A Lightweight Software Stack and Synergetic Meta-Orchestration Framework for the Next Generation Compute Continuum

D4.1 Initial Release of Hyper-distributed Applications Synergetic Meta-Orchestration Framework, Development Environment and Repository

Document Identification			
Status	Final	**Due Date**	30/11/2023
Version	1.0	**Submission Date**	27/12/2023

Related WP	WP4	**Document Reference**	D4.1
Related Deliverable(s)	D2.1, D2.2, D3.1	**Dissemination Level (*)**	PU
Lead Participant	IBM	**Lead Author**	Sofiane Zemouri (IBM)
Contributors	ATOS, CNIT, NTUA, ODINS, SMILE, UOM, ZHAW	**Reviewers**	Leonardo Militano (ZHAW)
			Dimosthenis Masouros (NTUA)

Keywords:
Synergetic Meta-Orchestration, Hyper-Distributed Application, Cloud-to-Edge-to-IoT, Computing Continuum, HDA Registry.

Document Information

List of Contributors	
Name	Partner
Giovanni Toffetti	ZHAW
Leonardo Militano	ZHAW
Alejandro Arias Jiménez	ODINS
Guillermo Gomez	ATOS (EVIDEN)
Hai Long Ngo	SMILE
Panagiotis Papadimitriou	UOM
Georgios Papathanail	UOM
Lefteris Mamatas	UOM
Ilias Sakellariou	UOM
Angelos Pentelas	UOM
Maria Drintsoudi	UOM
Ioannis Tzanettis	NTUA
Anastasios Zafeiropoulos	NTUA
Dimitrios Spatharakis	NTUA
Symeon Papavassiliou	NTUA
Eleni Fotopoulou	NTUA
Nikolaos Filinis	NTUA
Constantinos Vassilakis	NTUA
Ioannis Dimolitsas	NTUA
Manolis Katsaragakis	NTUA
Chiara Lombardo	CNIT
Sofiane Zemouri	IBM

Document History			
Version	Date	Change editors	Changes
0.1	01/09/2023	IBM	First draft ToC and initial content
0.2	01/11/2023	NTUA	Updated ToC
0.3 – 0.8	02/11/2023 – 04/12/2023	ALL	Content added to all sections in the deliverable by all contributing partners
0.9	07/12/2023	IBM, NTUA, ZHAW	Internal review by IBM, consortium review by NTUA and ZHAW
1.0	22/12/2023	IBM	Final version to be submitted

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Sofiane Zemouri, John Farren, Gavin Shorten (IBM)	07/12/2023

Internal Reviewers	Leonardo Militano (ZHAW), Dimosthenis Masouros (NTUA)	15/12/2023
Project Coordinator	Symeon Papavassiliou (NTUA)	27/12/2023

Table of Contents

Document Information 2
Executive Summary 11
1 Introduction 12
2 NEPHELE Synergetic Meta-Orchestration Overview 14
2.1 Orchestration Approach 14
2.2 Positioning of the Synergetic Meta-Orchestration Framework 17
2.2.1 Workload placement and configuration solution 17
2.2.2 Inter-Cluster Communication state of the art 19
2.2.3 Monitoring and Observability Mechanisms 21
3 NEPHELE Synergetic Meta-Orchestration Approach 22
3.1 Intent Formulation and Management 22
3.1.1 Intent Concept and Definition 22
3.1.2 Intent Principles, Functionality and Lifecycle 22
3.1.3 Intent Description Language 24
3.1.4 Distributed application intent for the compute continuum 25
3.1.5 Distributed application intent description language and data model for the compute continuum 29
3.1.6 Implementation Roadmap 31
3.2 Multi-cluster Resources Management 32
3.3 Network Management and Interfaces 35
3.3.1 Description 38
3.3.2 Requirements 42
3.3.3 Design ideas 42
3.4 Security and Trust Management 42
3.5 Monitoring, Data Fusion and AI-assisted Orchestration 47
3.5.1 Observability 47
3.5.2 AI-based Techniques for Autonomy in the Computing Continuum 48
3.5.3 AI-assisted Methodology 50
3.5.4 3D Monitoring 51
4 NEPHELE Dashboard and Development Environment 54
4.1 Development Environment 54

4.1.1 Hyper-distributed Applications Development Environment and Registry 54
4.2 Dashboard 64
5 Implementation Status and Roadmap 65
5.1 Experimental Comparison of Inter-Cluster Communication Solutions 65
5.2 3D monitoring and multidimensional alerting 70
5.3 Security 72
5.4 Multi-cluster AI-assisted deployment 77
5.5 Implementation roadmap 78
6 Conclusions 79
7 References 80

List of Tables

Table 1: HDA OCI Image-spec compliance summary 59
Table 2: hdarctl commands summary 61
Table 3: Average Throughput between clusters (Nephele1-Nephele2) 68
Table 4: Average Throughput between clusters (Nephele1-Nephele3) 69

List of Figures

Figure 1: Architectural approach for the support of synergetic orchestration mechanisms in NEPHELE 16
Figure 2: Integration of the synergetic meta-orchestration framework with the VOs, cVOs, and hyper-distributed application components within the continuum domains 17
Figure 3 Conceptual workflow of OpenHorizon and its intervention level in the context of the NEPHELE synergetic meta-orchestration framework 18
Figure 4: Intent lifecycle [3], [6] 24
Figure 5: Distributed application graph representation 27
Figure 6: Structural elements of a service template and their relations [14] 30
Figure 7: An abstract compute node template example [13] 31
Figure 8: The Federated Resource Manager in the context of the multi-domain infrastructure of NEPHELE 35
Figure 9: A Network ecosystem instance composed by five network services made up of a variable number of xNFs 37
Figure 10: The graph of a network ecosystem with anchor points highlighting the link between the xNFs and the physical infrastructure. 37
Figure 11: Mapping between the physical resources (e.g., gNobeB and servers/datacenters) and the geographical areas 38
Figure 12: Network ecosystem metamodel 39
Figure 13: CCNM internal architecture 40
Figure 14: Blueprint methods as SAGA patterns 41
Figure 15: Decentralized Identifier example. 44
Figure 16: Overview of Decentralized Identifiers architecture 44
Figure 17: Components of Verifiable Credentials and Verifiable Presentations 45
Figure 18: Verifiable Credentials flow 45
Figure 19: Methodology utilizing signals to manage Cloud applications 48
Figure 20: High-level workflows and topologies [42] 50
Figure 21: AI-assisted methodology 51
Figure 22: Example of an interactive Palindrome.js configuration 53
Figure 23: T4.1 development commitments 55
Figure 24: HDA artifacts and HDAG services 57
Figure 25: OCI Image-Spec manifests 59
Figure 26: NEPHELE's specific artifacts CICD pipeline 60
Figure 27: Artifact naming convention in distribution-spec 62
Figure 28: Functional blocks of the internal architecture of the HDAR System module of the SMO framework .. 63
Figure 29: Remote Node Scenario 66
Figure 30: Enable Multi-Clustering with LIQO/KARMADA 66
Figure 31: NEPHELE Clusters 67
Figure 32: RTT between clusters (Nephele1 - Nephele2) 67
Figure 33: RTT between clusters (Nephele1 - Nephele3) 69
Figure 34: General architecture for the case of combining 3D monitoring and multidimensional alert 70
Figure 35: Colour recognition as observed from the top view of a single-color Palindrome.js, accompanied by the system's state 71
Figure 36: Colours identification from the side/ top view of Palindrome.js alongside the according volumes and states. 72
Figure 37: Colours identification for the case of multi-state detection with a threshold 72
Figure 38: Overview of security components deployed 74
Figure 39: Access Token in JWT Format and decoded 75
Figure 40: OIDC4VP flow between a cVO and a VO 76
Figure 41: Initial testing set-up 77

List of Acronyms

Abbreviation / acronym	Description
A&A	Authorization and Authentication
ABAC	Attribute-Based Access Control
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ARIMA	Autoregressive integrated moving average
BDVA	Big Data Value Association
CCNM	Compute Continuum Network Manager
CICD	Continuous Integration Continuous Development
CLI	Command Line Interface
CN	Cloud Native
CNCF	Cloud-Native Compute Foundation
CRUD	Create, Read, Update and Delete
CSAR	Cloud Service Archive
CV	Computer Vision
cVO	Composite Virtual Objects
DC	Data Centre
DID	Decentralized Identifier
DLT	Distributed Ledger Technology
DNN	Data Network Name
DSBA	Data Spaces Business Alliance
DSL	Domain Specific Languages
Dx.y	Deliverable number y belonging to WP x
EC	European Commission
FD	Federated Distillation
FL	Federated Learning
FRM	Federated Resource Manager
HDA	Hyper Distributed Application
HDAG	Hyper-Distributed Application Graph
HDAR	Hyper-Distributed Application Registry
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IAM	Identity and Access Management
IDSA	International Data Spaces Association
IETF	Internet Engineering Task Force
IBS	Intent-based system

IBN	Intent-based networking
IBO	Intent-based orchestration
IM	Information Model
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
KNF	Kubernetes Network Function
KPI	Key Performance Indicator
LDAP	Lightweight Directory Access Protocol
MaaS	Metal as a Service
MANO	Management and Orchestration
MARL	Multi-Agent Reinforcement Learning
MCRM	Multi Cluster Resource Manager
MILP	Mixed integer linear program
ML	Machine Learning
MORL	Multi-Objective Reinforcement Learning
NBI	North-Bound Interface
NF	Network Function
NFV	Network Function Virtualisation
NFVO	Network Function Virtualisation Orchestration
NLP	Natural Language Processing
NRM	Network Resource Manager
NS	Network Service
NSD	Network Service Descriptor
NSI	Network Service Instance
OCI	Open Container Initiative
OH	OpenHorizon
OIDC	OpenID Connect
OIDC4VP	OpenID Connect for Verifiable Presentations
OPA	Open Policy Agent
O-RAN	Open Radio Access Network
ORAS	OCI-Registry As Storage
OSM	Open-Source MANO
PaaS	Platform as a Service
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PMP	Policy Management Point
PNF	Physical Network Function

QoS	Quality of Service
RBAC	Role-based access control
ReL	Representation Learning
REST	Representational State Transfer
RL	Reinforcement Learning
RTT	Round Trip Time
SBA	Service-Based Architecture
SBOM	Software Bill of Material
SIOPv2	Self-Issued OpenID Provider v2
SMO	Synergetic Meta Orchestrator
TAC	Tracking Area Code
TOSCA	Topology and Orchestration Specification for Cloud Applications
UC	Use Case
URI	Uniform Resource Identifier
VC	Verifiable Credential
VIM	Virtual Infrastructure Manager
VNF	Virtual Network Function
VNFD	Virtual Network Function Descriptor
VO	Virtual Object
VOStack	Virtual Object Stack
VP	Verifiable Presentation
VPN	Virtual Private Network
VXLAN	Virtual Extensible LAN
W3C	The World Wide Web Consortium
WP	Work Package
XACML	Extensible Access Control Markup Language
XML	Extensible Control Markup Language

Executive Summary

This deliverable regards the initial release of two iterations covering the design and implementation of the Synergetic Meta-Orchestration (SMO) framework, the development environment, and the repository of the NEPHELE platform. The deliverable is in conformity with Tasks 4.1 to 4.5 in the NEPHELE Grant Agreement (GA). The current deliverable represents an overview of the Synergetic Meta-Orchestration approach, as well as a critical evaluation of the main functionalities and open-source solutions that are considered for the integration into the framework. This includes the formulation of intents, the multi-cluster resource management, the network management and interfaces, security and trust management, as well as monitoring, data fusion and AI-assisted orchestration. In addition, this deliverable provides a detailed description of the work performed as part of the hyper-distributed application development environment and the registry, with a glimpse into the plans for a dual-purpose dashboard integration. Finally, an update on the implementation status is provided as a retrospective view of the efforts spent and the implementation roadmap serving as a guideline for the coming period until the final release of this deliverable. This document is in line with and structured around (i) the Grant Agreement task descriptions; (ii) feedback from the plenary meetings held at a consortium level; and (iii) progress made until M15 (Nov 2023). This document has also the following dependencies: (a) Requirements, Use Cases Description and Conceptualization of the NEPHELE Reference Architecture described in Deliverable 2.1 [1]; and (b) Initial Release of VOStack Layers and Intelligence Mechanisms on IoT Devices described in Deliverable 3.1 [2], which will be released in parallel with this deliverable.

1 Introduction

The NEPHELE project aims to enable the seamless orchestration of hyper-distributed applications across a continuum of computing resources – from the cloud to the far edge, including IoT environments. To achieve this, the project tackles several important technology barriers related to interoperability, and the convergence of technologies across the computing continuum. One of the main building blocks of such a solution lay in the enablement of a strong synergy between the components of a system of systems to devise an effective meta-orchestration of hyper-distributed applications on the continuum. Upon achieving this, NEPHELE strives to infuse automation and decentralized intelligence into this landscape.

The deliverable is part of WP4 and encapsulates the following tasks as defined in the Grant Agreement:

- Task 4.1. Hyper-distributed Applications Development Environment and Repository
- Task 4.2. Federated Compute Resources Management
- Task 4.3. Compute Continuum Network Management
- Task 4.4. AI-assisted Synergetic Orchestration
- Task 4.5. Cybersecurity and Trust Management

The primary goal of this deliverable is to provide an overview of the synergetic meta-orchestration framework. A "system of systems" approach is adopted, envisioning a framework that harmonizes elements across cloud, edge computing and IoT realms. An implementation roadmap is also described in detail.

The outputs from this deliverable outline the framework of the hyper-distributed applications synergetic meta-orchestration, development environment, and repository. This initial release marks a first milestone, showcasing the foundation upon which NEPHELE's orchestrated ecosystem will evolve.

Within the NEPHELE project framework, WP4, focusing on the development of the NEPHELE synergetic orchestration, serves as a pivotal component that interacts with various work packages, contributing to the overall success of the project. Its inputs, interactions, and outputs play a crucial role in shaping the project's trajectory.

- Inputs from WP2 (Synergetic Orchestration Requirements): WP4 draws its fundamental requirements from WP2, which serves as a foundational source for defining the essential prerequisites and specifications necessary for the design and development of the NEPHELE Synergetic Orchestration. The inputs from Deliverable 2.1 [1] provide crucial insights into the core functionalities and performance benchmarks required for the orchestration platform.
- Collaboration with WP3 (Autonomic Functionalities and Ad Hoc Clouds Management): WP4 and WP3 share a symbiotic relationship crucial to the development of the synergetic orchestration platform within the NEPHELE project. This collaboration extends beyond traditional orchestration requirements and includes the integration of the Virtual Object Stack (VOStack), which augments the functionalities and capabilities of the orchestration platform. WP4 operates in close collaboration with WP3 to delineate the autonomic functionalities and ad hoc clouds management aspects required for the synergetic orchestration platform. The outputs from WP3 and Deliverable 3.1 [2], particularly the orchestration management interfaces, serve as critical building blocks that inform and influence the design and implementation of NEPHELE's orchestration functionalities.

- Outputs towards WP5 (NEPHELE Platform Integration Testing and Refinement): The synergetic orchestration platform produced in WP4 constitutes a pivotal element that integrates and interfaces with the wider NEPHELE components. As part of WP5, this platform is deployed, tested, and refined within the broader NEPHELE ecosystem.
- Contribution to WP6 (Use Case Design, Implementation, and Evaluation): The NEPHELE synergetic orchestration capabilities are showcased in WP6's use case testing and evaluation, and rely heavily on the robustness and adaptability of the orchestration platform produced in WP4.
- Support for WP7 (Open Calls Support Programme) and WP8 (Dissemination, Communication, and Exploitation): The technical documentation generated in WP4 serves as an input for WP7's Open Calls Support Programme, providing comprehensive insights into the functionalities and methodologies employed within the synergetic orchestration framework. Additionally, the dissemination and exploitation materials produced by WP4 contribute directly to WP8, aiding in the communication of NEPHELE's achievements, capabilities, and potential impact to a wider audience.

The rest of this document is structured in 5 major sections:

Section 2 presents an overview of the NEPHELE synergetic meta-orchestration framework with an emphasis on the high-level orchestration approach followed and a brief state-of-the-art of the NEPHELE componentry.

Section 3 details the synergetic meta-orchestration concepts and building blocks including the high-level orchestration intent formulation and management, the multi-cluster resources management, the continuum network management and interfaces, the security and trust management, as well as the monitoring, data fusion and AI-assisted orchestration.

Section 4 describes the hyper-distributed applications development environment and the registry used as part of the NEPHELE Framework. The concept of dashboards is also addressed in this chapter.

Section 5 focuses on the implementation work performed to date in the context of the various aspects and tasks covered in this deliverable**.** An implementation roadmap is also devised, which describes the steps that will be undertaken to reach the objectives outlined.

Section 6 closes the deliverable by providing a high-level summary of the outputs of the deliverable, outlining the next steps, and providing concluding remarks.

2 NEPHELE Synergetic Meta-Orchestration Overview

In this section, the synergetic meta-orchestration overview is given, with an emphasis on the orchestration approach followed as well as the positioning of the solutions considered to enable the essential functions of the NEPHELE meta-orchestration.

2.1 Orchestration Approach

The NEPHELE synergetic orchestration framework is focused on the seamless and efficient orchestration of hyper-distributed applications across the compute continuum, which encompasses IoT, edge computing, and cloud infrastructure. The NEPHELE framework adopts a "system of systems" management approach, bringing together independent systems into a cohesive and unified orchestration ecosystem. Open-source modular orchestration platforms are not only integrated but will also be extended to ensure interoperability within the overarching NEPHELE meta-orchestration framework. Even if the proposed approach is generic enough and can be applicable in various orchestration solutions, NEPHELE recognizes Kubernetes[1] as the pivotal cornerstone of its operational landscape.

However, the uniqueness of NEPHELE goes beyond this fundamental “system of systems” approach. The key distinguishing features revolve around the critical role it plays by providing support for essential functions, particularly emphasizing:

1. Workload placement and reconfiguration: NEPHELE places strong emphasis on workload initial placement and dynamic runtime re-allocation. The framework should allow applications to be dynamically and intelligently placed across various clusters (e.g., Kubernetes clusters), ensuring they run where they are most needed. This adaptability extends to real-time reconfiguration to address evolving conditions and requirements.
2. Inter-cluster Connection and communication across Kubernetes clusters: A robust, secure, and efficient inter-cluster communication is a fundamental requirement of the NEPHELE synergetic orchestration framework. It intends to overcome the complexities often associated with cross-cluster communication and ensuring that resources and services flow seamlessly (e.g., across Kubernetes environments), to promote interoperability and synergy.
3. Storage and data high-availability: The nature of hyper-distributed applications prompts the need for remote execution of stateful workloads within a potentially scattered geographic area. This needs to ensure heightened reliability without the complexities of managing numerous independent application replicas.
4. Monitoring, adaptability, and optimization: Monitoring capabilities that are tailored to orchestration environments, providing deep insights into cluster status and performance are an essential building block to enable true hyper-distribution of application workloads in the continuum. These insights will empower the framework to adapt and optimize orchestration dynamically, based on real-time data. The focus is on achieving optimal resource utilization, system performance, and response to multi-cluster orchestration challenges.
5. Security and trust management: The system requires a robust security layer that safeguards hyper-distributed applications across the continuum. This encompasses comprehensive mechanisms ensuring secure communication and interactions between diverse elements, spanning from cloud to edge to IoT environments. Encryption protocols, authentication frameworks, and authorization mechanisms, will all contribute to improve the system's

[1] https://kubernetes.io/

resilience against potential threats and ensuring data integrity and confidentiality throughout the orchestration process.

The implementation of the NEPHELE synergetic orchestration framework is based on Kubernetes environments, making them the focal point of its orchestration strategy. This approach enables NEPHELE to address the unique challenges and opportunities within such environments, with a focus on inter-cluster communication, workload (re-)placement, and the capacity to monitor, adapt, and optimize hyper-distributed applications. The aim of the NEPHELE synergetic orchestration framework is to provide a resilient and adaptive orchestration solution, ensuring that the various orchestration components operate seamlessly and efficiently while catering to the diverse and ever-changing demands of hyper-distributed applications and modern computing in general.

A more detailed view of the architectural approach for the support of synergetic orchestration mechanisms in NEPHELE is depicted in Figure 1. A system of systems approach is adopted where an upper-level entity has the responsibility for the overall lifecycle management of the deployment and runtime of a distributed application, while the control is distributed in various entities following a hierarchical approach.

The Synergetic Meta-Orchestrator is the main entity that undertakes the responsibility for serving the deployment and lifecycle management of distributed applications over programmable resources[2] in the computing continuum. It undertakes a deployment request through the dashboard and prepares a deployment plan by considering both the current status and available resources of the underlying infrastructure as well as user-defined intents. Continuous monitoring and event-triggered optimization actions may be produced and forwarded to managers in the lower levels of the hierarchy in the system of systems approach. Specifically, the SMO interacts with the multi-cluster resource manager and the network resource manager.

The multi-cluster resource manager is responsible for managing the compute resources multi-cloud clusters. The Multi Cluster Resource Manager assumes that different clusters (each operated by a different Kubernetes instance) are registered to a unified continuum. Such clusters may span from the IoT to the edge to the cloud part of the infrastructure. Actions related to placement, security, migration, elasticity, and VO discovery are supported by the multi-cluster resource manager across the available clusters. A federated monitoring engine is also available that fuses information coming from the various monitoring engines at cluster level.

The network resource manager is responsible for providing network performance guarantees to the distributed application deployments in a multi-cluster infrastructure. It is activated in the event that a network operator is engaged in the deployment. In this case, end-to-end network slice lifecycle management is provided by the network resource manager, including processes for continuous optimization of network resources usage. In the event that a network operator is not engaged, connectivity services among the clusters is provided from the multi-cluster resource manager. In the latter case, less network performance guarantees can be supported. Interaction of the multi-cluster resource manager and the network resource manager is also envisaged based on the specification of Application Programming Interfaces (APIs).

At cluster level, a Cluster Manager is activated per cluster, responsible for guiding the deployment of a part of the application graph in its cluster. Local orchestration actions and collection of monitoring information is applied at this level. A cluster manager can be responsible for the management of a cloud computing cluster, an edge computing cluster or managing resources within a computationally strong IoT device or cluster of IoT devices.

[2] https://www.ciena.com/insights/what-is/What-Is-a-Programmable-Infrastructure.html

A development environment and a dashboard are made available to provide a set of tools to application developers and providers to use the NEPHELE platform. Application developers can use the development environment to validate and register VOs, cVOs and application graphs, while also producing indicative helm descriptors that can be used at the deployment phase. The dashboard provides a set of views to application developers and providers to support validation actions for the developed descriptors, both for application graphs and their intent, as well as interfaces for managing the registered images in the platform, monitoring the registered resources, and providing access to deployment views.

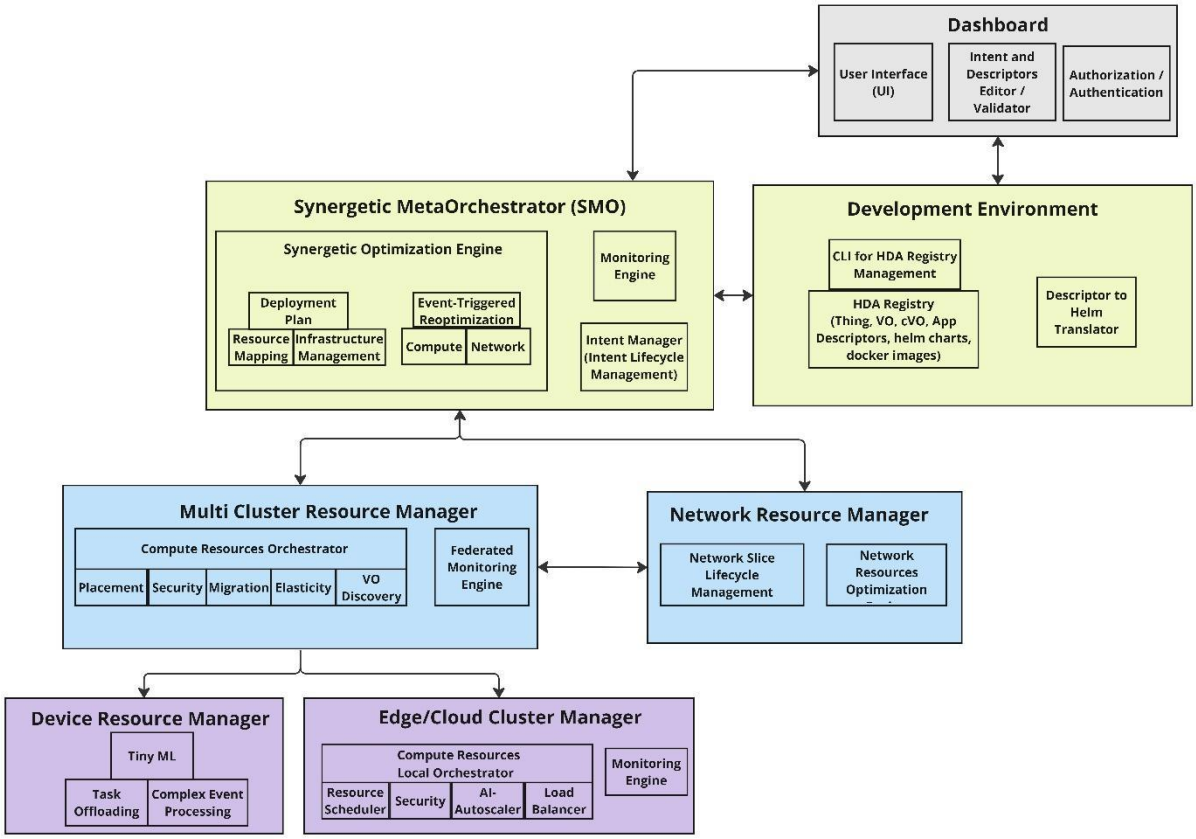

Figure 1: Architectural approach for the support of synergetic orchestration mechanisms in NEPHELE

The synergetic meta-orchestration framework sits in the cloud domain within the computing continuum. However, part or all of its functions and components can be offloaded towards the edge. Its relationship to the different hyper-distributed application components developed in WP3 (such as Virtual Objects, Composite-Virtual Objects) is described in Figure 2. The application artefacts (Application component, VO, cVO) span across the Cloud and Edge domains, while the IoT devices sit in the far edge of the continuum.

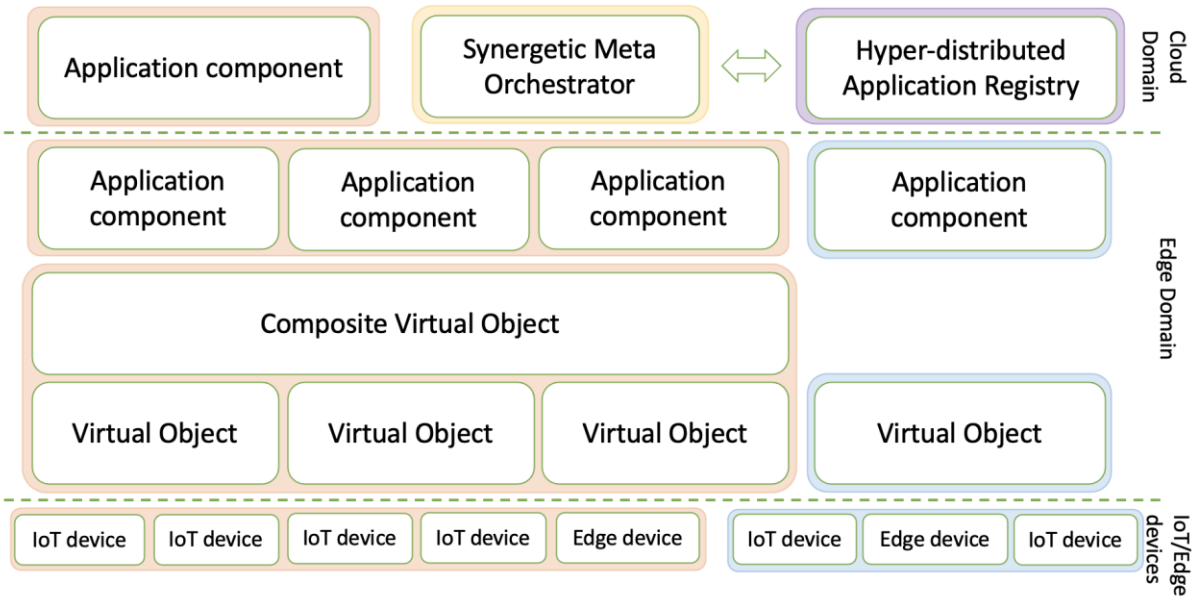

Figure 2: Integration of the synergetic meta-orchestration framework with the VOs, cVOs, and hyper-distributed application components within the continuum domains

2.2 Positioning of the Synergetic Meta-Orchestration Framework

This section depicts the positioning of the NEPHELE's synergetic meta-orchestration with regards to the emerging orchestration approaches in the context of the computing continuum.

