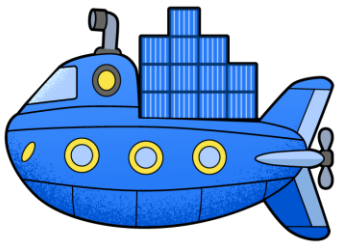

NEMO

D1.2 NEMO meta-architecture, components and benchmarking. Initial version

Document Identification			
Status	Final	**Due Date**	31/08/2023
Version	1.0	**Submission Date**	31/08/2023

Related WP	WP1	**Document Reference**	D1.2
Related Deliverable(s)	D1.1	**Dissemination Level (*)**	PU
Lead Participant	SPACE	**Lead Author**	Nikos Drosos (SPACE)
Contributors	ATOS, ATOS IT, ENG, INTRA, TID, WIND3, SYN, STS, AEGIS, UPM, ICCS, RWTH, TSG, MAG	**Reviewers**	Dimitris Christopoulos, (FHW) Wafa Ben Jaballah (TSG)

Keywords:
Meta-OS, meta-architecture, architecture, verification, validation, multi-cluster orchestration

Document Information

List of Contributors	
Name	Partner
Aitor Alcázar-Fernández	ATOS
Victor Gabillon	TSG
Antonello Corsi	ENG
Andrea Sabatino	ENG
Dimitrios Skias	INTRA
Panagiotis. Karkazis	MAG
Astik Samal	MAG
Nikos Drosos	SPACE
Emmanouil Bakiris	SPACE
Alejandro Muñiz Da Costa	TID
Luis Miguel Contreras	TID
Fabrizio Brasca	WIND3
Gianluca Rizzi	WIND3
Antonis Gonos	ESOFT
Theodore Zahariadis	SYN
Terpsi Velivassaki	SYN
Harry Skianis	SYN
Konstantinos Psychogyios	SYN
Spyros Vantolas	AEGIS
Alberto del Rio	UPM
Dimitris Siakavaras	ICCS
Stefan Lankes	RWTH

Document History			
Version	**Date**	**Change editors**	**Changes**
0.1	08/05/2023	N. Drosos (SPACE)	Table of Contents
0.2	24/05/2023	A. del Rio (UPM), T. Velivassaki (SYN), K. Psychogyios (SYN), D. Skias (INTRA), A. Corsi (ENG), S. Vantolas (AEGIS), S. Lankes (RWTH), D. Siakavaras (ICCS)	Updates on Section 3
0.3	09/06/2023	A. Alcázar-Fernández (ATOS), A. Muñiz and LM. Contreras (TID)	Updates on Section 5
0.4	07/07/2023	N. Drosos (SPACE), Th. Zahariadis (SYN), T. Velivassaki (SYN)	Updates on section 2 and 4
0.5	21/07/2023	P. Karkazis (MAG). A. Samal (MAG)	Updates on Section 6
0.6	31/07/2023	T. Velivassaki (SYN), N. Drosos (SPACE)	Contributions to sections 1, 2 and 7

0.7	02/08/2023	D. Skias (INTRA), A. Gonos (ESOFT), S. Vantolas (AEGIS), N. Drosos (SPACE), T. Velivassaki (SYN)	Updates on Section 5
0.8	04/08/2023	N. Drosos (SPACE)	Document consolidation; Ready for peer review
0.8.1	08/08/2023	D. Christopoulos (FHW)	Peer review
0.8.2	10/08/2023	W. Ben Jaballah (TSG)	Peer review
0.9	30/08/2023	N. Drosos (SPACE), E. Bakiris (SPACE), A. Muniz (TID), D. Siakavaras (ICCS), A. Sabatino (ENG), T. Velivassaki (SYN)	Addressing peer review comments; Document finalization
0.91	30/08/2023	N. Drosos (SPACE)	FINAL VERSION TO BE SUBMITTED
1.0	31/08/2023	Rosana Valle (ATOS)	Format check & submission to CE

Quality Control		
Role	**Who (Partner short name)**	**Approval Date**
Deliverable leader	N. Drosos (SPACE)	31/08/2023
Quality manager	R. Valle Soriano (ATOS)	31/08/2023
Project Coordinator	E. Pere Pages Montanera (ATOS)	31/08/2023

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
AAA	Authentication, Authorization, and Accounting
ABAC	Attribute-Based Access Control
AD	Architecture Description
AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
BDVA	Big Data Value Association
CBAC	Context-Based Access Control
CFDRL	Cybersecure Federated Deep Reinforcement Learning
CI/CD	Continuous Integration & Continuous Delivery/Continuous Deployment
CMDT	Cybersecure Microservices' Digital Twin
CN	Core Network
CNCF	Cloud Native Cloud Foundation
CNI	Container Network Interface
COAP	Constrained Application Protocol
CPO	Charging Point Operator
CPU	Central Processing Unit
DAC	Discretionary Access Control
DAIRO	Data, AI and Robotics
DLT	Distributed Ledger Technology
DN	Distinguished Name
DoA	Description of Action
DVL	Data Virtualization Layer
Dx.y	Deliverable number y belonging to WP x
E2E	End-to-End
EC	European Commission
FI-PPP	Future Internet Public-Private Partnership
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
HIL	Human-in-the-Loop
HLA	High-Level Architecture
HPC	High Performance Computing
HTTP	Hypertext Transfer Protocol
HW	Hardware
IAKM	Infrastructure-assisted Knowledge Management
IAM	Identity and Access Management
IAS	Intent-based API/SDK
IDSA	International Data Spaces Association
IIoT	Industrial Internet of Things

Abbreviation / acronym	Description
IMC	Intent-based Migration Controller
IoT	Internet of Things
JTC	Joint Technical Committee
K8s	Kubernetes
KPI	Key Performance Indicator
LCM	Life-Cycle Manager
LDAP	Lightweight Directory Access Protocol
LSP	Large Scale Pilot
MAC	Mandatory Access Control
MANO	Management and Orchestration
MEC	Multi-access Edge Computing
meta-OS	Meta-Operating System
ML	Machine Learning
mNCC	Meta Network Cluster Controller
MOCA	Monetization and Consensus-based Accountability
MQTT	Message Queuing Telemetry Transport
NFV	Network Function Virtualization
NGIoT	Next-Generation IoT
NGSI	Next Generation Service Interface
NMB	NEMO Message Broker
mRA	meta-Reference Architecture
OS	Operating System
OTP	One-Time Password
PPEF	PRESS & Policy Enforcement Framework
PRESS	Privacy, data pRotection, Ethics, Security & Societal
RAM	Random Access Memory
RAN	Radio Access Network
Raas	Resources as a Service
RBAC	Role-Based Access Control
RL	Reinforcement Learning
ROS	Robot Operating System
RTT	Round Trip Time
RuBAC	Rule-Based Access Control
SDK	Software Development Kit
SDLC	Software Development Life Cycle
SDN	Software Defined Networking
SDO	Standard Development Organization
SD-WAN	Software-Defined Wide Area Network
SEE	Secure Execution Environment
SLA	Service Level Agreement
SLO	Service Level Objective
SSD	Solid-State Drive
SSO	Single Sign-On

Abbreviation / acronym	Description
SUT	System Under Test
TEE	Trusted Execution Environment
TSN	Time Sensitive Networks
UBAC	User-Based Access Control
UC	Use Case
V&V	Validation & Verification
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VR	Virtual Reality
WASM	webassembly
WP	Work Package
YAML	Yet Another Markup Language

Executive Summary

NEMO aims to develop the *meta–Operating System* (metaOS), which will enable multi-cluster and multi-network orchestration of containerized workloads across the Internet of Things (IoT), edge and cloud continuum. As a (meta-)OS, NEMO will be user-centric, facilitating users to develop and deploy on top of NEMO. Moreover, NEMO will enable cloud and infrastructure providers to integrate their computing and networking resources into NEMO's infrastructure.

The present document provides architectural specifications towards achieving this vision. The main outcomes reported in this deliverable include:

- Definition of the methodology to be followed for the NEMO meta-architecture and architecture description, adopting the conceptual model defined in ISO/IEC/IEEE 42010 for architecture descriptions.
- Analysis of notable Reference Architectures which aim at the development of ecosystems in the continuum.
- Presentation of the NEMO Meta-Architecture Framework (MAF), which can be used as a reference for the description of metaOS architectures.
- Specification of the first version of the NEMO architecture, following the proposed MAF. This first version provides specifications for the Network, User, Logical, Functional and Process views of the architecture.
- Presentation of the NEMO Verification and Validation methodology, which aims to guide the relevant activities in WP4.

The information and specifications provided in this deliverable aim to guide the development, integration, validation and pilot activities of the project. Moreover, they may inspire metaOS architects and developers to build applications, services and plugins for the proposed metaOS.

