


Figure 7: Future OSS within a Digital Operations Ecosystem

1.7 Conclusion

The Future OSS architecture defines a set of functional domains for fulfillment and assurance, built around common orchestration and closed loop autonomic principles, using the OODA model as a technical basis. This architectural vision accommodates multiple cooperating and federated domains, which use a common information model to provide policy-based, metadata-driven management and control. The Future OSS accommodates multiple cooperating domains with federates both the control processes that Observe, Orient, Decide and Act, and the metadata that these processes use, via the Catalogue.

The scope of this architecture extends to all types of physical and virtual systems (including the control and monitoring of the Future OSS itself), along with their associated underlying processes. These processes, and those of the Future OSS, are exposed through service-based APIs in an “Everything-as-a-Service” (XaaS) paradigm.

2. CLOUDIFICATION

"Building a new digital OSS for a new digital operating model"

2.1 Introduction

The Future OSS platform must be cloudified. Cloudifying the Future OSS provides for better OSS resiliency, which is important for establishing automation with increased real-time and dynamic action capabilities. It also allows harmonizing OSS implementations (which for today for Orange are frequently siloed by technology domains, or by operational countries). The following concepts define the digital, cloud-native Future OSS:

- Embraces a platform model, which is made up of independent, componentized modules that can be assembled into applications and tools
- API-based and able to fulfill service requests in real-time, without human touch. Service request completion, such as alarm adjustment, must be immediate and without service disruption. Platform maintenance activities must be achieved without disruption
- Cloud native and resilient, and provisioned across diverse infrastructure pools. Able to scale and heal autonomously in response to new tenants, new resource pools, capacity exhaustion, load, and failures
- Provides an extensible southbound interface into which technology and application adaptors can be plugged, enhancing the reach and capabilities of the Orchestration, Fulfillment and Assurance engines

2.2 Key Characteristics of a Cloudified OSS Platform

Role-Based Tenants

The Future OSS must be viewed as a platform similar to other cloud platforms (e.g., IaaS, PaaS). Each Future OSS "workload" (application or service being fulfilled/assured by the OSS) belongs to an independent tenant of the OSS platform that is playing a particular role (i.e., has a set of unique characteristics, behaviours, and responsibilities). Note that this enables different resources and services to be provided to the same tenant as a function of the particular role that the tenant is playing.

This represents a significant change to current operating models. Just as a public cloud tenant (workload/account) does not ask the service provider for fulfillment assistance to create or adjust their infrastructure, an OSS tenant does not seek involvement from the OSS team to use the OSS platform. This is self-service and allows more agility. Once a

service is designed, with its resources, analytics and policies, it may be promoted to an operational state and instantiated.

Workloads register one or more event handlers (and trigger conditions) with the Future OSS, allowing it to respond intelligently to various scenarios, such as resource utilization and component failure. This responsibility is only passed to humans as a last resort for previously unseen conditions, outlying failure modes, and major incident coordination.

Highly Dynamic

Digital services require a much more dynamic view of alarming and event management than in a traditional OSS, and this has large implications on the design of the platform. It requires re-architecting OSS support processes from large, up-front, manual definition during system build (a "create" bias), to one of frequent adjustments in the form of configuration changes and code releases, all via open APIs (an "update" bias). In the Future OSS this is achieved by employing a model-driven engineering process, where applications are built from reusable modules, and modules are built from reusable components. This enables applications and tools to be assembled by matching the capabilities of a set of modules to the needs at hand.

The Future OSS will eliminate the current cycle times of days and weeks. Infrastructure requirements and service footprints change on a second-by-second basis in a microservice/container based world and normal lifecycle activities such as on-boarding new tenancies must be automated and real-time. The Future OSS must be designed for a highly dynamic inventory and rapidly changing profiles.

Cloud Native

The Future OSS platform must be cloudified. The resiliency of the platform must be pervasive, from alarm ingestion and transmission to the dashboards, orchestration and autonomous workflows. The scalability of the platform requires that it must be able to dynamically adjust its infrastructure footprint, mediating new probes and agents, expanding the big data processing ring, and adding new tenants and infrastructure resource pools as required.

A major focus of platform management is to ensure continuity of service and transparency of maintenance activities, including software upgrades of the OSS. No tenancies should be impacted by the normal, or indeed abnormal operations of the platform. The cloud-native nature of the Future OSS platform assists with both of these goals.

Open and Extensible

The Future OSS will not be supplier or technology specific and must be open and extensible. The Future OSS will support open APIs and be able to incorporate new

technologies, services and resources, with a degree of abstraction that allows total flexibility in the domains that can be controlled and onboarded. The Future OSS cloudified environment is extensible using modern techniques, allowing it to scale as required (e.g. to meet the big data requirements of the Analytics module of the Future OSS among others).

This extends the reach of the Future OSS, allowing more complex service domains and applications to become part of an end-to-end service chain, supporting ever more complex customer service and product offerings.

Security

The introduction of SDN and hybrid service domains into a single self-service OSS enlarges the security surface area significantly, potentially to third parties, and many additional attack vectors must be considered.

As with other operational requirements, security must be embedded throughout the lifecycle. It must be considered at design time, during testing and continuous delivery, and in operations. Thought must be paid to which metrics to monitor, how to automate security testing and verification activities, how to detect a variety of security related conditions, and the responses to those conditions, both autonomous and human-led. Part of the policies within the Future OSS will deal with security issues. Security must consider the usage of the Future OSS components themselves to detect any anomalies and take appropriate measures to mitigate any risk, so that the core of the Future OSS must be built of secure components which will monitor usage and anomalies and implement security policies.

As the Future OSS automates the operation of the hybrid network with dynamic and real-time actions, it is important that the Future OSS embeds self-management capabilities with superior resiliency. The Future OSS incorporates self-management capabilities, for instance to monitor, scale and heal its own functions within this cloud native environment.

Furthermore, the Future OSS must allow for the incorporation of constraints as imposed by local regulatory bodies (e.g. ANSSI, the French regulator). Especially, the Future OSS must provide the ability to control the scope of access to its components including allowed actions and data visibility following security best practices, with segregation controlled based on the access rights deemed appropriate to each user.

One aspect of security that requires specific attention is data privacy. This becomes of particular concern in the context of multi-region operators, and the potential to be required to comply with data sovereignty legislation. The data gathered by the Future OSS is operational data, but could potentially still include information which might trigger sovereignty concerns. One way to deal with such a problem is the use of data masking to hide sensitive information that is not required by the OSS platform. The layered nature of the Future OSS also supports running separate instances in a region where data masking is not deemed appropriate or adequate.

2.3 Conclusion

The design of Future OSS as a self-service, cloud-native platform has implications for how the platform is deployed, how it is managed and maintained, and how it evolves over time to encompass more legacy and hybrid workloads. The Future OSS will consist of many components, and will manage workloads across diverse geographic locations and resources, which is of special importance for a multi-country operator such as Orange.

The data acquisition edge will naturally be a distributed edge that feeds a Common Event Bus from as close to the data source as possible. Similarly, the distributed nature of the Future OSS will provide various compute, storage, and networking facilities that extend the edge of the cloud to the edge of the network, and perhaps even within various clusters of devices. This enables bespoke functionality to be used to accomplish intermediate compute and storage tasks (e.g. filtering, aggregation, and correlation), and to provide targeted, actionable data to cloud-based components. This produces higher quality, context-focused data, and simplifies the operation of the Future OSS while making it more resilient.

The core components should be instantiated per cloud best practices across diverse infrastructure (multiple availability zones) to ensure continuity under all failure conditions.

The Future OSS core platform should be hosted on a private cloud, though some functions could be potentially hosted on a public cloud, subject to data latency, security, and privacy issues. OSS-as-a-Service is another hosting option that can be made available through cloudification which can offer benefits such as allowing affiliates to start their digital transformation without large up-front CAPEX investments.

The Future OSS platform itself will evolve continually, using Continuous Integration, Continuous Delivery, and Joint Agile Delivery process models.

3. DESIGN SYSTEMS

"Enabling DevOps and DesignOps through Model-Driven Design"

3.1 Introduction

The Design Systems provide an intuitive Graphical User Interface (GUI), the Visual Design Studio, to on-board and create, read, update and delete (CRUD) objects and their metadata. The Future OSS Common Information Model enables objects and their metadata to be related to the managed entities at various levels of abstraction via a design "palette". For example the Visual Design Studio relates the resources and services that are required to fulfil an SLA, defines the data/events required to be sent to analytics systems to monitor the SLA and creates policies to manage data collection, analytics, monitoring, configuration, and other processes required in the lifecycle of the service. The Design Systems in cooperation with Support Systems enable DevOps [19], DesignOps, Site Reliability Engineering (SRE) [20], and Joint Agile Delivery (JAD) [21] operating models to be established within the Future OSS.

3.2 State of Play and Future Direction

Existing Approaches and Limitations

Many of the challenges identified in existing OSS, such as increased service agility, velocity, and the need for a completely integrated and customer focused approach are leading to a rethink of how services are conceived and progressed from ideas through to reusable operational models.

The existing OSSs today do not embrace the DevOps, SRE and JAD operating models. The approach to implement these IT based operating models has been limited due to:

- Lack of a cloudified Operating System (OS): an OSS currently running on bespoke hardware can lack the ability to host a DevOps environment
- Lack of the ability to translate business requirements into resource and service designs, and to track the change in business requirements to changes in offered resources and services
- Incomplete/Patchwork Design. Design of a new service is generally limited to designers interacting with an incomplete set of service and resource APIs without the ability to change the resource or service itself
- Hard to change configurations. Few systems have considered a common and metadata based method of configuration. Hence, changes in functionality have required the direct restructuring of data models, database schemas and associated APIs
- Hard to change software. A new or changed capability often needed a new software feature, or some form of change linked to a software delivery. The development practices associated to monolithic, tightly coupled software and their associated software delivery practices result in inefficiencies

Future OSS Approach and Benefits

The Future OSS is design holistic - it aims at allowing the design of anything that is needed in Operations. The Future OSS establishes a Visual Design Studio that provides a common cloudified working development environment, which incorporates design processes specific to each of the design modules of service, resource, analytics and policy.

A new approach to how cross OSS domain information is modelled and shared is required. A more complete common information model allows the integration of a wide variety of metadata (such as linkage of service specifications with service policies), enabling an Orchestration that can both deploy and lifecycle manage a service automatically. This information model must be driven by a sophisticated and intuitive Visual Design studio capability that integrates with wider definitions (such as those of 3rd party APIs, VNF packages and hybrid product offers) via federation through Support Systems.

The Visual Design Studio utilizes the cloudified Future OSS to link the Design modules to the Future OSS Run-Time via CI/CD PaaS based tooling. This establishes an environment where new designs and enhancements can be deployed to the Future OSS Run-Time using DevOps, JAD, SRE, and other modern IT led processes.

The use cases described in section 1.3 have illustrated how the Future OSS Design Systems can support the redesign of a VoLTE Service from a physical to a virtual EPC and IMS. Design Systems provides the intuitive Visual Design Studio to enable resource design to guide the designer through the rapid discovery and on-boarding of the physical and virtual resource definitions. The Resource Design discovery and on-boarding process works in cooperation with the Support Systems Adaptation and Catalogue modules to transform and expose the newly on-boarded resources within the Catalogue.

Design Systems then utilizes the Catalogue to create the new VoLTE service design and link the necessary analytics, policies and resources using the Visual Design Studio to ease in its creation. Once the design has been constructed, the Design Systems provide the DevOps/DesignOps environment with integrated CI/CD tools in cooperation with Support Systems to test and verify the service before deploying to the Future OSS Run-Time environment.