2.2.1 Workload placement and configuration solution

To enable orchestrating workload placement and configuration within the NEPHELE platform, the open-source project OpenHorizon[3] is a potential tool of choice. OpenHorizon serves as a sophisticated tool that enables the seamless deployment, management, and dynamic allocation of workloads across the diverse computing continuum present in NEPHELE. The following is an overview of the OpenHorizon features and capabilities:

1. Dynamic workload placement: OpenHorizon's core functionality lies in its ability to intelligently place workloads across various nodes within the NEPHELE ecosystem. Leveraging policy-based decision-making, OpenHorizon assesses the requirements of individual workloads and dynamically places them on nodes that best match their needs. This dynamic placement considers factors such as resource availability, node capabilities, latency considerations, and workload characteristics, ensuring optimal performance and resource utilization.
2. Policy-driven configuration: One of the key strengths of OpenHorizon is its policy-driven approach to workload configuration. Through defined policies and rulesets, OpenHorizon governs the configuration of workloads across the continuum. These policies encompass aspects such as resource allocation, security requirements, networking configurations, and performance expectations. By adhering to these policies, OpenHorizon ensures consistent and compliant configuration of workloads throughout the NEPHELE platform.
3. Multi-cluster resource management: In a distributed environment like NEPHELE, comprising Kubernetes clusters, telecom networks, and independent Linux nodes, OpenHorizon's multi-

[3] https://lfedge.org/projects/open-horizon/

cluster management capabilities are perfectly suited. It seamlessly manages resources across these diverse clusters, ensuring that workloads are optimally placed while taking advantage of the available resources in different environments. OpenHorizon's adaptability to various cluster architectures and its ability to extend the orchestration across different infrastructures make it an asset in the NEPHELE platform.

4. Adaptability and autonomy: OpenHorizon's adaptability and autonomy significantly contribute to the resilience and responsiveness of the NEPHELE platform. It possesses self-healing capabilities, allowing it to automatically respond to changes in the environment or workload requirements. This autonomy ensures that the platform maintains its operational efficiency even in dynamic and challenging scenarios, reducing manual intervention, and enhancing system reliability.
5. Integration and interoperability: OpenHorizon's design emphasizes interoperability and integration with other components within the NEPHELE ecosystem. Its compatibility with various frameworks, orchestration tools, and computing environments ensures seamless integration with NEPHELE's architecture, enabling effective communication and coordination between different components.

OpenHorizon is composed of a centralised component called the Management Hub, which usually sits closer to the Edge; as well as an agent, which is deployed in each edge cluster/resource. Figure 3 shows the conceptual workflow of OpenHorizon and the level at which it is integrated in the NEPHELE synergetic meta-orchestration framework.

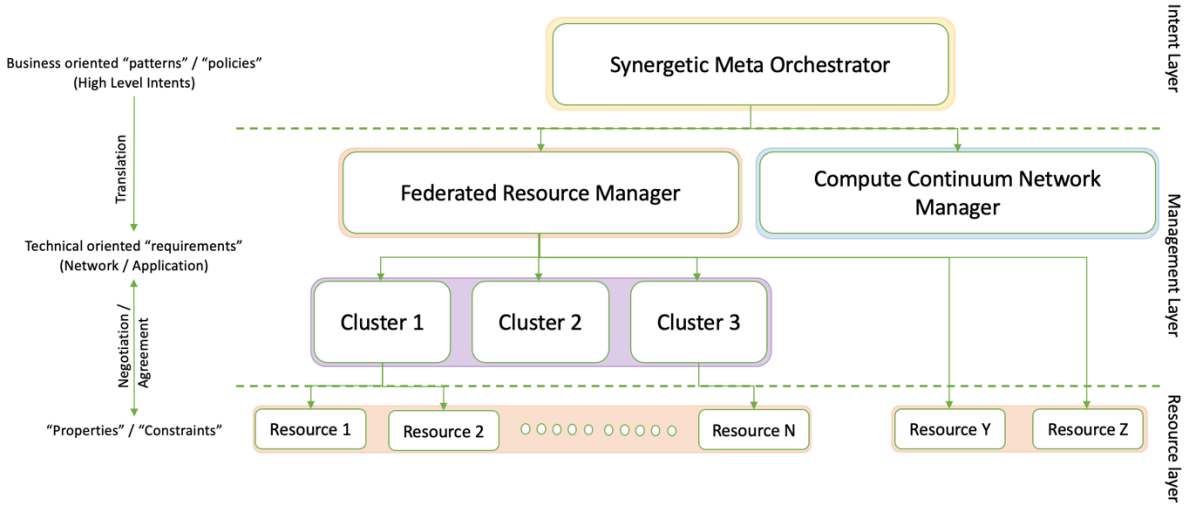

Figure 3 Conceptual workflow of OpenHorizon and its intervention level in the context of the NEPHELE synergetic meta-orchestration framework

The following represents the terminology used in Figure 3, and a description of its meaning in the context of the workload placement task.

- **Patterns**: related to high level business intents
- **Policies**: related to high level application intents
- **Requirements**: related to network / application KPIs
- **Properties**: specific to the resource / device type

- **Constraints**: specific to the state of the resource / device

2.2.2 Inter-Cluster Communication state of the art

To address the requirement for enabling and provisioning inter-cluster communication in Kubernetes environments, it is crucial to consider the integration of open-source technologies into the NEPHELE framework, which offer innovative solutions for inter-cluster communications and augment the NEPHELE platform's capabilities.

Examples for such technologies are LIQO[4] and SKUPPER[5]. LIQO is a Kubernetes extension that simplifies inter-cluster communication by creating a dynamic and secure overlay network between clusters. SKUPPER and KARMADA are two technologies that can be considered for inter-cluster communication within the NEPHELE platform. In the following, we will devise a comparative study of all three approaches with the pros and cons of each, in accordance with the synergetic orchestration requirements reported in Deliverable 2.1.

LIQO's operating mode transforms the conventional Kubernetes architecture into a federated, distributed system where clusters interact seamlessly and share resources and workloads across geographies and infrastructure types. This approach enhances resource utilization, resilience, and flexibility, making it well-suited for multi-cloud, edge, and hybrid environments.

Pros:

- Simple setup: LIQO provides a straightforward setup process, making it suitable for integrating with NEPHELE's complex framework.
- Simple setup: LIQO provides a straightforward setup process, making it suitable for integrating with NEPHELE's complex framework.
- Namespace-based isolation: NEPHELE can leverage LIQO's namespace-based isolation to ensure that clusters can communicate without risking data interference.
- Single interfacing point: LIQO offers a unified point of interaction (main Kubernetes cluster), streamlining communication across clusters.
- Virtual node for resource monitoring: NEPHELE can utilize LIQO's virtual node to monitor resources on remote clusters efficiently.
- VPN-based networking: LIQO employs VPN-based networking for robust and secure cluster communication, aligning with NEPHELE's need for reliability.
- Control and data plane separation: NEPHELE can explore LIQO's potential for separating control and data planes to enhance overall flexibility.

Cons:

- One-way offloading only: LIQO primarily supports one-way offloading, which may limit certain use cases in NEPHELE.
- Complex configuration for cluster mesh: Creating a mesh of clusters using LIQO could lead to complex configurations in NEPHELE.

[4] https://liqo.io/
[5] https://skupper.io/

SKUPPER enables secure communication across Kubernetes clusters by defining an ad-hoc virtual networking substrate. It implements multi-cluster services in namespaces exposed in the SKUPPER network. When a service is exposed, SKUPPER creates endpoints, making them available on the entire cluster.

Pros:

- Simple setup: Like LIQO, SKUPPER offers a straightforward setup process, simplifying integration with NEPHELE.
- Layer 7 networking: SKUPPER operates at layer 7, allowing each cluster to act as a layer 7 router, which enhances the versatility of inter-cluster communication.
- Mesh of clusters: SKUPPER's support for creating a mesh of clusters can facilitate NEPHELE's multi-cluster communication requirements.
- Namespace-based Isolation: NEPHELE can utilize SKUPPER's namespace-based isolation to ensure secure and controlled communication.
- Integration at higher layers: SKUPPER allows for integration at higher layers with workload placement solutions, aligning with NEPHELE's extensibility goals.

Cons:

- Performance degradation: SKUPPER uses queuing-based solutions at layer 7, which could introduce performance overhead, impacting certain latency-sensitive applications within NEPHELE.
- Complex monitoring: NEPHELE may need to invest more effort in monitoring and managing SKUPPER's complex configuration.
- No offloading: Unlike LIQO, SKUPPER doesn't support offloading, which can limit certain use cases.

Kubernetes Armada, also known as KARMADA[6], is a Kubernetes-native management tool that can be used across multiple clouds and clusters. Karmada lets users run their cloud-native apps across various Kubernetes clusters and clouds without making any changes to the apps themselves. This is achieved by providing centralized multi-cloud management, high availability, failure recovery, and traffic scheduling. Although Karmada is not considered an inter-cluster communication tool, as it relies on Submariner[7] to connect the clusters, the following is a list of its pros and cons in the context of NEPHLE.

Pros:

- Simple setup: Like LIQO and Skupper, Karmada provides a simple setup process well documented.
- Layer 3 networking: Karmada uses Submariner to provide cross-cluster L3 connectivity using encrypted or unencrypted connections.
- Easy integration: Karmada works well with Kubernetes resources that are already in place, making operations easy and reliable.

[6] https://karmada.io/
[7] https://submariner.io/

- Scalability: Karmada is designed to support large-scale cluster management.

Cons:

- No offloading: Compared to LIQO, Karmada does not offer offloading.
- Learning gap: Users may find it difficult to understand the concept of Karmada, and how the Karmada API works.
- Scheduling granularity: Karmada's coarser scheduler might not match the fine-grained control some use cases may require.
- Networking troubleshooting overhead: Submariner adds some troubleshooting overhead.

2.2.3 Monitoring and Observability Mechanisms

In NEPHELE we are considering various solutions to support observability of the deployment of distributed applications over multi-cluster infrastructure. The objective is to devise an observability stack that can support monitoring of different type of signals, including metrics, traces and logs for both compute and network resources. Open-source tools originating from the Cloud Native landscape[8] are going to be adopted. Some of the tools considered include the Prometheus Monitoring engine for collecting time series data for resources consumption and application metrics, for creating compound metrics and the specification of alerts. Distributed monitoring tools such as Thanos[9] are also considered for collecting metrics coming from different monitoring engines (e.g., when having a monitoring engine per cluster). OpenTelemetry[10] specifications are considered for the representation of metrics and traces. For the latter, tools such as Zipkin[11] and Jaeger[12] are being considered. Logging systems such as flutentd[13] and logstash[14] are also considered as part of the monitoring infrastructure developed in NEPHELE. Based on the different type of signals, data fusion mechanisms will be developed, as detailed in Section 3.5. Over the collected and fused data, proper interfaces will be provided to make such data available to the various orchestration mechanisms developed in NEPHELE per component.

[8] https://landscape.cncf.io/
[9] https://thanos.io/
[10] https://opentelemetry.io/
[11] https://zipkin.io/
[12] https://www.jaegertracing.io/
[13] https://www.fluentd.org/
[14] https://www.elastic.co/logstash

3 NEPHELE Synergetic Meta-Orchestration Approach

3.1 Intent Formulation and Management

3.1.1 Intent Concept and Definition

Intent refers to a collection of operational goals and outcomes expressed in a purely declarative manner, outlining what needs to be achieved rather than delving into implementation specifics [3]. The concept of intent finds diverse applications. Intent-based networking (IBN) aims to automate and manage network infrastructures, while in the realm of services and applications, it focuses on deploying and managing modern services executed within cloud infrastructures, encompassing compute, storage, and network resources. In all instances, intent offers a high-level abstraction, enabling the management of complex infrastructures or deployments in a domain-agnostic manner. Examples expressed in natural language, indicating what is and what is not an intent, for the case of IBN are listed in [3]. A representative subset follows here:

1. Examples of intent
    a. *"Steer networking traffic originating from endpoints in one geography away from a second geography, unless the destination lies in that second geography"* (states what to achieve, not how.)
    b. *"Maximize network utilization even if it means trading off service levels (such as latency, loss) unless service levels have deteriorated 20% or more from their historical mean"* (a desired outcome, with a set of constraints for additional guidance, that does not specify how to achieve this.)
    c. *"Ensure VPN services have path protection at all times for all paths"* (a desired outcome of which it may not be clear how it can be precisely accommodated.)
2. Examples of what would not constitute an intent
    a. *"Configure a given interface with an IP address"* (This would be considered device configuration and manipulating configuration parameters, not an intent.)
    b. *"When interface utilization exceeds a specific threshold, emit an alert"* (This would be a rule that can help support network automation, but a simple rule is not an intent.)
    c. *"Configure a VPN with a tunnel from A to B over path P"* (This would be considered as a configuration of a service.)
    d. *"Deny traffic to prefix P1 unless it is traffic from prefix P2"* (This would be an example of an access policy or a firewall rule, not intent.)

An intent does not deal with the how to achieve the goals set or how to implement what is declared. Thus, a configuration of a device or a service is not an intent; a policy which is a set of rules modelled around the variation of events/conditions/actions is not an intent; a service model which is a data model to describe instances of services offered to the customers is also not an intent. Intent is a high-level declaration of goals and desired outcomes without any specification on the how the goals are satisfied, or how outcomes are achieved; this would require the intent layer not to be agnostic to the underlying infrastructure, and for declarations to be low-level, specific to devices and services operation.

3.1.2 Intent Principles, Functionality and Lifecycle

The following *properties/principles* are those governing an intent. An intent is:

Non-prescriptive: Intent requests specify "what" services are requested but do not detail "how" those services are to be delivered. Decisions concerning assignment and use of resources are left entirely to the infrastructure/services provider. Implementation choices regarding technology, vendor, media, node, port, link, path, server, virtual machine, etc. are left to the provider and therefore such parameters never appear in intent requests [4].

Declarative: A consumer "declares" in an intent what it is expected from the provider to deliver. The consumer system does not have to "ask" the provider system for any information (such as topology) to do so.

Contextual: An intent brings context. Key to detecting and resolving conflicts is to be able to examine the "what I need" description, rather than the "how to do it" description.

Invariant: Intents don't change because of changes in the infrastructure. Intents can change dynamically, but a given expression of intent is invariant [5]. The set of validated invariant intent expressions, and the records of the operational states, enable comparison of the intended/desired state and actual/operational states of the system and determine any drift between them.

Portable: The same intent request may be presented – without variation – to any provider (**Provider agnostic/independent)** with which the consumer is permitted to inter-operate; it is portable across implementation choices and changes. Because intent is invariant with respect to implementation choices, it is portable across implementation choices and changes.

Refinable: Ideally the user expresses an intent, and then the system takes over all subsequent operations (zero- or one- touch). However, reaching the state of a well-formed and valid intent expression is not a one-shot process. Depending on the level of abstraction, the intent expressions may initially contain implicit parts and imprecise or unknown parameters and constraints. Parsing, understanding, and refining the intent expression is required to reach a well-formed and valid intent expression that can be further used by the system for the fulfilment and assurance operations. Intent conflicts should be detectable and resolved, either they are static (compile-time) conflicts or dynamic (run-time) conflicts [7].

Furthermore, as an implication of the above properties; an intent is **composable** and **scalable.** An easy-to-use interface should be provided to simplify the interaction of the intent users with the intent system.

An intent-based system (IBS) should support all the intent properties listed while at the same time support autonomy and supervision, learning and capability exposure. The basic *functionalities* of an IBS are:

Intent Formulation: Provides all the functions for the generation of a well-formed intent. The intent description is done with a simple set of instructions by the user through a typical human-machine interface. Analysis and interaction with the user should follow in an intent recognition process, to clarify any existing conflicts and provide for completeness of requirements declared. Then, an actionable user intent is generated and processed, with the context of the intent, for detailed analysis of its requirements and compatibility for deployment.

Intent Fulfilment: Provides all the functions and interfaces that allow users to communicate an intent to the provider and that perform the necessary actions to ensure that intent is achieved; intent ingestion and interaction with users, intent translation, intent orchestration.

Intent Assurance: Provides functions and interfaces that allow users to validate and monitor that the provider is indeed adhering to and complying with intent. This is necessary to assess the effectiveness of actions taken as part of fulfilment, providing important feedback that allows those functions to be trained or tuned over time to optimize outcomes. In addition, Intent Assurance is necessary to address

"intent drift." Intent drift occurs when a system originally meets the intent, but over time gradually allows its behaviour to change or be affected until it no longer does, or does so in a less effective manner. Related functionality includes Monitoring, Intent Compliance Assessment, Intent Compliance Actions, Abstraction, Aggregation, Reporting.

In Figure 4 the intent lifecycle is presented [3][6]. In the user space a well-formed well-expressed intent is formulated after probably several iterations. In the processing space an intent is translated to a machine-readable form to be implemented in the implementation space. An “inner” control loop between the processing and the implementation spaces makes sure automatic adjustments would take place. These are the result of monitoring, analysis and validation in order to maintain intent’s goals and expected outcomes and avoid “Intent drift”. An “outer” control loop extends to the user space. It includes the user acting and adjusting its intent based on observations and feedback.

Reporting back to the user space may include intent fulfilment information, intent conflict information (static or run-time) and intent feasibility information (regarding the required capabilities or the availability for deployment) [7].

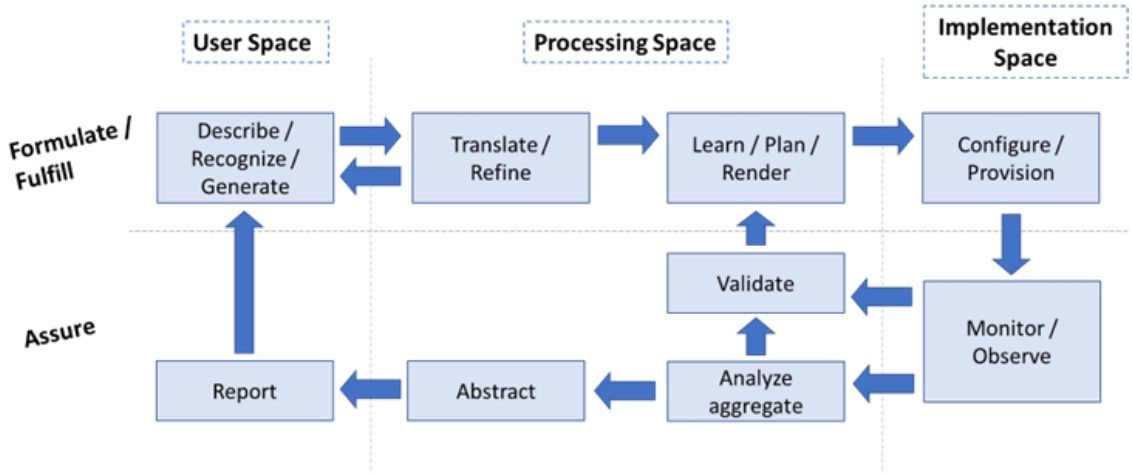

Figure 4: Intent lifecycle [3], [6]

According to IETF [7] intents are classified into transient and persistent base on their life-cycle management. A transient intent has no lifecycle management; as soon as the specified operation is successfully carried out, the intent is finished and can no longer affects the target object. A persistent intent is always active and follows a life-cycle management like the one pictured in Figure 4 until it is deactivated or removed. Furthermore, IETF [8] clarifies what an intent represents for different stakeholders through a classification on various dimensions, such as solutions, intent users, and intent types.

3.1.3 Intent Description Language

An intent is described in a high-level human-readable language before properly translated in a machine-readable language specifying low-level implementation specifics. To facilitate the intent lifecycle, from a well-formed well-expressed description to implementation, the description should be mapped into a proper structured data model. Thus, an intent description should use a human-readable intent description language supporting the level of expressiveness required to enable a user to describe the goals, constraints, and desired outcomes. The intent description should also be mapped into a well-structured data model to facilitate parsing and validation of the intent description itself, as well as intent fulfilment and implementation. It is considered that it would be ideal if an extensible

data model could support all the way from the abstract description of an intent to a machine-readable instantiation of the intent to the infrastructure.

Approaches presented regarding the intent description process, either develop a custom-made language and data model usually employing Natural Language Processing (NLP) and Machine Learning (ML) techniques for the intent translation [9] or utilize and extend Domain Specific Languages (DSL) and data models initially targeted to network or cloud automation to capture intent properties.

NEMO language [10] [11] is a DSL based for the abstraction of network models and conclusion of operation patterns. It provides North-Bound Interface (NBI) fashion in the form of language. NEMO language enables network users/applications to describe their demands for network resources, services, and logical operations in an intuitive way. The NEMO language description can be explained and executed by a language engine.

YANG (Yet Another Next Generation) [12] is a data modelling language for the definition of data sent over network management protocols such as the NETCONF and RESTCONF. The YANG data modelling language can be used to model both configuration data as well as state data of network elements and define the format of event notifications emitted by network elements. The language, being protocol independent, can then be converted into any encoding format representing data structures, e.g. XML or JSON, that the network configuration protocol supports.

OASIS TOSCA (Topology and Orchestration Specification for Cloud Applications) provides a domain-specific object-oriented language to specify applications and hardware resources independent of a particular cloud platform [13] [14]. Application developers create TOSCA templates according to this language while TOSCA orchestrators parse the templates and translate the statements into deployment actions.

Regardless of the domain addressing, NEMO, YANG and TOSCA are targeted to automate the configuration and management either of networks devices or cloud entities; they have been used alone or in combination in several implementations. However, all of them have been utilised to capture the properties of an intent or in other words to capture the abstraction and expressiveness required to formulate an intent. The advantage is the use of the same language and data models all the way from expressing a human-readable abstract intent to expressing a machine-readable configuration and implementation specific. To our perception TOSCA is a highly sophisticated object-oriented language supporting all the necessary data structures and customisation properties to facilitate the creation of a well-formed abstract intent which will be then refined and translated into a specific deployment expressed in the same enhanced TOSCA model.

3.1.4 Distributed application intent for the compute continuum

A distributed application is represented as an application graph consisting of application nodes and interconnection links. The nodes of the graph represent application components or consumed third-party services illustrating part of the business logic of the application. All of them are appropriately linked together in an overlay network, delivering the full application functionality in a synergetic way. More details about application graphs can be found in Deliverable 3.1. Declarative statements accompany the application graph in the form of tags (reflecting a value range for a metric or a set property) or simplified expressions. These statements address nodes and/or links or the whole application and aim to specify the application's objective and constraints regarding its deployment and operation to deliver the required results. An **application's objective** refers to an optional high-level goal set to pursue when the application is deployed in allocated infrastructural resources and executed along its lifecycle (e.g., energy efficiency, real time response, high proximity to the users etc.). There may be more than one objective declared. An **application's constraints** refer to:

1. Measurable **requirements** for each node and overlay link mostly specifying quantity and quality demands regarding required resources (i.e., computational power needed for an application component to run, overlay link bandwidth required between two application nodes, Quality of Service level expressing the value range for a set of network metrics, required for an overlay link etc.);
2. **Properties** enabling or setting required characteristics or functionalities for nodes and/or links (i.e., a secure overlay connection, autoscaling support for an application component etc.);
3. **Restrictions** applying to single node or link or a group of nodes or links, expressed as simplified expressions (i.e., a required exclusive access to an application component, locality specifications for a node or a group of nodes, a collocation demand for a group of nodes etc.).

The set of constraints are further characterized as hard and soft if they should be satisfied by all means during the application's deployment and operation or if they should be treated in a best effort manner to be satisfied, respectively. The application's optional objective set is soft by nature since during deployment and operation there will be a pursuit for optimizing the associated objective function in the best possible way.

In Figure 5, a visual representation of an indicative application graph is provided. In Figure 5, the main application components that compose the application graph are depicted, along with their constraints in terms of requirements, properties, and restrictions. A high-level application objective (such as energy efficiency) is also declared. This abstract visual representation is in fact a visual representation of the application's intent to be injected into the processing space (see Figure 4) and be enhanced with implementation specifics before the application is implemented. This is a visual representation of an intent that is described in a proper intent description language and a data model, to then serve as an input to the intent fulfilment phase.

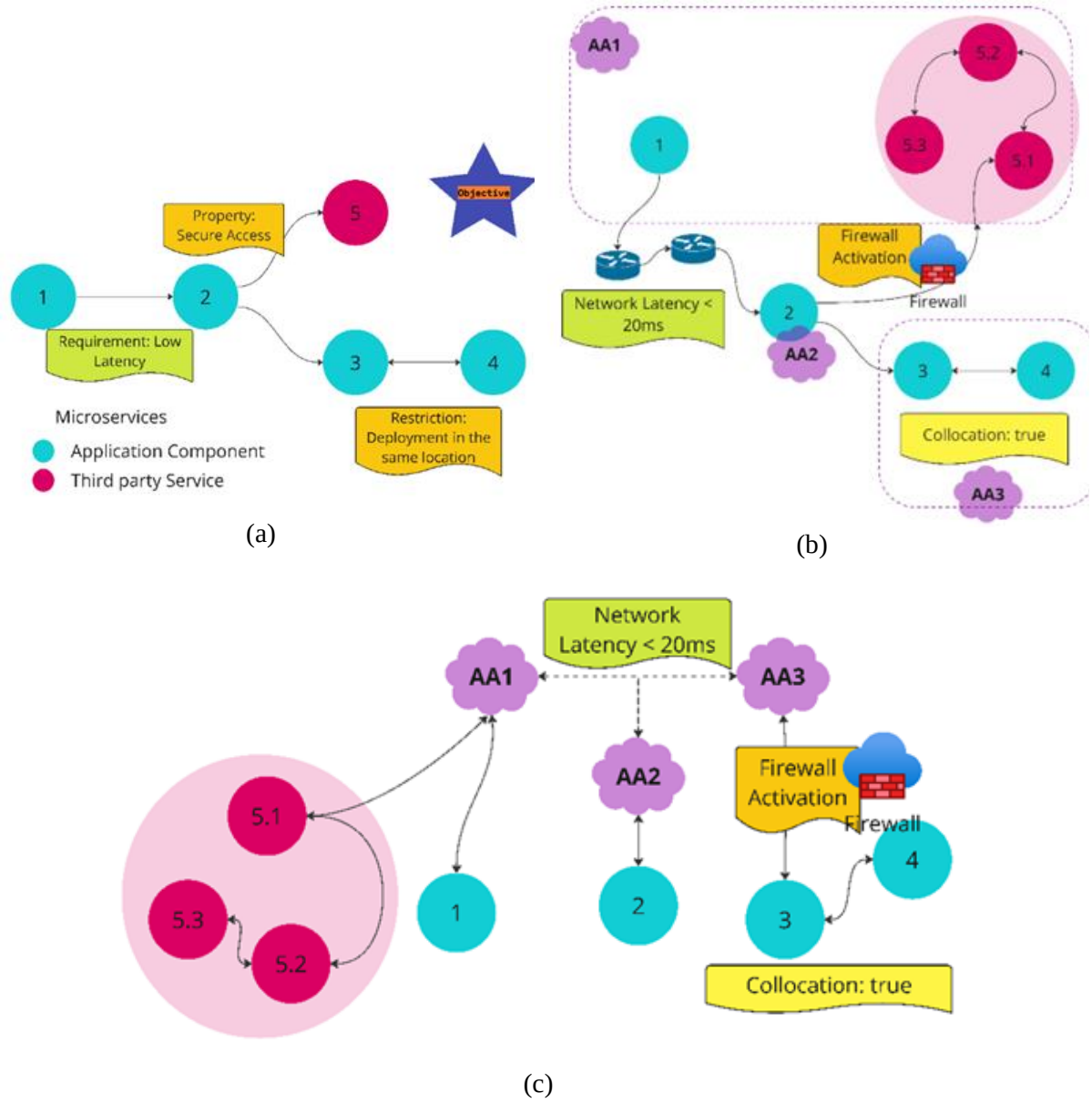

Figure 5: Distributed application graph representation

This intent-based declarative definition of the distributed application and associated overlay network is extended in Figure 5b as a first step to the intent fulfilment phase; the application graph is augmented with the inclusion of services that illustrate specified requirements, properties, and restrictions. Third-party services chain structure is revealed while network-oriented functionalities are decomposed to network services required to be activated for the illustration of each functionality (e.g., activation of a network firewall, enforcement of a routing policy, activation of network observability mechanisms). Furthermore, each component in the augmented application graph is tagged with the administration authority (AA) that is responsible for the deployment and operation of each application component. In the computing continuum various providers (compute infrastructure providers, network infrastructure providers, services providers, edge and far edge infrastructure providers) collaborate to make the offerings from each different administrative domain accessible to the

application developer and achieve through a negotiation and observability framework a synergetic deployment and orchestration of distributed applications not running under the control of a single authority. The high-level objective declared in the intent, depending on the case, might be at this step expressed as simple as a Key Performance Indicator (KPI) reaching a target value or even an objective function of an optimisation problem requiring to be minimised or maximised.