The future work and updates over the architectural specifications are expected in D1.3 "NEMO meta-architecture, components and benchmarking. Final version", due on M24.

1 Introduction

The future of the digital world is modular and containerized. As digitalization penetrates with increasing rates in our personal and professional activities, myriads of software pieces arise to complete the puzzle of our digital well-being, supporting use cases hard to even imagine in the recent past and which demand their housing in the digital grounds of the connected world. As technology evolves, newer and more devices spring up, and uninterrupted online presence becomes a necessity. But how to support delivery of any service, anywhere and anytime?

The response to this is challenging, even in the cloud and edge computing era. Cloud provides virtually unlimited resources and high availability is considered a commodity. Inherent cloud features, like elasticity and multitenancy, have allowed production-level delivery of myriads of services, while providing distinct user's personal space (tenant). However, advances in technologies, like Augmented/Virtual/Mixed Reality, as well as advances in image/video resolution (reaching 8K to date) are just examples that have led to the emergence of applications demanding ultra-low latency. In addition, the increasing penetration of Artificial Intelligence (AI) has created stiff requirements on availability and access to large data volumes, raising privacy concerns. Here is where computation and intelligence at the 'edge' came into play and together with advances and enhancements in communication technologies through 5G & 6G embrace private/local computation and networking, which aspire to connect seamlessly to various clouds.

This would possibly not have happened without cloud native applications and services. Microservices' based architectures, i.e. independent, loosely coupled software modules providing small functional pieces together with the cloud and workloads' containerization have leveraged the flexible use of resources while delivering high-QoS (Quality of Service) and high-QoE (Quality of Experience) services since the cloud-only computing era. Those distinct software pieces, the microservices, may run at different points (nodes) across clouds and more recently across edges and even IoT devices. Depending on their design, microservices usually need to communicate in order to form/deliver together a greater, coordinated application logic.

The next step is on services'/microservices' deployment and lifecycle management across the various points of presence of ambient computing, including the cloud, the edge and the IoT planes, in short, continuum. The orchestration of cloud native containers across the plane of dispersed resources, even in hybrid cloud settings, relies on container management, as the one delivered through Kubernetes [1]. Kubernetes, or -in short- K8s, allows managing containerized workloads and services, that facilitates both declarative configuration and automation. That is, through K8s, a single entity may organize the deployment of their services into clusters of nodes, which may be dispersed in remote locations and across infrastructure providers.

K8s has dominated the container world, as regards the cloud and servers, including also edge servers. The challenges that need to be faced now are mainly two. First, the extension of K8s, e.g. through lightweight distributions, in order to embrace lowest capabilities' devices. There are already mature solutions around lightweight K8s, like MicroK8s [2], K3s [3], K0s [4], minikube [5], KubeEdge [6], which, in the general case, apply to devices of as low as 0.5 GB RAM. However, the integration of least capable devices is less mature, with some attractive solutions, such as Akri [7], allowing to easily expose heterogeneous leaf devices (such as IP cameras and USB devices) as resources in a Kubernetes cluster.

The second challenge relates to the seamless execution and coherent orchestration of services and microservices across K8s clusters, integrating nodes across the IoT, edge and cloud continuum, even across administrative domains. We need to allow microservices being able to flexibly use the available resources, while respecting user-defined requirements, PRESS (Privacy, data pRotection, Ethics, Security & Societal) compliance, while guaranteeing seamless *application* delivery. This implies that the microservices composing together the functionality in the context of an application, while each of them may run at any place in the IoT-edge-cloud continuum, must ensure that the application services reach the end users at the appropriate (defined) performance, QoS and QoE levels.

NEMO aims to address this challenge by building the *meta-Operating System* (metaOS), which will enable multi-cluster and multi-network orchestration of containerized workloads across the IoT, edge

and cloud continuum. As a (meta-)OS, NEMO will be user-centric, facilitating users to develop and deploy on top of NEMO. Moreover, NEMO will enable cloud and infrastructure providers to integrate their computing and networking resources into NEMO's infrastructure.
The functional stack vision of NEMO for this kind of metaOS is depicted in Figure 1.

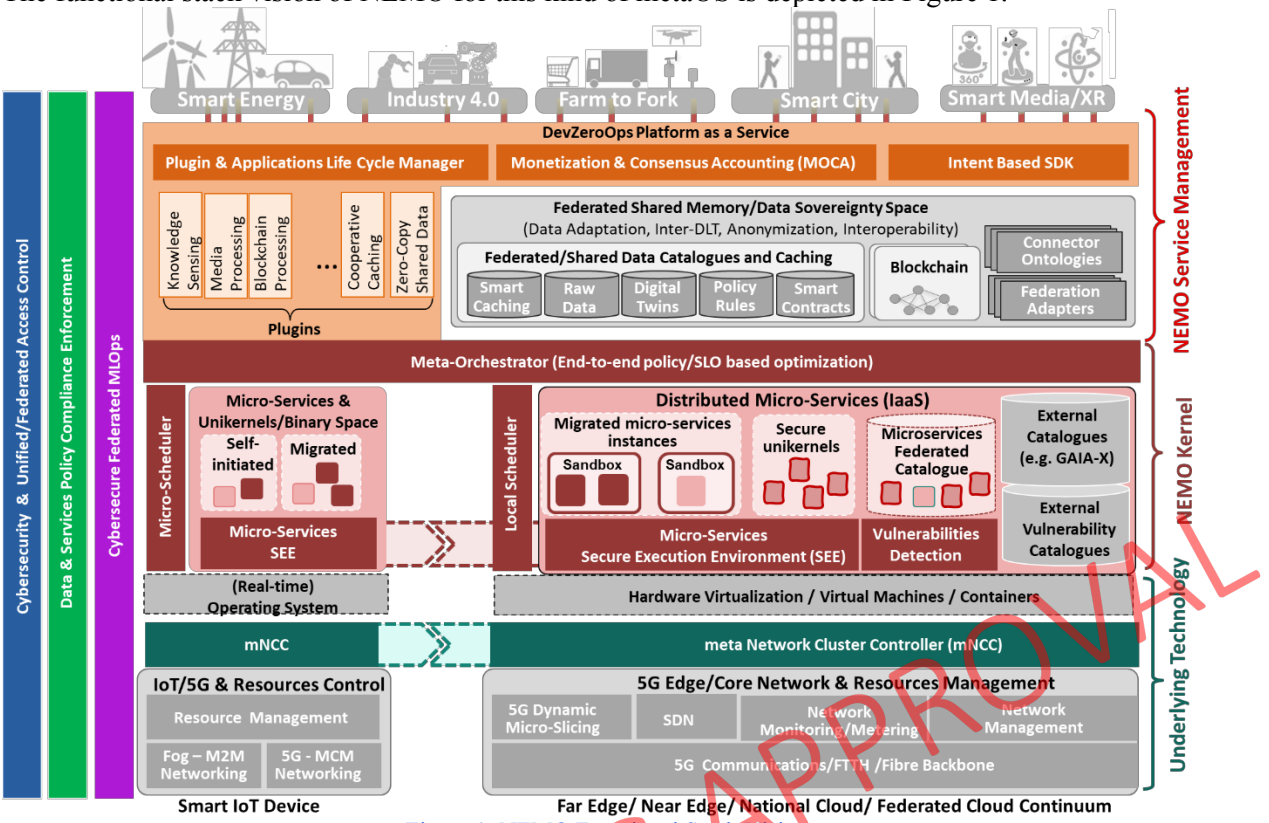

Figure 1: NEMO Functional Stack Vision

This vision foresees three broad functional layers, which abstract metaOS components' role into functional groups, as well as three vertical (cross-cutting) functions, which apply at all metaOS levels. The functional layers of the NEMO vision include:

- Underlying Technology, which includes the infrastructural elements delivering communication and network services, as well as the management of those resources. In the metaOS, the role of this layer includes meta network management, i.e., abstracting and homogenizing resources and network management for the underlying infrastructure elements.
- NEMO Kernel, which –in correspondence to the Linux Kernel [8]– is the main meta-OS component, and the core interface between both the virtual and physical infrastructural resources and the processes, i.e., applications and services running on the metaOS. The NEMO Kernel is envisioned to support registration and scheduling of the metaOS tasks and processes, as well as cater for security and privacy preservation during these operations. As the underlying infrastructure elements can be quite diverse, as well as there can be varying orchestration clusters, the role of the Kernel is to offer meta-orchestration of the available resources, as a meta-control plane on top of the existing container orchestration (K8s) clusters.
- NEMO Service Management, which deals with service management from the end-user perspective following the ZeroOps approach. This layer is envisioned to bring GitOps practices for managing multi-cluster settings, aiming to address the complexity in integrating cloud and edge computing, while supporting accountability and development on top of NEMO. This layer is envisioned to support the plugin architecture for applications and services. With the NEMO Kernel in the role of the core system, plugins are meant to allow providing additional features as plugins to the core, providing extensibility, flexibility, and isolation of application or custom metaOS logic.
- The cross-cutting functions include:

- Cybersecurity and unified/federated access control, which ensures the security of metaOS operations across the metaOS layers, as well as federated access and identity management across the metaOS components.
- Data & Services Policy Compliance Enforcement, which ensures that PRESS rules and GDPR, as well as user-defined rules, are respected across the metaOS layers and components.
- Cybersecure Federated MLOps, which provides inherent integration of Artificial Intelligence (AI) operations and services into the metaOS, yielding AI-based decisions and or controls alongside the metaOS. This function aims to support the complete Machine Learning (ML) lifecycle, e.g., from ML development and training to serving and inference performed within metaOS components, ensuring AI cybersecurity.