3.3 Detailed Solution

The Future OSS Design Systems Domain is shown below:

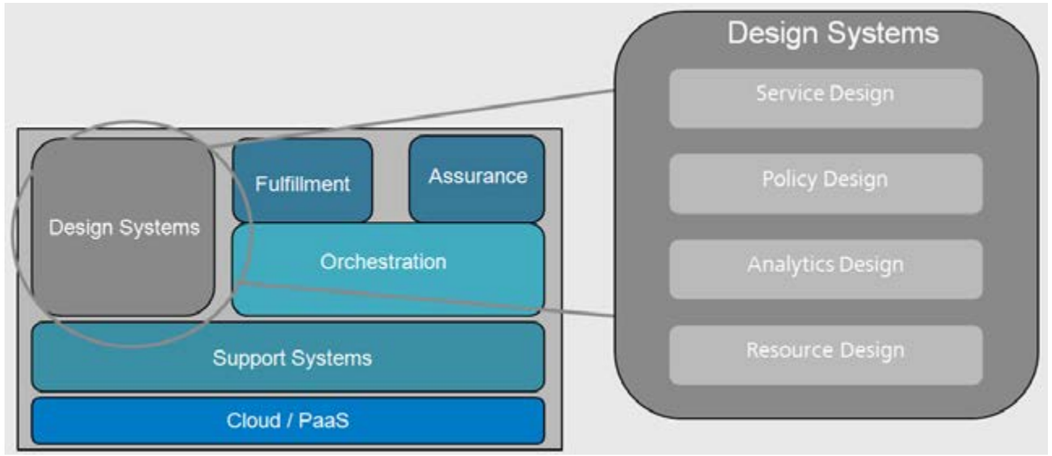


Figure 8: Future OSS Design Systems Domain and Modules

Resource Design

All available objects which can be leveraged to deliver a service or feature offered by or used within the Future OSS are considered resources – whether those resources be physical, logical or virtual, e.g. legacy resources through a domain manager (see chapter 8.2). Resource Design allows for:

- Resource specification – defining and modifying resource specifications and resource configuration, or template definitions
- Deployment specification – defining and modifying the executables, workflows, tools, dependencies and mappings necessary to implement the specification
- Resource specification discovery/on-boarding – searching for or discovering the required resources and loading or referencing all the required definitions/artefacts. These may be referenced via catalogue federation, imported through 3rd party VNF catalogue on-boarding, as swagger API artefacts, TOSCA blueprints, or other means via the Adaptation module

Resources are thus viewed as reusable objects. Resource specifications will be available in the Catalogue and can be assembled together to form larger resource specifications with additional functionality. Metadata can be added at each stage of the design process. For example, metadata can describe best practices for using a particular resource, and/or prescribe how it should be installed. The resources can be certified, tested, deployed, and activated through the use of consistent and open APIs.

Analytics Design

The Analytics Design module enables the user to build-up metadata, models and algorithms to provide context-aware intelligence in support of knowledge acquisition and understanding. This is critical for providing closed-loop control of business and operational processes.

The Analytics Design module is responsible for defining the appropriate data, controlled by metadata and policies, for the analysis of service performance. It defines associated events as well as the algorithms that will process, store, and conduct further analysis (e.g. performing trending, identifying patterns) as necessary. It defines dependencies on this data, and defines where the data should be sent to, and how often. It also defines what types of applications and tools are needed to analyse the history of the service lifecycle to discern patterns governing usage, thresholds, events, policy effectiveness, and other health-related processes.

Analytics design focuses on defining analytics based on a use case or service definition. In the VoLTE Network Service design for example, Analytics design will define VoLTE metrics model (KPI and KQI) and analytics functions required. The Analytics Design supports the collection of available artefacts from appropriate catalogues, along with their manipulation and testing, using a collaborative visual designer. The final analytics model is then defined and stored in the Catalogue, for use as part of service design. This approach facilitates the reuse of the analytics module in a stand-alone or as part of a composite template.

In practice, many of the inputs for Analytics Design will come from data sources that need to be carefully defined. Some will not be available at the initial time of design, and must be dynamically on-boarded, quickly modelled/designed and added to the system while the use of machine learning enables the detection and formalisation of patterns used to discover these resources.

Analytics are thus viewed as reusable objects, consisting of a set of algorithms, data sources, data targets, and metadata. Analytics models can be assembled together to form larger analytics models with additional functionality. Metadata can be added at each stage of the design (and run-time) process. Analytics models will be available in the Catalogue. For example, metadata can describe best practices for using a particular analytics model, and/or prescribe how it should be installed. The analytics models can be tested, deployed, and activated through the use of consistent and open APIs.

Once finalized, the models can be retrieved from the Catalogue for both operations-driven analytics requirements, and for association to resources, policies and services.

Policy Design

The ability to design, modify, and fine-tune policies that change the network operation behaviour is at the heart of operational agility and responsiveness. This process is completed in a CI/CD mode, with the designer having to ability to deploy, operate and test the policies which will increasingly constitute the major part of human operations activities.

Policy Design provides a combination of intuitive, natural language methods for policy development with a wizard-based approach to build policies and their appropriate associations to resource, analytics, or service models Command line and GUI versions of the Policy Design Studio are available to suit the needs of different actors. In addition, open APIs can be used to define how policies are shared both between Future OSS domains and with external systems via Policy Management and Adaptation modules.

Importantly, Policy Design supports Event-Condition-Action (e.g. imperative), Goal (e.g. declarative) and Utility Function paradigms [22]. This enables the designer to focus on the logic executed by policies, as opposed to their syntax. Since policy concepts are part of the Future OSS Common Information Model, it is straightforward to associate them with the parts of the information or data model that they control. The role and relationship of policy types appropriate to a given orchestration scenario can then be expressed as:

- Event-Condition-Action Policies (rules), which operate on the current state space
- Goal and Utility Function Policies (rules), which operate on the intended state space

In other words, to describe an orchestration operation using ECA policies, a designer must anticipate the necessary events that trigger the evaluation of a set of conditions; if the conditions evaluate to true, then the designer can select a set of actions to be executed (those actions may be related to physical, logical or virtual resources). The action statements can use metadata to determine which actions get executed in which order, and what to do if an action fails to execute correctly.

This requires working with the Analytics module to gather historical data, create relevant metrics or analytics models, and test the events and conditions used to ensure that they provide the correct semantics and granularity of action. If multiple simultaneous policies are anticipated, prioritization or other in-built conflict resolution mechanisms must be added.

On the other hand, a goal-based policy has the advantage of defining a preferred state as an intention, relying on the Orchestration logic to decompose this to a set of actions as it sees fit. Utility Function policies operate on a similar basis, but apply a full range of cost/benefits to allow Orchestration to perform optimization. Therefore, guiding the type of policy paradigm used, as well as the components that are contained by the policy for given scenarios, is an important function of the Policy Design module.

Policies are thus viewed as reusable objects, consisting of a set of statements, a set of targets to which they are applied, and metadata. Policy objects can be assembled together to form larger policy objects with additional functionality. Metadata can be added at each stage of the design process. Policy specifications will be available in the Catalogue.

Service Design

The Service Design module enables resources, analytics, and policies to be treated as “building blocks” to create, manage, and retire services. Service specifications, like resources, policies, and even analytics, are objects represented in the models; hence, they are reusable and easily associated with each other.

Hence, this function is capable of building service models from business requirements (top-down) as well as from resource specifications (bottom-up). As with the other modules of the Design Systems, the Service Design module utilizes an online Catalogue from Support Systems for the service designer to create new reusable building blocks and combine those blocks in different ways to build new services. There are three basic types of building blocks: service, dependencies (e.g. this service requires a resource with a set of specific features and behaviors), and connectors that govern the chaining of service components and their interactions.

Each of these three building blocks includes data required by the building block, how to install it, and which set of resources it is compatible with. When the service is defined, its set of building blocks and metadata are then packaged into a new object; this new object is then associated with additional metadata and other objects (e.g. analytics and policy objects). The service can be tested, deployed, and activated. The design of tests is an integral part of the service design, for example, tests to check that onboarding is successful. This is achieved through the use of consistent and open APIs.

3.4 Conclusion

Design Systems create a holistic design environment that can co-operate cohesively with Future OSS Run-Time components, leading to the following benefits:

- Establishes the necessary toolset to enable DesignOps, DevOps, JAD and SRE operating models to be implemented within the Future OSS
- Enhances agility in providing the ecosystem to allow a user to conceive new OSS services and leverage new technology capabilities via the Visual Design Studio to move these ideas rapidly into production
- The benefits of virtualization are brought to legacy networks by the ability to expose legacy resources alongside virtual resources within the Visual Design Studio to enable true E2E service design and operations capabilities

4. SUPPORT SYSTEMS

"Providing the bridge supporting hybrid end-to-end Think, Build and Run"

4.1 Introduction

The Support Systems include modules that are shared across the Future OSS to act as the bridge from Design to Run-time. The shared modules within Support Systems include Adaptation, Catalogue, Policy Management and Dynamic Inventory.

Adaptation supports the interchange of metadata between Future OSS components and the transformation engine required for the Future OSS to communicate with external systems. **Dynamic Inventory** is responsible for maintaining up-to-date, dynamic views of inventory items to support the Fulfillment, Orchestration and Assurance domains. The **Catalogue** contains the previously created or onboarded objects and acts as a design "palette" to support the construction of new services, policies, analytics and resource models. **Policy Management** directs policy federation and distribution across Orchestration, Fulfillment, Assurance, and external policy control domains.

Support Systems work in cooperation with Design Systems in DevOps, DesignOps, JAD and SRE operating models to help distribute and mediate designed artefacts to the Future OSS Run-Time and external systems.

4.2 State of Play and Future Direction

Existing Approaches and Limitations

The root cause of existing limitations is the lack of a true model-driven implementation. In a model-driven approach, the model provides both the syntax and grammar to enable disparate systems to understand and communicate with each other. The lack of true model-driven implementation results in the following limitations:

- The interchange of metadata between Design and Run-Time systems within the existing OSS platforms is not possible: each application has its own definition of objects tightly coupled to specific metadata. Metadata should instead be viewed holistically, able to be attached to any object that can benefit from it
- Events from underlying Network elements are not propagated to all relevant interested OSS components in an efficient and common manner
- OSS in practice today have not fully embraced open APIs, resulting in systems that

are closed in nature. In addition, the APIs exposed often are not consistent

Future OSS Approach and Benefits

The model definitions that conform to common information model must be consolidated within a Catalogue, which integrates federated capabilities and allows external interactions via an API Manager.

The dynamic nature of the data required to support the Orchestration, Fulfillment and Assurance domains demands a new view on dynamic inventory management to support flexible views of this data, such that it can be used for multiple purposes in real-time.

The use cases described in section 1.3 have illustrated how Future OSS Support Systems acts as the bridge between the Future OSS Design and Run-Time systems to support the redesign of a new virtualized VoLTE Service.

The design processes use the Catalogues centralized view of all relevant model information to compose them into the published services which the Future OSS will orchestrate, fulfill and assure. Design metadata will define the relationships of service, resource, and policy, which will in turn be used by the Policy Management and Dynamic Inventory functional modules to manage a federated cross domain view. The Catalogue using the API Manager publishes available methods and brokers API requests to other Future OSS components and external systems enabling a more holistic view.

4.3 Detailed Solution

The Future OSS Support Systems Domain and modules are shown as follows:

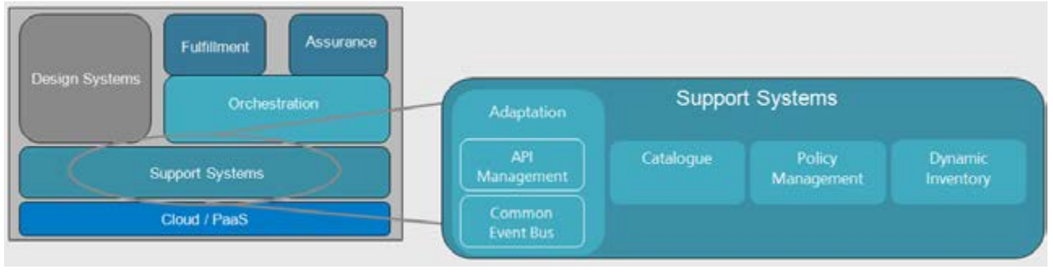


Figure 9: Future OSS Support Systems Domain and Modules

Catalogue

The Support Systems Catalogue provides the design "palette" or repository of service, resource, analytics and policy designs as these are progressed and made available to the functional Future OSS Run-Time systems (e.g. Orchestration, Assurance) with support from Adaptation.