In Figure 5c the augmented application graph is partitioned per authority responsible for the deployment and operation of a group of nodes, while connection between authorities is represented with virtual links between Administrative Authorities service offering connection points. The latter are considered as the negotiation and communication points that illustrate bilateral agreed policies towards making it possible for a distributed application partially operating in different administrative domains to efficiently meet its objective and satisfy its constraints.

After these enhancements, the model is further enhanced to include instantiation and configuration specifics and lead to application's implementation and execution governed by all the necessary functions for intent assurance along intent's lifecycle.

The description of an application intent for the compute continuum is composed by **several sub-intents**. The application intent declares goals and outcomes regarding application software components, third party services, network interconnection and network services, cloud and edge hosting environments, virtual objects (VOs) and composite virtual objects (cVOs) as virtual representations and enhancements of the IoT domain, business perspective. Thus, it can be seen as it comprises of several sub-intents each of them potentially to be injected and be managed in a different authoritative domain. Furthermore, the same sub-intent may have a different mapping depending on the provider assigned to illustrate it. Depending on the management schema, the life cycle of the application intent may be considered as a whole or as involving several sub-intent lifecycles interconnected and each of them fulfilling and assuring part of the application intent towards satisfying the overall set goals and expected outcomes. Another way to express the same saying is that an intent application intent comprises of several sub-intents each of them matching to a **different intent stakeholder** (service provider, network provider, IoT provider, cloud provider).

Intent description is not considered as a one-shot process. Intent refinements are done during the intent formulation phase but also during the intent fulfilment and assurance phase either in an automated manner or manual after a recommendation to the user.

Reaching the state of coming up with a well-formed well-expressed intent involves a series of refinements coming out of the following validation processes at different phases of the intent life cycle:

1. **Conflict detection and resolution**: It is the process of detecting and resolving a conflict which is the case of incompatible declarations, conflicting declared objectives (improvement in reaching one objective may mean getting away from reaching another), conflicting constraints with objectives (some constraint may not let an objective to be reached). It is a process that should run during design but as well during run-time.
2. **Model efficiency validation**: It is the process to detect and resolve the case of unnecessary declarations or redundant ones during intent design time.
3. **Request feasibility check**: It is the process to detect the case when an intent declares objectives and constraints that cannot be satisfied by the provider due to fact that it does not possess or offers the required capabilities.
4. **Deployment feasibility check**: It is the process to detect the case when the provider does not have the requires availability in resources to fulfil the intent. Such a case may be true when an application is about to be deployed but also during run-time at some time in a deployed application's lifetime.

5. **Run-time intent assurance validation**: Apart from the previously mentioned processes, a constant validation happens in the intent assurance phase to assure intent fulfilment; capturing the intent "drift", in terms of satisfying constraints and objectives, by automatic reconfigurations in the implementation space and reporting back to the user space.

3.1.5 Distributed application intent description language and data model for the compute continuum

Representing and describing an application intent requires a DSL and data model able to express the several components comprising an application graph, their relationships, constraints as defined; applied on sole or grouped components and relationships, objectives. Furthermore, it may facilitate decompositions (of complex services or objects) and enhancements taking place at the intent fulfilment phase. It is considered beneficial to be able to use the same language and data model, or at least with minimum required translations all the way from the abstract definition of an intent to a machine-readable implementation plan.

OASIS TOSCA [13] [14] object-oriented language and associated data model is considered to support the level of expressiveness, flexibility and customisation required to capture most of the requirements. *Furthermore, translation to other DSLs such as NEMO and YANG is supported while TOSCA also has been used to represent the abstraction of infrastructural resources as well as the description of SLAs.* TOSCA supersedes several fit for purpose automation deployment technologies since, as a sophisticated language and as it is widely supported, not only may cover such automation tasks but also may describe and support complex management schemas as those in the compute continuum.

"The TOSCA language introduces a YAML-based grammar for creating service templates that define the lifecycle management of application, infrastructure, and network services. The language defines a *metamodel* for specifying both the structure of a service as well as its management aspects. Within a TOSCA file, a *Service Template* defines the *structure* of a service. *Interfaces, Operations, and Workflows* define how service elements can be created and terminated as well as how they can be managed during their whole lifetimes. Policies specify operational behaviour of the service such as quality-of-service objectives, performance objectives, and security constraints, and allow for closed-loop automation." [14] The major elements defining a service are depicted in Figure 6.

TOSCA service templates consist of topology templates, where application developers instantiate pre-defined node and relationship types into node templates and relationship definitions [15]. A node may provide several capabilities, while requiring capabilities (requirements) from other nodes. A node can also have properties and expose interfaces for performing certain actions. The operation implementations for the interfaces are external and can for example be Bash scripts or Ansible playbooks, which interface with cloud platforms. Query functions are supported, and properties may take values according to defined inputs. Custom new types of relationships and new types of nodes can be defined.

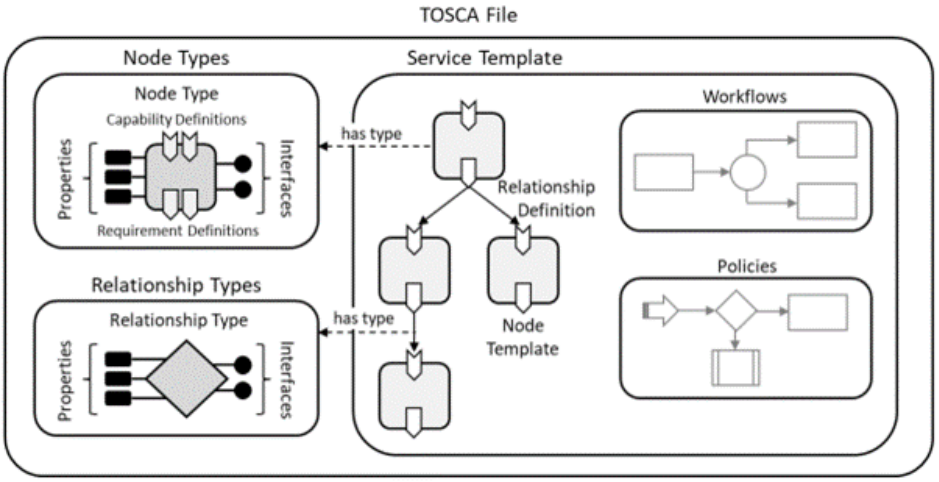

Figure 6: Structural elements of a service template and their relations [14]

The TOSCA specification also supports workflows to instruct an orchestrator to execute tasks in a specific order. Policies can further instruct orchestrators to make deployment decisions within the policy bounds (e.g., a scaling policy). Service templates and type definitions are packaged into a Cloud Service Archive (CSAR), which can also contain artifacts, such as installers, software container images or VM images.

TOSCA supports inheritance as well as selection and substitution mechanisms. These two mechanisms are used by TOSCA templates to connect to TOSCA nodes and services defined by other TOSCA templates; (1) the selection mechanism allows a node instance created a-priori by another service template to be selected for usage (i.e., building relationships) to the current TOSCA template, (2) the substitution mechanism allows a node instance to be represented by a service created simultaneously via a substitution template.

```
tosca_definitions_version: tosca_simple_yaml_1_3

description: Template with requirements against hosting infrastructure.

topology_template:
  inputs:
    # omitted here for brevity

  node_templates:
    mysql:
      type: tosca.nodes.DBMS.MySQL
      properties:
        # omitted here for brevity
      requirements:
        - host: mysql_compute

    # Abstract node template (placeholder) to be selected by provider
    mysql_compute:
      type: Compute
      directives: [ select ]
      node_filter:
        capabilities:
          - host:
              properties:
                num_cpus: { equal: 2 }
                mem_size: { greater_or_equal: 2 GB }
          - os:
              properties:
                architecture: { equal: x86_64 }
                type: linux
                distribution: ubuntu
```

Figure 7: An abstract compute node template example [13]

TOSCA supports abstract node templates. This is a node template that doesn't define any implementations for the TOSCA lifecycle management operations, as depicted in Figure 7. Service shows an abstract compute node template example. designers explicitly mark node templates as abstract using the substitute directive. TOSCA orchestrators provide implementations for abstract node templates by finding substituting templates for those node templates [14].

An abstract service template along with the other properties TOSCA offers may clearly be used to represent an application intent and appropriately be enhanced during the fulfilment phase.

3.1.6 Implementation Roadmap

The development roadmap to fully support an application intent lifecycle follows.

1. **Intent description language customisation**: Definition of the data schema, customization, and extension of TOSCA entities to support an application intent as described.
2. **Intent definition and visualization toolbox development**: Development of a toolbox to support visual representation of an application intent conforming with intent description language and data model.
3. **Intent validation processes development**: Algorithmic design, development, and integration of all the processes to validate an application intent; parsing for syntax and model quality check, conflict detection and resolution, feasibility checking.
4. **Intent integration and life-cycle support**: Definition and development of all those processes to support intent fulfilment and assurance; intent decomposition, model translations, orchestrators interoperability.

OASIS TOSCA wide adoption resulted in the availability of several different kinds of tools in the context of TOSCA service templates. TOSCA modelling tools allow creating TOSCA service templates with a

graphical user interface, where for example nodes can be connected using relationships and configurations can be entered before generating a CSAR file [16]. TOSCA processors or parsers can parse TOSCA service templates [17]. TOSCA orchestrators process the information extracted from CSAR files, make deployment decisions employing scripts or Infrastructure-as-Code tools to carry out the actual deployment [18] [19]. Furthermore, several implementations are available to facilitate converting modules described in other DSLs (such as YANG, NEMO, ANSIBLE, HELM etc.) to TOSCA type definitions and vice versa [19] [20].

3.2 Multi-cluster Resources Management

Orchestration in the context of multi-cluster resource management within the compute continuum involves coordinating and managing various resources across multiple clusters to ensure efficient, reliable, and scalable operation of applications. The orchestration aspects that emerge in such a complex environment are multifaceted and the specification and implementation of a holistic framework, which will incorporate all the necessary mechanisms, encompasses several challenges. Given the distributed deployment of services, which are described based on an application graph, the challenges for a multi-domain orchestration framework are, mainly, related to:

- How to efficiently place components of a distributed application across multiple domains Decisions regarding when to deploy application components on specific clusters must be optimized. The overarching objective is to minimize the overall cost of resource usage while making judicious placement decisions. This involves considering several factors such as resource utilization, network latency, resource availability, elasticity, and energy consumption.
- How to enable the dynamic scaling of resources to adapt to fluctuating workloads by making real-time adjustments based on demand patterns. By adapting to dynamically conditions the orchestration framework will be able to ensure efficient resource allocation and optimize the overall multi-domain system performance.
- How to provide resilient deployments, by ensuring fault tolerance through distributed resource allocation and avoiding single points of failure and dynamically adjust resource allocations (placement re-optimization) to meet defined Quality of Service metrics and guarantee optimal application performance under varying conditions.

The inherent challenges in multi-domain orchestration unfold as specific optimization problems, each requiring sophisticated solutions. Particularly, the distributed placement of an application graph can be considered a case of virtual network embedding problem, which in its distributed version aims at determining the placement of virtual service nodes among distinct edge and cloud infrastructures, under several resource, network, and application-related constraints. The complexity of such problems increases considering the compute continuum application ecosystem, that involves IoT-related applications, where parameters such end-device mobility, are included in the optimization modelling. The NEPHELE meta-orchestrator aims at including holistic optimization engines that undertake the placement of application graph and the scheduling of the containerized application nodes in the compute continuum. To deal with the respective challenges, several placement strategies will be considered. Typical problems like bin packing, where the goal is to efficiently allocate virtual network elements within physical resources, find relevance. Heuristic approaches, leveraging rule-based methods for quick decision-making, are appropriate in scenarios demanding rapid application deployments, and dealing with the increased complexity of such problems. Multi-Criteria Decision Making (MCDM) methods, by considering multiple factors simultaneously, offer a holistic perspective, aligning well with the intricate demands of virtual network embedding. In navigating these placement

challenges, NEPHELE meta-orchestration framework will employ a spectrum of strategies, incorporating established methodologies and innovative approaches, to ensure optimal resource utilization and responsiveness in the distributed deployment of applications across diverse infrastructures in the compute continuum. Besides the general goal of minimizing the overall cost of resource usage while meeting Quality of Service (QoS) requirements of applications, specific placement objectives of these strategies concern:

- **Dependency**-aware placement**:** Consider dependencies and communication patterns between application components to minimize inter-component communication latency.
- **Resource**-affinity placement**:** Identify and leverage resource affinities (e.g., requirement for GPU accelerators), ensuring that certain components are placed on clusters with specialized hardware or configuration.
- **Latency-aware** placement**:** Minimize end-to-end latency for applications with components deployed across multiple clusters.
- **Intent-aware** placement: Taking into account aspects of the intents such as energy consumption and security when determining the placement of application components within the infrastructure.

An important aspect of multi-cluster resource management also concerns the need for dynamic allocation of resources, particularly in response to the ever-changing demands and conditions within a distributed computing environment. This dynamic allocation is a strategic approach that seeks to optimize resource utilization across multiple clusters in real-time, adapting to fluctuations in workload, user demands, and the availability of resources within each cluster. In a multi-cluster environment, workloads may vary significantly across clusters. Static resource allocation models, which allocate a fixed number of resources to each cluster, may result in suboptimal performance and resource wastage during periods of low demand, or conversely, resource shortages during peak demand. Dynamic resource allocation addresses these challenges where the meta-orchestration system continually assesses the current state of each cluster, monitor the performance of applications, and adjust resource allocations dynamically based on the observed demand. This adaptability could enhance the efficiency and responsiveness of the entire multi-cluster system, ensuring that resources are allocated where and when they are needed the most.

Moreover, dynamic resource allocation aligns with the principles of scalability and elasticity in the compute continuum. It enables the system to scale resources up or down based on the current demand, ensuring that applications receive the necessary resources for optimal performance without overprovisioning or under provisioning. This approach supports the efficient use of resources, cost-effectiveness, and the ability to handle varying workloads.

To tackle the challenges of dynamic resource allocation and autoscaling, focusing on the corresponding optimization problems, the NEPHELE meta-orchestrator excels in integrating intelligent autoscaling policies, collaborating with the local cluster managers, and implementing algorithms that autonomously adjust resources. This entails a constant optimization process, ensuring that resource utilization aligns with real-time demand patterns. It incorporates predictive scaling algorithms leveraging historical data, and machine learning (unsupervised learning, reinforcement learning) models to anticipate future demand patterns. It also incorporates reactive strategies by performing dynamic resource redistribution and continually optimizing allocations. In doing so, it aims to achieve a balance between high applications performance and low-cost resource usage. This, also, involves the development of mechanisms that enforce policies at a cluster level, continuously monitor the workload, trigger scaling events, and consider migration strategies to adapt to changes in the environment.

A key objective of the framework is to achieve resilient application deployments, necessitating the seamless integration of the designed mechanisms with the local cluster orchestration tools. The above discussed global-optimization objectives regarding distributed resource management demand a holistic view and control over the entire multi-cluster deployment, cross-cluster coordination, and timely high-level policy enforcement. Thus, the development of essential mechanisms, such as resource discovery and policy enforcement, becomes paramount. These mechanisms are critical for ensuring the timely collection of monitoring metrics that feed into global optimization processes. Finally, it is worth mentioning that the network resource management is of great importance for efficient orchestration and management, as it ensures seamless communication between application components. Therefore, ensuring effective network management is key to the efficiency of optimization mechanisms and the resilience of application deployments.

Recent zero touch network and service management initiatives in the cloud and telecommunications industry aim to streamline complex infrastructure operations and management tasks in an automated fashion. Traditionally, decision-making systems in this domain rely on heuristics and linear programming techniques, such as mixed integer linear programs (MILPs). However, these approaches have limitations in handling the complexity and dynamic nature of modern cloud infrastructures. This is where data-driven frameworks, such as reinforcement or deep learning, offer significant advantages. More specifically, employing ML-based resource orchestration approaches by the NEPHELE meta-orchestrator can help address the following aspects:

Complexity handling. Cloud networks have become highly complex, with numerous interconnected elements, diverse services, and dynamic traffic patterns. Heuristic and MILP-based approaches often struggle to effectively handle this complexity. This can stem from lack of modelling accuracy, scalability limitations, etc. In contrast, ML-based approaches can learn from vast amounts of network data and adapt to complex scenarios. More importantly, it can learn the underlying connection of raw system metrics (which describe arbitrary network states) to generic optimization objectives (which drive actions towards desirable network states), making it better suited to handle intricate cloud management tasks.

Uncertainty handling. Cloud networks often face uncertain and unpredictable conditions, such as fluctuations in user demand or equipment failures. Heuristic and MILP approaches may struggle to handle such uncertainties effectively. ML, by learning from historical data and exploring different actions, can devise robust decision-making policies by accounting for such uncertain conditions, thus optimizing network performance under varying circumstances.

Self-learning and improvement. Heuristics and MILP models require manual design and frequent updates to account for network changes. Conversely, ML enables intent-based networking, where the user merely specifies some quantitative objective (e.g., in the form of a service level objective), and the ML method shall learn from network data to adapt its policy on par with the changes within the network. This reduces the need for human intervention and manual updates.

Association of intents. A critical limitation of centralized resource orchestration with ML is the association of the resource allocation intents of the orchestrator with the respective intents of the underlying cluster managers. That is, as it is common in hierarchical resource management systems with global and local controllers, the global controller queries the local controllers about their available resources and uses this information to compute a resource allocation decision (which is eventually realized by the latter). Effectively, this limits the inherent capacity of the local controllers to express their own resource allocation intents. In fact, global and local intents are not necessarily conflicting; on the contrary, fostering the acknowledgement of local intents can improve the resource allocation efficiency of the overall system, since each local controller has a more precise view of its actual state.

Decision-making with incomplete information. The common assumption that a single learning agent, positioned at the orchestration layer (which is the standard practice in centralized resource orchestration using ML), has a precise view over the entire substrate, is somehow unrealistic. In fact, the higher we move along the orchestration hierarchy, the more coarse-grained the information we have at our disposal (because of data aggregation), which implies higher uncertainty. Conversely, if a centralized ML approach admits partial observability, it follows that the decision-making process relies on incomplete information.

Dependence of action space on topology nodes. In most works that propose centralized ML schemes for service mapping, the action space of the learning agent coincides with the set of available clusters. This is somewhat natural, since the centralized agent must infer which is the best cluster to host a particular VNF or cloud-native app. However, the dependence of the agent's architecture on the physical nodes of the substrate makes the ML scheme particularly hard to scale. That is, the larger the action set, the longer the training time. Our argument is further strengthened by studies which employ techniques for shrinking the action space, e.g., grouping a set of clusters, or placing the entire application graph within a single cluster.

Figure 8 shows the Federated Resource Manager in the context of the NEPHELE multi-cluster and multi-domain infrastructure with potential technologies and platforms that could be considered further down the line, in addition to the Kubernetes-centric approach expressed in this document.

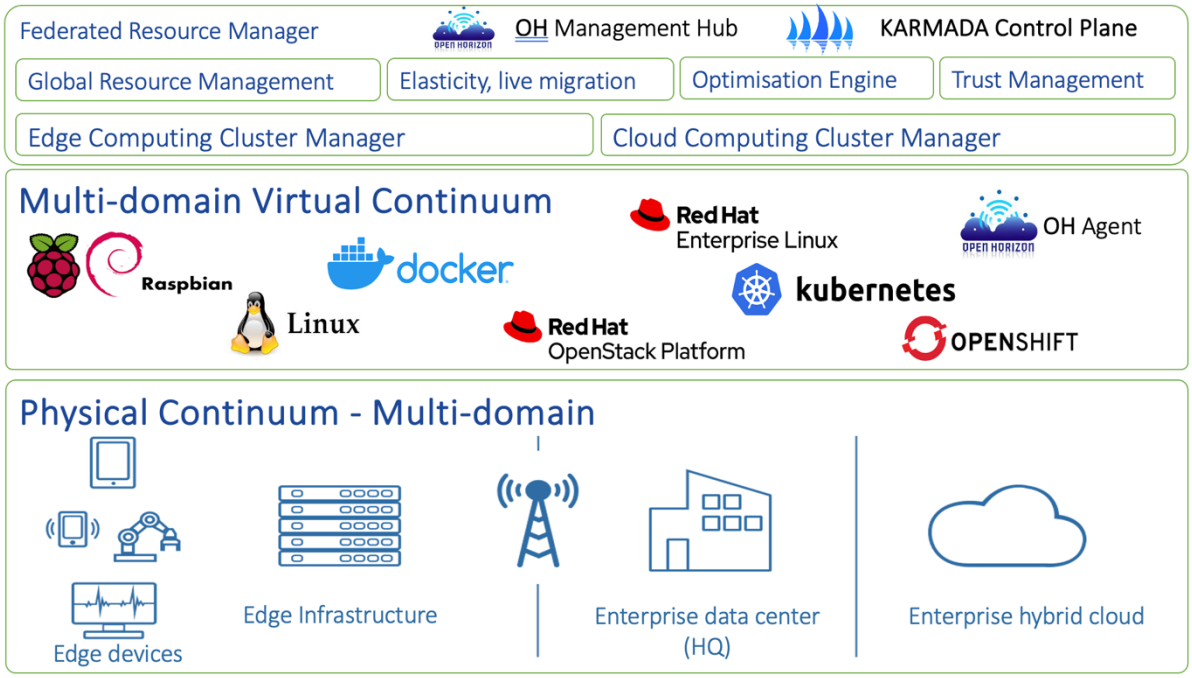

Figure 8: The Federated Resource Manager in the context of the multi-domain infrastructure of NEPHELE

3.3 Network Management and Interfaces

The Compute Continuum Network Manager (CCNM) is responsible for managing network resources and functionalities across the edge-cloud compute continuum. This role encompasses several critical points that are addressed by this component: clearly, there is the identification of the computing and network resources and the setup of the related slice according to the QoS requirements of the

considered application, followed by the management of its lifecycle, but the fact that these resources are distributed over the continuum adds further complexity to an already demanding role.

Resources across the continuum are geographically distributed among data centres that can be in the edge, hence within the premises of a telecom operator, or under the ownership of a cloud provider. This means that the interoperation must be guaranteed in a way that preserves the ownerships and isolates the resources of the different stakeholders, for example by providing individual tenant spaces and abstractions able to conceal their actual, physical locations.

To this end, the objective of the CCNM is providing a level of abstraction for the flexible and high-level management of the complete lifecycle (i.e., Day 0/1/2/N) orchestration of network services (e.g., a 5G system, an overlay system for network cybersecurity or a simple application service mesh). The reason why the CCNM layer was introduced is twofold: on the one hand, it was designed to support different NFVOs (since it implements standard ETSI NFV-SOL 004 [21], 005 [22] and 006 [23] APIs); on the other hand, it allows to cope with some shortcomings of ETSI NFV specifications, as explained herein. The Network Service Descriptors (NSDs) implemented in ETSI NFV are static, as the number of VNFs and the related internal/external connections are pre-determined in accordance with the VNF Descriptors (VNFD). Aside from the initial decision on the Virtual Infrastructure Manager (VIM) to allocate the service on, each NSD dictates a fixed list of PNFs and SBA functions, including their connections, which makes it impossible, for example, to perform network service internal topology change on any of the functions without destroying and re-deploying the whole service, let alone to modify the in-life operations on-the-fly upon changes on the related slice.

The aforementioned network services, as shown in Figure 9, can be composed of any number of heterogeneous Network Functions (xNFs – i.e., Physical NF (PNF), Virtual NF (VNF) and cloud-native Kubernetes NF (KNF)), which are usually to be realized over highly distributed infrastructures. More specifically, every network ecosystem can be thought of as a graph, where the vertexes are composed by the sets of xNFs instances and the sets of interconnection networks, while edges represent the interconnectivity between xNFs and networks.

As defined in the ETSI NFV standard, xNFs are managed by the NFV Orchestrator (NFVO) (usually ETSI Open-Source MANO) through end-to-end Network Service Instances (NSIs). Every Network Service can include one or more xNF instances, and it is meant to be deployed over a single geographical facility, which may correspond to a computing facility and/or a physical device (e.g., a gNodeB, an O-RAN Radio Unit, a P4 switch, etc.).

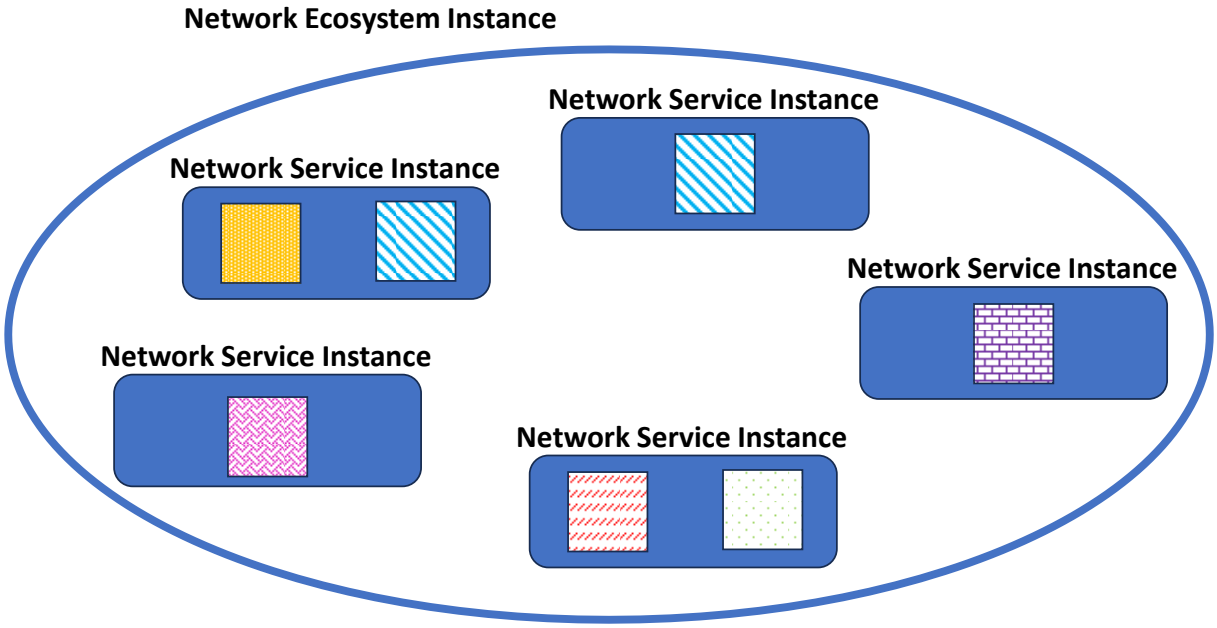

Figure 9: A Network ecosystem instance composed by five network services made up of a variable number of xNFs

As shown in Figure 10, the graph of a network ecosystem is meant to be annotated with "anchors" that represent the placing/binding of the ecosystem endpoints over the physical infrastructure topology. An anchor can be associated to a network in the set, or to a PNF to be instantiated over a physical device.