Under this vision, the meta-OS meta-architecture and architecture are defined in this document.

1.1 Purpose of the document

The present document is the second deliverable of Work Package (WP) 1 (D1.2) and reports the activities of Task 1.2 "NEMO meta-architecture design and components specifications" and Task 1.3 "Benchmarking definition and GDPR/Ethical compliance". Moreover, D1.2 reports the outcome of activities which contribute to meeting the following WP1 objectives:

- Analyze the challenges, define the requirements and specification of the NEMO meta-Architecture.
- Produce the test reports format, parameter, test points and benchmarking for a unified and reliable outcome.
- Provide continuous technology monitoring on next generation IoT advancements and alignment with NEMO.

This document aims to provide the first version of the NEMO metaOS meta-architecture, which will materialize the NEMO vision. In order to achieve this, we first define a methodology for such a definition, starting by analyzing state of the art flagship Reference Architectures.

Then, we define the NEMO meta-architecture framework (MAF), following the definitions of ISO/IEC/IEEE 42010 for the *architecture framework*, slightly adapted to support the meta-architecture concept. The NEMO MAF includes conventions, principles and practices for the description of a metaOS meta-architecture and may inspire future metaOS architecture designers.

Moreover, MAF is applied for the NEMO metaOS instantiation. The present document covers most parts (viewpoints) of the NEMO MAF, while there are references for the ones planned in other NEMO activities.

Last, but not least, the present document includes an introduction to the NEMO Validation & Verification (V&V) benchmarking framework, guiding the V&V activities within WP4.

1.2 Relation to other project work

The relation of D1.2 with other NEMO activities is tabulated in Table 1.

Table 1: Relation of D1.2 with other NEMO activities.

WP	Relation to D1.2
WP1	D1.2 reports activities conducted within Task 1.2 and 1.3. Moreover, it considered feedback from D1.1, identifying functional and non-functional requirements, used as input for the definition of NEMO components' functionalities in the functional viewpoint of the architecture. Moreover, the use case scenarios' definition in D1.1 complement the NEMO architecture specifications, as they constitute the operational view of the NEMO architecture. D1.2 will feed further activities in WP1, providing the first version of the metaOS architecture, subject to updates in D1.3 "NEMO meta-architecture, components and benchmarking. Final version", due on M24.

WP	Relation to D1.2
WP2	D1.2 aligns to the activities of WP2, reporting the functional specifications for the NEMO components developed in WP2. D1.2 will provide feedback in future WP2 activities, for defining the technical specifications (development view) for WP2 components.
WP3	D1.2 aligns to the activities of WP3, reporting the functional specifications for the NEMO components developed in WP3. D1.2 will provide feedback in future WP3 activities, for defining the technical specifications (development view) for WP3 components.
WP4	D1.2 aligns to the activities of WP4, reporting the functional specifications for the NEMO components developed in WP4. D1.2 will provide feedback in future WP4 activities, for defining the technical specifications (development view) for WP4 components. Moreover, the process view of NEMO provides feedback for the integration activities which will be conducted within WP4. D1.2 provides V&V methodology for the NEMO framework validation, also part of WP4 activities. Such activities will also provide feedback to WP2 for updating D1.2. Last, but not least, the NEMO metaOS architecture reported in D1.2 will be complemented by the deployment view, which will be part of future WP4 activities.
WP5	D1.2 provides architectural specifications, which are useful for the instantiation of the NEMO metaOS in the Living Labs. Thus, D1.2 may be used as a guide to NEMO deployments on the Living Labs.
WP6	D1.2 reports significant outcomes around the metaOS (meta-)architecture specifications, which may be used for impact creation, dissemination, communication and exploitation activities.
WP7	D1.2 provides architectural specifications, which may be useful to Open Call candidates and winners.

1.3 Structure of the document

This document is structured in 7 major chapters.
Chapter 2 presents the methodology followed by NEMO for the definition of the NEMO MAF and architecture.
Chapter 3 analyses state-of-the-art reference architectures.
Chapter 4 presents the NEMO MAF.
Chapter 5 presents the NEMO architecture specifications.
Chapter 6 introduces the NEMO V&V framework.
Chapter 7 draws conclusions and next steps.

2 Methodology for NEMO architecture

NEMO aims to facilitate design and development of higher-level (meta) operating systems for the smart Internet of Things with strong computing capacity at the smart device, system and edge-level, embedded in a compute continuum from IoT-to-edge-to-cloud. The NEMO meta-OS would then provide what expected by an operating system, such as hardware abstraction, device control, service orchestration and management, as well as meta-OS functionalities, allowing flexible and seamless communication and management of services across nodes, whether they belong to cloud, edge or the IoT side.

2.1 Defining the purpose for the meta-OS meta-architecture

The NEMO meta-architecture is conceived as a Reference Architecture on top of existing reference architectures, which aims to provide guidance on evolving or creating new meta-OS architectures.
Reference Architectures capture knowledge from existing architectures. Based on an elaboration of mission, vision, strategy, and on customer needs, the Reference Architecture is transformed into an architecture that provides guidance to multiple organizations that evolve or create new architectures.
Reference Architectures should address technical aspects, business needs, and context. The aim and positioning of the Reference architecture in the technological and business landscape is depicted in Figure 2.

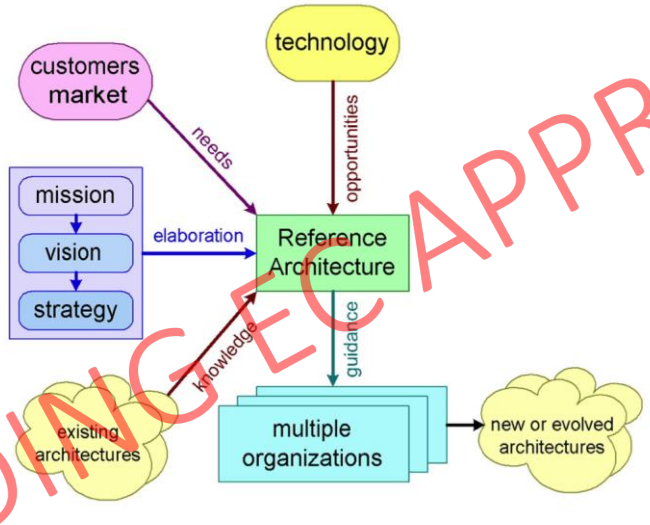

Figure 2: Positioning of the Reference Architecture in the technological and business landscape [9]

2.2 Standardization Landscape around (meta-) Architecture Descriptions

ISO/IEC/IEEE 42010:2011 “Systems and software engineering — Architecture description” [10] addresses the creation, analysis and sustainment of architectures of systems through the use of architecture descriptions (AD). The standard, which has been revised by ISO/IEC/IEEE 42010:2022 [11], has shifted its focus from the ‘system’ to the ‘entity’, including software, systems, enterprises, systems of systems, families of systems, products (goods or services), product lines, service lines, technologies and business domains. It provides the requirements for describing an entity’s architecture via a set of architecture views and architecture view components (‘models’ in the first edition). This set is governed by architecture viewpoints and model-kinds, respectively. The second edition introduces ‘Stakeholder Perspectives’ as a means to group ‘Concerns’ and therefore to organize ‘Viewpoints’ framing those ‘Concerns’. This edition also introduces ‘Architecture Aspects’ as characteristics of the entity of interest that are reflected in Architecture Views.
The standard is domain-neutral and is aimed to be used as the primary reference for specific Architecture Descriptions, allowing software and system architects to communicate in a common language.

Figure 3 presents the conceptual model – or "meta model" – of the Architecture Description in terms of a UML class diagram, as revised in the 2nd edition of the standard. The updates are depicted in purple in the figure.