Hence the Catalogue takes a primary role between the Design and Support Systems – providing not only the "specification database" but also the means of controlling and managing the lifecycles and availability of those specifications. It ensures their completeness and validity, allowing disparate parts of Future OSS or 3rd Party components to achieve a common view on objects and relationships under management via a Visual Design Studio.

The concept of a Catalogue has progressed significantly over recent years. Within the Future OSS, there is a need to evolve further, expanding from the classic means of providing definitions for products, services, and resources to becoming a centralized point for all relevant information that affects how the system is running.

This includes: policies, capacities, capabilities, metrics, analytics, microservices, profiles, workflows, recipes, etc. designed in the DevOps and DesignOps. More importantly, since Future OSS is a model-driven system, these entities are objects in their own right, and hence can have associated metadata. The intention is that by basing this capability on a common information model, any client component is able to discover a wide variety of relevant information through one federated access point.

Policy Management

Policy is a fundamental enabling capability to automation and is addressed from multiple perspectives within the Future OSS architecture. One perspective is in the active management of policies working in cooperation with Orchestration. Policy Management supplies Orchestration with the ability to efficiently process context data for a given event and provide a decision.

A second perspective is in the organization of policies. Policies can be accessed through the Catalogue and applied to any process that requires a decision to be made. Policy federation will require specific open APIs to permit the co-ordination and negotiation of policies with management entities in different policy domains. As such Policy Management is available across all Future OSS Run-Time components as a Support Systems shared module.

For external policy domains, translation from the Future OSS internal policy model to supplier or platform-specific formats is provided via a policy language translation capability within Policy Management. The Future OSS Policy Management component will utilize a policy broker for all interaction between policy domains, which includes policy language translation (e.g. DROOLS, XAMCL++, RUBY, etc.) to handle the exchange of policy and related knowledge.

Policy Management also covers contention/conflict and methods for resolution. This implies the use of a broker to identify appropriate domains and ensure successful co-ordination between multiple systems via policy APIs. While the initial policy validation can achieve some level of semantic resolution of policy conflicts, run-time conflict detection and resolution is included within the Future OSS Policy Management architecture to allow for resolution, for instance, of

multiple resource claims from overlapping policies using a knowledge plane for context [11].

Finally, policy takes two very different forms. Intra-operator policies are federated, and those existing at a higher level of abstraction define the behaviour for those existing at a lower-level of abstraction. Such policies are used to define behaviour. However, inter-operator policies do not define behaviour (e.g., one carrier cannot tell another carrier how to build their service); rather, such policies present an offering that can be negotiated.

Dynamic Inventory

The Dynamic Inventory forms the core instance repository within the Future OSS. Its first role is to provide validation of instances during CRUD operations as activities occur e.g. that the entity and its relationships are valid for the given context in terms of time, relationships and state.

The inventory repository information represents a subset of the overall Future OSS Common Information Model as instantiated by various domain data models and made externally visible via the Dynamic Inventory. It represents a real-time "registry" of how products, services, resources and other objects are being used, including their relationships. All components within the Future OSS refer to the Dynamic Inventory as the "source of truth". However, to achieve this within the goals of the Future OSS implies a number of design decisions which differ from historical solutions.

Firstly, the Catalogue provides for definitions of service and resource domains and can either define their capabilities statically or request details through open APIs. The capabilities include categories of objects and supported operations, and they allow a linked master data map to be assembled. This allows inventory federation to navigate through a number of domains including the integration to legacy inventory systems to form a complete picture of how services and resources were configured, and are currently performing. The Future OSS can first be deployed on top of existing inventory systems including clouds which can remain the "source of truth" for their own objects in order to avoid data duplication.

Secondly, the federated inventory view is exposed as a graph of nodes and relationships where each domain is ideally able to provide this graph through an open API based topology (e.g. to assist in the Incident Management operations via graphical interfaces). Where this is not available, the Future OSS will need specific adapters to synthesize the topology from the server domain information. This information will allow rapid navigation of relationships to support different use cases such as provisioning or fault analysis.

Finally, due to the fast-changing and complex nature of data sets, the Dynamic Inventory will support real-time maintenance of key parts of its topology information (e.g. through event subscription and notification), and will provide time managed views wherever possible, not only of changing relationships between objects, but in the states and information held by objects.

As such, the Dynamic Inventory will provide a repository for all instance information within the Future OSS – not just what has traditionally been associated with “inventory” such as lists of equipments, attributes and their relationships. Beyond any association of relevant virtual to physical mapping, the Dynamic Inventory provides support for management of context, through associations to metrics, policies, locations, people etc. from which the relevant data can be aggregated and inferences made. State and time are also essential components of this context information from which to make historical and predictive analysis, and support short-term and long-term planning. This is a fundamental feature of the Future OSS (in the use case described in section 1.3, the virtualized VoLTE operation is designed by using the Design Systems, Support Systems (including Dynamic Inventory).

Adaptation

Adaptation supports the interchange of metadata between Future OSS components and the transformation engine required for the Future OSS to communicate with external systems (e.g. networks, clouds, legacy OSS and BSS) through open APIs.

The use cases described in section 1.3 have illustrated how the Future OSS Adaptation module can support the on-boarding process in the redesign of a new virtualized VoLTE Service. The on-boarding process will break each resource definition into its constituent parts. The parts can be expressed in a domain specific language such as TOSCA, as node types, relationship types, policy types, artefact types, artefact templates, and the artefacts themselves such as virtual machine images. A link will be established between the Catalogue definitions of these artefacts and the repositories containing the actual artefacts.

Adaptation provides the transformation engine necessary for Future OSS components to communicate with external systems. The transformation engine is extensible allowing adapters to be plugged in as required to communicate with external systems. As both an internal and external communication medium, the Future OSS Adaptation module includes the following components:

- **API Management:** Any large system runs the risk of having inconsistent APIs. The Future OSS adds many different types of managed entities (e.g. analytics, policies and microservices) and hence, it is critical that APIs are consistent, not just in naming, but also, how they operate. The Future OSS enables a model to be used as a common vocabulary and grammar. By basing the API design on the model, this ensures consistency and simplifies their management. The API Management component within the Future OSS provides a framework to design, test, secure, version, publish, and monitor APIs across the ecosystem in support of collaboration. As resources are on-boarded, the API Management will support auto-generation and publishing of APIs from the Adaption module to the API Stores. The API Management platform must put APIs security as a priority and will govern

the lifecycle of the APIs from creation to retirement and offer versioning control over the APIs. For a better CI/CD introduction; the Design of API test monitoring must be anticipated

- **Common Event Bus:** A Common Event Bus will serve as a scalable communication mechanism both between Future OSS components and between the Future OSS and external entities. The Common Event Bus will support multiple event types, from micro-service registration events performance event, to events passed between loosely coupled nested control loops (as defined by Design Systems). This is achievable because the Common Information Model provides a rich definition of events, their payloads, their metadata, and models the communication infrastructure found in Enterprise Service Buses. The Common Event Bus will support the large scale publish and subscription of events, backed by routing and forwarding using a number of selectable abstraction mechanisms. Hence the Common Event Bus provides a unified, high-throughput, low-latency communication medium for the Future OSS. Coupled with modern techniques and software around communicating and handling shared state, the Common Event Bus can be used to form the core of federation and consensus building within the Future OSS repositories

4.4 Conclusion

The Support Systems form the bridge between the Future OSS Design and Run-Time systems so that they can co-operate cohesively, which leads to the following benefits:

- A **Dynamic Inventory** that provides a real-time repository for all instance information within the Future OSS and the ability to navigate it graphically
- A **flexible API Management system** is defined that can cater to the demands of individual CSPs by allowing controlled, secure access to APIs
- An **Adaptation module** that allows for the transformation of on-boarded resources into Future OSS metadata and an extensible transformation engine to adapt the Future OSS data/metadata as required to communicate with external systems
- An **extensible Policy Management system** that uses multiple policy programming paradigms that can be combined as needed: visual design of policies is emphasised
- Enhanced collaboration and sharing of the information model via the **Catalogue** across Future OSS domains, to allow innovation to thrive to increase efficiencies and increase automation

5. ORCHESTRATION

"Automating the engine of hybrid end-to-end Service Management"

5.1 Introduction

Orchestration deals with the coordination of all needed actions to create or change a service or maintain existing service functions. Some actions may require human intervention, if not yet automated. Within the scope of this Future OSS document it covers the Fulfillment and Assurance parts of the TMF Operations domain, providing the necessary automation, responsiveness, and conformance to KPIs in these areas. Because of the growing complexity of services and networks, one key principle is to split control and to delegate.

Orchestration encompasses all closed loop functions described in the architecture section – functions that collect data, analyze, consider, act, and check the results. The discussion in this section touches on both the coordination and detailed functionality of all stages of this cycle.

A number of challenges need to be addressed by any orchestration solution:

- Massively growing network scope and information pools
- The growing need for on-demand services, some of them with stringent performance requirements
- The nature of services is rebalancing from network centric to data and application centric, but at the same time legacy networks will form part of the service mix for some time
- The ability of the network to self-organise and manage is growing, presenting both advantages to automation and challenges to coordination. In a similar way services are increasingly composed of a mix of B2B/B2C/B2B2X players which need to build flexible ecosystems as assets for creating agile business and value at the touch points between the interacting actors.
- From the CSPs point of view, the orchestration process must be able to improve Quality of Service (QoS) and increase operation efficiency, and help achieve an optimum, cost efficient use of network and IT resources

5.2 State of Play and Future Direction

Existing Approaches and Limitations

In the past, a level of resource orchestration has extensively been achieved through workflow engines, which typically employ a form of graphical modelling using notations such as BPMN.

These specify the coordination of flow across both software systems and human process, and where tasks such as detailed network design and configuration are required, scripting languages may be used to accomplish this.

While these workflows and scripting approaches clearly have a flow model as the outer layer, one problem with this approach is the difficulty in maintenance that comes with the code-based approach and detailed configuration tasks. Workflow alone also does not capture the higher level goals or requirements that an organisation needs to place on system behaviour.

The ability to delegate decisions to the network has been limited in the past, so that provisioning individual controllers has been at a low level. This has meant more detailed knowledge of the network needed to be incorporated at the service orchestration level leading to a complex, monolithic implementation. The CSP has then to rely on specialized teams (internal or integrators) with the supplier support.

Policy management may be present in some systems but is always restricted to low level event handling, input validation etc. While this is needed, there is no ability to build higher level rules, adding to the complexity human administration.

Product, service and resource catalogues may be used in conjunction with service orchestration, but as with policy, they are used in isolation from other catalogues, so don't directly address the needs of B2B/B2C/B2B2X or network integration.

Future Approach and Benefits

A future approach needs to provide more integration between the various models that control orchestration behaviour, from the rules that govern higher level goals and SLAs, to those that describe service and resource composition, and the processes of fulfillment and assurance that go around them.

Model-driven orchestration works with a catalogue-based description of the product, service and resource types, enhanced with metadata such as event and API definitions, along with a policy rule base. Instead of requiring separate workflow or scripts - that describe how to implement the catalogue model, the orchestration engine is able to use the model relationships to determine the ordering of processes, and the parameters and APIs needed to achieve those processes.

These relationships do not have to be restricted to a single catalogue-based system, but can extend across systems and organisations, allowing orchestration to manage any type of service including end-to-end services that are dynamically constructed and supporting a rapid development process. A key dependency of hybrid orchestration is on model translation and federation for areas such as inventory and catalogue, as dealt with in Support Systems.

Policy is used both to direct and verify event based reactions, to specify what the high level goals or SLAs are, and to determine the most cost effective solution. Refinement into low level

actions is done automatically as far as possible. This places a much greater demand on the orchestration and policy functionality but leads the way to the next level of self management.

Workflow or scripting capability may be needed to supplement model and policy interpretation in cases where a system does not provide a declarative API to create or change a configuration that can be linked to the model. For instance, in the example of hybrid VoLTE scale-out, the model may specify a dependency on ”traffic steering” which is mapped to a physical SBC and its management system, but the dependency requires several private APIs (or direct CLI) to be executed in a specific order to carry this out, which needs a script to be linked to the model. In addition, if the VoLTE scale-out fails and falls out to manual processes, then human workflow is invoked as part of the assurance process.