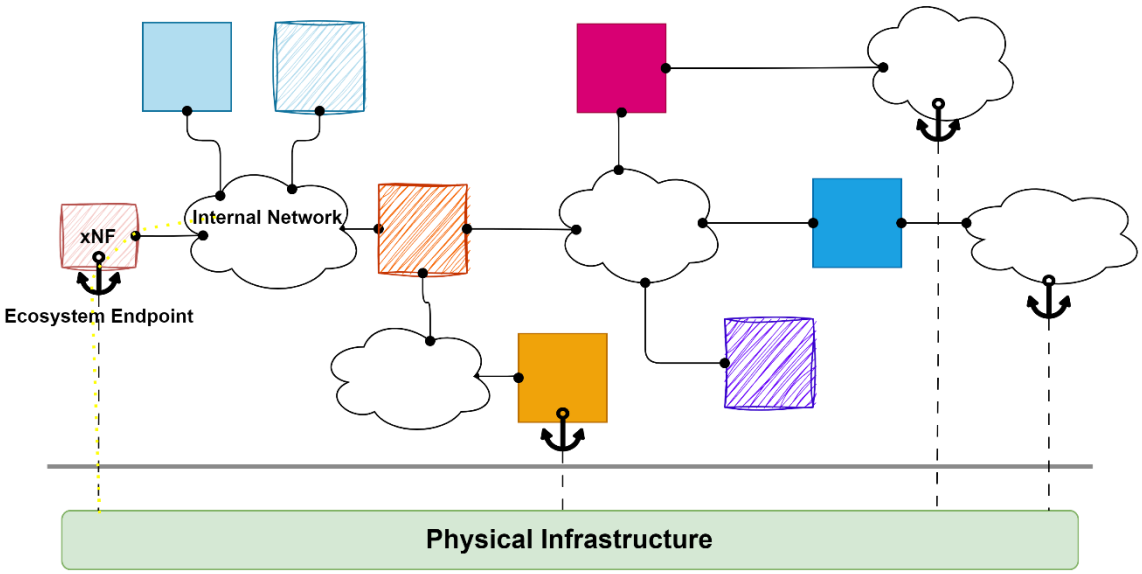

Figure 10: The graph of a network ecosystem with anchor points highlighting the link between the xNFs and the physical infrastructure.

To support the Edge Computing technology enabled by 5G, it is fundamental to map the physical resources on the continuum. The CCNM implements this by tagging the resources (e.g., gNodeBs and

servers) with an Area ID (see Figure 11). In this example, there is one gNodeB and three datacenters (which in the reality are composed on one server each). Each of these physical resources are marked with an Area ID that is selected according to their geographical location, in fact they are assigned on the Tracking Area Code (TAC) basis. Regarding the servers, the correspondence is maintained when deploying virtual resources by exploiting the label feature of Kubernetes is exploited: Kubernetes worker nodes are tagged with an Area ID label and then, the pods (i.e., wrappers of containers) can be deployed on worker nodes having a specific Area ID.

Moreover, Figure 11 highlights the need for having different levels of virtualization which can be exploited by the NSs. Platform as a Service (PaaS) allows the deployment of KNFs, Infrastructure as a Service (IaaS) of VNFs and finally Metal as a Service (MaaS) allows to bypass virtualization and to deploy services directly on the hardware (e.g., a Kubernetes bare-metal cluster). PaaS can be either deployed on bare metal or on top of an IaaS.

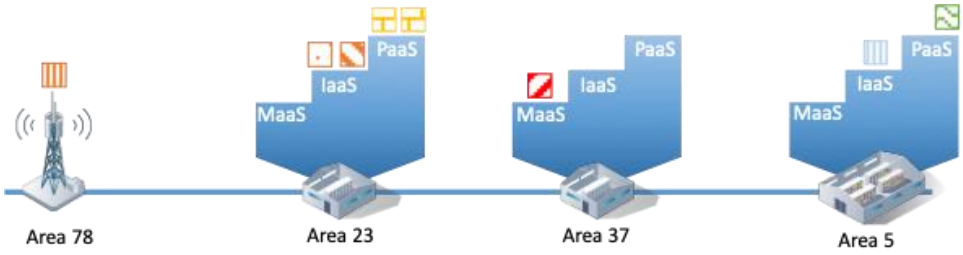

Figure 11: Mapping between the physical resources (e.g., gNobeB and servers/datacenters) and the geographical areas

3.3.1 Description

The CCNM has been built over a modular and flexible architecture that can be easily extended to support new xNF and ecosystems. At the foundations of this architecture, the metamodel in Figure 12, Network ecosystem metamodel, has been specifically designed to augment extensibility and flexibility and to drive clear interaction patterns among the different internal modules during Lifecycle Management (LCM) operations.

As shown in Figure 12, every ecosystem instance is built based on a Blueprint, which in turn falls into a Category. The Blueprint Category corresponds to the high-level network ecosystem function type, like a 5G system or a network security toolchain, etc. The Blueprint is meant to support ad-hoc operations for specific implementations falling into that Category. For instance, the CCNM currently provides 2 different 5G system implementations, based on different open-source projects, namely Free5GC, Open5GS.

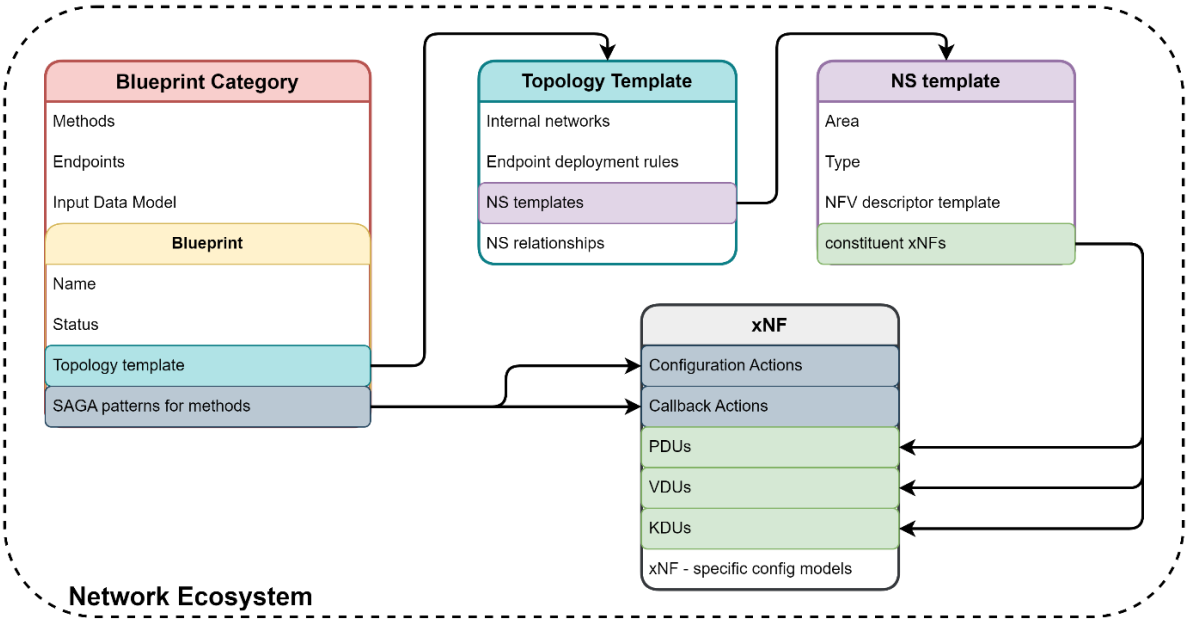

Figure 12: Network ecosystem metamodel

The Blueprint Category allows to have a homogeneous north-bound interface against the different implementations available for an ecosystem, since it defines a single input meta-data model (including the possible ecosystem endpoints) and the associated ecosystem-level LCM methods. For example, the 5G System Blueprint Category exposes operations to add/remove/reconfigure RAN over specific geographical areas, to create/modify/destroy network slices, etc., and it fixes the endpoints to be physical devices like base stations or ORAN radio units, and networks to be used as 5G Data Network Name (DNN).

A Blueprint provides the implementation-specific means to support the Category methods and to translate the metadata model into sets of NSIs and xNFs, interconnected and running with coherent (but implementation-specific) configurations. To this end, Blueprints define the template of the ecosystem internal topology, as well as the specification of the internal procedures to be executed for every supported Category method. These internal procedures are realized as SAGA pattern interactions among specific CCNM modules.

The topology template defines the graph pattern including the templates of internal network and of NSIs that can be applied, and their possible relationship bindings. On its turn, an NSI template specifies the list of possible constituent xNFs, and the logic to build NFV SOL-006 descriptors to be onboarded and used by the NFV Orchestrator.

Finally, the metadata model of xNFs plays a key role in the CCNM architecture. It defines not only the specific physical/virtual/Kubernetes deployment units to be used to materialize NS templates, but it also defines the implementation-specific methods and callbacks that can be executed on an xNF, and the models of its configuration. In other words, xNF templates represent a sort of glue between NFV-driven LCM operations to instantiate or remove artefacts from the ecosystem (e.g., creating a RAN NS in a new area), and management operations affecting the configuration of running xNFs (e.g., add a new 5G subscriber, add a new policy, etc.). Each of these operations might include a variable number of different actions to add NSI instances (Day 0 and 1 actions), to change the configuration settings of xNFs and to retrieve information from the deployed xNFs (Day 2 actions), as well as to remove one or more deployed NSIs.

The CCNM internal architecture, shown in Figure 13, encompasses the meta-models introduced so far. A first module, named CCNM North Bound Interface aims to expose CRUD REST APIs for ecosystem LCM through the methods defined in the Blueprint Category meta-models that are available and onboarded onto the CCNM. Among these methods, the ecosystem creation and deletion are mandatory (and correspond to HTTP POST and DELETE messages respectively).

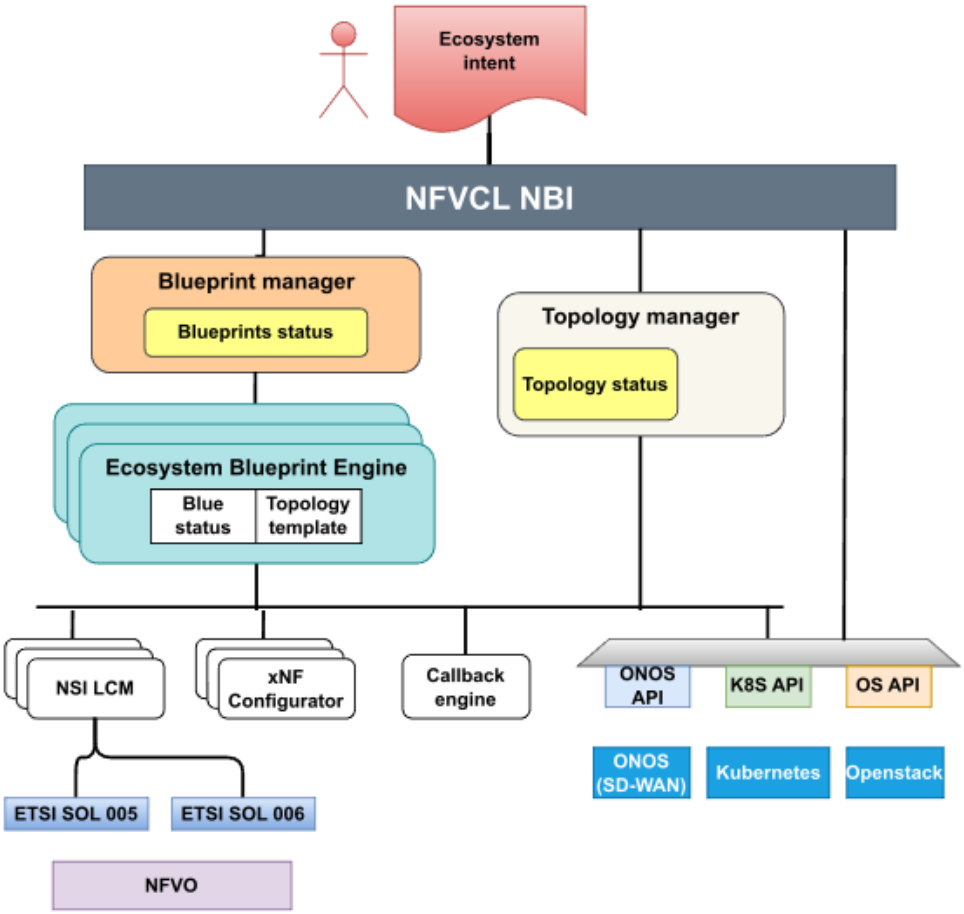

Figure 13: CCNM internal architecture

The Topology manager offers resources needed by the ecosystems. Topology statuses include the VIM list with their detailed networks status, available Kubernetes clusters that can be used to deploy KDUs, a list of available PNFs and metric servers that can be used to store KPIs. A powerful feature offered by the Topology Manager is to terraform resources on the VIM, in other words, when a blueprint needs something that does not exist it can create it on demand.

The Blueprint Manager takes care of all the requests towards blueprints, from creation (Day 0) to deletion (Day N) operations. An Ecosystem Blueprint Engine instance (Operator) is instantiated for every active ecosystem at its initialization, in this way, it is possible to work on multiple requests, for different blueprint instances, at the same time.

The Operator acts as a sort of worker, dedicated to handle and to serialize incoming LCM initialization/change requests on the ecosystem. This component maintains the information related to both the Blueprint meta-model (status) and the Topology template. It is in charge of binding any supported blueprint category method into a coordinated set of multiple implementation-specific operation requests against resources in the topology, the LCM of NFV Network Service Instances (NSI),

or configuration changes within one or multiple xNFs. While these operations are executed into further dedicated components, namely, the Topology Manager, the NSI LCM engines, and the xNF configurators, the Ecosystem Blueprint Operator acts as a central coordinator for the distributed transaction through a *publish-subscribe* communication system (in the current version, Redis [24] is applied in this regard). This architecture is known in software engineering as *orchestrator-based saga pattern* [25].

As shown in Figure 14, any method is defined as a saga pattern composed of multiple rounds, and each round is an ordered sequence of topology, Day 0/1 and Day N, as well as Day 2 optional actions, which are handled by the Topology Manager, the NSI LCM, and the xNF Configurator modules, respectively.

Such actions are execution commands aimed to add/modify/remove entities or to trigger commands in the ecosystem xNFs. Actions might be optionally followed by callbacks. Callbacks are meant to collect those pieces of information and parameters that have been produced by the action execution. For example, an action on a xNF acting as a "master" server might produce a key to be used by other "slave" xNFs in the ecosystem. Day 2 callbacks are meant to retrieve this key string and move it to the Blueprint Engine, so that it can be used on further actions involving "slave" xNFs. Finally, Day 0/1 callbacks are meant to retrieve dynamic deployment information of xNFs (e.g., IP addresses, etc.).

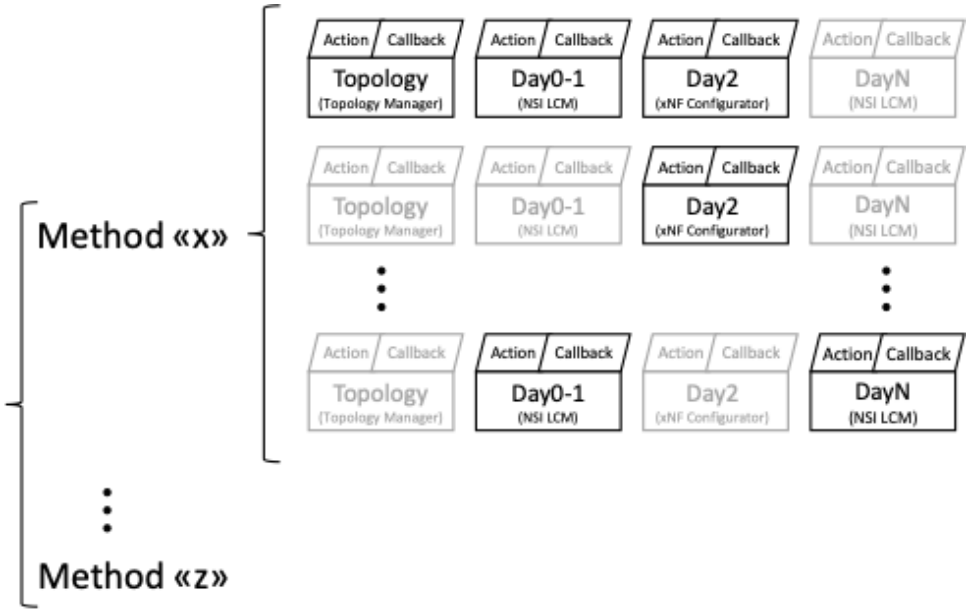

Figure 14: Blueprint methods as SAGA patterns

Finally, the CCNM can interact directly with external entities, like Kubernetes or Openstack, to enable features not supported by the VIM. For example, when working with OSM, there is no way to create/update images for VDUs, they must exist on the VIM. Kubernetes APIs are strongly used for managing and configuring clusters deployed through the CCNM, some useful operations performed in this way are plugins installation and user creation.

All dynamic vital information for the operation of the CCNM are saved into a MongoDB database. The two main collections to be saved are the status for the topology and, for every instantiated blueprint, the status, and the topology template. When the CCNM starts it firstly loads the topology information

and then, when an LCM operation comes for a blueprint, it loads the status for the specific blueprint instance.

In general, every blueprint ecosystem needs different input data, this strongly affect the generated status of the blueprint to be saved and maintained. To this end, every blueprint type, also belonging to the same category can have different status data structure. The saved data of the blueprint can also include past actions, like executed LCM primitives.

Regarding the collection of data from the xNFs, the CCNM offers, through the Topology manager, a list of Prometheus servers ready to collect data. These instances can be used from blueprints to setup metric exporters on xNFs. The collected data depends only on the configuration of the exporter, injected on demand in a VNF.

3.3.2 Requirements

The current requirements are:

- An OpenStack instance.
- An Ubuntu (22.04 LTS) instance where the CCNM will be installed and run.
- Having OSM 14 running on a reachable machine.
- Python 3.11.
- Recommended: 4 CPUs, 16 GB RAM, 80GB disk and a single interface with Internet access. Minimum: 2 CPUs, 8 GB RAM, 50GB disk and a single interface with Internet access.

3.3.3 Design ideas

The research and development efforts that have brought to the realization of the CCNM stem from an initial prototype developed for the MATILDA Project [26]. Initially, the goal was to provide additional features that were not delivered by the NFVOs available at that time, while following extensions have provided, among others, the management of the topologies, of Day 2 operations, and of xNFs. Within NEPHELE, the support for edge-cloud continuum deployment will be refined by integrating it within the Synergetic Meta-Orchestrator and providing the interaction with the other components.

3.4 Security and Trust Management

Data spaces play a crucial role in enabling secure, compatible, and reliable data-sharing among businesses and societies. This is a vital step towards realizing the Data Economy of tomorrow. In September 2021, the Data Spaces Business Alliance (DSBA) [27] was established through a collaboration between the Big Data Value Association (BDVA), FIWARE Foundation, Gaia-X, and the International Data Spaces Association (IDSA). Their collective aim is to promote the adoption of data spaces not only in Europe but also on a global scale.

As part of their initiative, DSBA members committed to developing a unified technological framework. This framework is based on the integration of existing architectures and models. Capitalizing on the progress made by each organization in terms of specifications and implementations. The objective is to achieve seamless interoperability and portability of solutions across different data spaces by aligning technology components and other essential elements.

To achieve the intended technical convergence, an implementation-driven plan is proposed around evolution through subsequent versions of a Minimum Viable Framework (MVF) enabling creation of data spaces. A first version of the MVF was the result of a first workstream targeted to provide a minimum set of building blocks required to cover three major technology pillars for creation of data spaces:

- **Data interoperability**: NGSI_LD API and smart data models for actual data exchange, extending the interoperability mechanisms of the IDS-RAM with a special focus on the IDS-Infomodel and the Vocabulary Hub.
- **Data Sovereignty and Trust**: A decentralized Identity and Access Management (IAM) framework based on:
    - A set of Verifiable Credential issuing protocols (OpenID Connect for Verifiable Credential Issuance, Self-Issued OpenID Provider v2 (SIOPv2), via DIDComm channel, etc).
    - A set of Verifiable Presentation protocols (ex: OpenID Connect for Verifiable Presentations (OIDC4VP), Presentation Exchange, etc).
    - An ABAC (Attribute Based Access Control) framework based on Verifiable Credentials, comprising components implementing PEP, PDP, PAP/PMP, and PIP functions.
- **Data value creation**: Centralized Service Catalogue and Marketplace functions based on TM Forum standards.

To establish **authentication**, we suggest employing the same mechanism as used in EBSI and the EUDI Wallet for online processes. This involves the utilization of OIDC4VP and SIOPv2. These methods rely on the well-established, reliable, and secure protocols of OpenID Connect. This allows for the following:

- Facilitation of the transmission of Verifiable Credentials/Presentations within the framework of OpenID Connect, enabling Relying Parties to employ familiar methods for issuing and receiving Verifiable Credentials.
- Empowerment of all participants (via SIOPv2) to transmit identity information and Verifiable Credentials to other participants. This is achieved without the need for large, centralized Identity Providers, which is unfortunately common in standard OpenID Connect implementations.

This approach allows us to establish a decentralized, resilient, reliable, and effective Identity and Access Management (IAM) system, eliminating the need for centralized Identity Providers (IdPs). By leveraging widely adopted standards such as OIDC and W3C Verifiable Credentials, we significantly lower the entry barriers for participants implementing IAM. The utilization of OIDC for Verifiable Credentials transport permits the integration of verified data within the credential, enabling sophisticated and adaptable Authorization schemes. Those who implement this Decentralized Identity and Access Management Framework can leverage credential data for advanced RBAC/ABAC access control and policy enforcement.

Decentralized Identifiers

Decentralized Identifiers (DIDs) [28] are unique identifiers that enable verifiable, self-sovereign digital identity. They are a crucial component in a decentralized identity system, allowing individuals, entities, and things to have a persistent, globally unique identifier independent of any centralized authority. DIDs are foundational in enabling secure and private interactions in digital environments.

As shown in Figure 15, DIDs are divided in 3 parts:

- The **URI scheme** identifier: this is the “did:” at the beginning of the identifier that serves as a Uniform Resource Identifier (URI) scheme that indicates it is a Decentralized Identifier.

- The identifier for the **DID method**: this part specifies the method used to create, resolve, and manage the DID. It indicates the underlying decentralized network or system. Each method may have its own rules and processes for creating and managing DIDs.
- The DID method-specific **identifier**: This is a unique identifier assigned to a particular entity or individual within the chosen DID method. The format and length of the specific identifier may vary depending on the method. It uniquely identifies the subject of the DID within the chosen decentralized network or system.

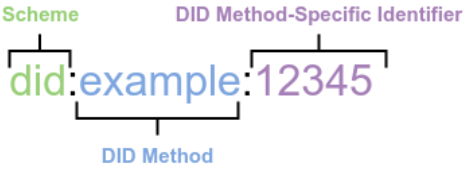

Figure 15: Decentralized Identifier example.

As depicted in Figure 16, DIDs resolve to DID documents, which are JSON-LD representations associated with DIDs. They contain key information about the subject of the DID, including public keys, authentication methods, service endpoints, and additional metadata. DID Documents serve as a crucial component for verifying and interacting with decentralized identities within a given DID method. They enable secure and standardized management of identity-related information.

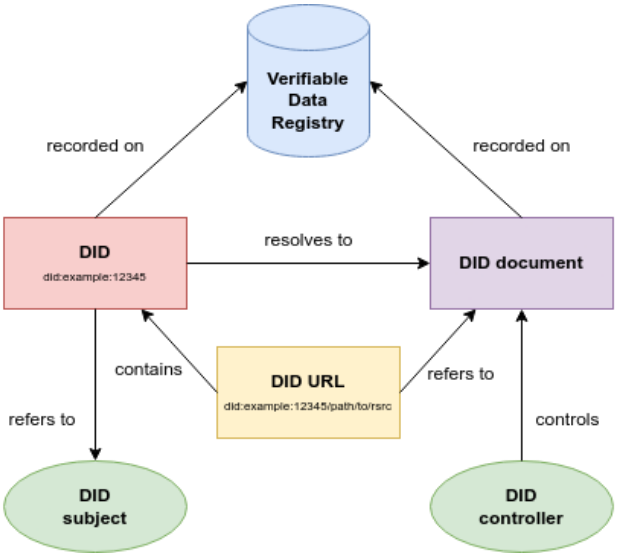

Figure 16: Overview of Decentralized Identifiers architecture

Verifiable Credentials and Verifiable Presentations

Verifiable Credentials (VCs) [29] are digital declarations that include information in form of claims about a subject (individuals, organizations, or devices). VCs are unique the way that they can be independently verified thanks to cryptography techniques, making VCs more tamper-evident and more trustworthy than their physician counterparts. This cryptographic assurance ensures the integrity and authenticity of the information, allowing third parties to independently verify the credential without needing to contact the Issuer. Furthermore, the underlying protocols, such as the Decentralized Identifiers and the Verifiable Credential Data Model, provide a standardized way to

structure and exchange these digital statements, making VCs play a crucial role in advancing the technical foundations of secure and user-controlled digital identity ecosystems. Also, VCs include a feature called 'selective disclosure' that allows Holders to share only relevant aspects of their credentials, thus minimising data exposure.

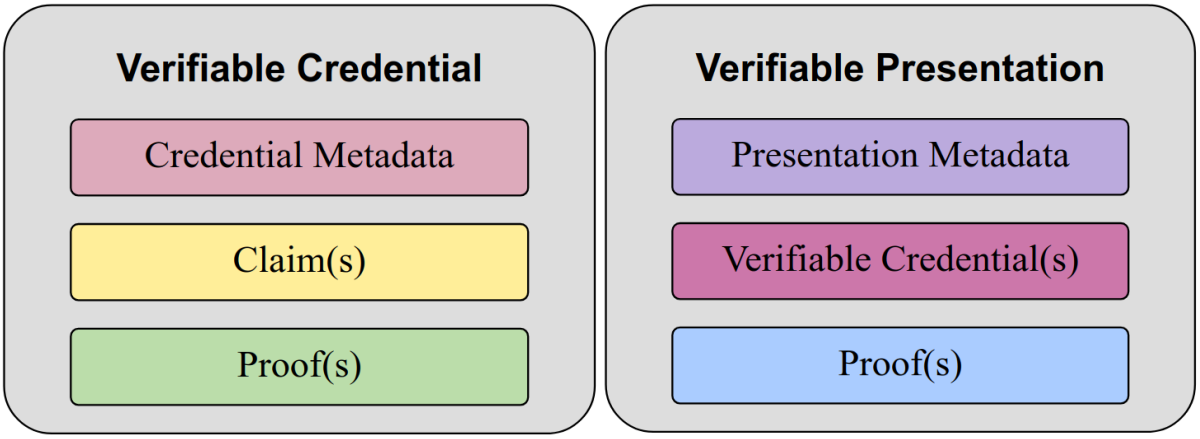

Figure 17: Components of Verifiable Credentials and Verifiable Presentations

Verifiable Presentations (VPs) are tamper-evident presentations encoded in such a way that authorship of the data can be trusted after a process of cryptographic verification. VPs includes data derived from one or more VCs. These presentations enable individuals to selectively disclose specific verifiable information from their credentials to different parties or relying parties. VPs contribute to the user-centric paradigm by allowing individuals to assert their digital credentials in a privacy-preserving manner. This capability is particularly valuable in contexts such as identity verification, where individuals can provide proof of specific attributes without divulging unnecessary personal details (selective disclosure). The development and adoption of Verifiable Presentations further strengthen the foundation of decentralized identity systems, emphasizing security, privacy, and interoperability in the evolving landscape of digital interactions. Both VCs and VPs can be transmitted rapidly. Figure 17 shows the components of Verifiable Credentials and Verifiable Presentations while Figure 18 shows Verifiable Credentials flow.