ISO and IEC Joint Technical Committee (JTC 1) for information technology, which is a consensus-based, voluntary international standards group, has identified Meta Reference Architecture and Reference Architecture for Systems Integration as key components of the work program and has established an Advisory Group (AG) on this topic, namely AG 8 "Meta Reference Architecture and Reference Architecture for Systems Integration". Meta Reference Architecture and Reference Architecture for Systems Integration needs to provide the highest level of abstraction for multiple horizontal business domains under a systems-of-systems view, and Meta Reference Architecture and Reference Architecture should allow business value assessments to select among potential alternative models and or scenarios. JTC 1 AG 8 aim is to standardize architecture practices across JTC 1. The group has adopted ISO/IEC/IEEE 42010 as the primary point of reference for the Meta-Reference Architecture standards its members are developing [12]. The resulting document entitled mRA (for meta reference architecture) describes a canvas with several viewpoints and model-kinds to provide a standardised way for architecting reference architectures. It also leverages the concept of patterns, or reusable artefacts that can be used in the construction of architectures.

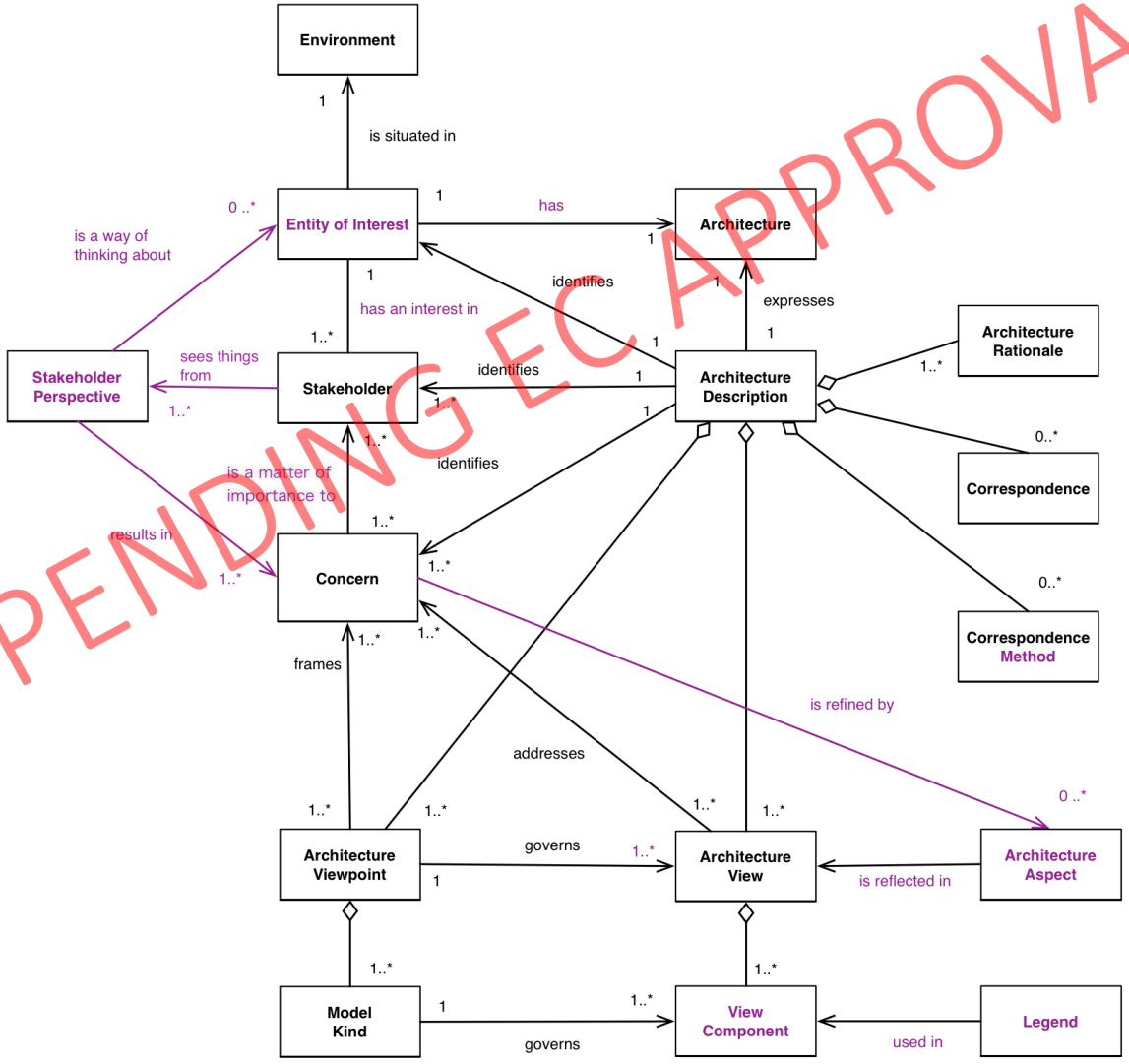

Figure 3: Conceptual model of Entity's Architecture Description in ISO/IEC/IEEE 42010:2022 [13]

2.3 The NEMO meta-Architecture approach

The NEMO meta-architecture definition has been derived, according to the following steps.

- Define architecture objectives.
- Review other architectures, styles and patterns and gather lessons from past experience.
- State architecture principles
- Decide on concepts and mechanisms to ensure architectural integrity and consistency.

3 Reference Architectures for the IoT, Edge and Cloud Continuum

3.1 GAIAX

Gaia-X [14] is a project that aims to create a federated and secure data infrastructure for Europe. Spearheaded by the European Union and various key industrial players, Gaia-X seeks to establish a framework that promotes data sovereignty, interoperability, and transparency. The primary goal of Gaia-X is to build a trusted ecosystem where businesses, organizations, and individuals can securely store, share, and utilize data in a standardized manner. By fostering data mobility and enabling secure data exchange, Gaia-X aims to empower European businesses to compete globally while maintaining control over their data. With its emphasis on privacy, security, and compliance with European data protection regulations, Gaia-X envisions a future where data is harnessed as a strategic asset for innovation, economic growth, and societal benefits.

As of 2023, the Gaia-X Association for Data and Cloud AISBL counts more than 340 members and three working groups. The backbone of the ecosystem are the Gaia-X hubs, which are designed to enable seamless interoperability between different data infrastructures, allowing businesses and organizations to leverage diverse data sources while maintaining control over their data assets. It should be noted, that access to the data hubs or a technical specification on data access is currently not publicly available.

3.2 IDSA

The International Data Spaces Association (IDSA) [15] is a coalition of more than 140 member companies, a non-profit organization, that share a vision of a world where all companies self-determine usage rules and realize the full value of their data in secure, trusted, equal partnerships. The goal of IDSA is a global standard for international data spaces (IDS) and interfaces, as well as fostering the related technologies and business models that will drive the data economy of the future across industries. The International Data Spaces initiative proposes a Reference Architecture Model (RAD) for data sovereignty and related aspects, including requirements for secure and trusted data exchange in business ecosystems, aiming to establish an international standard.

In compliance with common system architecture models and standards (e.g., ISO 42010, 4+1 view model), the RAD uses a five-layer structure expressing various stakeholders' concerns and viewpoints at different levels of granularity. The general structure of the Reference Architecture Model is illustrated in Figure 4.

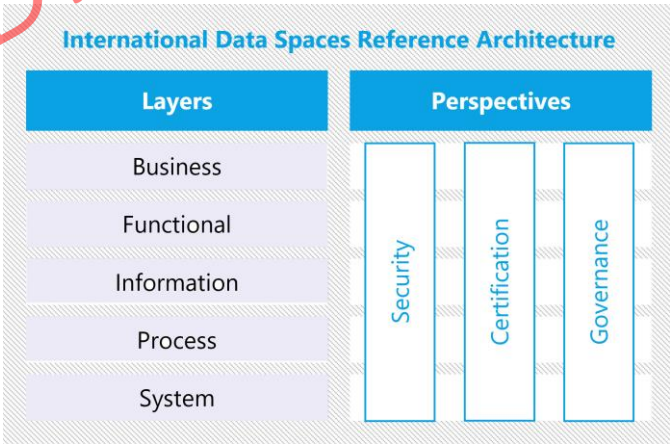

Figure 4: IDS Reference Architecture Model

The model consists of five layers:
The **Business Layer** specifies and categorizes the different roles, which the participants of the International Data Spaces can assume, and it specifies the main activities and interactions connected with each of these roles (Figure 5).