The sheer size and complexity of networks such as IoT, and the need to have instant access to key data on these networks, makes the split into numerous domains of control (and the delegation of control) inevitable, where decisions must be made locally to each domain. This reality is fully consistent with the Future OSS architecture and the patterns of distribution and data driven principles.

5.3 Detailed Solution

Orchestration Architecture

The phases of Orchestration can broadly be divided into Data Collection, Analytics, Decision (based on the events and context provided), and Implementation of actions (where the decision is translated into service and resource changes). This is represented below where it can be seen that the basic components can be mapped to an Observe, Orient, Decide, Act (OODA) loop. The architecture is distributed, and OODA loops nested, in that fulfillment actions may be delegated to other orchestrators / controllers / domain managers (e.g. VIM, ONAP SDN-C or APP-C, NFV-O, legacy local loop domain managers) via declarative APIs. These orchestrators may also through coordinated policy assure the parts of an end to end service that fall within their domains and only pass event notifications to a higher level of closed loop orchestration in cases of shared resource conflict or other inability to completely close their loop.

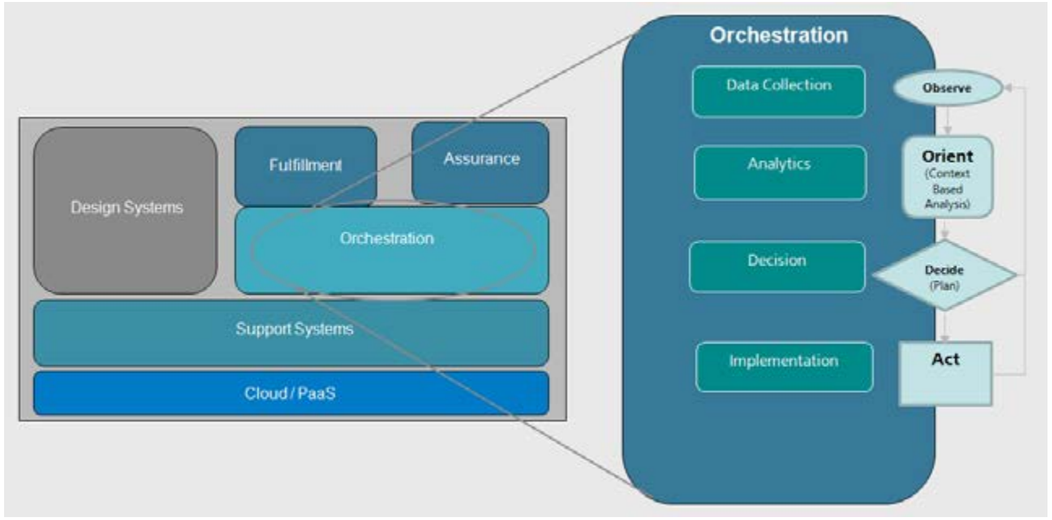


Figure 10: Future OSS Orchestration Domain and Modules

This end-to-end operation and interaction of Orchestration with other domains begins with the Observe function and is represented below for an assurance case (see Fulfillment chapter for a related case):

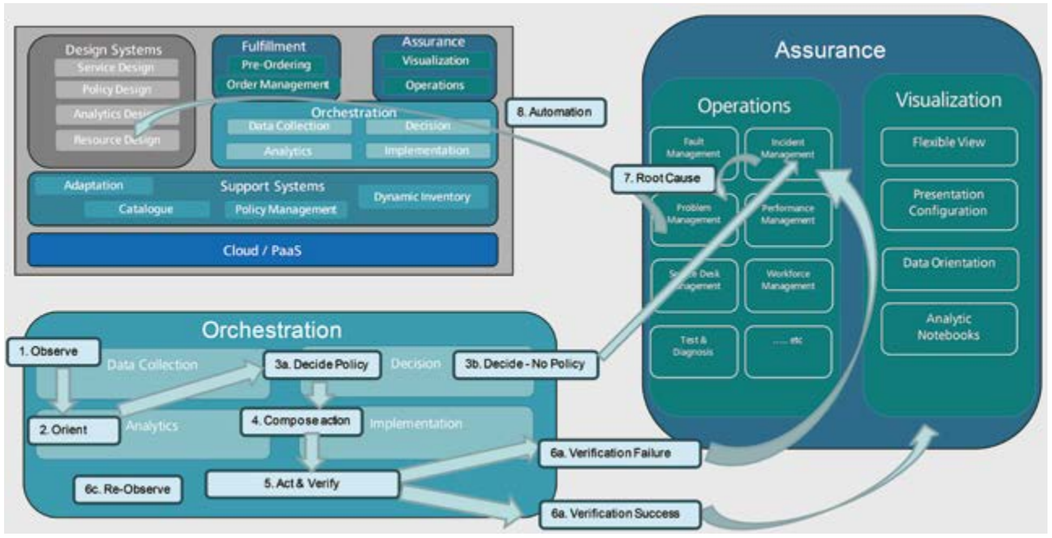


Figure 11: Orchestration Control Loop

1. **Observe** - the event collection trigger is the starting point for Orchestration (e.g. collect VoLTE traffic data)

2. **Orient** – this function executes Analytics with additional information to provide context (e.g. apply KPI models, capacity models and topology model to generate traffic analytics model and prediction)

3. **Decide**

   a. Policy Satisfied – decide **automatic** course of action (possibly delegated) based on analytics output and relevant policy (e.g. spin up a new instance, extend capacity of existing if supported)

   b. No Policy Satisfied – fall out to **manual** Incident and Problem Management as described in Assurance

4. **Compose Action** – design components to support the decision (e.g. virtualised IMS component)

5. **Act & Verify** – perform the action and verify the outcome (e.g. check the KPIs are within expected bounds)

6. **Verification Result**

   a. Verification Failure - fallout to Incident and Problem Management as described in Assurance

   b. Verification Success - visualise and monitor as described in Assurance

   c. Re-Observe – whether success or failure, re-observe

7. **Root Cause** –Problem Management performs analysis and defines permanent solutions as described in Assurance

8. **Automation** - Problem Management uses design environment to implement automation as described in Assurance

Data Collection

Data Collection forms the entry point for Orchestration to ingest the necessary context from the hybrid network to form intelligence. Orchestration requires the ability to limit or throttle the incoming event data or request additional event data as needed from the hybrid network via Data Collection.

Data from both the physical and virtual resources in the Future OSS Platform shall be collected and consolidated onto the Common Event Bus within Support Systems by the use of Data Collection mediation agents. This function is best achieved by embedding mediation agents

where possible within the physical Network Elements, via the legacy domain managers, controllers, VIM, SDN-C and the many other data sources directly.

To ease Integration to the Future OSS, embedded mediation agents should be a prerequisite for on-boarding new components. Embedded mediation agents may be simpler to mandate via VNF on-boarding processes. In specific cases, the VNFs could directly provide VNF application events. However in cases such as legacy Network Elements where it may not possible to embed the mediation agent directly, a mediation server may be used to host the mediation agents. The mediation agents can use existing mechanisms such as SNMP, CORBA, and REST to collect the data from these legacy Network Elements and transform it into the Common Event Bus format.

Well documented SDKs enable the agile and cost effective development of these mediation agents and promote easy adoption. They accept event throttling, heartbeat interval adjustments and configure event measurement intervals from a remote management component of the Future OSS platform to allow for the control of event data entering the common event bus.

These agents also provide inventory discovery and delivery of Future OSS related inventory events to the Common Event Bus to enable a Dynamic Inventory view to be maintained. Events include automatic broadcasting of changes in state of both the PNF/VNF/component that the mediation element is embedded in and consequent state change events for federated inventory resources.

The section has focused on the Future OSS real-time Data Collection, which is the primary enabler to supply real-time context for Orchestration to act on. Batch Data Collection is also considered within the Future OSS in support of Data Analytics which is explored below.

Analytics

Analytics involves the generation of context based intelligence for the Orchestration and Assurance systems to decide and act. It is the Orient part of the OODA loop utilising the Observed data, and other available information such as historical data, topology, SLAs, costs, policy definitions to generate a context aware intelligence for other systems to act on. It will support automatic anomaly detection, context formed metrics, making predictions, and issuing actionable intelligence to guide the selection of policies and hence intents.

Key features of the Future OSS Analytics capabilities include:

- Open source technologies – allows leverage of collaboration, best practices and innovation for large data volume, variety and veracity management, combined with effective real-time and historical analytics
- Data storage, manipulation and management – it is costly and unnecessary to store all data either centrally or at edge storage centres. Decisions need to be made on what data

to store, what level to store it at, how long to keep it and most importantly the aggregation necessary at the edge for transportation over managed bandwidth to a central storage. These decisions are determined based on the use case, domain expertise and the applications which require access to that data

- Big Data and Fast Data architecture solutions, in-memory and Massively Parallel Processing (MPP) repositories - required for the processing and storage of these data sets to support the expected data loads. Data storage technologies will support raw, structured and unstructured data and make it available to the relevant applications in near real-time, or as historical batch data (this latter part is of special importance for problem management and trouble shooting, which constitutes one of the main human activities in the operation of the future). In certain cases, large volume data aggregation is required where MPP and in memory databases support the speed and aggregated data volumes required. In other cases, it will be necessary to run analytics on the raw data, particularly in use cases to find unknown or unexpected patterns in network operations and predict outcomes or faults
- Real-time analytics – the Future OSS will create and maintain a library of analytics models that operate on real-time data via Complex Event Processing (CEP). CEP analytics orientation will be based on federated policy to form the context enriched data that provides intelligence for automatic anomaly detection. This enriched intelligence context shall enable nested Orchestration closed loops to decide the best action within the overall Orchestration process (e.g. handle it locally or enrich the event and relinquish control to a parent control loop). The Future OSS real-time analytics will participate in providing intelligence for Future OSS Orchestration control loops that the VNF, SDN or PNF could not handle
- Batch analytics - the Future OSS will also create and maintain analytics models based on longer term analysis of historical data. The analytics models are composed of the metadata describing the models (e.g. data models, service models, capacity models, traffic models as well as the analytics techniques that need to be applied). The analytics models are executed within the analytics run-time environment, providing output and storing results for further action by Orchestration and Assurance functions

The Future OSS will support analytics tools and applications allowing Operators to work in a collaborative manner on descriptive analytics and more advanced data scientists to generate complex analytic models using Machine Learning (ML) and Artificial Intelligence (AI). This AI element will be critical as the system evolves and constantly learns based on human analysis.

Decision

The decision module must determine the optimal course of action for assurance problem resolution, fulfillment requests and requirement solutions.

Orchestration **may** support procedural interfaces to allow explicit provisioning but it **must**

support intent-based interfaces where requests are expressed as a desired goal plus a set of constraints. To enable this, a request mapping functional component must often be used to translate the ”what” of a request to the ”how” of a set of actions. The Catalogue is a reference point to enable, for instance, the mapping of a request for service between two locations with a certain set of QOS parameters to a specific Catalogue blueprint for building that service, or a request to resolve an imminent performance violation on a specific resource. This may be done in a number of ways:

- The Catalogue defines filtering, selection and mapping definitions that enable a direct linkage between intent and a required template, or a number of templates, to be established
- The Catalogue contains specific capability definitions that may be matched against the requirements or intents present in the request to determine the optimum service and resource templates
- The Catalogue encompasses event type policy definitions that may be searched for matches on the incoming request to determine a specific course of action, and these may in turn be linked to specific Catalogue configurations or other targets of their action

Following the mapping process, the decision functional component either directly implements the change specified by the context of the request and the Catalogue mappings, or it must determine among several options which provides the best course of action, or resolves conflict among them. To do this, utility function policy provides the framework to evaluate alternative goals or states of an object such as a service or resource by specifying how objective (cost/benefit) functions may be used to determine the relative ”goodness” of the various goals, and then execute the optimum solution for a given set of inputs.

Although the phases of analysis, decision, composition and implementation processes are documented as if they were a single flow, in practice the architecture must support multiple analysis and decision points in the process of responding to a single intent-based request. The Future OSS may be split into several domains, with split of control. For instance, the decision to build a given service is broken down into one or more decision points for VNF/PNF design, plus SDN design, each one of which could need policy based analysis in the form of objective functions, and optimization decision taking.