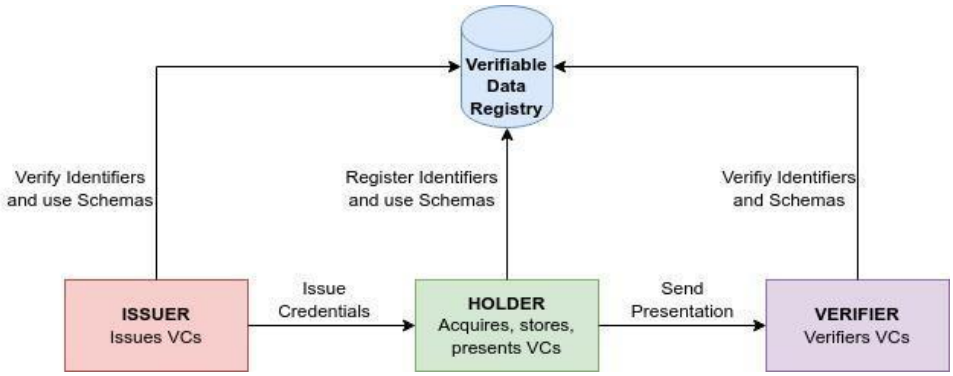

Figure 18: Verifiable Credentials flow

OpenID Connect and Self-Issued OpenID Provider v2

OpenID Connect (OIDC) [30] is a simple, open standard identity protocol based on the OAuth 2.0 protocol that allows client applications to trust on authentication made by an OpenID Connect Provider to verify a user's identity. OIDC uses OAuth 2.0 for authorization and then creates the identities that exclusively identify users. Client applications can also obtain basic profile information about a user in an interoperable REST-like manner from OIDC Providers. As an addition to OAuth 2.0 authorization, OIDC provides authentication. Clients use the OpenID scope value in their authorization requests to request the use of this extension. An ID Token, which is a JSON Web Token (JWT) that contains information about the authentication process is returned. OpenID Connect for Verifiable Presentations (OIDC4VP) [31] is a protocol that extends the implementation of OIDC and enables the presentation of Verifiable Credentials as Verifiable Presentations. The main addition for OIDC4VP is the VP Token, which is used as a container to let End-Users use the Wallet to present Verifiable Presentations to Verifiers. A VP Token contains one or more Verifiable Presentations. Additionally, OIDC can also be extended with the OpenID Connect for Credential Issuance (OIDC4CI) [32] protocol, which consists of the request of Verifiable Credentials by an End-User (Holder) to a Verifiable Credential Issuer. The access to the Issuer's API is authorized using OAuth 2.0, making the issuance process secure, simple, and flexible thanks to OAuth 2.0 characteristics, and making it possible for OAuth 2.0 deployments and OpenID Connect OPs to be extended to become Credential Issuers.

Self-Issued OpenID Provider (SIOPv2) [33] is a specification that extends OIDC by making an OpenID Provider controlled by the End-User. This allows the En-Users to authenticate themselves with Self-Issued ID Tokens, presenting self-attested claims directly to the Relying Parties (RPs). These claims can also be issued by third parties (Issuer) trusted by the RPs and when used with the OIDC4VP protocol. This allows the interaction between the End-Users (Holders) with RPs (Verifiers) without the need for the RPs to interact with claim issuers (Issuers).

Security Architecture

The architecture is based on the utilization of **Holder, Verifier and Issuer** components. These components are deployed within Kubernetes on every element (Applications, cVOs, and VOs). Additionally, there is the option of deploying them on IoT Devices, if the limitations of the devices allow it. The Application will only deploy the Holder component because the interaction flow is always VO-to-App, i.e., the App will never receive requests from VOs, so the App will never have to verify credentials to produce an Access Token. VOs and cVOs will deploy all 3 components because they will have to both Verify and Authenticate depending on the type of interaction that is taking place (Direct VO-to-VO, VO-to-IoT-Device(s) or VO-to-App). The IoT Devices, if the conditions are met, will deploy just the Holder component for the same reason as the Application does, but in this case the interaction is VO-to-IoT-Device(s).

The **Holder** component is the individual/organization/thing who receive digital credentials that contain data about themselves from various sources (Issuers). By aggregating and storing such credentials in digital wallets, Holders can build holistic digital identities that are under their control and can easily be shared with third parties (Verifiers).

The **Verifiers** are the parties who rely on data to provide products and services can reliably verify and process data that has been provided by others (Holders). Verifiers, also called "Relying Parties", are usually organizations or individuals in their professional capacity.

The **Issuers** are parties who "issue" identity-related data to people or organizations (Holders) in the form of digital credentials. They are the original data sources of an SSI ecosystem. For example, a government issues digital passports to citizens or a university issues digital diplomas to graduates.

3.5 Monitoring, Data Fusion and AI-assisted Orchestration

As cloud applications grow larger, and the underlying infrastructures converge more to the complex nature of the compute continuum, monitoring and data correlation become crucial for automating the analysis and confrontation of problems such as the resolution of anomalies in application deployment and the optimization of the specified objectives. For this purpose, modern technologies emerge for collecting, storing, and analysing different kinds of data signals [49], while many state-of-the-art techniques are utilized for introducing system autonomy based on data analysis methodologies [42]. The NEPHELE platform will need to be aware of the latest research in the field and adopt any ongoing work that can enable the exploitation of the real-time collected datasets to identify patterns and optimal strategies towards efficient orchestration in the computing continuum.

3.5.1 Observability

One of the biggest obstacles in effectively managing distributed applications and services is the need to transition from traditional monitoring tools to modern cloud-native observability tools. Traditional tools are not tailored to distributed microservice environments and interactions among containers, as they are typically geared towards monolithic applications. Additionally, while tracing techniques have long been employed by developers to track an application's behaviour-related metrics, they are not well-suited for microservices-based applications, and do not account for the horizontal scaling abilities of these applications. The concept of cloud-native observability has emerged as a means of providing insight into the health and status of applications within cloud-native elements like microservices, containers, and orchestration tools. However, modern approaches to observability need to consider the necessity of integrating with existing or emerging **monitoring** tools. The existing monitoring mechanisms in open-source orchestration engines are not designed to support advanced monitoring that caters to the unique characteristics of distributed applications and the observation of metrics related to interactions among application components. Additionally, multiple third-party tools exist to support **distributed tracing** and **logging** mechanisms, albeit with limited integration with the monitoring tools. Over the collected data from modern observability tools, various analysis pipelines can be executed, including the pipelines that are based on machine learning techniques.

Observability in the compute continuum becomes even more complex considering the multi-layer and multi-cluster deployment of applications. It becomes critical to not only monitor execution performance, but also network performance since the continuum in collaboration with modern network technologies (5G/6G) aim to support more and more latency-intensive applications. So, obtaining **temporal and performance information** that can help identify issues in data **transfers** becomes part of the distributed tracing process at an application as well as a network infrastructure level. Furthermore, software **architectures** in the continuum take new forms, especially with the concurrent addition of multiple AI processes, moving away from pure server-client approaches and leveraging other more efficient or well-suited approaches. Thus, distributed observability tools need to adapt and support modern applications running at the continuum, embracing new requirements that come with it.

Some key challenges of an observability system [48] include context connectivity, easier and faster exploration, the identification of a single source of truth, capturing arbitrarily wide events and decoupling data sources from sinks, heterogeneous signal correlation, topology, and application profiling (historical and real-time), and event response.

The general methodology towards utilizing signals to manage cloud applications is sepicted in Figure 19. It can be modelled using three entities: Signals, Events, Actions. Observability signals are collected and analysed to identify hidden patterns that correlate to specific events such as system changes

(deployment modifications), issues (healthy states, system/app failures) or incidents (KPI violations). Based on these events and the underlying data, algorithms evaluate the status and select the best actions to apply based on the specific problem they try to address.

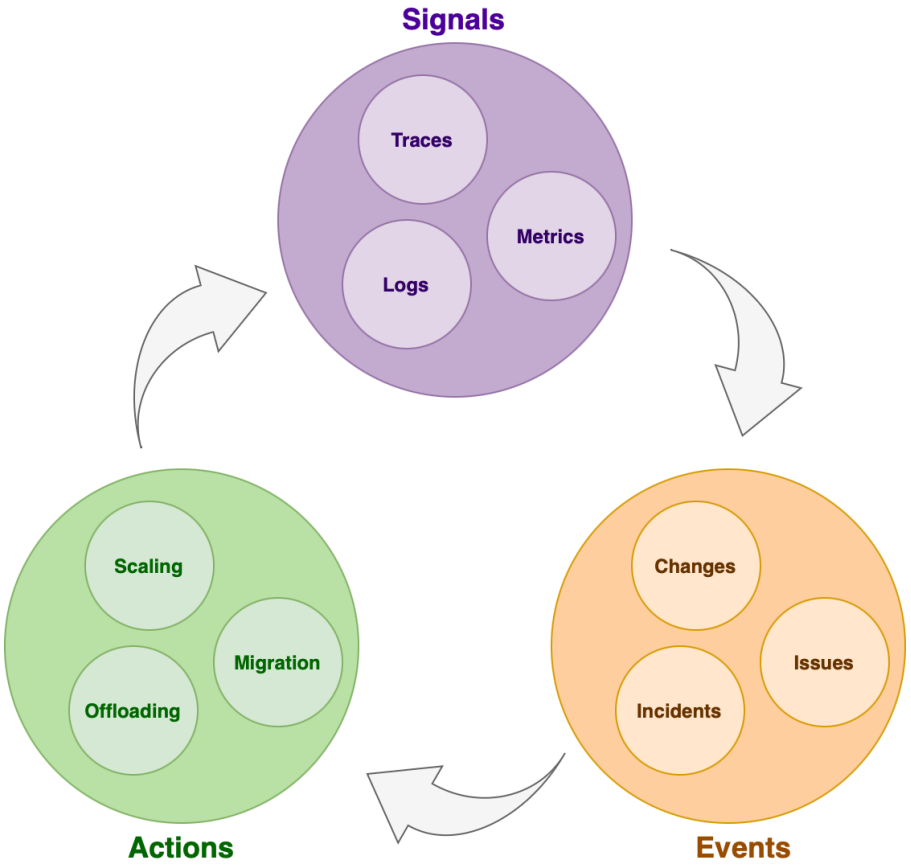

Figure 19: Methodology utilizing signals to manage Cloud applications

3.5.2 AI-based Techniques for Autonomy in the Computing Continuum

3.5.2.1 Reinforcement Learning

Reinforcement Learning is a popular paradigm which has been proved very effective in building autonomous agents for various tasks, such as request dispatching with Multi-Agent RL (MARL) [34] or fog cluster scaling [35], due to its ability to relate state-action pairs with their corresponding rewards. Such agents often demonstrate co-dependent requirements due to their complex relations, so agents that manage microservices in microservice networks can interact with each other to satisfy requirements that arise from their exchanges, while agents managing computing nodes in a network in the continuum can distribute incoming workloads.

MARL

In [39] the following taxonomy is proposed:

- **Independent Learners (IQL):** A simple approach to extend single-agent RL algorithms to multi-agent algorithms is to consider each agent as an independent learner. In this setting, the other agents' actions would be treated as part of the environment.

- **Fully Observable Critic (MADDPG, COMA):** In this setting, the critic model learns the true state-value and when paired with the actor can be used toward finding the optimal policy. When the reward is shared among all agents, only one critic model is required.
- **Value Function Factorization (QMIX):** a decomposition function is learned from the global reward.
- **Consensus (DQN, DAC):** The key idea is to put the learning agents on a sparsely connected network, where each agent can communicate with a small subset of agents. Then the agents seek an optimal solution under the constraint that this solution is in consensus with its neighbours.
- **Learn to Communicate (RIAL, DIAL [36]):** Learn to Communicate allows the agents to learn what to send, when to send, and send that to which agents. In particular, besides the action to the environment, the agents learn another action called communication action Game-theoretic analysis of practical MARL systems for networked system control (**NeurComm[37]**) [38].

Multi Objective Reinforcement Learning

RL methods usually aim to optimize a single objective. However, any orchestration solution in the computing continuum must inherently consider multiple objectives regarding cost, quality and resources. Multi-Objective Reinforcement Learning (MORL) is a field that aims to develop methods for RL problems where multiple different objectives must be optimized. This means finding an appropriate trade-off between competing objectives. A central solution concept in MORL for defining the set of best trade-off solutions is Pareto efficiency. [42]

Game Theory

Combining game theory concepts with RL opens many interesting possibilities. The benefit of evolutionary game theory over traditional game theory is the fact that it does not require agents to be hyper-rational players, and an equilibrium can change dynamically. Evolutionary game theory has even been proposed as the preferable framework for studying multiagent learning formally [45]. Furthermore, the compute continuum may entail many agents, when most of the approaches for reinforcement learning in a MAS are designed for or validated in settings with a finite number of agents. Mean-Field Games (MFGs) are designed for situations with possibly thousands of agents and so, Reinforcement Learning solutions can leverage on such methods to tackle scaling when agent numbers are high [46].

3.5.2.2 Federated Learning

FL aims to train a global ML model in a distributed manner. The global model is most typically an ANN model, but it can also be some other parameterized model. The original version of FL, often called vanilla FL, trains a global model in a centralized manner on decentralized data. Each agent participating in the training has their own training data that they use to train a local model. Then, the local parameter updates are sent periodically to a central server that aggregates the updates and sends the resulting global model back to agents [40]. An alternative approach termed as Federated Distillation (FD) exchanges model outputs instead of model parameters, which reduces communication costs. [41]

Fog learning enhances federated learning along three major dimensions: network, heterogeneity, and proximity. It considers a multi-layer hybrid learning framework consisting of heterogeneous devices with various proximities. It accounts for the topology structures of the local networks among the heterogeneous nodes at each network layer, orchestrating them for collaborative/cooperative learning through device-to-device communications. This migrates from star network topologies used for parameter transfers in federated learning to more distributed topologies at scale shown in Figure 20. [44]

FL relies on a central aggregator, which creates a single point of failure, compromises scalability and maintains higher communication costs due to the star topology of the network. An autonomous computing continuum requires less reliance on centralized entities and more independent and decentralized ways to coordinate model training. In decentralized learning, agents are organized as a virtual and possibly dynamic network graph with a small maximum node degree, and the nodes can only share their local model parameters in their neighbourhood. These local models should gradually converge to a global model, i.e., agents should reach consensus. [34]

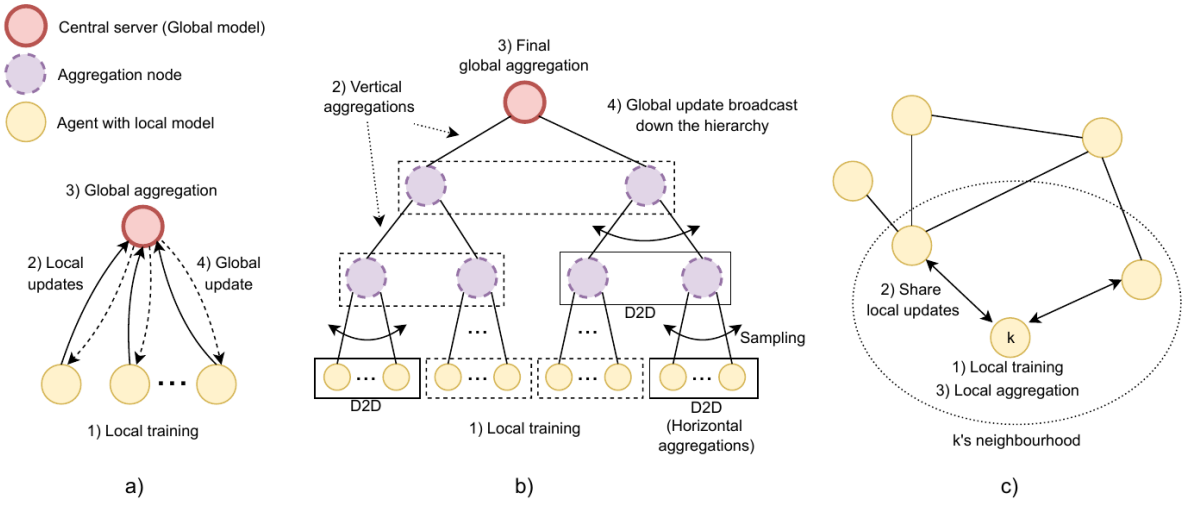

Figure 20: High-level workflows and topologies [42]

3.5.2.3 Representation Learning

The heterogeneous data collected by observability tools can potentially facilitate multiple orchestration mechanisms and solve related problems. For such mechanisms to achieve this though, they need to process **large amounts of data** and discover correlations between the recorded signals and possible deployment failures. This process can become very difficult due to the massive amounts of computing devices that participate in the continuum as well as **complex ways that application components connect to each other** and thus, it is necessary to advance to new designs and technologies. To this end, inspecting **Representation Learning** (ReL) techniques [47] for management in the Compute Continuum can help r**educe the complexity** of the data and **export useful information** out of big data lakes. Specifically, this set of approaches aims to extract high-level, information-rich representation of data that can be used for pattern recognition, behaviour prediction, or classification. Some examples include Matrix factorization, Random walk learning, Graph representation learning, Bayesian network structure learning and Contrastive representation learning.

3.5.3 AI-assisted Methodology

To efficiently exploit the knowledge obtained from relevant research on the specified scenarios, we design a methodology on how to combine mature AI-based techniques and identify the main components of such a framework and their interactions. As mentioned in the previous sections, some of the main methods identified can be summarized in the following list:

- Anomaly detection
  - Supervised
  - Unsupervised

- Workload prediction
- Representation learning
- Reinforcement learning
- Federated learning

The first three components mainly exploit input from monitoring/observability interfaces to provide more analytical and enriched representations of the deployment (infrastructure and application) state. Anomaly detection identifies events and changes, workload prediction offers a probable view of future metrics and representation learning offers more complex representations of the collected data which can facilitate the task at hand. Reinforcement learning is utilized for controlling the system in real-time, taking input from the previous three components, focusing on optimal decision making for the addressed scenario. Reinforcement learning work, whatever the orchestration scenario, focuses on the translation of deployment states and feedback calculated by the monitoring interfaces into orchestration actions. Federated learning is used to manage model training and inference of the former in cases where smaller models in federated compute nodes need to be used. The methodology is more thoroughly described in Figure 21.

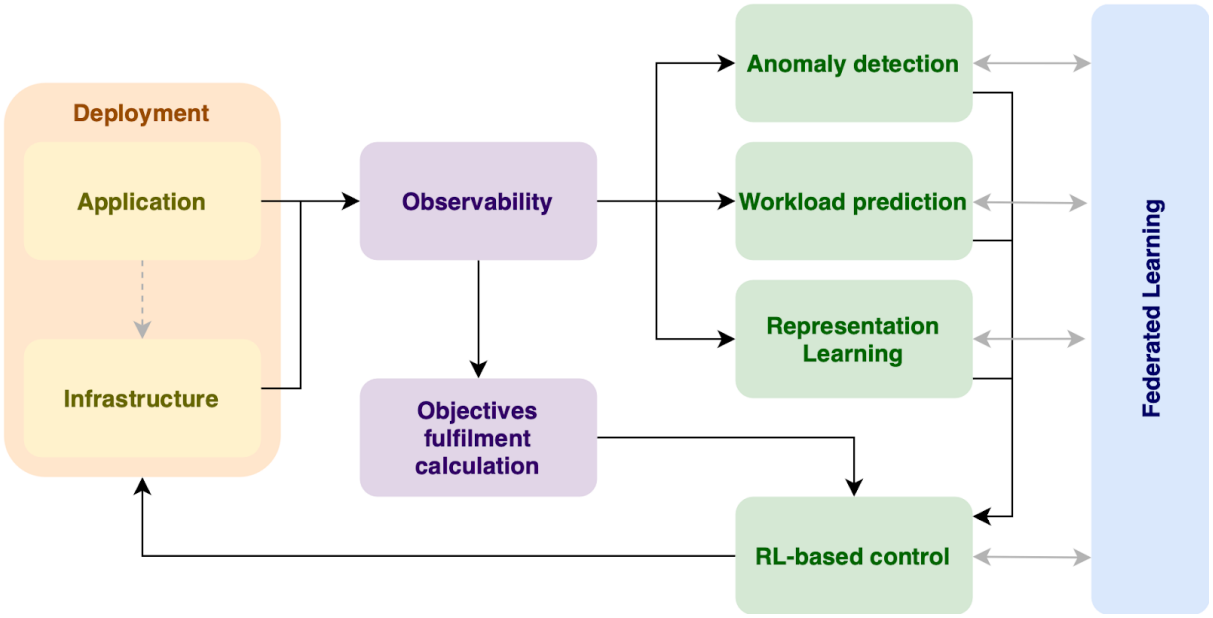

Figure 21: AI-assisted methodology

Observability data is collected from application and infrastructure components to be analysed for orchestration assistance. This data is used for calculating the degree to which the objectives have been achieved and is also exploited for anomaly detection, workload prediction and representation learning. Outputs from the components serve as input to the reinforcement learning algorithms, either as states of the system or as rewards, while orchestration actions for optimizing towards the objectives are selected by the RL agents. Federated learning is used for supporting the training and inference functionalities of the different ML models in cases where federated models are available, for data security reasons, model acceleration etc.

3.5.4 3D Monitoring

While the scheduling domain within the Cloud-Edge continuum has been extensively examined and has put forth numerous approaches, from high-level modelling to dynamic scheduling, there has been a noticeable dearth of discussions regarding monitoring methods. Addressing the challenges posed by

the ever-expanding array of metrics, which may vary in terms of units and update frequencies and cater to diverse system scales and topologies, primarily relies on conventional 2D representations, comprising charts, gauges, maps, and lists. However, these conventional representations do not offer visual abstractions specifically tailored for real-time comprehension and interactions in the context of machine-to-machine or machine-to-human interfaces. Furthermore, the contemporary requirement for monitoring in systems and applications extends beyond system metrics to encompass additional dimensions of information, for example, data acquired through IoT devices and sensors within the user space.

With the introduction of Palindrome.js[15], we present a novel 3D monitoring probe designed to facilitate multidimensional visual modelling and analysis. Palindrome.js, through the utilization of metric-based layers, effectively constructs a structured visual abstraction in the 3D space for addressing problems that are represented by metrics or KPIs.

Goal. The conceptualization and development of Palindrome.js encompass two primary objectives. Firstly, it aims to facilitate multidimensional display and analysis. Considering the escalating distribution complexity of systems and applications, the efficacy of enabling stakeholders to promptly monitor individual tenants, while simultaneously preserving a granular level of metric, hinges on the capacity of the sub-system to furnish multidimensional visual data. Secondly, it endeavours to provide a predefined visual representation to support computer vision analysis. The task of discerning various behaviours exhibited by heterogeneous metric sets can be tedious when dealing with large-scale and real-time in 2D. Leveraging computer vision can facilitate the dynamic identification of features, particularly when the subsystem furnishes a predetermined 3D visualization.

Methodology. We investigate the capacity of Palindrome.js to achieve these specified goals. This is accomplished by devising a concise metrics description framework, which embeds a 3 levels (min/med/max) range parameter for each metric organized into layers. Furthermore, we integrate a flexible 3D visualization engine, enabling the modelling of both generic and application-specific scenarios through customizable configurations, designed to cater to diverse visual requirements, whether targeted at human or machine-oriented visual outputs. The description interface is instantiated in JSON, the logic in JavaScript, and the 3D rendering engine using Three.js.

Figure 22 illustrates an example of an interactive Palindrome.js configuration on the web-based "storybook" environment, which offers the complete Palindrome.js API through interactive features within a graphical user interface (GUI). This allows users and developers to configure and test their settings before deploying them in a production environment. The library, inclusive of the testing environments, is publicly available as an open-source project under the Apache 2.0 licence.

[15] https://github.com/Smile-SA/palindrome.js

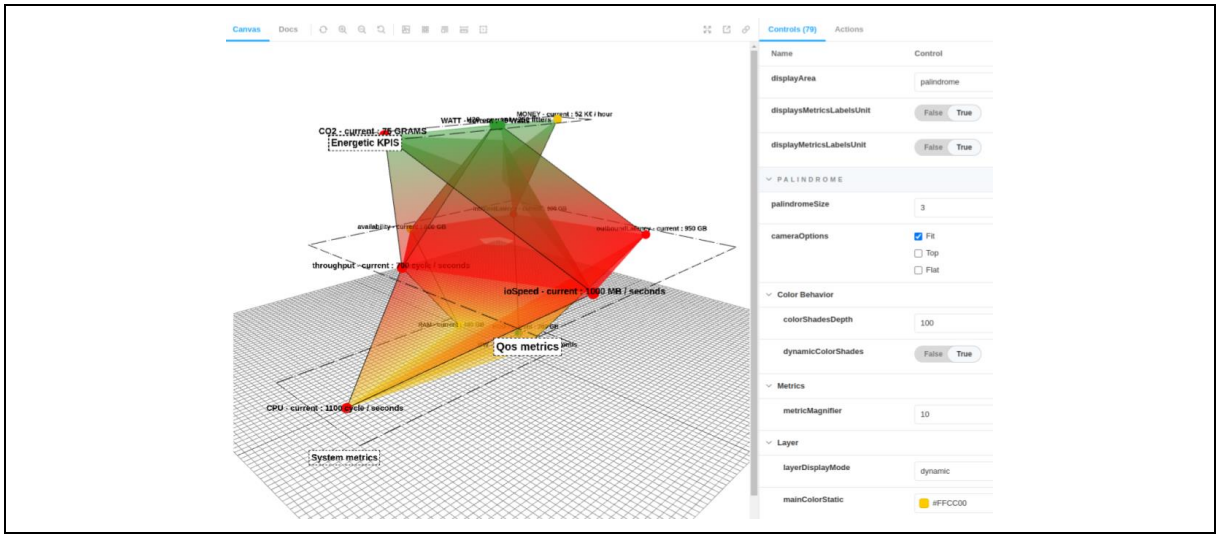

Figure 22: Example of an interactive Palindrome.js configuration

4 NEPHELE Dashboard and Development Environment

4.1 Development Environment

4.1.1 Hyper-distributed Applications Development Environment and Registry

All tasks in WP4 target the development of the Synergetic Meta-Orchestration framework (SMO) from a different perspective following a system of systems approach. This section compiles the progress and contributions of T4.1, which focuses on the developer side and the applications that they will create and, later deploy through the SMO, which are referred to as Hyper Distributed Applications (HDAs). T4.1 main contribution to the project is twofold, depicted in Figure 23, which are introduced in this deliverable:

1. A sandbox repository for developers will be built in a publicly available repository[16] within the Eclipse's development environment in GitLab. It will be used to host both, documentation, and practical resources. It will show how to create, verify, and distribute HDAs which will be made available to the SMO for deployment (see Section 4.1.1.1) through the HDA Registry.
    a. Documentation will be aggregated in a Gitlab repository to allow easy access to first, the NEPHELE's SMO framework documentation, and second, the technology landscape and Information Model (IM) that developers need to use to compose HDAs.
    b. A functioning Continuous Integration and Continuous Delivery demo repository for a ready-to-use HDA scenario.
    c. The necessary development utilities to successfully create an end-to-end HDA scenario.
2. A storage and distribution system for the HDAs named HDA Registry will be deployed as part of the SMO framework. It will offer custom functionalities for developers, by providing the hosting of repositories for all artifacts involved in the deployment of an HDA, and to the SMO itself (see Section 4.1.1.2).