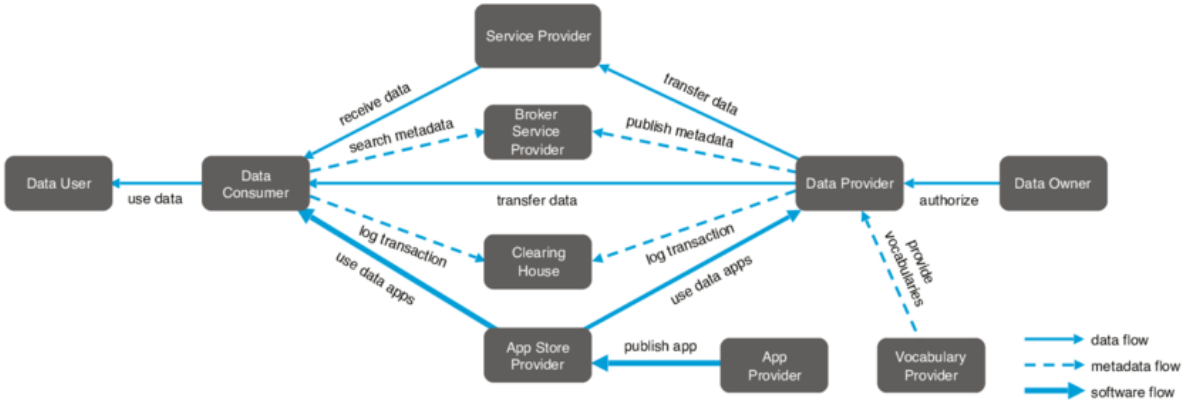

Figure 5: Roles and interactions in the Industrial Data Space

The Business Layer can be used to verify the technical architecture of the International Data Spaces. In this sense, the Business Layer specifies the requirements to be addressed by the Functional Layer.
The **Functional Layer** defines the functional requirements of the International Data Spaces, and the concrete features to be derived from these. A summary of functional requirements is depicted in Figure 6.

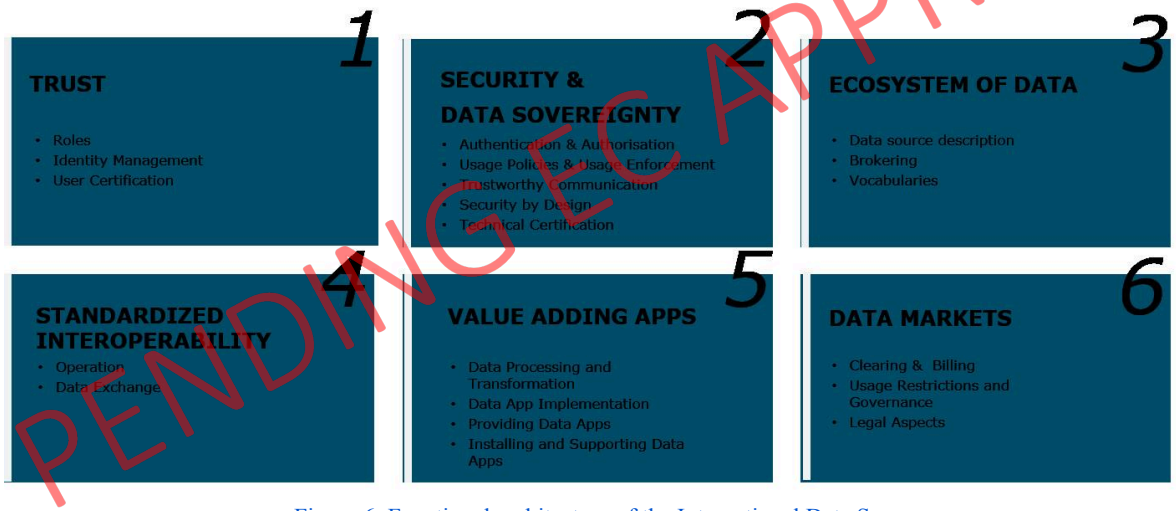

Figure 6: Functional architecture of the International Data Spaces

The **Process Layer** specifies the interactions taking place between the different components of the International Data Spaces and thus provides a dynamic view of the Reference Architecture Model. The layer contains three basic processes: 1 – Onboarding; 2 - Exchanging data; 3 - Publishing and using Data Apps.
The **Information Layer** defines a conceptual model which makes use of linked-data principles for describing both the static and the dynamic aspects of the International Data Space's constituents. The Information Model has been specified at three levels of formalization. Each level corresponds to a digital representation, ranging from this high-level, conceptual document down to the level of operational code, as depicted in Figure 7. Every representation depicts the complete Information Model in its particular way.

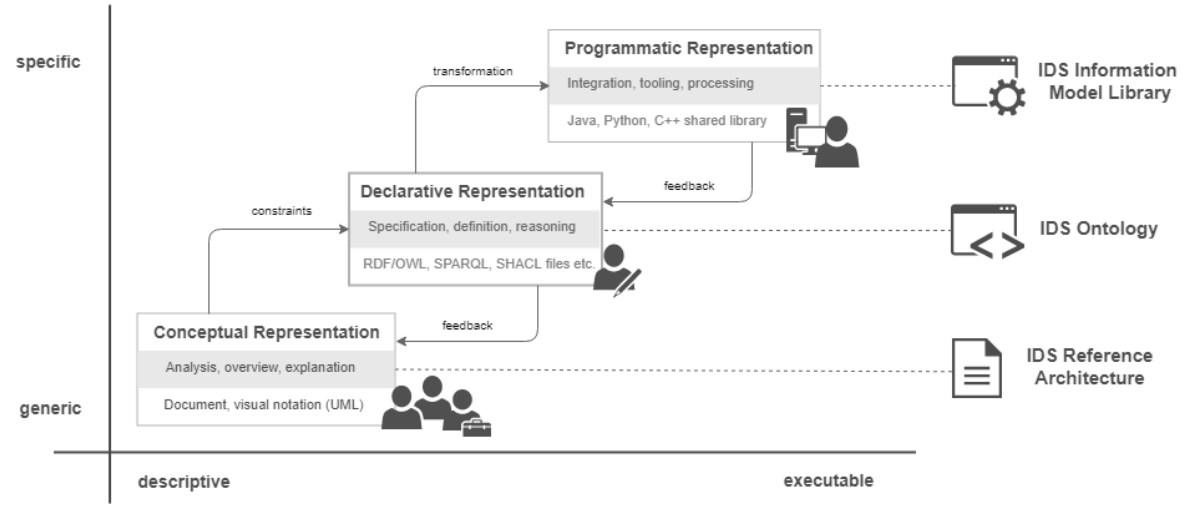

Figure 7: Representations of the Information Model

The **System Layer** is concerned with the decomposition of the logical software components, considering aspects such as integration, configuration, deployment, and extensibility of these components.

From the requirements identified on the Functional Layer, three major technical components are derived:

- the Connector
- the Broker
- the App Store.

How these components interact with each other is depicted in Figure 8. The components are supported by four additional components:

- the Identity Provider
- the Vocabulary Hub
- the Update Repository

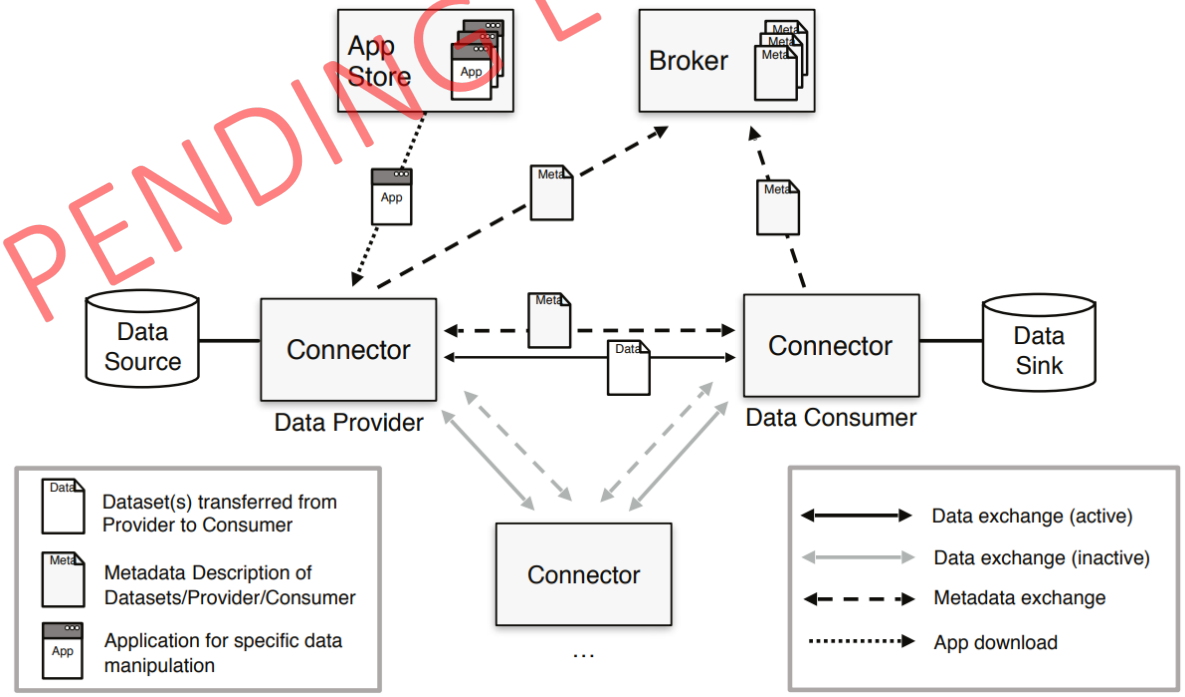

Figure 8: Interactions between components of the functional layer

3.3 BDVA / DAIRO

The Big Data Value Association [16] has been established in 2014 as the private counterpart of the European Commission in the Big Data Value Public Private Partnership. Since then, the Association has served the community within and outside the scope of the Partnership and has been instrumental in developing research and innovation agendas and roadmaps, guidelines for industry and policy makers, and in creating a forum for knowledge sharing and discussions on Big Data, Data Value and Data-driven AI at the EU level.
In 2020 and taking into account the end of the 2014-2020 Multi Annual Financial Framework and the advent of the post 2020 European Commission's programmes (i.e. Horizon Europe and Digital Europe), BDVA members decided to strengthen the Association by giving it a new mandate, a new name and by expanding its scope and breadth of activities. In 2021, BDVA thus became DAIRO. DAIRO stands for Data, AI and Robotics (DAIRO). This new name testifies the ambition of the Association to closely collaborate with other communities in order to jointly engage at the intersection of the key disciplines of Data, AI and Robotics.

The Big Data Value (BDV) Reference Model is a reference framework defined by the European Big Data Value Association (BDVA) in their Strategic Research and Innovation Agenda (SRIA) that describes logical components of a generic big data system. The BDV Reference Model is composed of horizontal and vertical concerns. Horizontal blocks refer to the data processing value chain, from data acquisition to data visualization. Whilst, vertical blocks address cross-cutting issues, which may affect all the horizontal concerns.

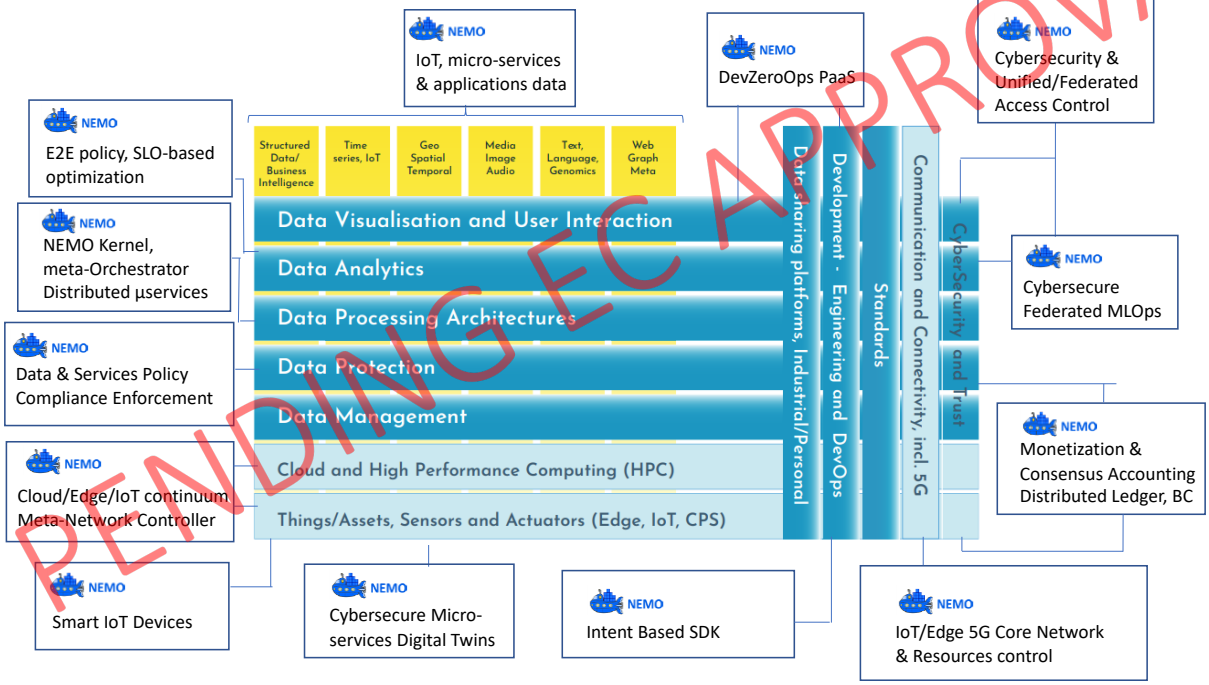

Figure 9: NEMO mapping on the BDV Reference Architecture

Although BDV Reference Model has no ambition to serve as a technical reference structure, the presented model is compatible with NEMO. As illustrated in Figure 9, NEMO functional elements can be easily mapped on the BDV RM. More specifically, the alignment of NEMO meta-OS towards the BDV RA's Horizontal and Vertical concerns is listed below:

- Data Visualization and User Interaction is also promoted in NEMO especially through DevZeroOps services that aim to provide to the NEMO user an intent-based SDK/API in a sense of an interactive interface.

- Data Processing is driven by the NEMO Kernel and the NEMO meta-Orchestrator, which enforce optimization and scalability techniques to the deployed micro-services.
- Data Analytics in NEMO feed the Cybersecure Federated Deep Reinforcement Learning (CF-DRL) and the PRESS framework providing for the Service Level Objective (SLO)-based optimization of the NEMO micro-services.
- Data Protection in NEMO is realized through Data & Services Policy Compliance Enforcement tools such as PRESS. PRESS in NEMO includes privacy protection and anonymization monitoring mechanism facilitating data protection in the meta-OS.
- Data Management techniques in NEMO are provided through the Monetization and Consensus-based Accountability (MOCA) where Distributed Ledger Technology (DLT) technology verifies data integrity.
- Cloud and high-performance computing in NEMO is facilitated on one hand by the NEMO Kernel layer which optimally orchestrates the deployed NEMO services and on the other, through the meta-Network Cluster Controller which facilitates the transparency of the underlying infrastructure that covers the complete IoT/Edge/Cloud continuum.
- Smart IoT and Edge (and cloud) devices and applications are the main source of data for NEMO.
- Development – Engineering and DevOps is aligned with NEMO's DevZeroOps Platform as a Service framework which aims to provide to the user DevOps automation at the highest degree.
- Communication and connectivity in NEMO are based on the underlying infrastructure that covers the whole IoT/Edge/Cloud continuum and is orchestrated by the meta-Network Cluster Controller. In addition, 5G core network and resource control mechanisms are also part of the NEMO meta-OS.
- Finally, Cybersecurity and Trust in NEMO is addressed by various components namely, PRESS framework and CF-DRL. In NEMO's reference architecture, Cybersecurity and Unified Federated Access Control is a vertical functionality as well. This means that is inherently incorporated in all of the horizontal layers of NEMO meta-OS architecture.

3.4 Open DEI

The digital transformation strategy of the European Union has, among others, a particular priority: the creation of common data platforms based on a unified architecture and an established standard. As part of the Horizon 2020 programme, the OpenDEI project [17] focused on "Platforms and Pilots" to support the implementation of next generation digital platforms in four basic industrial domains, namely Manufacturing, Agriculture, Energy and Healthcare.

The project's aim has been to enable a unified data platform, to create large scale pilots and contribute to a digital maturity model, to build a data ecosystem and to strive for standardisation.

The four domains on which Open DEI focuses are very closely related to the cloud-to-edge-to-IoT continuum since in all of them there is extensive use of sensors/drones/robots and/or other IoT devices which generate a large amount of data which needs to be stored and processed in different levels of the cloud-to-edge-to-IoT continuum.

Open DEI suggests a Reference Architecture Framework (RAF) for integrated data-driven services for Digital Transformation pathways, to guide their planning, development, operation and maintenance by adopting organizations. Open DEI RAF is depicted in Figure 10 and provides a modular conceptual model, which comprises loosely coupled service components interconnected through a shared common data infrastructure.