Implementation

This is essentially a process driven by catalogue, topology, and policy information. The Catalogue describes the required functional requirements, characteristics and relationships while the network topology describes the actualised capabilities and relationships, and policy describes the goals and constraints on the model. A design process then operates to ensure that each piece of the service can be satisfied by a particular service or resource domain, while the policy rules qualify this by dictating whether reuse of existing instances is permitted, and

what dependency interactions between instances are allowed.

The implementation module is composed of six components:

- **Decomposition:** Decomposition uses the Catalogue models and policy to decompose a decision action into a set of service or resource level changes. This may be necessary to implement both fulfillment and real time assurance changes. For instance it may take a business level fulfillment request and turn it into service order level requests. In some cases resource order and even work order requests may be needed for tasks such as CPE delivery/installation for services with physical elements. Real-time assurance may also require a defined change to a service or resource that references a model, for instance to perform scaling of resources, and this may decompose to configuration actions in several different PNFs and VNFs as a result
- **Topology and Placement:** As part of the translation of an intent to a detailed design, an up to date network topology that crosses domains is essential to determine the optimum placement of functions, locate domain boundaries, and qualify any inter-domain requests with location based information. Functions of explicit or intent based path finding at different levels need to be handled, and where explicitly managed, path finding and placement may need additional information on shared risk that the dynamic topology must provide. The process includes taking topology templates from the Catalogue for configurations such as network services and VNFs and superimposing them on the network topology provided by Dynamic Inventory
- **Configuration and Lifecycle Management:** This process determines the required reuse, creation, modification/state change, or removal of services and resources, and involves:
  - Mapping requirements dictated by the decision process to candidate existing or new service/ resource capabilities
  - Determining the lifecycle changes required to existing services and resources. This involves mapping target lifecycle states to existing states and determining the required changes to move to the target states via state machine models for each type of object
  - Mapping the target service and resource configurations to the existing ones and determining the required changes, such as increases in capacity, within limits dictated by the Catalogue and policy
- **Activity:** The Activity functional component drives the instantiation of the required components in the correct order of metadata-driven dependency, so that for instance where a virtual WAN connection depends on a set of VNFs which in turn depend on the VNF infrastructure, the appropriate domain managers are called in the correct order. This process determines the correct set of domains, APIs, and adaptations required for each service or resource component
- **Verification:** The Verification functional component ensures that a request has been correctly executed, and this extends to a service request as a whole as well as each component of a service requested from other domains. It is assumed that each domain is responsible to check the correct execution of what has been configured in each domain. For instance, in ONAP, the Application Controller (APP-C) is responsible to monitor the VNF health it has instantiated, not the Master Service Orchestrator (MSO). Questions that must be answered include whether the end result satisfies the intent and constraints of the interface that requested it, and whether any internal policy that controls its creation has been satisfied. In order to answer this, specific test and diagnostics functions from the Assurance Domain are needed
- **Fall Out Management** determines the course of action to take if a single request, or one of a group of requests fails. Again this process needs to be configurable and policy driven, so that the recovery action can be completely automated, or drop-out to manual action. The functionality to learn from successful actions, either manual or automatic, and update policy accordingly is also part of this process.

5.4 Conclusion

The Future OSS solution approach follows the closed loop architecture introduced in the first few sections of the document and covers the following key points:

- The existence of a number of distributed domains of Orchestration dealing with the lifecycle of even a single service is a fact of life. This can be best dealt with by open interfaces that allow the domains to communicate at a level that hides their internal workings in ”intents”, yet maintains the essential shared knowledge of areas like services and resources through notifications and federation
- A number of different linked models can be used to drive the Orchestration processes, but increasingly, the service and resource model itself can be used to do this by detailing the dependencies of each element of a service, reducing the dependency on traditional workflow
- Orchestration needs to be supported by a distributed network of data collection and mediation agents feeding a common event bus. This acts as a collection point for both state and configuration changes from other domains or network elements as well as real-time performance and alarms
- Orchestration requires a flexible set of analytic functions that support real-time Assurance

issues or offline planning functions, which must be architected to support extremely large data volumes and parallel processing. Pattern recognition, rule-based inference and machine learning will play an increasing part in the elimination of manual intervention in Orchestration

- A policy-based decision process needs to interact with the analysis to provide a Complex Event Processing function which can coordinate events from a number of distributed domains to support the end-to-end service assurance process. Policy based decision making also needs to support planning and fulfillment functions in the translation of the required service or network goals to the optimum set of lower level actions
- The implementation of a network fulfillment or assurance response follows the same pattern of design and checking of results over a number of distributed domains. Policy again forms the essential, means of guiding and verifying that the agreed goals, constraints and SLAs are achieved
- Orchestration is responsible to check the overall end-to-end correct execution. To this end, it is able to check what it has done itself, and check that what it has delegated to other domains has been acknowledged

6. FULFILLMENT

"Transforming business requests into agile and cost effective solutions"

6.1 Introduction

Fulfillment involves the management of requests from a number of customer sources, the translation and decomposition of customer requests to service and resource based requests, and the tracking and notification of the progress of order fulfillment. Although the business layer is not discussed here, the Future OSS requires a continual coordination of function from BSSs and OSSs.

The drivers for changes in fulfillment existed prior to the introduction of SDN/NFV, and many of them are underway, simply becoming more urgent in order to take advantage of virtualization.

This includes customer self-service and zero touch provisioning, driving to improve and speed the fulfillment experience for the user as well as reduce OPEX for the service provider.

From the CSPs point of view, the process of fulfillment also becomes a more dynamic process, with more complex systems, and a need to offer competitive products that may involve a chain of service suppliers.

Complete virtualization at the service level will take a number of years even for progressive service providers, and in some cases will not be possible, so that efficiencies of hybrid fulfillment within the physical network still need to be pursued.

6.2 State of Play and Future Direction

Existing Approaches and Limitations

Existing fulfillment systems are often served by stacks of catalogue based product/service/resource definitions with some degree of manual provisioning and varying degrees of integration. This may extend to APIs but is quite often proprietary with information exchange offline and file based. This may cope with a single tightly integrated stack, but is less able to cope with a rapidly changing environment with a steady stream of definition changes that in turn affect the order integration process. This is also unlikely to suffice in a self-service environment where the customer may create their own customized services from a mix of possibilities on offer.

Automation often extends to auto service and resource provisioning with the aim of reaching zero touch, but this is still too often hampered by disconnects and inaccuracies between the automation processes and the devices and management systems that they are trying to

provision over, resulting in high fall out rates. Those disconnects are not easy to correct in an agile way.

Future OSS Approach and Benefits

Previous approaches need to be extended and improved to operate in an increasingly dynamic and open ecosystem.

Providing a flexible B2B/B2C/B2B2X environment requires business level interfaces which expose product level definitions. These may in turn be reused within service and resource definitions of other organisations orchestration and fulfillment systems and repackaged. A simple, single hierarchy of product, service, and resource increasingly does not apply, and the functions described such as feasibility and decomposition must be metadata-driven and interoperate in a distributed way with similar systems to form a federated whole.

It is tempting to think that with the increased capabilities that virtualization brings to the IT systems and network, traditional functions such as service feasibility checks and reservation no longer apply and can be left to dynamic network adjustment, and indeed these functions will change in nature but they cannot be discarded, particularly where application-level physical resources are needed (e.g. SIM, physical equipment, etc.). The order processes still need to be aware of potential issues that can either permanently or temporarily affect a request for service, but they will do this through the processes of Orchestration, Data Collection and Analytics described in the previous chapter.

Within the Future OSS, the Fulfillment Domain relies heavily not only on common functions such as the Catalogue, but on the general Orchestration functions that both instantiate services and assure them. The approach to business and network policy management discussed in previous sections applies equally to Fulfillment, governing what orders can be accepted from various parties and how they are processed. The orders include manual work orders that are necessary (for instance CPE availability).

The goals of fulfillment to increase automation and reduce fallouts will, in the short term at least, depend on how well hybrid networks are managed. Several important factors relate:

- The use of a more dynamic inventory means that the legacy resource systems are not necessarily replaced but wrapped in APIs. The emphasis is on topology extraction aided by smart stitching of data, to increase the speed and accuracy of the design process, often using specialized graph database software
- The use of analytics to trouble shoot and to attack the fallout queues and reduce the amount of manual intervention
- The use of predictive analytics to feed the planning process and indirectly reduce capacity

fallouts

- The modification / improvement of the fulfillment process

6.3 Detailed Solution

This end-to-end operation and interaction of Future OSS Fulfillment with other domains is represented below:

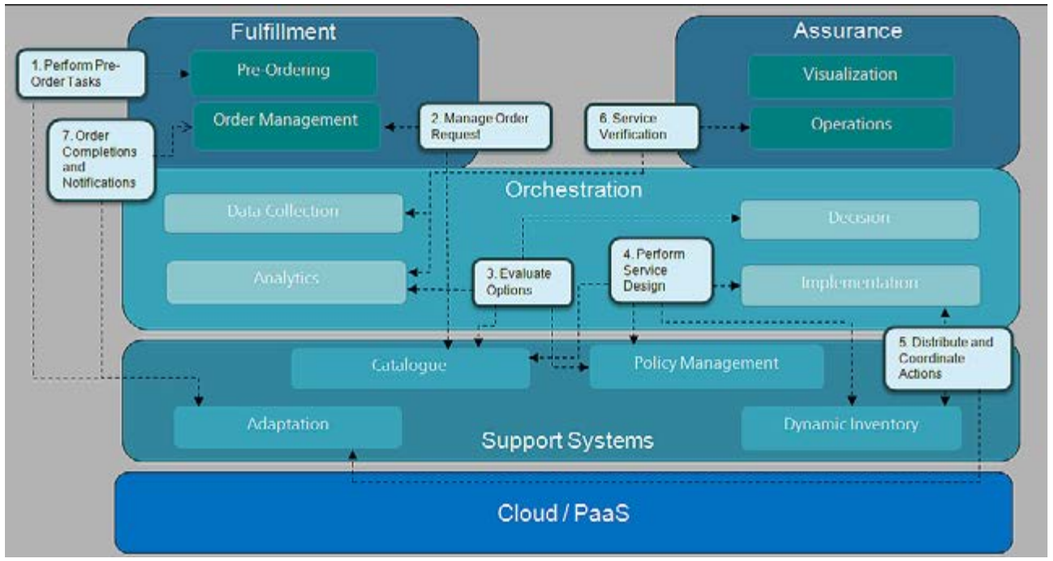


Figure 12 - Request to Fulfill Interactions

The following steps refer to the example of a vEPC network slice order use case described in section 1.3:

1. **Perform Pre-Order Tasks.** The feasibility of the vEPC slice request is checked by ensuring, for instance, that the virtualised slice can cover all the access regions required by the order
2. **Manage Order Request.** If feasible, the vEPC slice order request is made persistent as product/service orders and passed to Orchestration. Further order management may be needed for the associated resource orders as implementation proceeds
3. **Evaluate Options.** The Orchestration process takes the vEPC order request and maps it to Catalogue predefined options. Where alternate options exist these are evaluated using analytics and policy to determine the best vEPC VNF and SDN resource option to use
4. **Perform Service Design.** The resulting action is decomposed using the Catalogue into the

required resource components. The required vEPC service and resource functions requests are mapped to existing EPC service and resource domains using federated Dynamic Inventory topology, and an overlay of new VNF and SDN path requirements is created to serve the new network slice

5. **Distribute and Coordinate Actions.** The required sequence of service and resource allocations are performed by using intent-based API requests to the southbound VNF and SDN orchestration systems
6. **Service Verification.** Any required policies relating to the vEPC Orchestration process are verified by:
   - Gathering analytics data and invoking policy verification
   - Performing explicit end to end vEPC slice service testing (present in the catalogue) through the Assurance operations components
7. **Order Completions and Notifications.** The vEPC slice service order completion is notified to the requesting MVNO system

Fulfillment Architecture

The roles of supporting modules to meet these fulfillment functions are represented graphically in the figure below, with key components discussed in more detail in subsequent sections:

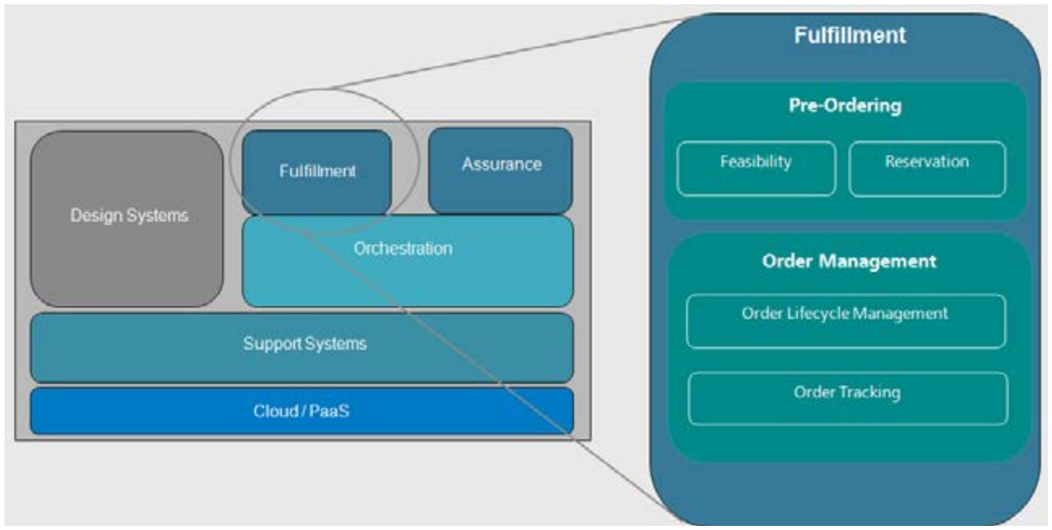


Figure 13: Future OSS Fulfillment Domain and Modules

The Fulfillment Domain key functions include:

- Accepting orders through open APIs and mapping them to a set of catalogue based service and resource order requirements
- Performing pre-checks on those requirements either internally or through APIs to other order management systems before determining and invoking the target systems for service orchestration
- Maintaining the lifecycle and persistence for the orders under its control, and coordinating that with other systems

Feasibility

TThe Feasibility functional component ensures that what can be accepted as an order can be delivered within all the required SLAs. This gives rapid answers and focus on the key constraints provided by the pre-ordering context supplied, and the model that guides it.

The supplied context is used to search for specific Catalogue products or bundles, either directly by name or by general user requirement. This will in turn give the associated service model configurations, and any associated feasibility check configurations describing the validation tasks (these may be specific or quite common to a set of services). Typical tasks include:

- Customer location validation for fixed services, using key address reference data
- Assigning component services to domains of responsibility, each of which will potentially have its own feasibility checks and APIs
- Performing selected resource checks. For example: if a CPE or fixed licenses are needed, are there reusable examples that associated services already have?
- Assessing time and cost based on service/resource and process dependencies
- Location placement validation. For instance if there is a need to site certain functions (e.g. vCPE ones) close to the user for performance reasons, or apart from each other for residency

If the customer requirements cannot be satisfied, the feasibility process should continue to examine the possible models and suggest a combination of options that will work.

Reservation

The Reservation process guarantees that a specified resource or amount of resources will be available to one or more potential future orders, via the appropriate domain resource manager. The reservation functions like a time limited order, and the action depends on the Future OSS or federated resource manager/inventory, for instance:

- Reservation of specific resources relevant to the type of service. For an intended IoT service for instance this may include interfacing to Subscriber Data Management (SDM) systems to reserve mobile subscriber numbers/SIM cards needed for a proposed device rollout

- Reservation of bandwidth for anticipated growth or a single one time use. The domain manager (e.g. for SDN) may directly support a bandwidth calendaring function. In addition, the proposed growth may trigger network planning and will combine into the analysis process for network augmentation

Order Lifecycle Management

Order Lifecycle Management maintains order persistence, state, and relationships throughout the fulfillment process. It responds to requests to create, modify or remove product orders from the business layer and to create, modify or remove service or resource orders. Changes may need coordination with Orchestration to ensure a coordinated action and response in cases such as resource creation, cancellation and rollback.

Operations will have the ability to troubleshoot fulfillment (looking at previous orders results, introducing collection of events, designing policies for fulfilment).

Order Tracking

Order Tracking tracks the completions against order requirements and dependencies as they are orchestrated, responds to status requests from business systems, and notifies them of status changes as required to complete the business level processes.

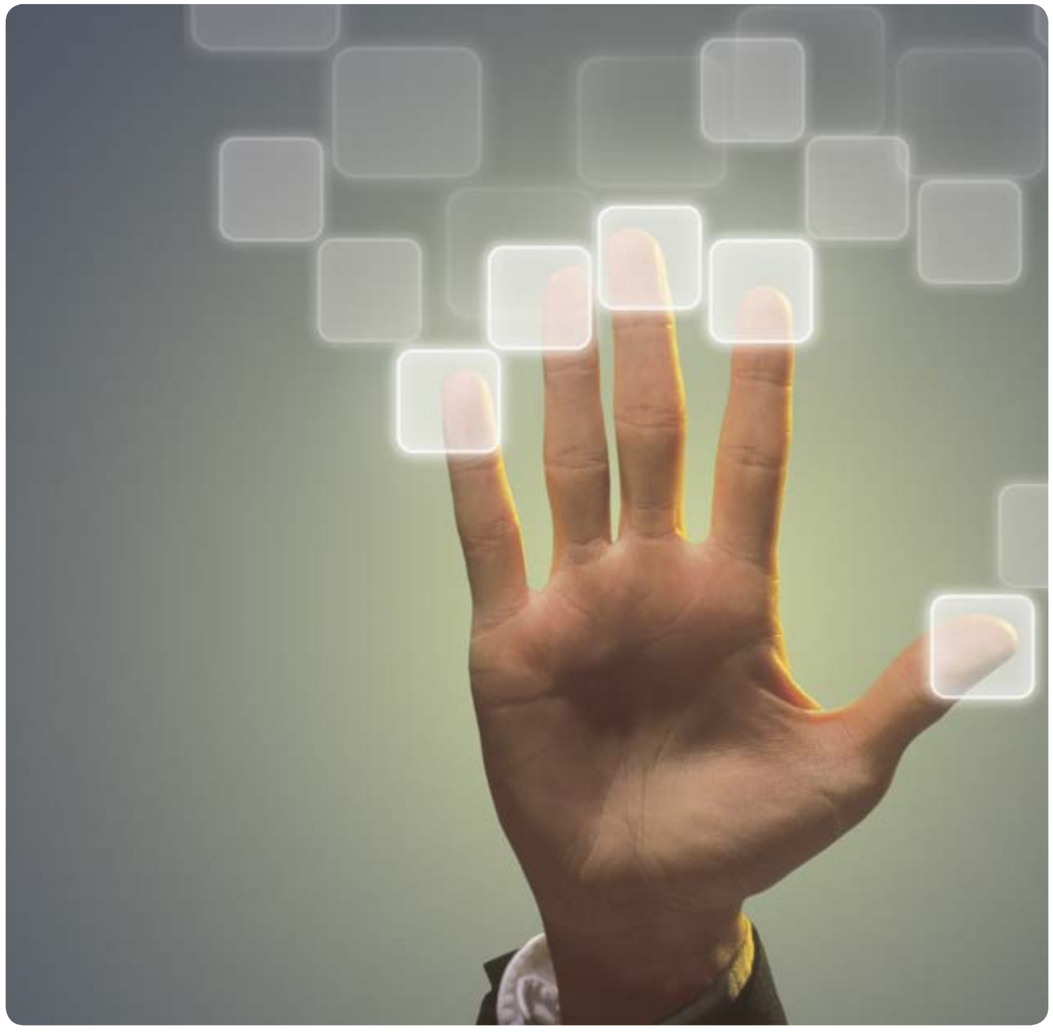
6.4 Conclusion

The fulfillment process, in common with the Future OSS as a whole, needs to address challenges of:

- Competitive pressures improve the speed of new service introduction along with their changing nature as flexible groups of multiple supplier services
- Continual needs for more customer centricity in terms of services which are available regardless of device and location, on demand, with short lead times to provision
- Internal pressures for OPEX reduction and zero touch provisioning along with the realities of hybrid physical/virtual networks

To achieve this, the architecture outlined:

- Performs policy and analytics based feasibility checking based on dynamic gathering of customer and order data to ensure that accurate service options are fed back to the customer
- Uses common federated catalogue and domain definitions throughout to flexibly bring in new B2B/B2C/B2B2X services, farms out requests internally and externally , and composes the best service mix on a per order basis via Orchestration
- Optimizes and assures the fulfillment process through a common cycle of orchestration as covered in the Orchestration section



7. ASSURANCE

"Dynamic and automated operations to assure service quality"

7.1 Introduction

Assurance deals with dynamic and automated operations, to provide confidence and certainty that defined service quality levels are maintained for optimum customer experience. It covers the functions of Visualisation and Operations. It focuses on the dynamic requirements of the supporting systems and the need to automate manual tasks where possible for a more efficient operating environment. This automation is supported by the Data Collection and Analytics components and policy-based decisions in Orchestration.

A number of challenges need to be addressed by assurance:

- Assuring service quality levels in the face of changing customer demands requires a dynamic operating environment and supporting systems
- Management and analysis of abnormalities which are not captured by existing automations and policy must become automated and support social collaboration
- Monitoring, verification and improvement of autonomic response

7.2 State of Play and Future Direction

Existing Approaches and Limitations

The assurance systems currently in production have originated from an era where the mechanisms for ensuring quality levels were designed and created in a reactive rather than proactive environment (issue alarms). Traditionally, assurance systems are closed with supplier-specific EMS, with a general lack of flexibility, and they are deficient in supporting an agile operating environment. Assurance is made up of many solutions, requiring integration and Orchestration across products, while existing systems do not support the necessary dynamic requirements and promote a mostly manual assurance response.

Today, there are specific and complex management interfaces, and solutions providers implement EMS solutions on top of network equipment.

Future OSS Approach and Benefits

The drivers for change in assurance are ignited from the fault tolerant autonomically assured systems that OTT competitors operate. When examining assurance in the Future OSS, it must be designed to allow a proactive not a reactive response to new service introductions. The future approach will put dynamic systems and an ability to automate as priorities. To support this dynamic environment, it is necessary that solution providers support openness and interoperability via open APIs. The open APIs will allow Assurance metadata to be exposed as Catalogue resources.

Assurance systems will request the necessary context from Analytics to automate operations where possible. Existing operations tools and procedures (change of parameters, reset, filtering, etc.) need to be incorporated into the Future OSS platform through open APIs, which in turn will support automation of Event, Incident and Problem Management processes.

The Future OSS will need to be assured, as OSS actions are increasingly dynamic and real-time. The Future OSS must monitor and manage its own usage and operation to detect any possible abnormalities and implement resolution.