[16] NEPHELE Project · GitLab (eclipse.org) | https://gitlab.eclipse.org/eclipse-research-labs/nephele-project

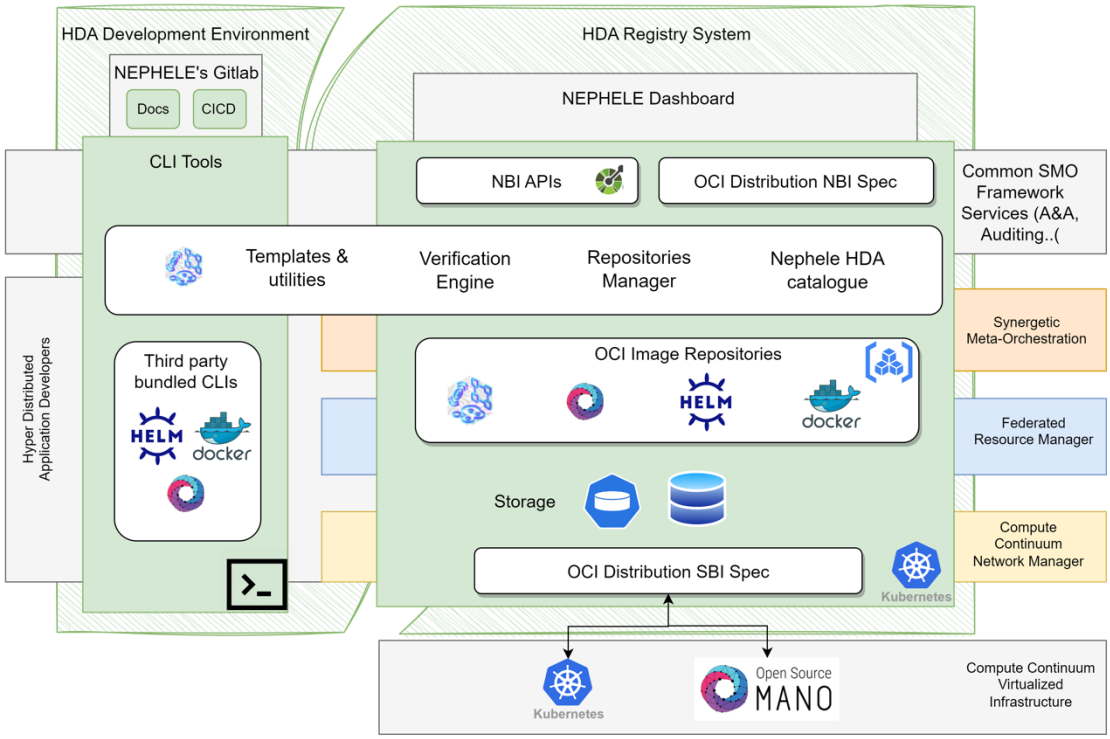

Figure 23: T4.1 development commitments

4.1.1.1 HDAs and Development Environment

The first part of this section relates to the definition of the HDA and the technology landscape from a developer perspective. In summary, an HDA is to be understood as a chain of services deployed by the SMO in the compute continuum. This is further defined in this section.

The second part of this section relates to creation of a development sandbox repository and the release of a Command Line Interface (CLI) utility to assist developers on the creation, verification and distribution of the artifacts related to the HDAs which are covered in the first part of this section. This is further defined in this section.

4.1.1.1.1 Hyper-Distributed Applications

All software applications are ultimately released as one or more types of artifacts. An artifact is anything that a system or user might need to successfully run a software. Some of the most known artifacts[17] in the Cloud Native (CN) industry include:

- Docker images.
- Helm charts.
- Precompiled binaries and installer packages.
- Software Bill of Materials (SBOMs).
- Open Policy Agent (OPA) bundles.
- Release signatures, certificates, and metadata

[17] OCI artifact | OCI Registry As Storage (oras.land) | https://oras.land/docs/concepts/artifact/OCI artifact | OCI Registry As Storage (oras.land)

T4.1 is responsible for the definition and alignment of the IM of the custom descriptors and artifacts of the HDA required by the SMO framework in taking into consideration the requirements described in D2.1, and with close collaboration with UCs and WP3 which will define the cVO implementations and IM. A HDA is realized as the combination of a graph descriptor (HDAG) that represents the end-to-end interlink of services running in the compute continuum along with an intent-based orchestration descriptor.

Although the artifact technology for the (c)VOs is already agreed to consist of a Helm Chart[18], each VO implementation may need a different set of configurations which may or may not finalise in the creation of a common descriptor with its own IM. Since they are yet under development and will continue to be defined as part of collaboration activities with WP3, the final decision will be part of the next release of WP4 deliverables.

Figure 24 depicts the complete set of services that take part in an HDA. From the perspective of the NEPHELE SMO framework, only the HDAG and the (c)VO Helm Chart need to be defined since they are specifically designed for the project while the remaining ones are based on mature technologies with large communities and documentation already available online.

1. A HDAG essentially describes, in a way that the SMO will understand, a chain of services, how and where to deploy each of them and any other relevant information to it. Figure 24 depicts the collection of artifacts that represent each service along with their interlinks for the most complex scenario. A HDAG is realized in the form of a custom artifact that consists of a descriptor file and a default intent orchestration file that the SMO uses to orchestrate the deployment and overall life cycle of each service. The intent file is to be customized at instantiation time or left with the provider's default content.
2. In the context of NEPHELE, (c)VOs have already been described in previous deliverables from a functional perspective mainly in WP3. From the point of view of the software development, a (c)VO will be handled by the SMO in the same way as a standard Helm application as they are indeed, a specialized CN application that provides functionalities to the NEPHELE ecosystem. The structure and syntax are defined by Helm, however, each implementation of (c)VO will need a minimum set of manifests and configuration files which is specific to the NEPHELE project, thus, T4.1 will provide the means to create, verify and catalogue the (c)VOs while the utilities to distribute the artifacts (push to and pull from the HDAR) will be delegated to the official Helm CLI.

[18] Helm Charts | https://helm.sh/docs/topics/charts/

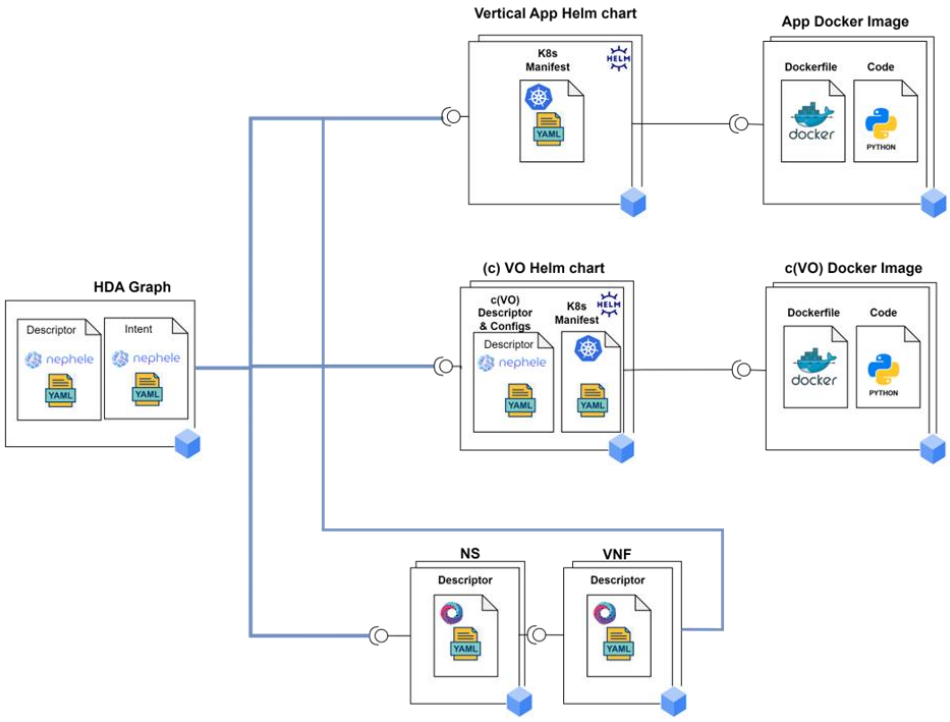

Figure 24: HDA artifacts and HDAG services

For each of the artifacts shown in Figure 24, there are references to the technologies used to create it. For the most part, artifacts are built as one or more YAML files (descriptors and manifests) distributed in a filesystem bundle (a directory) which describe what the orchestration system shall do.

In all cases, YAML will be used to create their descriptors. YAML[19] is a human-readable data serialization language that has a very simple syntax extensively used for declarative configurations files used by orchestration system such as Kubernetes[20] (K8s) or Docker Compose[21]. As a matter of fact, YAML has also been selected by T4.4 to compose the intent-based orchestration mechanisms which is also part of the HDAG artifact.

Resulting from WP3 and WP4 discussion it has been seen appropriate to agree on a common technology for all (c)VO implementations and for the most part for all Use Case (UC) applications. They will always be expressed as K8s workloads using Helm where the VO container image executed is configured using a series of descriptors and configuration files as defined by the VO implementation specification, which is prepared as part of WP3. There is additional interest to integrate Network Services (NSs) directly as part of the HDAG, for that, the initial approach will enable a way to also deploy NSs [50].

Originated from the Docker project in 2015, the Open Container Initiative[22] (OCI) is an open governance structure, maintained as part of The Linux Foundation[23] which is at the same time part of the Cloud-Native Compute Foundation (CNCF), for the express purpose of creating open industry

[19] What is YAML? (redhat.com)What is YAML? (redhat.com) | https://www.redhat.com/en/topics/automation/what-is-yaml

[20] Configuration Best Practices | Kubernetes | https://kubernetes.io/docs/concepts/configuration/overview/

[21] Overview | Docker Docs | https://docs.docker.com/compose/compose-file/

[22] Open Container Initiative | https://opencontainers.org/

[23] Linux Foundation | https://www.linuxfoundation.org/projects

standards around container formats and runtimes. OCI has resulted in three core standards that build the grounds of interoperability in CN environments[24] and that will be the basis of the HDAR for the creation, storage, and distribution of artifacts:

- The Runtime Specification (runtime-spec) details how to run a “filesystem bundle” that is unpacked on disk. It counts with implementors such as Docker Engine and Containerd which both enable CN workloads and the overall K8S ecosystem.
- The Image Specification (image-spec) outlines how to build the "filesystem bundle" that is used by the runtime-spec. However, image-spec has been extended to support generic image formats containing the released OCI image which may or may not be used by the runtime-spec. It has already been adopted to release Helm, SBOM, Digital signatures, Provenance data, Attestations and Vulnerability reports[25].
- The Distribution Specification (distribution-spec) defines an API protocol to facilitate and standardize the distribution of OCI images. The system that implements the distribution-spec is referred to as an OCI-Registry[26]. It has already been adopted by implementors such as CNCF, Amazon, Azure, GitLab, GitHub, Google, Docker and ZoT.

While most of the existing implementations for any of the three specifications focus on containerized workloads in the context of Kubernetes. The image-spec and the distribution-spec are by no means limited to K8s. Additionally, CNCF [51] continues to expand its work and bring more communities to the ecosystem to further enrich and enhance the interoperability of the containerization technology. An example of this alignment comes from the telecommunications industry, since the rollout of 5G where CN is a vital part of the technology landscape, processes and tools.

In fact, ETSI and CNCF have an existing agreement for collaboration to drive harmonization across Open Source and Open Standards [52, 53]. Collaborations between CNCF and the telco industry, with ETSI, have already resulted in successful projects such as the ONAP–project[27] which implements a platform for orchestration, management, and automation of network and edge computing services for network operators, cloud providers. From the ETSI side, their reference implementation of a similar orchestrator, the Open-Source Management and Orchestration[28] (OSM), also continues to align itself with CNCF in each new release.

Figure 25 depicts how the specification describes an OCI image as a collection of standardize manifests, namely, image manifest, config manifest along with the associated blobs of storage. For the initial release of the NEPHELE implementations, it is not expected to use the optional index manifest (top element in Figure 25) as it mostly serves as support file for giving additional information to different versions of the same image manifest. These are the common IM that an OCI-Registry uses handle the artifacts without the need to read the contents of it.

[24] https://github.com/opencontainers
[25] OCI artifacts on Docker Hub | Docker Docs
[26] Compatible OCI Registries | OCI Registry As Storage | https://oras.land/docs/compatible_oci_registries/
[27] Home - ONAP | https://www.onap.org/
[28] https://osm.etsi.org/

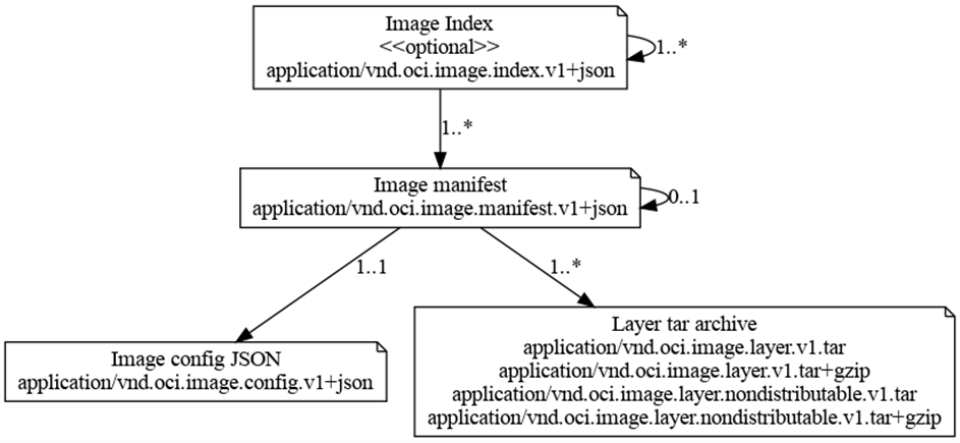

Figure 25: OCI Image-Spec manifests[29]

The core logic behind the image-spec and how it is used in the distribution-spec is that a "mediaType" defines the structural elements that are permitted in the manifest. In particular, the image config manifest is used to link to a custom configuration file, written in JSON format, which gives the required information for a consumer to correctly use the OCI image as a whole. The image manifest is the one that is consumed by the distribution-spec and thus, custom support is needed when moving away from the default version. At this moment, all implementors use the default image manifest "mediaType" while there is some work being done to generate a more generic one called artifact manifest or even to merge it with the current image manifest [54]. For each artifact shown in Figure 24, it must be possible to generate an Image and a Config manifest with a fixed structure and an associated "mediaType" identificatory. Table 1 depicts the summary of the compatibility with the OCI specifications for each of the artifact which will be supported by the HDAR.

Table 1: HDA OCI Image-spec compliance summary

Artifact	Provider	Config MediaType	Tool	Notes
App & (c)VO Container image	Docker	application/vnd.docker.container.image.v1+json	docker	
App Helm Chart	Helm (>=v3)	application/vnd.cncf.helm.config.v1+json	helm	
OSM NS	NEPHELE	application/vnd.etsi.nfv.v1+json	hdarctl	
OSM VNF	NEPHELE	application/vnd.etsi.nfv.v1+json	hdarctl	
(c)VO Helm Chart	NEPHELE & Helm	application/vnd.cncf.helm.config.v1+json	hdarctl & helm	(c)VOs are distributed as Helm Charts but their structure and syntax are managed by the hdarctl
HDAG	HEPHELE	application/vnd.nephele.hdag.v1+json	ıdarctl	

[29] https://github.com/opencontainers/image-spec/blob/main/img/media-types.png

For all NEPHELE's implementations of the Config "mediaType" from Table 1, the content will continue to be defined as the project continues, making sure that they are aligned with the corresponding orchestrator, SMO for the case of the HDAG and the (c)VO and OSM for the case of NS and VNF. At the time of submission of this deliverable, there is already a version 0.1.0 available in the HDAR repository which has been used as proof of concept.

4.1.1.1.2 Development Environment

This section relates to the creation of a sandbox repository for the development of all the services that are required to run the HDAs (see Figure 24) as well as to host the documentation on how to use the NEPHELE's SMO framework. This sandbox environment will consist of:

- A demo HDA Gitlab repository which will include an example of how to create each of the required services along with an automated pipeline that will perform CICD of the services towards the HDAR.

While for most services in the demo HDA repository, the CICD will be built on standard steps and based on already existing tools, for the (c)VO and HDAG services, there will be a CICD pipeline similar to what is described in Figure 26. Thus, T4.1 will contribute also with the creation of a CLI tool, named the hdarctl, that will enable a way to communicate with the HDAR following the OCI specifications and to run local verification steps.

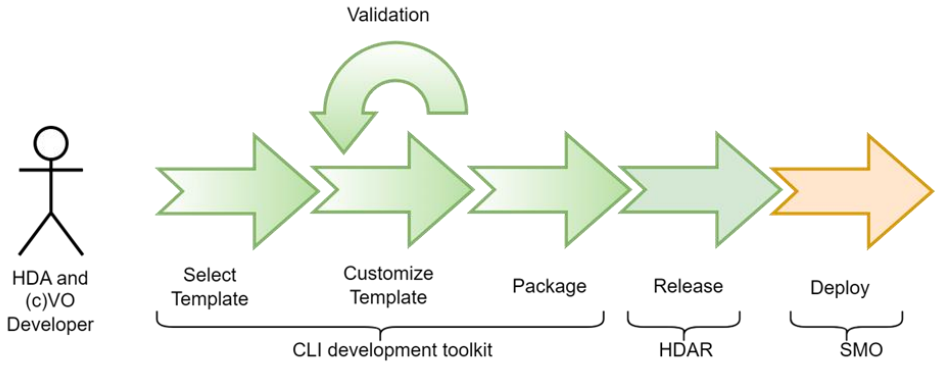

Figure 26: NEPHELE's specific artifacts CICD pipeline

- All documentation needed to create a developer guideline will be centralized in Gitlab using a static web page generator that will enable a way to keep the official documentation published automatically and up to date, exposed online using GitLab pages[30], read-the-docs[31] or similar, and containing:
  - Instructions to create each of the services of an HDA.
  - Single entry point to the description of the specifications of the HDAG and the (c)VOs, including how to create and configure them, linking to external documentation and communities when necessary.
  - Description of the demo HDA Gitlab development repository. Instructions on how to register and use the NEPHELE SMO platform, including HDAR and SMO interactions.

[30] Tutorial: Create a GitLab Pages website from scratch | https://docs.gitlab.com/ee/user/project/pages/getting_started/pages_from_scratch.html
[31] Full featured documentation deployment platform | https://about.readthedocs.com/?ref=readthedocs.com

The demo HDA development repository will leverage the built-in CICD capabilities of Gitlab using a ".gitlab-ci.yaml" configuration file. Said configuration file will be based on a template, documented as part of the developer guidelines, aiming to contribute and simplify the work of the developer. The Eclipse development environment has already set-up a series of repositories within a group accessible under the NEPHELE Project GitLab [32].

As described before, there will be a HDAR CLI, written in Golang, named hdarctl which will be delivered as a single executable file with no dependencies. The CLI will be developed using the official OCI-Registry As Storage (ORAS) libraries[33] to develop the corresponding implementations of the OCI image specification. This will greatly simplify the configuration of the environment for HDA developers and enable an easy automated a CICD pipeline. The hdarctl v0.1.0 has already been released, supporting the preliminary versions of the IM of (c)VOs and HDAGs.

The main requirements considered for the development of the hdarctl come mostly from D2.1 [1]. The final release of the hdarctl will provide the utilities shown in TABLE HDARCTL and its development will consider the following requirements:

- Small profile and simple installation
- Compatibility with Linux and Windows systems
- Simple set of command alignment with the syntax of other commercial-grade CLI utilities such as Helm.

Table 2: hdarctl commands summary

Comand	Description
Login/Logout	Will interact with the Authorization and Authentication system of the NEPHELE SMO framework to gain local access to the HDAR
Create	Will enable a way to create a baseline artifact, HDAG or (c)VO based on user input.
Lint	Will provide local verification tests over an artifact searching for syntax, structure, and functional errors.
Pull/Push	Will enable a way to interact with the HDAR repositories to download/upload artifact.
Package	Will enable a way to create an OCI-compatible artifact based on a filesystem bundled.

4.1.1.2 HDA Registry System

The HDAR is the storage, distribution, catalogue and verification system for all services involved in a HDA within the NEPHELE project (see Figure 24). The terms "repository" and "registry" are often used interchangeably, however, there are some key differences between the two that are important to fully understand the HDAR system in the NEPHELE project. A registry is a storage and distribution system for those artifacts. A registry is organized into repositories, where a repository holds all the versions of a specific artifact[34] (see Figure 27 for reference).

[32] https://gitlab.eclipse.org/eclipse-research-labs/nephele-project
[33] OCI Registry As Storage | https://oras.land/docs/client_libraries/go/
[34] Access repositories | https://docs.docker.com/docker-hub/repos/access/

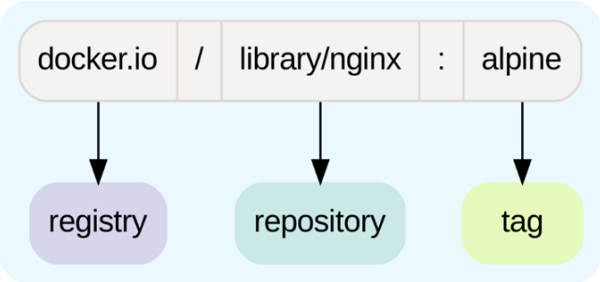

Figure 27: Artifact naming convention in distribution-spec[35]

It is of great importance to T4.1 to leverage innovative solutions to provide a consistent registry system that can unify how all of them are handled while making sure that the proposed system is based on common standards and aligned with each technology's roadmap.

Aiming to contribute to this alignment of communities when applied to the compute continuum ecosystem, the HDAR will be implemented as a microservice-based system which will include an OCI-Registry compatible with custom image-spec implementations of all the services involved in the deployment of the HDAs which cover NEPHELE-specific implementations and others that are standard:

- The HDA Graph
- OSM NSs and Virtualized Network Functions (VNF) (rel >=V8)
- Docker containers images
- Helm Charts (rel >=V3) including the VO and cVO.

The HDAR will be deployed as a centralized component of the SMO framework in K8s. The OCI-Registry implementing the distribution-spec that will be selected will be the official one maintained as part of the CNCF codebase[36] [55] which will be configured to use the SMO framework's Authorization and Authentication service (A&A), auditing of artifacts and other cross-framework services that may be offered.

Figure 23 depicts the overall design architecture of the HDAR as well as its relationship with other components of the SMO framework. Its design has taken into consideration the requirements collected in D2.1 and the initial release of the remaining systems of the SMO framework.

- OpenAPI North Bound Interface to expose the functionalities required by the SMO framework.
- OCI-Registry and compatibility with standard and custom artifacts offering OCI image-spec; HDAG, VO, cVO, OSM NS and VNF, docker images, Helm charts.
- Private and public repositories for all artifacts.
- A&A (RBAC or LDAP) to enable multi-tenancy.
- Hosting for VO and cVO base docker images.
- Exposure of events related to artifacts for their auditing.
- Support for HDA development CICD.

[35] https://oras.land/docs/concepts/reference/#what-a-reference-means-in-the-registry

[36] The toolkit to pack, ship, store, and deliver container content | https://github.com/distribution/distribution

The HDAR system will additionally count with two complementary componets, see Figure 24, developed as golang-based microservices to offer first, an OpenAPI V2[37] based interface to interact programmatically with the HDAR, and second a verification engine to execute a CICD pipelines for artifacts. Figure 28 extends what was shown in Figure 23 and depicts the functional blocks that compose the internal architecture of the HDAR System module of the SMO framework. Each of the components shown in Figure 28 will be built following a micro-service architecture with the additional help of module-wide services such as databases, for the persistence the artifacts, auditing of events and other configurations, and a message broker for the delivery of messages among the module components and the scheduling of verification tasks.

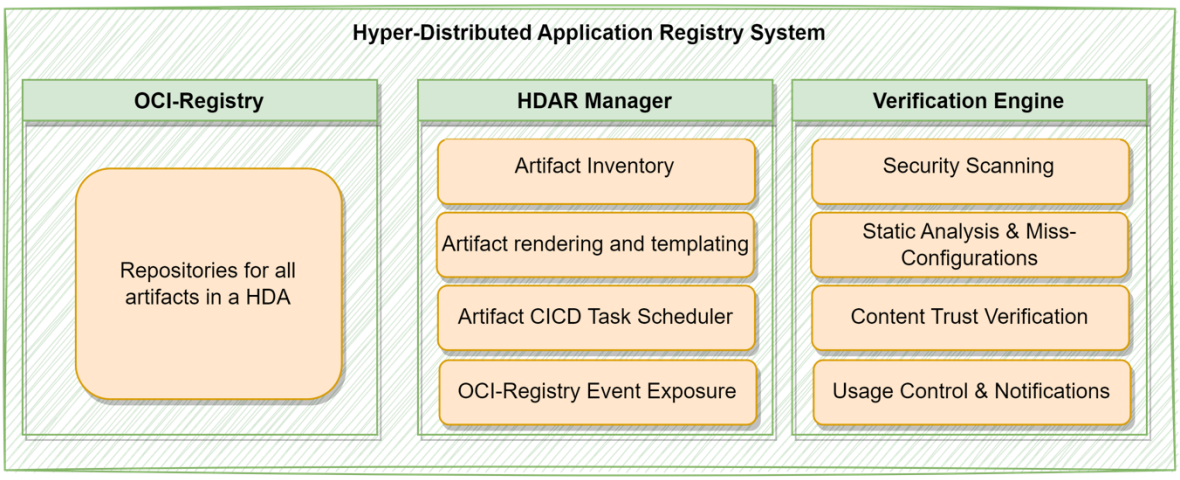

Figure 28: Functional blocks of the internal architecture of the HDAR System module of the SMO framework

The OCI-Registry will be based on the baseline implementation from CNCF[38] which is essentially is a stateless, highly scalable server-side application that implements the distribution-spec to store and distribute container images and other OCI artifacts. The client-side application that will be needed to interact with it has been described as part of the development environment outcomes in the previous subsection. By removing links to other vendor specific implementations, the HDAR system will provide a custom-tailored pipeline for the artifacts and will additionally result in an easier integration with the rest of the modules in the NEPHELE platform.

The HDAR Manager will contain the core logic of the pipelines for handling OCI artifacts to support the HDA Development Environment. This component will be exposed with a custom REST API following the OPEN API v2 specification which is being designed in aligenment with the needs of the Dashboard as well as with those functionalities required by others modules in the SMO framework.

The Verification Engine will work in parallel with the HDAR Manager following a Pub/Sub approach to carry out a series of checks over the artifacts to assist the HDA developers. It will use the artifact-specific tool whenever existing (e.g., Trivy[39]), or else, the HDAR will execute its own set of verification and scanning tests to provide information about; 1) security scanning to raise Common Vulnerabilities and Exposures (CVEs); 2) syntax errors and other miss-configurations respect to the corresponding IM;

[37] OpenAPI Specification - Version 2.0 | Swagger | https://swagger.io/specification/v2/
[38] https://distribution.github.io/distribution/
[39] aquasecurity/trivy: Find vulnerabilities, misconfigurations, secrets, SBOM in containers, Kubernetes, code repositories, clouds and more | https://github.com/aquasecurity/trivy

3) anti-tampering validation based on content digital signatures; 4) auditing, notifications and custom control over the usage of artifacts.

4.2 Dashboard

In the NEPHELE project, we plan to integrate a dashboard with a dual-purpose interface. On the one hand, it will offer visualisations that will allow users to gain comprehensive insights into the orchestration ecosystem, enabling informed decision-making. On the other hand, it will serve as an interactive platform where users can input high-level intents, enabling direct communication with the system to trigger specific actions or configurations. In the coming period, we will investigate existing tools for telemetry and visualisations, that will be integrated in the NEPHELE platform.

5 Implementation Status and Roadmap

5.1 Experimental Comparison of Inter-Cluster Communication Solutions

The multi-cluster approach entails the deployment of an application on one or more Kubernetes clusters, either individually or collectively. This facilitates the enhancement of application availability, isolation, and scalability. The implementation of multi-cluster systems is significant to ensure compliance with diverse and perhaps contradictory regulations.