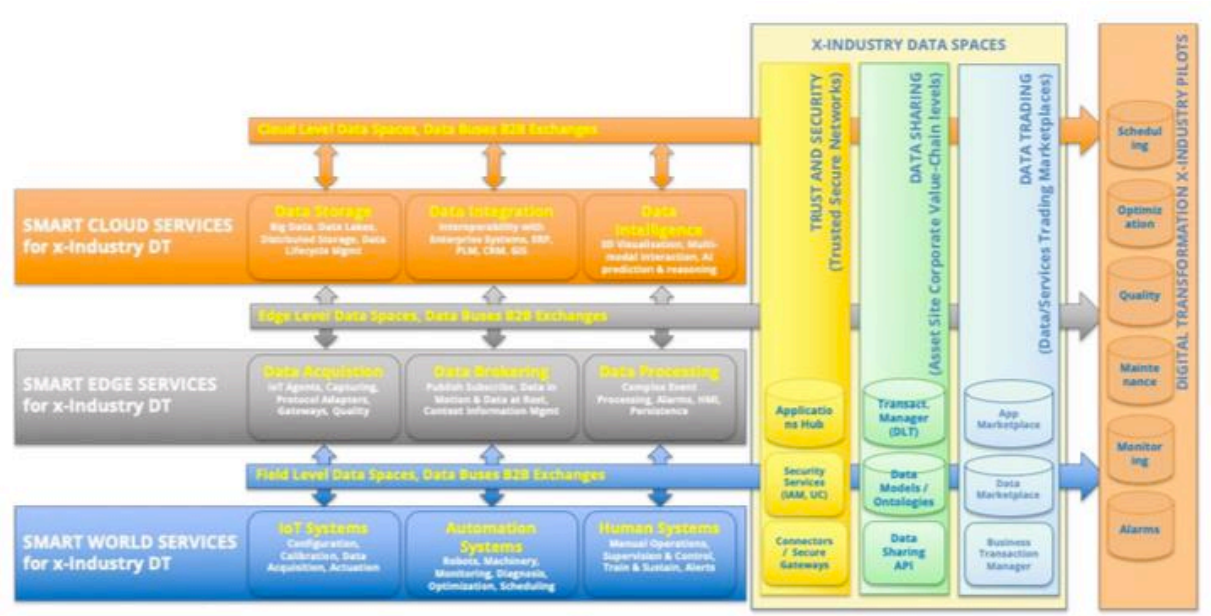

Figure 10: Open DEI Reference Architecture Framework

Regarding the connection with NEMO, Open DEI RAF considers distinct services in the IoT, edge and cloud continuum, which could be flexibly orchestrated by the NEMO meta-OS. It also provides a reference for integrating data spaces concept into the meta-OS.

3.5 AIOTI

The Alliance for IoT and Edge Computing Innovation (AIOTI) [18] aims to lead, promote, bridge and collaborate in IoT and Edge Computing and other converging technologies research and innovation, standardisation and ecosystem building, providing IoT and Edge Computing deployment for European businesses creating benefits for European society.

Within the AIOTI Standardisation Working Group, a High-Level Architecture (HLA) for IoT has been proposed, which aims to address the challenges of architectural convergence and interoperability in the IoT domain. It recognizes the need to align with existing standard development organizations (SDOs), alliances, and consortia to promote a unified approach. The HLA intends to provide a foundation for standardization activities and accommodate the requirements of AIOTI Large Scale Pilots.

The main objectives of the HLA proposal are:

- Converging Architectures: The proposal seeks to align with existing efforts from SDOs, alliances, and open-source projects to promote convergence and interoperability across different IoT architectures.
- Large Scale Pilots (LSP): The HLA aims to serve as a framework for AIOTI Large Scale Pilots, enabling the incorporation of feedback from working groups involved in pilot projects.
- Incremental Development: The proposal adopts an incremental approach, avoiding duplication of existing standards and projects, allowing flexibility, and supporting evolution over time.

The HLA proposal leverages the ISO/IEC/IEEE 42010 standard, which provides guidelines for describing architectures. This standard serves as a reference for capturing and organizing architecture descriptions. The proposal focuses on two key models: the Domain Model and the Functional Model.

The Domain Model is derived from the IoT-A (Internet of Things Architecture) Domain Model. It captures the main concepts and relationships within the IoT domain. The model highlights the interaction between a User and a physical entity (Thing), mediated by an IoT Service and an IoT Device. Emphasis is placed on semantic interoperability and the inclusion of metadata to consistently describe things.

The Functional Model defines the functions and interfaces within the IoT domain. It is structured into three layers: Application, IoT, and Network. Each layer represents a set of modules that provide specific services. The Application layer focuses on user-application interaction, the IoT layer handles device-service interaction, and the Network layer facilitates network connectivity and communication between devices and services.
In addition, the HLA emphasizes the digital representation of physical things as IoT Entities. These entities enable services such as discovery, actuation, and measurement. The proposal acknowledges that multiple representations of a physical thing can exist based on user needs, and these representations can coexist within an IoT system.
Security and management are critical aspects of IoT systems. The proposal suggests that security and management functionalities should be intrinsic to interface specifications. It highlights authentication, authorization, and encryption as essential security features. Additionally, the proposal addresses various management aspects, including device and gateway management, infrastructure management, data life cycle management, digital rights management, and compliance management.
Identifiers play a crucial role in identifying components within IoT systems. The proposal categorizes identifiers into Thing, Application & Service, Communication, User, Data, Location, and Protocol identifiers. It recognizes the existence of diverse identification schemes and suggests that IoT applications should accommodate different schemes based on specific requirements and contexts.
In conclusion, the proposal for an HLA for IoT within the AIOTI WG Standardisation presents a comprehensive framework for achieving architectural convergence and interoperability. It builds upon the ISO/IEC/IEEE 42010 standard, emphasizes the Domain Model and Functional Model, addresses security and management considerations, and provides guidance on the use of identifiers within IoT systems. The proposal aims to collaborate with existing standards bodies, align with ongoing projects, and support the development of AIOTI Large Scale Pilots.

3.5.1 Functional model

The AIOTI functional model provides a framework for describing the functions and interfaces within an IoT system. It emphasizes that the functions described in the model do not dictate any specific implementation or deployment approach. In other words, the model does not assume that each function must correspond to a physical entity in an operational setup. Instead, it allows for the grouping of multiple functions within a single physical equipment in practical implementations.

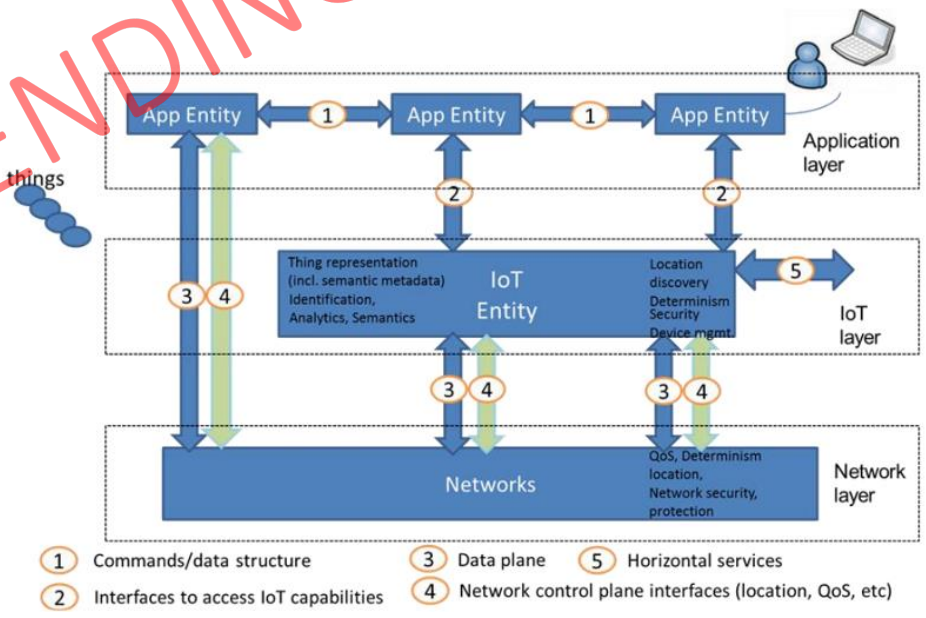

Figure 11: AIOTI HLA functional model

The model is depicted in Figure 11, referred to as the "AIOTI HLA functional model." It presents a high-level overview of the various functions involved in an IoT system. Let's delve into the functions depicted in the figure:

- App Entity: An App Entity represents an entity in the application layer that implements the logic of IoT applications. It can exist within devices, gateways, or servers, and there is no predetermined assumption of a centralized approach. The App Entity encapsulates the specific application functionalities and can be tailored to different use cases. For example, it could represent a fleet tracking application entity or a remote blood sugar monitoring application entity.
- IoT Entity: An IoT Entity represents an entity in the IoT layer that exposes IoT functions either to App Entities via Interface 2 or to other IoT entities via Interface 5. The IoT Entity serves as a bridge between the application layer and the underlying IoT infrastructure. It encompasses a wide range of functions such as data storage, data sharing, subscription and notification mechanisms, firmware upgrades, access right management, location services, analytics, semantic discovery, and more. The IoT Entity utilizes the data plane interfaces of underlying networks (Interface 3) for sending and receiving data. It may also access control plane network services (Interface 4) for tasks like location or device triggering.
- Networks: Networks represent the underlying connectivity infrastructure of the IoT system. These networks can be realized using various technologies such as Personal Area Networks (PAN), Local Area Networks (LAN), Wide Area Networks (WAN), and others. Networks are composed of interconnected administrative network domains, and the Internet Protocol (IP) often serves as the common interconnection mechanism between heterogeneous networks. Depending on the requirements of App Entities, the network may offer best-effort data forwarding or premium services with Quality of Service (QoS) guarantees, including deterministic guarantees for specific flows.