7.3 Detailed Solution

The Future OSS Assurance Domain is shown in the figure below:

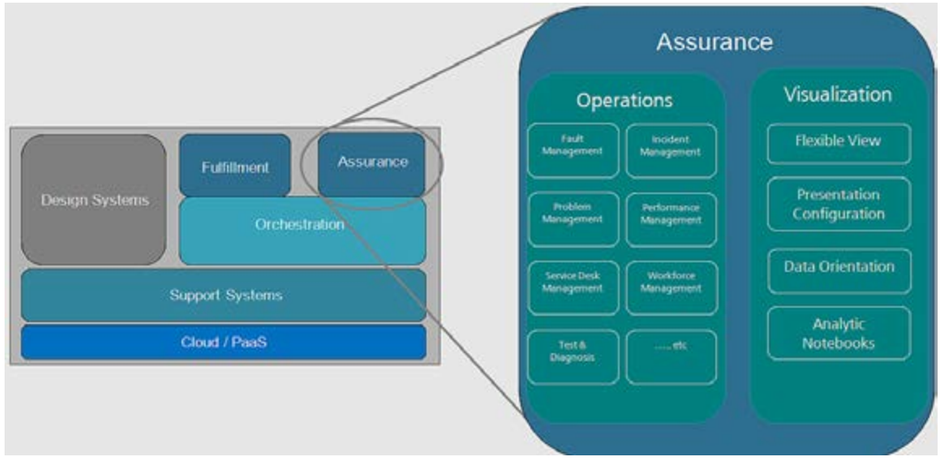


Figure 14: Future OSS Assurance Domain and Modules

Operations Module

The Operations module of the Assurance Domain contains the processes and applications that support human activities. The Future OSS will also allow operations to create new or updated applications by chaining the required microservices as necessary. The Operations module includes processes and a number of applications such as:

- **Dynamic Fault and Performance Management:** The dynamic nature of the environment necessitates that operations have the flexibility within Performance Management to define and create KPIs based on observations and incidents, and incorporate metrics from a combination of virtualised functions and legacy systems. Fault management systems require the ability to rapidly turn on or off, combine and modify optional alarm elements. Additionally, these applications will provide intelligent fault diagnosis, localisation and dynamic impact analysis utilising alarm pattern and service impact libraries. These functions will support resolution in an increasingly automatic way

- **Service Desk and Workforce Management** will be operated in an increasingly automated manner allowing operations to use analytics models to identify patterns, and define rules to manage tickets and work orders more efficiently. These applications will use the analytics models to automate ticket management through intelligent analysis and diagnosis that automatically processes tickets, creates work orders and provides intelligent work order dispatch, routing, verification and automatic closure

- **Test & Diagnosis** will support policy verification, incident resolution validation and the ability to test the end-to-end service chain on design or instantiation. For PNF, it is possible to activate probes to collect more data. Test & Diagnosis will verify policy and fault resolution by performing specific tests and analysis of logs. Policy verification requires the ability to monitor the changes performed as a result of policy based actions for a configurable period of time, after which the actions may be verified or rolled back. When a service chain is created, the Future OSS active test agents will also be instantiated and the chain tested with real data to ensure that the fulfillment has succeeded. Test and Diagnosis results will be analysed periodically to enable fine tuning of the processes, metadata and policy that drive assurance and fulfillment. Initially the process of fine tuning will be manual, but the Future OSS needs to support and evolve towards increasing self learning and regulation

- **Incident Management.** On assignment of opened tickets from Events/Alarms, will pursue all possible investigations to restore normal service operation as quickly as possible with the need to implement “work around” solutions. It will require the ability to track the history of automated and manual actions in support of rollback where these actions did not have the desired outcome. Incident Management will utilize all of the dynamic applications and analytics models available to it to diagnose and resolve the incident. Furthermore, a Runbooks & Social Collaboration component will allow operations to share, update and expand the knowledge base for automatic execution of routine procedures and operations

- **Problem Management** aims to find and resolve the root causes of incidents which are not resolved by Incident Management. This requires higher skills and cross-department cooperation, and will constitute a major part of the manual Operation work using DesignOps and DevOps

In the VoLTE Network Service use case described in Section 1.3, dynamic applications such as Fault and Performance Management will be used to troubleshoot the extent of the issue and attempt to resolve. If resolution is not possible, for example the automated response in Orchestration continues to fail, the Incident will be handed-over to Problem Management. Problem Management will perform fault diagnosis, localisation and impact analysis whilst also utilising Test and Diagnosis to verify the problem. It may require manual intervention to manage virtual functions to offload traffic or to create a work order to add physical capacities. Once resolved a follow up investigation and update of the automated response via the DevOps and DesignOps will be implemented.

Visualization

Operations need to quickly and easily visualize actionable insights. Visualization will provide views on the status of the network, fulfillment orders & transactions. It will provide a representation of the network configuration, provide a portrayal of the data quality and support social collaboration. It will monitor the status of the closed loops in Orchestration and support monitoring of event and incident management. Additionally it will provide the ability to support Problem Management allowing operations to perform historical insight into what happened and use root cause analysis information to automate future response. Visualisation will support a view of the history of actions performed to allow operations perform rollback if necessary to resolve incident.

Dynamic visualization should provide a number of aspects and qualities:

- **Flexible View.** A view of the network or service based on the who, why and what of the issues and analysis being examined. Additionally if problems are detected in the autonomic operation of a service, then these issues need to be represented in an operations view for further action

- **Presentation Configuration.** This is to configure the different types of views (for instance the dynamic inventory) supporting the required level of data analysis, including the integration to external systems to provide context or additional information to be displayed

- **Data Orientation.** Providing a data model to support the metrics, events, aggregation levels, configuration data and other relevant external data

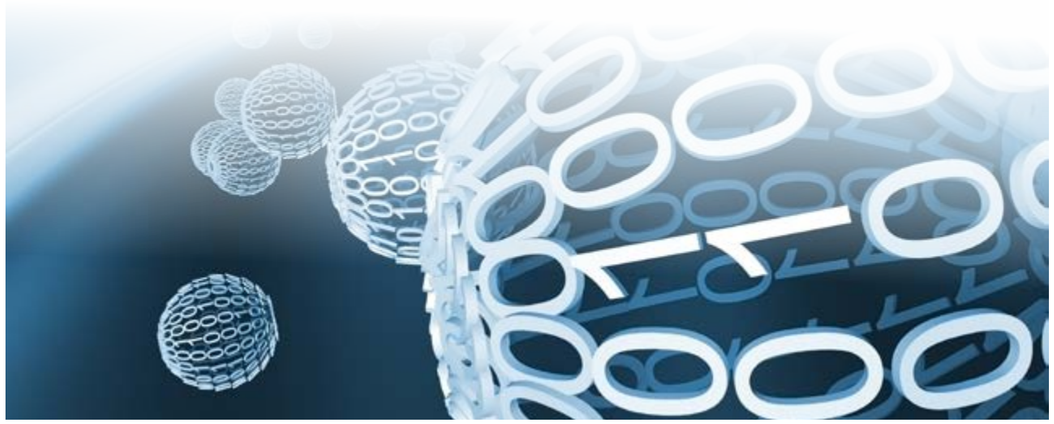
- **Analytic Notebooks.** To visualize the modelling and machine learning experimental work, these notebooks show the work in progress that can be shared and accessed by multiple parties for direct cooperation and execution

A flexible operating view is driven by a DevOps model, supporting the ability to create, view, update, share and delete views as monitored services change dynamically. Visualization will support open APIs for customer/partner/supplier portals, cross department analysis, integrations to context information.

7.4 Conclusion

The Future OSS Architecture contains an Assurance functional domain that supports the operations and dynamic visualization of hybrid networks for the following benefits:

- Operations empowerment providing dynamic applications where metrics and events can be enhanced, created and combined as per network and service conditions
- Allows the analysis of Incident Management to support increased ticket automation, intelligent dispatch and automation of repetitive, manual and time consuming tasks
- Facilitates the application of domain and operations knowledge in Problem Management to close the loop with Orchestration, utilizing DevOps, DesignOps, SRE and JAD to increase the automation of incident responses
- Increased efficiency, skills and knowledge through enhanced operations collaboration
- Provides operations with the tools to test and diagnose resolution, assist policy verification and support service instantiation
- Dynamic visualisation which supports the entire Future OSS and allows for more effective resolution, communication and greater collaboration



8. TRANSFORMATION

"Addressing the cultural shift and embracing Agile and Digital Operations"

8.1 Introduction

The Future OSS framework utilizes DevOps, DesignOps, Joint Agile Delivery (JAD), and Site Reliability Engineering (SRE). It is built on a technology based on cloud, APIs, microservices, SaaS, containers and PaaS. It includes agile practices such as Continuous Integration and Continuous Delivery, Push on Green, and autonomic event responses.

The process of transformation towards this new operating model is critical to ensuring that the Future OSS does not become just another product in a pool of point solutions. The Future OSS should be leveraged as a transformation enabler. Supporting this new OSS operating model requires significant changes to organizational structures, tearing down the traditional lifecycle-oriented silos, and aligning to new customer-centric KPIs across all teams, implying skill transformation [23] towards trouble shooting, problem management and design of policies, analytics, etc.

8.2 OSS Service Migration

OSS and Cloud Native Services

The nature of lifecycle management for cloud native and cloud scale services requires that OSS is treated as an integral part of design and build, and not just left as an operational handover matter. This is a structural (and sometimes difficult) transformation for people working on Think and Build. Cloud native services must also be capable of auto-scaling and autonomic fault response.

On-boarding of OSS components into systems (including installation and configuration of agents and network probes), and reconfiguration of OSS systems must be self-service and via API.

Legacy and Hybrid Workload Integration

The simple approach to legacy OSS workloads has been to leave them on the incumbent system, and use the new generation OSS only for new, cloud based workloads. This bi-modal approach severely limits the value of the operational transformation required.

The Future OSS approach recommends that the legacy workloads should be viewed the same as cloud workloads. Legacy workloads are merely those that – in their present state – rely more on human-led responses. There is no reason that, once an appropriate OSS framework and

culture is in place, that these, too, cannot benefit from automation and self-managed event response.

Legacy Networks and Resource Pools

The transformation of existing inventories may be difficult. Therefore the Future OSS is able to work with existing inventories (including cloud) as an input of its Dynamic Inventory, focusing on the repository of dynamic information not existing anywhere else. The Future OSS provides a flexible framework which can completely replace existing OSS applications while also allowing the retention of some of the existing legacy OSS applications (e.g. OSS applications that are costly, difficult or time consuming to transform). The retained legacy OSS applications will be "wrapped" within the Future OSS to allow them to participate in the hybrid end-to-end service orchestration.

Therefore, the goal of the Future OSS is that it must manage digital, legacy, and hybrid workloads. The division between legacy physical networks and cloud native virtualised network resources can be done as follows:

- **Cloud.** Cloud assets are those that are natively API based and support the self-service tenancy model. New asset purchases should all be of this type. These are full participants in the digital service ecosystem
- **Hybrid or Legacy "adapted".** Hybrid resources are those that can be converted into service domains. This would allow them to participate in the Future OSS hybrid service ecosystem to the extent that lifecycle services are defined for that service domain. Legacy resource pools can be brought under the control of this OSS by introducing a domain manager to provide a lifecycle-oriented API, and present this as a service domain to the Future OSS. The underlying legacy information model can be mapped to the common information model, and a consistent federated view of the data models can be produced through adaptation. Not all legacy domains will be able to fully participate in this ecosystem, but the ability to "wrap" such a service domain around manual fulfillment requests allows them to still participate in the wider service fulfillment landscape as shown in the figure below
- **Legacy not "adapted".** Certain physical assets will likely be deemed not worth conversion. This might be because of where they are in their asset lifecycle, or the cost of conversion. For example, it may be difficult to introduce a resource manager that supports automation of fulfillment functions. This would be evaluated against the benefit derived from revenue uplift in automated usage of that resource. In these circumstances, the strategy would be to minimize the footprint of these legacy services, and lifecycle them out as practical

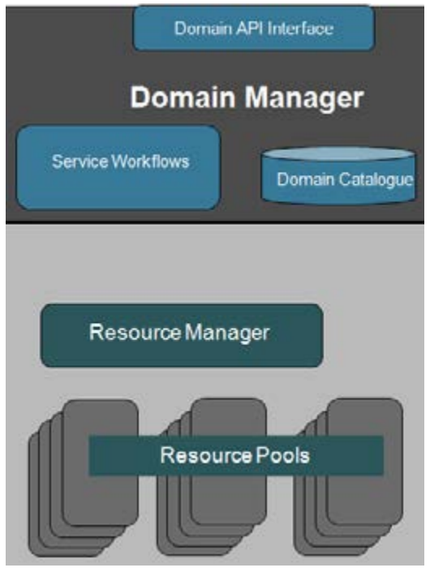


Figure 15: Legacy Resource Adaptation

The domain managers should all be located close to their resource pools, but also subject to resilience best practice. The Common Event Bus will link these many locations.