This is due to the adaptability of individual clusters, which can be modified to adhere to specific local requirements. The enhancement of software delivery speed and safety can be achieved by adopting a strategy where individual development teams deploy applications to segregated clusters and selectively expose specific services for testing and release purposes.

Benefits of Multi-Clustering

1. Increased scalability & availability
2. Application isolation.
3. Security and Compliance.

Multi-Cluster orchestration in NEPHELE

In the context of the NEPHELE, multi-cluster orchestration is essential due to the nature of VO. The latter aims to enable the convergence of IoT, edge and cloud computing technologies and facilitate the development of distributed IoT applications that can be manageable across resources in the computing continuum. To this end, we can consider, that a VO can either be deployed on an edge data centre or on a cloud data centre if its functionality needs more computing capacity.

To facilitate the orchestration of multiple Kubernetes clusters, we evaluate different tools. Two well-known implementations of such platforms are LIQO and Karmada. The pros and cons of each tool have been described before.

Scenarios

Baseline. In the preliminary phase of our experimentation, we conducted an evaluation of three distinct scenarios. In the first scenario referred to as the baseline setup, a single Kubernetes cluster has been deployed within a data centre (DC). This cluster consists of a single master node and three worker nodes. Worker node 1 is co-located with the control plane in the same DC, whereas worker nodes 2 and 3 are deployed at separate remote sites. In this setup, the nodes communicate over the internet. We can consider that the Worker Node 2 is running at the edge of the network, while Worker Node 3 is running in core cloud. An illustration of this setup is depicted in Figure 29.

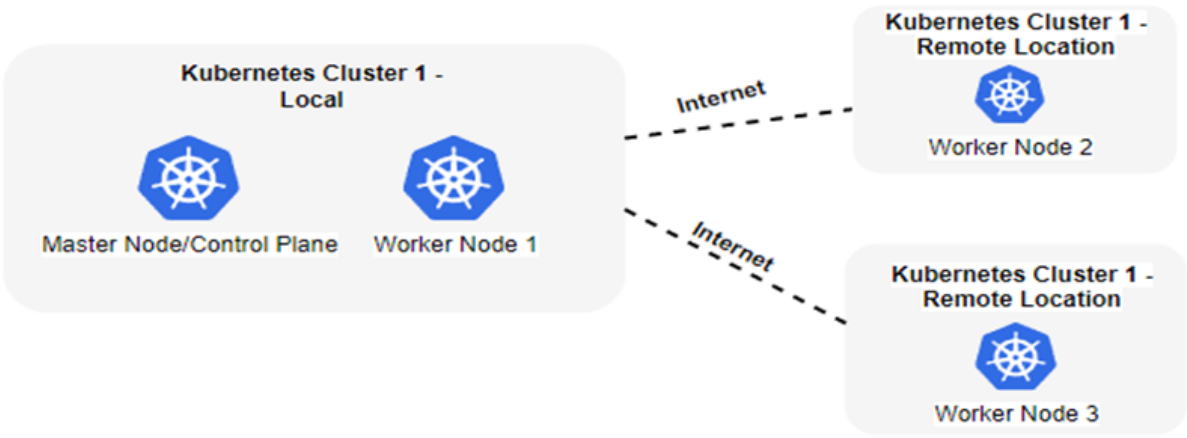

Figure 29: Remote Node Scenario

LIQO/KARMADA.

The next scenario depicted in Figure 30 is associated with three Kubernetes Clusters that are deployed on different locations. Nephele1 is the local cluster and Nephele2 and Nephele3 act as the remote clusters. Inter-cluster communication is enabled using LIQO and KARMADA. The Nephele2 cluster is considered as a cluster at the edge, while Nephele3 serves as a cluster residing in a core cloud (see Figure 31). The main goal of this experiment is to examine potential trade-offs in terms of VO orchestration across the compute continuum.

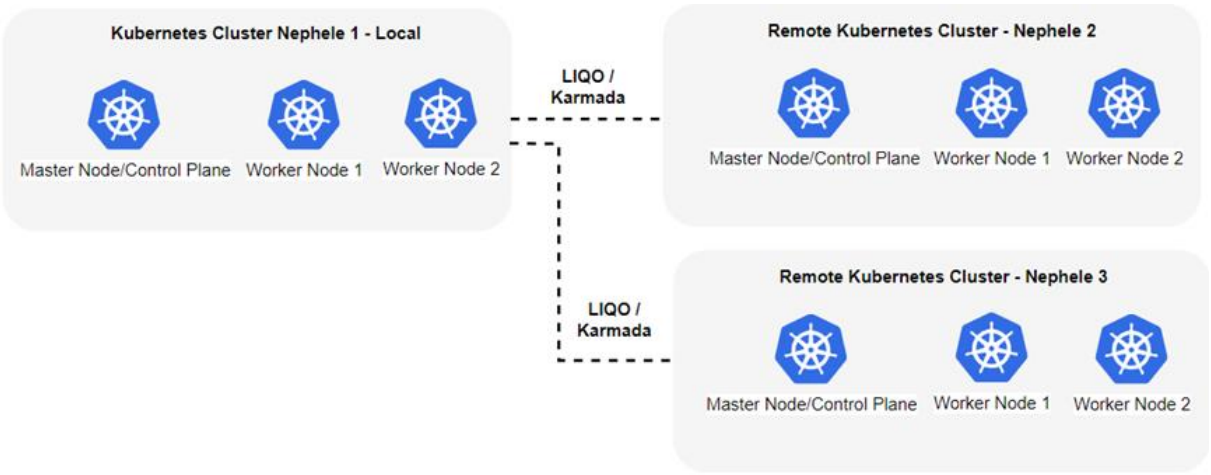

Figure 30: Enable Multi-Clustering with LIQO/KARMADA

Based on the above topologies, we measure the latency between the Nephele1 cluster and clusters Nephele2, and Nephele3. These clusters host pods that fulfil the role of VOs.

```
clusterslice@apt034-apt:~$ kubectl get clusters
NAME       VERSION   MODE   READY   AGE
nephele1   v1.28.3   Pull   True    21h
nephele2   v1.28.3   Pull   True    47h
nephele3   v1.28.3   Pull   True    47h
```

Figure 31: NEPHELE Clusters

To measure the latency between two VOs that are running in different parts locations of the network, we inject ICMP traffic from the nephele-client pod to the nephele-server pod with packets of different sizes: 64, 128, 256, 512, 1024, and 1500 bytes. For each packet, we send 100 ICM– requests. The corresponding results are illustrated in the following.

Local - Edge (Nephele 1 - Nephele 2)

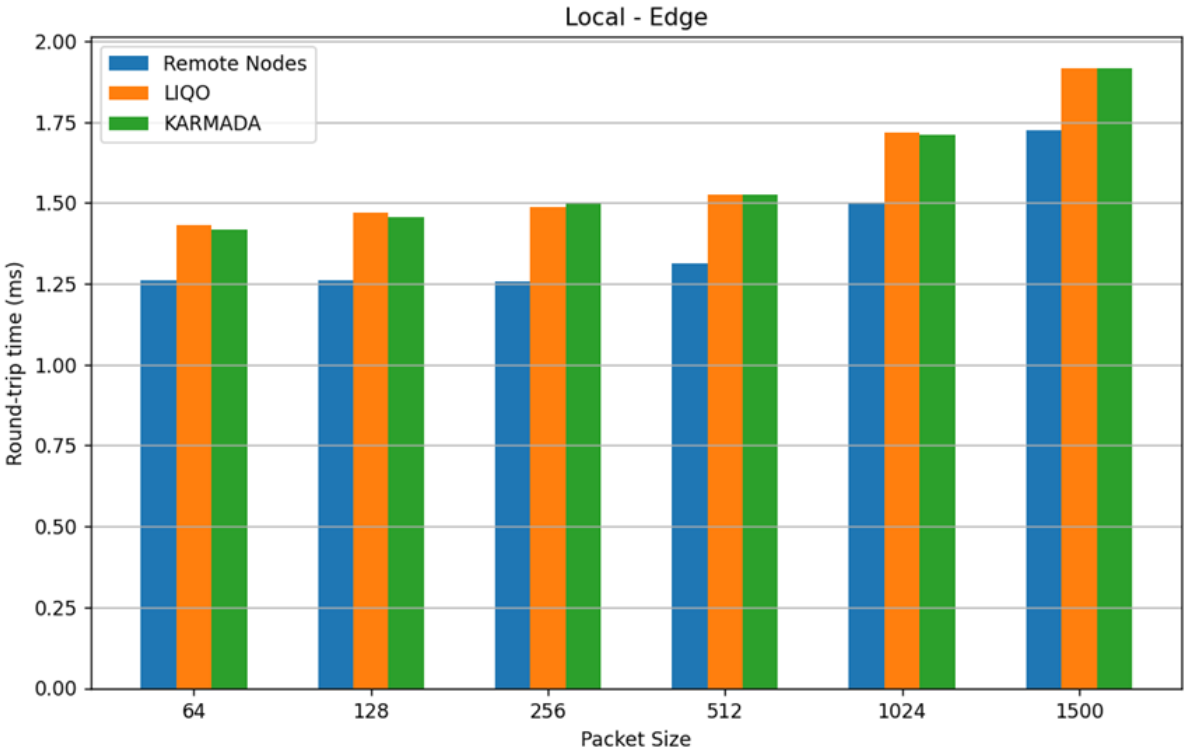

Figure 32: RTT between clusters (Nephele1 - Nephele2)

Figure 32 shows that LIQO and KARMADA yield almost the same RTT across the whole range of packet sizes. This stems from the fact that they both employ VXLAN for pod communication among the different clusters. Overall, LIQO and KARMADA exhibit similar characteristics in terms of latency. On the other hand, the use of remote nodes yields lower RTTs. Although the margin in terms of latency between the remote nodes and the multi-clustering tools is noticeable, we would refrain from using the setup with the remote nodes, due to the security concerns that it introduces (i.e., traffic is exchanged among clusters over insecure Internet paths).

Throughput

Next, we measure the throughput between the two pods. To this end, we use the same packet sizes and we run iperf (in TCP mode) between the two pods running in clusters Nephele1 and Nephele2 and Nephele3. Table 3 illustrates the average throughput values for packet sizes ranging from 64 to 1500 bytes. Like the previous experiment, we perform throughput measurements with Remote Node, LIQO,

and KARMADA. The throughput of LIQO exhibits a slightly higher degree of variability compared to that of the Remote Node. However, this fluctuation remains low, as it consistently falls within the range of 88.8 to 89.6 Mbit/sec. Likewise, the throughput of KARMADA exhibits stability, with a slight fluctuation ranging from 88.7 to 89.7 Mbit/sec.

The observation that LIQO is on par with KARMADA in terms of throughout (across the various packet sizes) implies a correlation with their utilisation of identical technology for pod communication, as previously stated. The Remote Node's marginal advantage in throughput performance may be attributed to its utilisation of a more direct communication approach, which, although efficient, may pose security vulnerabilities if the sent data is inadequately safeguarded.

When considering (inter-)cluster networking, it is important to evaluate the potential advantages and drawbacks of a Remote Node. Although it may provide a slight increase in terms of throughput, the associated security issues must be carefully taken into consideration. The decision to choose between Remote Node, LIQO, or KARMADA would be affected by other aspects, extending beyond raw performance. Security considerations would play a significant role in this decision.

Table 3: Average Throughput between clusters (Nephele1-Nephele2)

Average Throughput (Mbps)			
Packet Size	Remote Node	LIQO	KARMADA
64	**91.5**	**89.3**	**89.5**
128	**91.6**	**88.8**	**89.5**
256	**91.6**	**89.6**	**88.7**
512	**91.4**	**89.5**	**89.4**
1024	**91.7**	**89.3**	**89.7**
1500	**91.6**	**89.4**	**89**

Local – Cloud (Nephele1 – Nephele3)

We conduct the same experiments between the clusters Nephele1 and Nephele3. Recall that Nephele3 acts as a core cloud cluster. The delay between the pods is depicted in Figure 33, whereas the corresponding throughput measurements appear in Table 4. The observations drawn from the results are like the previous experiment. We only observe an inflation in latency, which is attributed to the longer distance between the clusters.

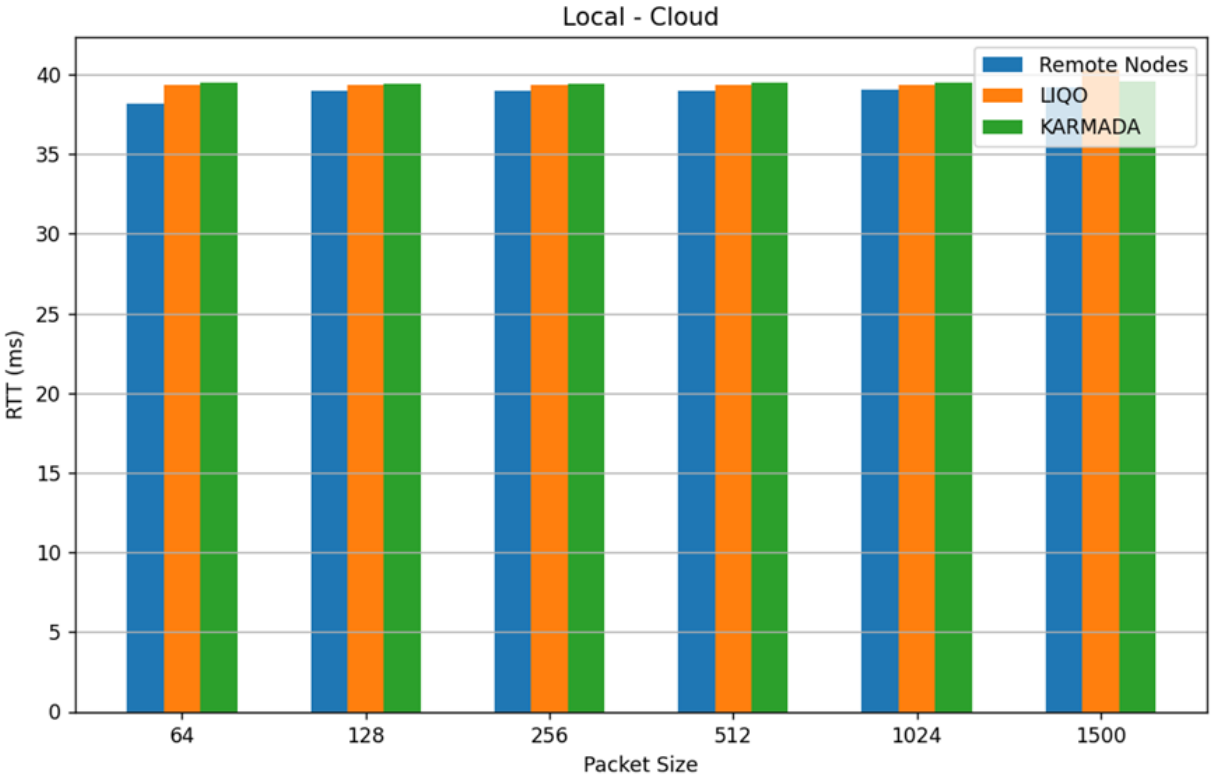

Figure 33: RTT between clusters (Nephele1 - Nephele3)

Table 4: Average Throughput between clusters (Nephele1-Nephele3)

Average Throughput (Mbps)			
Packet Size	Remote Node	LIQO	KARMADA
64	**552**	**550**	**475**
128	**550**	**537**	**470**
256	**549**	**516**	**475**
512	**552**	**533**	**466**
1024	**551**	**519**	**456**
1500	**551**	**522**	**475**

Overall, LIQO and KARMADA yield similar data plane performance, which is slightly deteriorated in comparison to a setup with remote nodes. This degradation stems from the performance penalty induced by header encapsulation/decapsulation (due to VXLAN) and packet encryption operations (due to TLS). Nevertheless, we consider that the secure communication between clusters can outweigh this small performance penalty, so both tools will be under serious consideration for inter-cluster communication.

5.2 3D monitoring and multidimensional alerting

As depicted in subsection 3.5.4, 3D monitoring, to demonstrate the ability to provide a multidimensional scheduling mechanism, customisations over code and configuration were applied to the open-source library Palindrome.js. Serving as a 3D visualisation probe, Palindrome.js empowers multidimensional information visualisation.

Code changes were considered to provide a formal display result. Namely, specific visual orientation (known as the flat mode), and colouring declarative system and display (where the user has a better control over displayed colours).

Palindrome.js was experimented in parallel with the developments as a headless information provider for OpenCV. Subsequently, OpenCV was employed to analyse colours, including indexes, and visual volumes, and map them with predefined states.

The experimental code setup is made available as a private repository to partners on the project Gitlab platform.

As metrics used within the conducted experiment were randomly generated data at runtime, possible foreseen developments are integration of real data providers, such as VO metrics collectors, KPIs providers or AI modules.

Within the NEPHELE project, we introduce the concept of computer vision assisted Palindrome for synergetic orchestration tasks. Specifically, we explore its capacity to facilitate synergetic scheduling through the integration of Palindrome.js and OpenCV. Synergetic scheduling aims to deliver coherent scheduling decisions from either single system states or multiple systems of systems states. While Palindrome.js can display multi-dimensional data representations, the implementation of an automated state detection mechanism is expected to validate novel scheduling directives (see Figure 34).

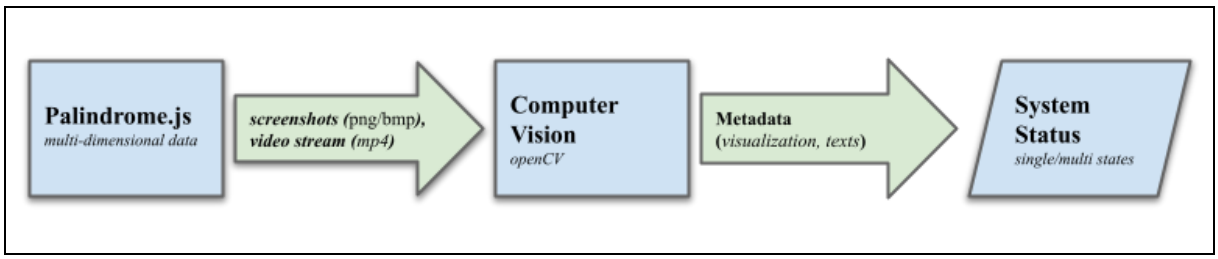

Figure 34: General architecture for the case of combining 3D monitoring and multidimensional alert

The computer vision (CV) assisted Palindrome experiment employed the OpenCV library to identify colours and their respective volumes in Palindromes.js. Derived data from side views or top views functions as a fundamental basis for making further informed decisions. Through the analysis of colours and their volumes, valuable insights can be gleaned, potentially extending their applicability to diverse applications and scenarios.

Use case 1: Single state detection. In this context, our experiment considers a single-color configuration for Palindrome.js. It becomes feasible to define metric value intervals, categorizing them into levels such as low, medium, and high. Furthermore, associated colours can be allocated to each stratum of the Palindrome.js system based on its respective value range. Specifically, the colour green signifies low values, yellow designates those in the medium range, and red is indicative of high values. Each colour serves as an indicator for a predefined system state. Colour detection and contour analysis are shown. Then subsequent system states are extracted, discerning whether the system is idle or fully utilized. This information is subsequently utilized for making informed decisions regarding the system's

capacity to accommodate additional services. Figure 35 provides an illustrative representation of the state detection as observed from the top view of a single-color Palindrome.js.

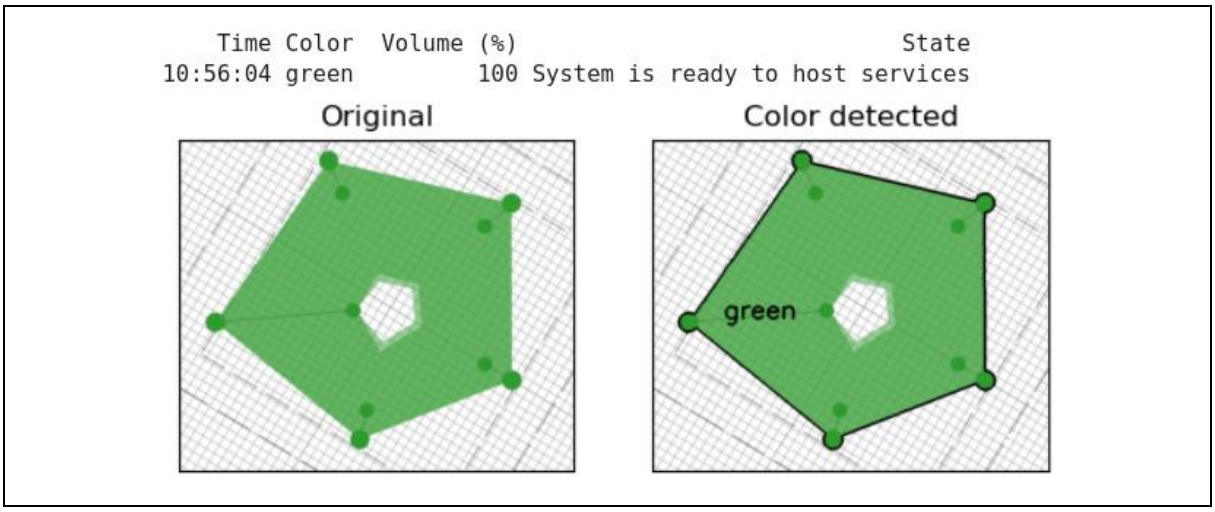

Figure 35: Colour recognition as observed from the top view of a single-color Palindrome.js, accompanied by the system's state

Use case 2: Multi-state detection without threshold. In this context, we consider a multiple-colour configuration for Palindrome.js. The CV assisted Palindrome experiment detects contours and colours from various viewpoints of Palindrome.js. This approach enables the comprehensive analysis of Palindrome.js from a multitude of orientations, facilitating an in-depth comprehension of its inherent properties and characteristics. Figure 36 illustrates the system's states by the analysis of colours and according to volumes.

In Figure 36, it is evident that four distinct colours are discernible, each representing distinct states within the context of the multivariate problem. Notably, each layer of the 3D object is associated with a state unique to itself. Consequently, this observation presents a considerable challenge to the local scheduler. In response to this challenge, we contemplate various strategies for the selection of the entire system's state:

- Predominant colour: The layer with predominant colour, ascertained by its highest frequency or biggest volume among the layers, is selected to represent the whole system.
- State hierarchy: In certain instances, the various states may exhibit a hierarchical interrelation, wherein one state is described as more important than others. Hence, this state is considered the overall representation for the whole system.

Use case 3: Multi-state detection with a threshold. We conduct simulations for multiple systems in 3 distinct scenarios: 1. Ready to go: a system is deemed online only when its metrics values exceed a predetermined threshold. When all the systems are online, they are considered "ready to go"; 2. Not good to go: one or more systems transition from online states to a sleeping state. Then they are considered "not good to go"; 3. Exit: if the metric values of a system fall below predetermined thresholds, it is deemed to be offline. One offline system will result in the exit of all these systems. Figure 37.a illustrates our use cases from the "Not good to go" scenario to "Ready to go". Figure 37.b depicts the "Exit" scenario, when any system exits, all the systems exit.

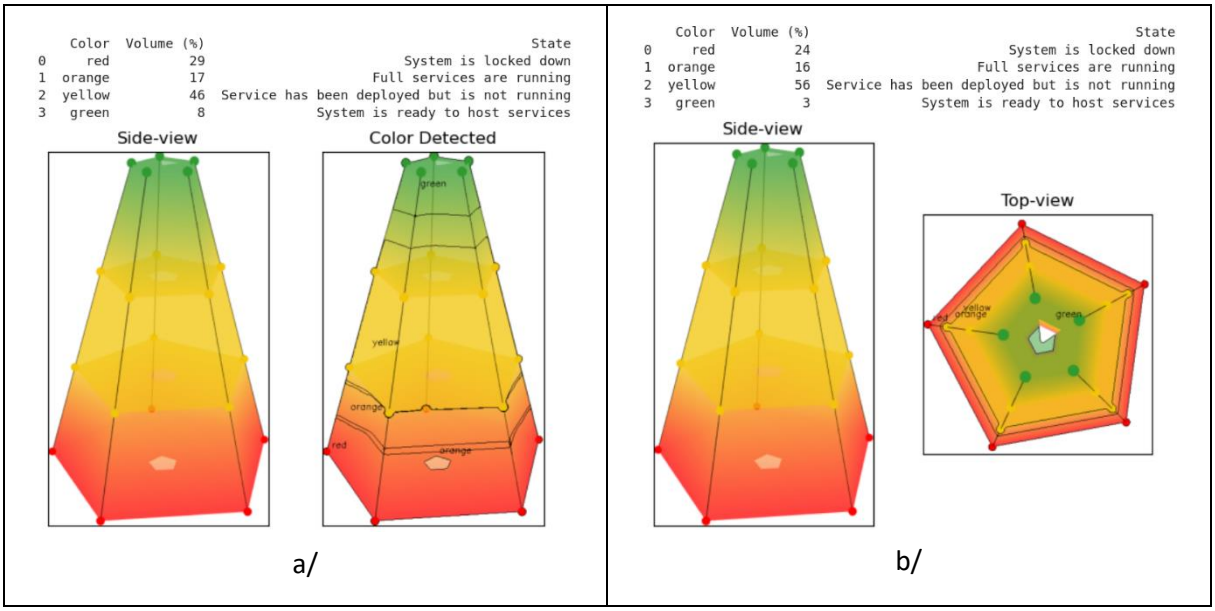

Figure 36: Colours identification from the side/ top view of Palindrome.js alongside the according volumes and states.

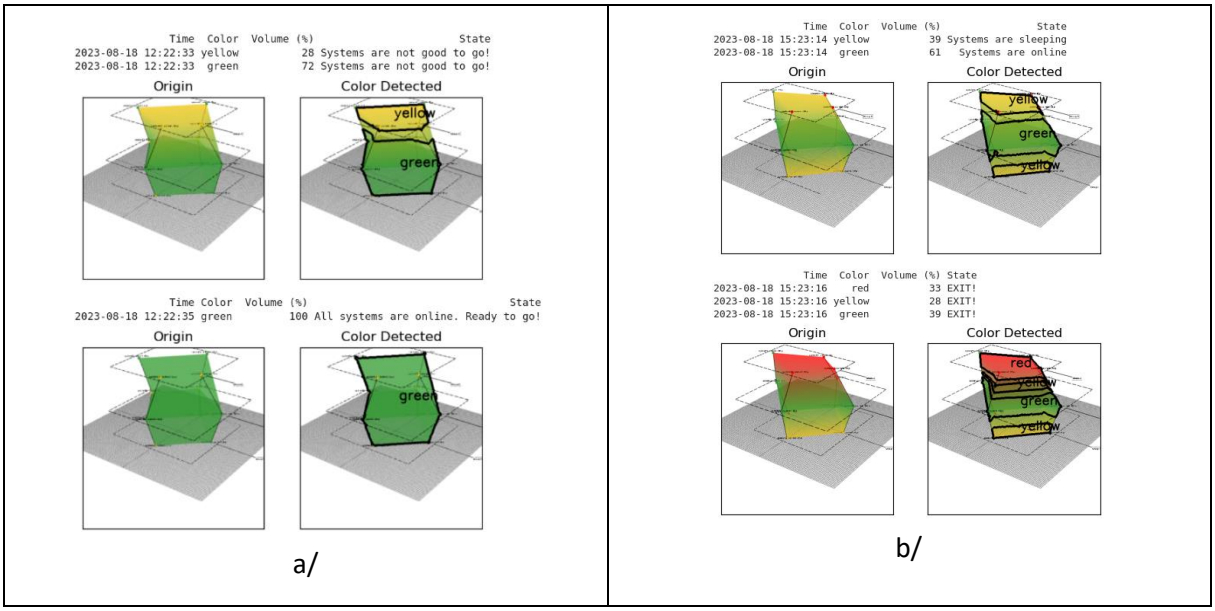

Figure 37: Colours identification for the case of multi-state detection with a threshold

5.3 Security

A security demo has been implemented to be integrated in the VO-WoT demo for the OpenCall. Walt.ID[40] has been chosen as the source for the implementation of security. Walt.ID is an open-source solution written in Kotlin, offering libraries that aligns with the use of DIDs, VCs and VPs. Specifically, we have used the SSI KIT library from Walt.ID, which is a comprehensive solution that streamlines the

[40] https://walt.id/

development of end-to-end use cases by eliminating the need for researching, combining, or adjusting various libraries for pilot projects or production systems. It simplifies the process by abstracting complexity and low-level functionality through interfaces such as CLI and APIs. The system is modular, composable, and adheres to open standards, allowing for customization and extension of functionality with both proprietary and a third-party implementation to prevent lock-in. Its flexibility enables deployment on-premises, in a (multi) cloud environment, or as a library within your application. The SSI Kit also offers the following functionalities for Issuers, Holders, and Verifiers:

- **For Issuers**: process and authenticate data request by people or organizations, import data (from local storage or third parties), create re-usable VC templates, create VCs in different formats (JSON/JWT, JSON-LD) [55, 56], sign VCs using different key types (e.g.,. ed25519, secp256K1, RSA), manage the lifecycle of VCs (e.g. revocation), issue VCs (e.g. via OIDC/SIOP).
- **For Holders**: interact with Registries (read, write), create/store/manage keys/data (DIDs, VCs) and other secrets, request and import data (VCs) from third parties, selectively disclose data (VCs/VPs) for authentication and identification, manage consent and data access in a user-centric fashion.
- **For Verifiers**: request data (VCs/VPs) from stakeholders, verify dada (VCs/VPs; incl. Integrity, validity, provenance, authenticity), trigger pre-defined actions following the verification.