Relationship with Existing OSS Platforms

Whilst the intention and target state is to consolidate all OSS platforms down onto the single Future OSS platform, such a goal will take a significant amount of time. During this intervening period, two key patterns are expected to emerge:

- **Legacy OSS as a sub-domain of Future OSS.** The most powerful pattern is to install the Future OSS as the master OSS, sitting above all other OSS. This would use the service domain pattern to place each existing OSS as a service domain, and have it advertise a subsidiary OSS domain service catalogue. This approach would provide consistency of a northbound service catalogue (from the Future OSS), and introduce closed loop capabilities into services managed by existing OSS platforms
- **Retiring Legacy OSS.** Legacy systems could be left alone, with their legacy OSS in place. Such an arrangement would not bring any of the benefits of Future OSS to those systems. However, the Future OSS could be introduced directly into key systems in parallel with the legacy OSS, and ultimately replace the OSS for that system. This would ensure continuity of service, and also enable the full advantages of Future OSS as the system to be brought under its control, including the participation in digital and hybrid service chains, advanced alarm controls and analytics, and autonomic responses. Transformation may then be introduced in a phased manner where legacy systems that become fully supported by the Future OSS can be retired

9. CONCLUSION & BENEFITS

This White Paper has explained how the Future OSS can achieve agility for enabling digital operations transformation. The White Paper has defined that a common information model, model–driven processes, closed loops and autonomic operations should be the key concepts for designing the Future OSS architecture. These concepts need to be comprehensively adopted through modern cloud-based approaches.

The White Paper presents a holistic Future OSS architecture, where the traditional silos between different OSS processes are instead replaced with more cohesive building blocks where functional components for the Fulfillment and Assurance domains are built around Orchestration and closed loop autonomic principles. This cohesiveness is reinforced by the Design Systems that bridge the three Run-Time domains (e.g. Fulfillment, Assurance, Orchestration) via Support Systems, all hosted within a cloudified environment.

Agility is enabled by the integrated design and configuration of resources, analytics, and policies to support rapid service creation and deployment. The Design Systems within the Future OSS architecture supplies the necessary DevOps, DesignOps, SRE and JAD toolset for Think and Build processes. The cloud-native Future OSS includes a Visual Design Studio which facilitates CI/CD process models for unified operations.

In the Future OSS architecture, the Run-Time processes are supported by the Fulfillment and Assurance domains under the intelligent coordination of Orchestration. The interaction between these three domains is essential to ensure the cohesion from order capture through fulfillment, resource allocation and network configuration. Real-time data collection and analytics produces context-aware intelligence as input to the Orchestration to ensure appropriate restoration of the network and quality of service. Assurance deals with the automation of fallout cases where guided human intervention may be necessary.

The whole Think, Build and Run processes require the Support Systems to provide a common integrated view of metadata via the Catalogue, and instance data and state information via Dynamic Inventory. Adaptation supports the federation of this information with external cooperating systems.

The Future OSS architecture provides the microservice based platform to on-board OSS components into an open API enabled ecosystem, allowing “OSS-as-a-Service” to be realized. This allows physical network functions (PNF), virtual network functions (VNF) and hybrid service models to be incorporated into service operations in a cohesive manner, establishing true hybrid network management.

By adopting open APIs, the Future OSS architecture can be easily adopted and be interoperable with any industry initiative including open source and other industry developments. By enabling common event monitoring, the Future OSS establishes the ability to define, capture, collect, orient and enrich events in real-time to react with agility through designed-in policies.

The Future OSS is a common framework architecture and approach to create new operating models, and provides the following benefits:

- A blueprint from which CSPs can begin their digital operations transformation
- Technology best practice to create an end-to-end operations ecosystem
- A holistic OSS architecture for hybrid networks and services management across physical, virtual and legacy
- Intelligent building blocks to support autonomic operations and continuous improvement

The Future OSS is designed to act as a standardized framework that can be adopted by the whole industry, including CSPs and Open Source communities. The Open Source communities (e.g. ONAP, OPNFV, etc.) may use the concepts of the Future OSS for streamlining and aligning their architecture for the management of hybrid networks and services. For Orange, it is important to minimise the diversity of approaches, a way to improve responsiveness, the heart of Orange Essentials2020 plan.

The Future OSS framework will guide Huawei Infrastructure Enabling System (IES) product development and roadmap, and will act as a reference for Huawei’s OSS transformation initiatives (e.g. Open Roads [24], other customers, etc.).

This White Paper is an initial step in the cooperation between Orange and Huawei. The next release of the White Paper will provide practical use cases, examples and details for standardizing the implementation of the Future OSS in a real world. The program will produce further results in the future as this is an ongoing collaboration between both companies.

10. GLOSSARY

Term	Explanation
Architectural Domain	A top-level grouping of similar functional modules within the Future OSS arranged by a defined common purpose.
Artefacts	A generic term to refer to any deliverable which could be produced by any role in the software development lifecycle - anything from specifications to software to configurations.
Autonomics	A computing environment with the ability to manage itself and dynamically adapt to change in accordance with business policies and objectives. Self-managing environments can perform such activities based on situations they observe or sense in the IT environment rather than requiring IT professionals to initiate the task. These environments are self-configuring, self-healing, self-optimizing, and self-protecting. **Source: IBM MAPE**
Catalyst	Catalysts are proof-of-concept projects developed collaboratively by TM Forum members. These projects bring together companies large and small to create innovate solutions to common challenges demonstrating how this can be achieved leveraging key TM Forum best practices and standards. **Source: TM Forum**
Closed Loop	See OODA as an example of the Closed Loop.
Constraint	Additional information provided as part of an intent which qualifies it, or by policy internal to the provider of the intent capability – e.g. the QOS or bandwidth required between two points.
Context	A collection of measured (facts) and inferred knowledge that describes the state and environment in which an entity exists or has existed. **Source DEN-ng**

Control Domain	A segmentation of operational control focussed on a particular type of activity. This can be reflected in dedicated Future OSS functional modules and components responsible for achieving the outcome for that control domain, and can act on declarative of detailed intents from a peer or parent Future OSS system (or elsewhere).
DesignOps	A communication enabler between designers and engineers teams. In a way, DesignOps is the translator between these two teams.
DevOps	A clipped compound of "software DEVelopment" and "information technology OPerationS" is a term used to refer to a set of practices that emphasize the collaboration and communication of both software developers and information technology (IT) professionals while automating the process of software delivery and infrastructure changes. It aims at establishing a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably. **Source: Wikipedia**
Functional Component	Components represent lowest layer of function and final layer of recursion in the Future OSS architecture. Can be used within use cases and detailed description of operation, and form the basis of groupings of the microservices which are used to construct them.
Functional Module	The Future OSS main application ‘modules’ (which are constructed through one or several components at a lower layer). Allows description of the Future OSS through interaction between modules, and their grouping into overall Architectural Domains.
GANA	Generic Autonomic Network Architecture.
G-VNFM	Generic - Virtual Network Function Manager.
Intent	A specification of a desired end result or capability to be provided expressed between a provider and consumer of a service.
JAD	Joint Agile Delivery, referring to an adoption of best practices in collaborative software delivery.

Knowledge Plane	Knowledge Plane concept was first defined by David Clark. The ETSI GANA Knowledge Plane is inherited from the Clark's Knowledge Plane as a pervasive system within the network that builds and maintains high-level models of what the network is supposed to do, in order to provide services and advice to other elements of the network.
Metadata	Metadata generally refers to "data about data ", which can include specifications, configurations, recipes, workflows which can be associated with any entity within the Common Information Model.
Microservice	Microservice is a method of developing software applications as a suite of small, independently deployable, modular services in which each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goal.
Model-Based Configuration	Use of metadata applied to metadata-linked objects through a model structure to allow for reconfiguration of the system.
OODA	Observe, Orient, Decide, and Act – a methodology for describing human and non-human decision making. **Source: Wikipedia**
Policy	A definite goal or course of action to guide and determine present and future decisions. Policies are implemented or executed within a particular context. **Source: IETF RFC 3198**
SDO	Standards Development Organisation. An organisation whose primary activity is developing or interpreting technical standards for a group of adopters.
SRE	Site Reliability Engineering – use of software engineering expertise in operations, to automate efficiently as tasks grow.
WIM	WAN Infrastructure Manager, an example of a specialized VIM, typically used to establish connectivity between PNF endpoints in different NFVI-PoPs. **Source: ETSI NFV MANO GS**

11. REFERENCES

01. TM Forum Catalyst: Model-Driven Service Orchestration via FMO Architecture: https://www.tmforum.org/about-tm-forum/awards-and-recognition/catalyst-team-awards/

02. Ranganai Chaparadza, Tayeb Ben Meriem, John Strassner, Steven Wright; Joel Halpern - Joint SDOs/Fora Workshop hosted by TMForum during TMForum Live 2015 Nice! on "Industry Harmonization for Unified Standards on Autonomic Management & Control of Networks and Services, SDN, NFV, E2E Orchestration, and Software-oriented enablers for 5G": http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7063423

03. TMF Open APIs - https://www.tmforum.org/open-apis/

04. Orange researches on Open Platform and Orchestration: https://recherche.orange.com/vers-une-plateforme-ouverte-et-programmable-globalos

05. TMF GB922 Information Framework: https://www.tmforum.org/resources/collection/gb922-information-framework-sid-r16-5-1/

06. ONAP Reference: https://www.onap.org/

07. AT&T, Orange, Colt, MEF and TM Forum Pave the Way Toward Worldwide Ecosystem of Connected Service Provider Networks: https://www.orange.com/en/Press-Room/press-releases-2017/Industry-Leaders-Collaborate-on-Common-SDN-Business-API-Standards-for-Orchestrated-Services

08. AT&T and Orange Collaborate on Open Source and Standardization Initiatives for Software-Defined Networking: https://www.orange.com/en/content/download/38070/1158617/version/5/file/Final%20PR_AT%26T_Orange_SDN%207.20.16.pdf

09. OODA Loop: http://www.artofmanliness.com/2014/09/15/ooda-loop/

10. Strassner, John C, Agoulmine, Nazim and Lehtihet, Elyes. FOCALE - A Novel Autonomic Networking Architecture: http://repository.wit.ie/189/1/2006_LAACS_Strassner_et_al_final.pdf

11. ETSI GANA 16 Reference authored by Orange, Vodafone, Verizon, Telefonica, Fokus: http://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp16_gana_Ed1_20161011.pdf

12. David D. Clark, Craig Partridge, and J. Christopher Ramming - A knowledge plane for the Internet, In SIGCOMM, 2003: https://groups.csail.mit.edu/ana/Publications/PubPDFs/A%20

knowlege%20plane%20for%20the%20internet.pdf

13. ETSI NFV Architecture: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf

14. Roberto Kung on NFV: https://www.youtube.com/watch?v=9gPyfPCvX6g

15. Laurent Leboucher on ZOOM: https://www.youtube.com/watch?v=kvy_mLfd0vs

16. TR262 - Hybrid Network Management Platform Blueprint R16.5.1 Technical Report: https://www.tmforum.org/resources/standard/tr262-hybrid-network-management-platform-blueprint-r16-5-1/

17. IG1118 OSS/BSS Futures – Architecture R15.5.1 Exploratory Report: https://www.tmforum.org/resources/standard/ig1118-ossbss-futures-architecture-r15-5-1/

18. Digitizing With Huawei TelcoOS: http://www.huawei.com/en/events/mwc/2016/topics/digitizing-with-huawei-telco-os

19. IG1137 Transformation of NetOps to DevOps R16.0.1 Exploratory Report: https://www.tmforum.org/resources/collection/ig1137-agile-transformation-suite-r16-5-0/

20. GOOGLE Book: Site Reliability Engineering edited by Betsy Beyer, Chris Jones; Jennifer Petoff & Niall Richard Murphy: https://books.google.com/books/about/Site_Reliability_Engineering.html?id=81UrjwEACAAJ

21. IG1137A Joint Agile Delivery: Accelerating Value to the End User in a Value Fabric R16.5.1 Exploratory Report: https://www.tmforum.org/resources/standard/ig1137a-joint-agile-delivery-accelerating-value-to-the-end-user-in-a-value-fabric-r16-5-1/

22. TR235 - ZOOM Policy Model and Architecture Snapshot R14.5.1Exploratory Report: https://www.tmforum.org/resources/standard/tr235-zoom-policy-model-and-architecture-snapshot-r14-5-1/

23. IG1148 Communication Services Provider Workforce Transformation Implementation Experience R16.5.0 Exploratory Report: https://www.tmforum.org/resources/standard/ig1148-communication-services-provider-workforce-transformation-implementation-experience-r16-5-0/

24. Open Road Community: https://openroadscommunity.com