For the development of the security implementation carried out for the VO-WoT demo, the source code of Walt.ID had to be modified to meet the project's requirements, thus adapting it to the needs of the demo. These modifications are mainly because the solution offered by Walt.ID is more focused on being deployed on the web and requires user interaction. The adjusted version of the Walt.ID implementation for the demo is also open-source and is public for everyone in the NEPHELE Eclipse Repository[41].

Since it's a demo and the development is in an early stage, the security and trust management implementation is based on deploying the Holder and Verifier components as services in each cVO/VO/App of the demo, aiming to achieve an approach as decentralized as possible. The Issuer is not deployed for the demo, so the VCs are pre-established and not generated at runtime, as they should be. Figure 38 shows a top view of the components deployed for each element of the architecture.

[41] https://gitlab.eclipse.org/eclipse-research-labs/nephele-project/vo-security

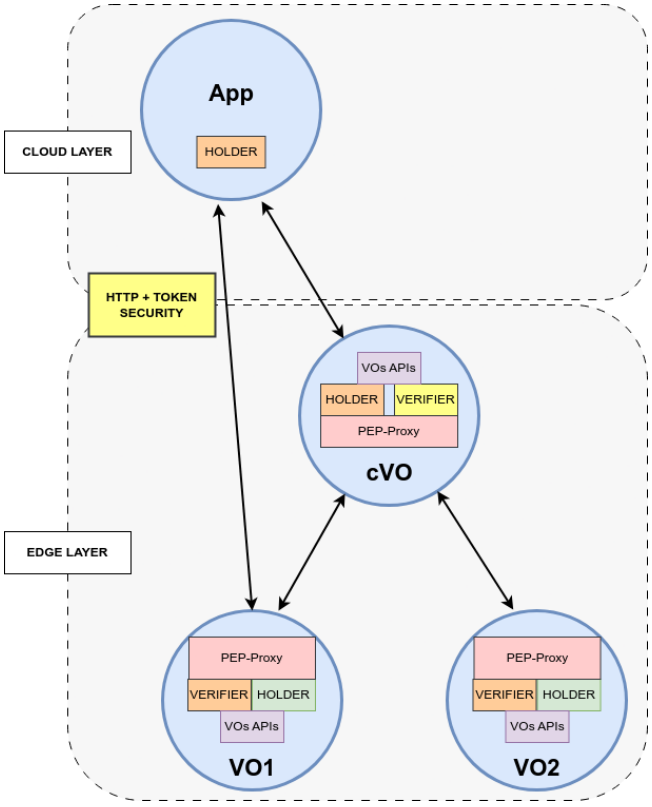

Figure 38: Overview of security components deployed

The Holder acts as a wallet. The Holder functionality is to obtain the SIOP request, select the Verifiable Credentials indicated in the SIOP Request, create the VP and send it to the correspondent API (also indicated in the SIOP Request) to be verified and to obtain the Access Token (OIDC4VP Protocol). As it acts as a wallet, the Holder also stores the VCs and the DIDs. This component offers internally the API '/auth' to receive data from the IVO/App where it is deployed. The Holder acts as a server by offering this API, but it also acts as a client when communicating with the target Verifier to obtain the Access Token, e.g., the Holder is responsible of triggering the OIDC4VP protocol.

The Verifier generates the SIOP Request and then verify the credentials to generate the Access Token. The SIOP Request is a request that is sent by the Relying Party (in this case, the Verifier) to an individual's self-issued OpenID Provider asking for an OpenID Token. The request includes information about the relying party and the specific claims or information it requires from the Holder. The OpenID Token is composed of two parts: the ID Token and the Verifiable Presentation Token (VP Token), and both are generated by the Holder. The Access Token, shown in Figure 39, is a Bearer Token in JSON Web Token (JWT) format and signed with the private key of the device where the Verifier involved is deployed. This Access Token includes information about the authenticated entity or the context of the token such as the Issuer, the subject, expiration time, etc. This component only acts as a server by offering internally two APIs, the '/obtainVP' API to generate the SIOP Request and the 'verifyVP' API to verify the VCs and to generate the Access Token in case the VCs are verified correctly.

Figure 39: Access Token in JWT Format and decoded

Another component that has been added to the implementation is the PEP-Proxy. This component is also deployed in every cVO/VO of the demo and acts as an entry point for them. This component offers the same APIs (including Verifier/Holder APIs) as those offered by the (c)VOs on which the PEP-Proxy is deployed, so the PEP-Proxy just redirects the requests to the corresponding component, thus ensuring that the (c)VOs and its deployed Holder/Verifier are not exposed and can only be accessed through the PEP-Proxy. This component is also responsible for verifying the Access Token when receiving a request to a resource with the Access Token included in the 'x-auth-token' header.

The main objective of the OIDC4VP is to obtain an Access Token that gives the (c)VOs/App the permission to get specific resources from specific (c)VOs. This process to get the Access Token can be divided into four steps. For example, for the cVO (acts as the 'initiator') trying to get the resource '/temperature' from VO1 (acts as the 'responder'), the OIDC4VP flow to obtain the Access Token (shown in Figure 40) would be as follows:

1. The cVO sends an HTTP Post to his Holder including a JSON that contains information about what is the request the cVO wants to make. Specifically, the JSON contains the following data:
    a. **Device**: VO1's PEP-Proxy URL (PEP-Proxy IP + Port).
    b. **Method**: HTTP Method the cVO wants to perform on the resource. This info is included because maybe the cVO can have permission to make a GET on the resource but not to make a DELETE.
    c. **Resource**: VO1's resource that the cVO wants to get.
    d. **Requester**: cVO's IP. This info will be included in the Access Token and will help the PEP-Proxy verify if the cVO that is sending the Access Token is the same that requested it.
2. The Holder starts the process to obtain the Access Token. First, the Holder parses the JSON received from the cVO and obtains all the info it needs. Then, with the info obtained, the Holder can now make an HTTP GET to the '/obtainVP' API of VO1's PEP-Proxy to get the SIOP Request (step 2.0). The PEP-Proxy forwards/redirect the request from the cVO's Holder to the VO1 Verifier's '/obtainVP' API (step 2.1), which generates the SIOP Request and sends it to the cVO's Holder. This SIOP Request contains relevant info, such as what VCs the cVO must present to the VO1's Verifier or the redirectURI where the Holder need to send the VP to be verified.1

3. The Holder receives and parses the SIOP Request, selects the correspondent Verifiable Credential/s and, if the Holder have all VCs requested by the VO1's Verifier, the Holder creates the VP and the ID_Token. Once both the ID_Token and the VP_Token are created, the Holder makes an HTTP POST to the redirectURI included in the parsed SIOP Request (step 3.0). The VO1's PEP-Proxy forwards the HTTP POST sent by the Holder to the VO1's Verifier. The Verifier validates and verify the Verifiable Presentation and, if the validation is correct, the Verifier generates the Access Token and sends it to the cVO's Holder (step 3.1).
4. The cVO stores the Access Token and the cVO make an HTTP GET to the VO1's PEP-Proxy '/temperature' API including the Access Token in th' 'x-auth-token' header (step 4.0). The PEP-Proxy validates the Access Token and, if the validation is successful, the PEP-Proxy redirects the request to VO1's '/temperature' API.

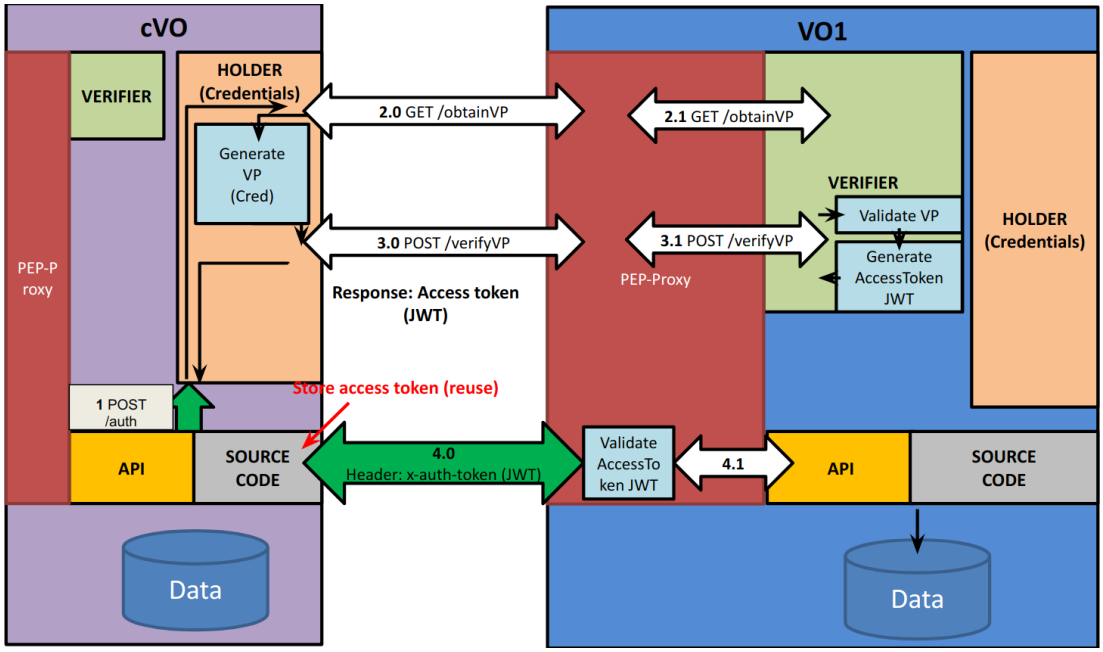

Figure 40: OIDC4VP flow between a cVO and a VO

Future Work

For the upcoming months, the development will focus on implementing and enhancing key components of the security architecture, including the Issuer, an XACML Framework and a Distributed Ledger Technology.

The **Issuer** functionality is almost defined, but the deployment of this component is still something that needs to be reviewed and decisions need to be taken.

The implementation of an **XACML Framework** will enhance our ability to manage access policies in a more flexible and granular manner. This implementation will empower administrators to establish specific and detailed rules regarding who has access to which resources, thereby strengthening access control measures.

Furthermore, the integration of a **Distributed Ledger Technology** will bring a new level of transparency, security, and traceability to our platform. This decentralized ledger system will enhance the overall reliability of our services, fostering trust among users and stakeholders. As we move forward, these components will collectively contribute to a more robust and feature-rich platform,

elevating the user experience and ensuring the highest standards of security and accessibility. This DLT could store DIDs, schemas or domain access control policies.

Also, the implementation made in this first period is not a final version of the components implemented, so during the following months, the components and functionality explained above will be re-studied and reviewed to try to improve it as much as possible.

5.4 Multi-cluster AI-assisted deployment

Towards the evaluation of the presented technologies in the context of this deliverable, the methodology described in Section 3.5 was instantiated providing a first view of what a closed-loop system should look like.

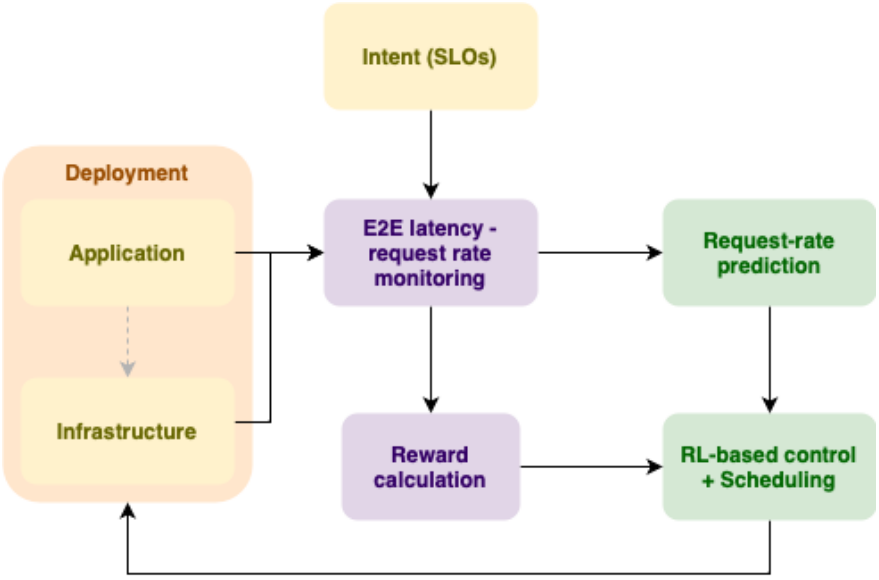

Figure 41: Initial testing set-up

Two Kubernetes clusters were deployed, i.e., to simulate the edge and cloud infrastructure and the network communication between the two clusters was achieved by utilizing the LIQO open-source tool that allows the deployment of applications across multiple clusters. For this initial deployment the serverless paradigm was considered, so the OpenFaaS serverless platform was deployed for handling the deployment of the serverless functions. Regarding monitoring, Prometheus enabled the collection of metrics such as the end-to-end latency of the application and the incoming rate of requests to the function chain. The OpenAI Gym library was used to model the system environment and create an interface so that our agents can interact with it, triggering scaling actions and receiving a reward in return. The interfaces that are used to interact with the testbed are those provided by the Kubernetes API.

For functionally testing the deployed system, a function chain with specific SLOs was used whose high-level objective was "energy efficient" and "latency sensitive". Thus, for the calculation of the reward, QoS violations are considered (if a request takes more time than the maximum value for the latency defined in the selected SLA, we assume a QoS violation) along with the amount of the utilized resources. For the experiments, the RL-driven Autoscaler and the Scheduling Optimizer are invoked at a specified interval. To create an HTTP load for our application, the Vegeta tool was used and for the workload prediction mechanism, an ARIMA forecasting model. Figure 41 descries this initial testing set up.

5.5 Implementation roadmap

The Implementation Roadmap delineates the sequential stages and strategies that NEPHELE will undertake to realize the development, integration, and validation of the Synergetic Meta-Orchestration Framework. This roadmap encompasses the following phases:

- **Requirements Expression**: Project requirements have been systematically articulated and documented comprehensively in Deliverable 2.1. Key functionalities and performance expectations in the form of KPIs are also continually identified and defined as foundational guidelines for the activities of this WP4.
- **Synergetic Meta-Orchestration Framework Architectural Design**: Building upon the requirements, we delved into the design phase, conceptualizing the architecture and specifications of the synergetic meta-orchestration framework. This work was initiated as part of Deliverable 2.1 and is currently ongoing as part of Deliverable 2.2.
- **Open-Source Tooling Selection**: Dedicated efforts has been made to select and open-source tools to be integrated into the framework, addressing critical aspects such as workload placement, inter-cluster communication, data high availability, monitoring, adaptability, optimization, and security and trust management.
- **Testbed Preparation**: Work has been initiated to focus on preparing the testbed and the development environment, aligning it closely with the framework's specifications and requirements to facilitate comprehensive testing procedures.
- **Integration of Framework Components and Interfaces**: The integration phase will be centred on incorporating and unifying the components and interfaces of the synergetic meta-orchestration framework, ensuring seamless functionality and interaction. This work will be carried out as part of joint efforts in WP4 and WP5.
- **Synergetic Meta-Orchestration Framework Testing**: A comprehensive testing methodology will span multiple phases, encompassing unit tests, component tests, integration tests, and system (end-to-end) tests to asses' the framework's functionalities and adherence to the design. This work will also be carried out as part of joint efforts in WP4 and WP5.
- **Validation and User Acceptance**: The solution will undergo rigorous validation and user acceptance procedures to ensure alignment with predefined criteria and user expectations.

6 Conclusions

This deliverable reports a detailed overview of the Synergetic Meta-Orchestration Framework, the Development Environment, and the Repository in the context of the NEPHELE project and for the purpose of deployment and hosting of Hyper-Distributed Applications on the Cloud-to-Edge-to-IoT computing continuum.

The components involved in the Synergetic Meta-Orchestration from an architectural point of view were described and detailed. In addition, the Synergetic Meta-Orchestration approach was exposed in detail with a clear view of the main functionalities that are considered as part of the Orchestration Framework, including the following aspects:

- Workload placement
- Inter-cluster communication
- Storage and data high availability
- Monitoring, Adaptability, and Optimization
- Security and trust management

Solutions for the above functions and open-source tools have been studied extensively and results, analyses and syntheses have been reported in this deliverable. The hyper-distributed application development environment approach has also been dissected and specifications of the Registry were expressed in detail. Plans to incorporate a dual-purpose dashboard and investigate monitoring tools and solutions was also touched on.

Finally, the implementation status of various components was reported in this deliverable with intermediary results and conclusions, and an implementation roadmap was provided, which will serve as a guideline for the next period (up to M24 and beyond), and pave the way for the deployment, testing and validation activities.

The next release of this deliverable will deliver the final specifications of the Synergetic Meta-Orchestration Framework as well as the Hyper-Distributed Application Development Environment and Repository. It will also report on a more tangible technical progress as the focus will shift from conceptual design and definition of specifications, to a more “hands-on” implementation work.

7 References

[1] NEPHELE Deliverable 2.1: Requirements, Use Cases Description and Conceptualization of the NEPHELE Reference Architecture, May 2023

[2] NEPHELE Deliverable 3.1: I–itial Release of VOStack Layers and Intelligence Mechanisms on IoT Devices, December 2023

[3] IETF, Intent-Ba–ed Networking - Concepts and Definitions, Available at: https://datatracker.ietf.org/doc/html/rfc9315

[4] Intent NBI – Definition and Principles, October 2016, ONF TR-5“3

[5] Lenrow, D., "Intent As The Common Interface to Network Resources", Intent Based Network Summit 2015 ONF Boulder: IntentNBI, February 2015.

[6] Kashif Mehmood, Katina Kralevska, David Palma, Intent-driven autonomous network and service management in future cellular networks: A structured literature review, Computer Networks, Volume 220, 2023, 109477, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2022.109477.

[7] 3GPP, Intent Drive Managements, Available at: https://www.3gpp.org/technologies/intent

[8] C. Li, O. Havel, W. Liu, A. Olariu, P. Martinez-Julia, J. Nobre, D. Lopez, Intent Classification, Report, IETF, 2020.

[9] Y. Elkhatib, G. Coulson, G. Tyson, Charting an intent driven netwo[rk], in: 2017 13th International Conference on Network and Service Management, CNSM, 2017, pp. 1–5, http://dx.doi.org/10.23919/CNSM.2017.8255981

[10] NEMO project, Available at: https://wiki.opendaylight.org/display/ODL/NEMO

[11] Huawei, Intent nbi – definition and principles, 2016, Available at: http://wwwfile.huawei.com/media/CNBG/Downloads/Technical%20Topics/Fixed%20Network/Intent%20NBI%20for%20Software%20Defined%20Networkingwhitepaper

[12] T. Zhou, S. Liu, Y. Xia, S. Jiang, Yang data models for intent-based network model, 2015, Available at: https://datatracker.ietf.org/doc/html/draftzhou-netmod-intent-nemo-00

[13] P. Lipton, C. Lauwers, M. Rutkowski, C. Noshpitz, C. Curescu, Tosca simple profile in yaml version 1.3, 2020, Available at: https://docs.oasisopen.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-ProfileYAML-v1.3-os.html

[14] TOSCA Version 2.0, Committee Specification Draft 05, 19 January 2023, Available at: https://docs.oasis-open.org/tosca/TOSCA/v2.0/TOSCA-v2.0.html

[15] Heiko Koziolek, Pablo Rodriguez, Phuoc Sang Nguyen, Nafise Eskandani, Rhab[an] Hark. “TOSCA for Microservice Deployment in Distributed Control SystemsExperiences and Lessons Learned”, 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C). 13-17 March 2023 L’Aquila, Italy, 2023, ISBN 978-1-6654-6459-8

[16] O. Kopp, T. Binz, U. Breitenbucher, and F. Leymann–a modeling tool for tosca-based cloud applications,” in International Conference on Service-Oriented Computing. Springer, 2013, pp. 700–704.

[17] Puccini TOSCA Parser, Available at: https://puccini.cloud/tosca/parser/

[18] OpenTOSCA, Available at: https://www.opentosca.org/

[19] Turrandot, Compose and orchestrate Kubernetes workloads using TOSCA, Available at: https://turandot.puccini.cloud/

[20] Known TOSCA Implementations, Available at: https://github.com/oasis-open/tosca-community-contributions/wiki/Known-TOSCA-Implementations
[21] ETSI GS NFV-SOL 004, "Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; VNF Package and PNFD Archive specification", V3.5.1, 2021.
[22] ETSI GS NFV-SOL 005, "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point", V2.7.1, 2020.
[23] ETSI, ETSI GS NFV-SOL 006 V2.7.1, "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors based on YANG Specification", December 2019.
[24] Redis. https://redis.io/
[25] S. Aydin, C B. Çebi, "Comparison of Choreography vs Orchestration Based Saga Patterns in Microservices" in Proc. of the Internat. Conf. on Electrical, Computer and Energy Tech. (ICECET), July 2022, Prague, Czech Republic.
[26] https://www.matilda-5g.eu/
[27] Gronlier, P., Hierro, J., and S. Steinbuss, "Data Spaces Business Alliance Technical Convergence", 21 April 2023, <https://data-spaces-business-alliance.eu/wp-content/uploads/dlm_uploads/Data-Spaces-Business-Alliance-Technical-Convergence-V2.pdf>
[28] Sporny, M., Guy, A., Sabadello, M., and D. Reed, "Decentralized Identifiers (DIDs) v1.0", 19 July 2022, <https://www.w3.org/TR/did-core/>
[29] Sporny, M., Noble, G., Longley, D., Burnett, D. C., Zundel, B., and K. Den Hartog "Verifiable Credentials Data Model v1.1", 03 March 2022, < https://www.w3.org/TR/vc-data-model/ >
[30] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", 8 November 2014, <httpp://openid.net/specs/openid-connect-core-1_0.html>
[31] Terbu, O., Lodderstedt, T., Yasuda, K., Lemmon, A., and T. Looker, "OpenID for Verifiable Presentations", 20 May 2021, <https://openid.net/specs/openid-4-verifiable-presentations-1_0.html>
[32] Loddersted, T., Yasuda, K., and T. Looker, "OpenID for Verifiable Credential Issuance – draft 12", 26 November 2023, <https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html>
[33] Yasuda, K., Jones, M. B., and T. Lodderstedt, "Self-Issued OpenID Provider V2", 1 January 2023, <https://openid.bitbucket.io/connect/openid-connect-self-issued-v2-1_0.html>
[34] Tailored Learning-based scheduling for Kubernetes-oriented Edge-Cloud System, **DOI:** https://doi.org/10.48550/arXiv.2101.06582
[35] FScaler: Automatic Resource Scaling of Containers in Fog Clusters Using Reinforcement Learning, **DOI**: https://ieeexplore.ieee.org/document/9148401.
[36] Learning to Communicate with Deep Multi-Agent Reinforcement Learning, **DOI**: https://doi.org/10.48550/arXiv.1605.06676
[37] Multi-agent RL for Networked System Control, **DOI:** https://doi.org/10.48550/arXiv.2004.01339
[38] A game-theoretic analysis of networked system control for common-pool resource management using multi-agent reinforcement learning, **DOI**: https://doi.org/10.48550/arXiv.2010.07777
[39] A review of cooperative multi-agent deep reinforcement learning, **DOI:** https://doi.org/10.1007/s10489-022-04105-y
[40] Federated Learning in Mobile Edge Networks: A Comprehensive Survey, **DOI:** https://doi.org/10.1109/COMST.2020.2986024
[41] Communication-Efficient On-Device Machine Learning: Federated Distillation and Augmentation under Non-IID Private Data, **DOI:** https://doi.org/10.48550/arXiv.1811.11479

[42] Autonomy and Intelligence in the Computing Continuum: Challenges, Enablers, and Future Directions for Orchestration, **DOI:** https://doi.org/10.48550/arXiv.2205.01423
[43] Decentralized Collaborative Learning of Personalized Models over Networks, **URL**: https://proceedings.mlr.press/v54/vanhaesebrouck17a.html
[44] From Federated to Fog Learning: Distributed Machine Learning over Heterogeneous Wireless Networks, **DOI:** https://doi.org/10.1109/MCOM.001.2000410
[45] Evolutionary Dynamics of Multi-Agent Learning: A Survey, **DOI**: https://doi.org/10.1613/jair.4818
[46] Mean-field Multi-Agent Reinforcement Learning, DOI: https://doi.org/10.48550/arXiv.1802.05438
[47] The Promising Role of Representation Learning for Distributed Computing Continuum Systems, url: https://ieeexplore.ieee.org/abstract/document/9912640
[48] A Survey on Observability of Distributed Edge & Container-Based Microservices, doi: 10.1109/ACCESS.2022.3193102
[49] Data Fusion of Observability Signals for Assisting Orchestration of Distributed Applications, doi: https://doi.org/10.3390/s22052061
[50] ANNEX 3: Reference of OSM Information Model — Open Source MANO documentation (etsi.org) URL: https://osm.etsi.org/docs/user-guide/latest/11-osm-im.html
[51] CNCF: The Cloud Native Telco | URL: https://www.telecomtv.com/content/etsi-nfv-evolution-event/cncf-the-cloud-native-telco-41268/
[52] ETSI - Linux Foundation and ETSI Further Collaborate to Drive Harmonization Across Open Source and Open Standards. URL: https://www.etsi.org/newsroom/press-releases/2278-linux-foundation-and-etsi-further-collaborate-to-drive-harmoniz%E2%80%93tion-across-open-source-and-open-standards
[53] CNCF launches Cloud Native Network Functions (CNF) Testbed | Cloud Native Computing Foundation. URL: https://www.cncf.io/announcements/2019/02/25/cncf-launches-cloud-native-network-functions-cnf-testbed/
[54] New OCI Artifacts Project - Open Container Initiative. URL: https://opencontainers.org/posts/blog/2019-09-10-new-oci-artifacts-project/
[55] *JSON-LD 1.1: A JSON-based Serialization for Linked Data*. Gregg Kellogg; Manu Sporny; Dave Longley; Markus Lanthaler; Pierre-Antoine Champin; Niklas Lindström. W3C JSON-LD 1.1 Working Group. W3C Working Draft. URL: https://www.w3.org/TR/json-ld11/
[56] *JSON Web Token (JWT).JSON Web Token (JWT)* M. Jones; J. Bradley; N. Sakimura. IETF. May 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7519