The AIOTI functional model also outlines the interfaces between these functions:

- Interface 1: This interface defines the structure of data exchanged between App Entities. The actual connectivity for data exchange on this interface is provided by the underlying Networks. Examples of data exchanged through Interface 1 include authentication and authorization details, commands, measurements, and more.
- Interface 2: Interface 2 enables access to services exposed by an IoT Entity. App Entities can use this interface to register, subscribe for notifications, consume or expose data, and interact with IoT functions provided by the IoT Entity.
- Interface 3: Interface 3 facilitates the sending and receiving of data across the Networks to other entities within the IoT system. It serves as the data plane interface, ensuring the seamless transfer of information between different components.
- Interface 4: This interface enables the IoT system to request network control plane services. These services could include device triggering (similar to "wake on LAN" in IEEE 802), device location (including subscriptions), establishment of QoS bearers, and deterministic delivery for specific flows.
- Interface 5: Interface 5 enables the exposing and requesting of services between IoT Entities. It allows entities to exchange data and services with each other. For instance, a gateway may use this interface to upload data to a cloud server or retrieve software images of gateways or devices.

The AIOTI HLA is highly relevant to NEMO, as it represents a family of IoT systems which can be part of the meta-OS. The IoT and Network Layer relate to the infrastructural elements of NEMO, that can have monitoring interest for the meta-OS as potential nodes and network management elements. Also, the application layer provides workload that can be hosted and orchestrated across the meta-OS nodes.

3.6 FIWARE

FIWARE [19] has been a flagship project of the Future Internet Public-Private Partnership (FI-PPP) program, a joint action by the European Industry and the European Commission. FIWARE

Community's current mission is to "build an open sustainable ecosystem around public, royalty-free and implementation-driven software platform standards that ease the development of new Smart Applications in multiple sectors". FIWARE realizes this vision via open-source Generic Enablers (GE) or Domain Specific Enablers, which support generic or domain specific functions, respectively.
The Reference Architecture of FIWARE is designed to provide a flexible and scalable framework for developing smart applications and services in the context of the IoT and data-driven solutions. It follows a modular approach, where different components work together to enable the processing, management, and utilization of data in real time.
At the core of the FIWARE architecture is the Context Management layer. It consists of the Context Broker, which acts as a central repository for managing and storing context information. The Context Broker follows the Next Generation Service Interface (NGSI) standard and provides a uniform API for accessing and manipulating context data. It enables real-time updates, retrieval, and querying of context information from various sources.
Building upon the Context Management layer, FIWARE offers various tools and components for Data Processing and Analysis. This includes components for real-time data processing, data fusion, and complex event processing. These components allow for real-time analysis and extraction of valuable insights from the context data. They can perform tasks such as data aggregation, filtering, correlation, and pattern recognition, enabling intelligent decision-making and triggering of actions based on the analyzed data.

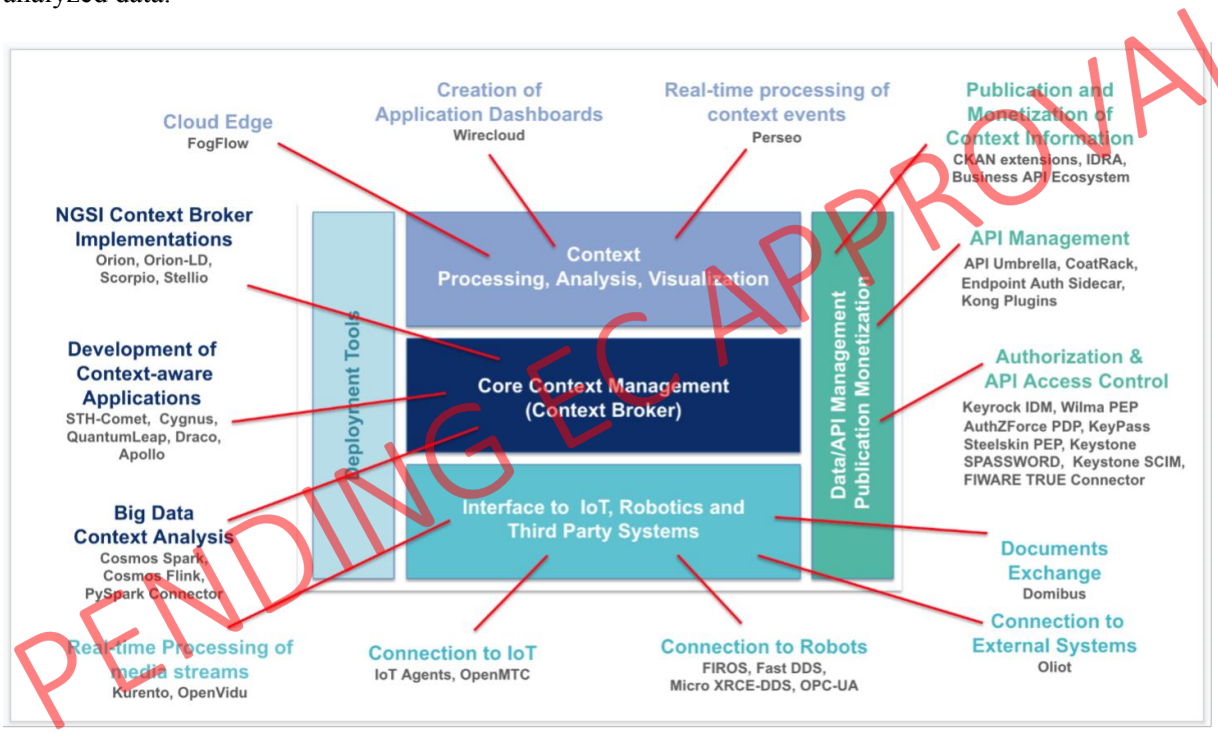

Figure 12: FIWARE Reference Architecture

FIWARE supports the publication and subscription of context information through its Data Publication and Subscription mechanisms. Publishers can push data updates to the Context Broker, while subscribers can express their interest in specific context data and receive notifications when changes occur. This pub/sub model allows applications and services to efficiently consume and react to real-time data changes, facilitating dynamic and responsive behavior.
To facilitate the integration of IoT devices and systems, FIWARE includes IoT Agents. These agents handle device registration, discovery, and communication with the Context Broker. They support various communication protocols, such as Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), Hypertext Transfer Protocol (HTTP), and provide a standardized way to connect and manage IoT devices. IoT Agents ensure interoperability and ease the process of incorporating diverse IoT devices into the FIWARE ecosystem.

Security is a crucial aspect of the FIWARE architecture. It incorporates mechanisms for Security and Access Control to protect data and ensure controlled access. This includes features such as authentication, authorization, and secure communication between components. Access control policies can be defined to restrict data access based on user roles and permissions, ensuring data privacy and confidentiality.
FIWARE provides a set of tools, APIs, and frameworks to enable the development of applications on top of the platform. This Application Enablement layer includes features for data visualization, user interface development, and application logic implementation. It allows developers to build user-friendly interfaces, create innovative applications, and leverage the power of the underlying FIWARE components.
The FIWARE architecture is designed to be cloud-friendly and supports deployment on various cloud platforms. It offers compatibility with popular cloud services and can seamlessly integrate with other cloud-based applications and infrastructure. This allows for scalability and flexibility in deploying FIWARE-based solutions, leveraging the advantages of cloud computing.
The modular and extensible nature of the FIWARE architecture empowers developers to select and combine the components that best suit their application requirements. It promotes interoperability, scalability, and reusability, enabling the development of robust and flexible smart solutions in various domains, including smart cities, agriculture, industry, and transportation.
The FIWARE RA may accommodate a lot of IoT systems, which may run on top of NEMO, implementing the FIWARE conceptual model. In addition, individual elements, such as the Context Broker and IoT agent could be interesting for data collection from IoT nodes participating in the meta-OS.

3.7 H2020 IoT RIA projects

In the following, reference architectures proposed by EU research and innovation actions, aiming to accommodate next-generation IoT systems are presented. The architecture proposals by IoT-NGIN [20], ASSIST-IoT [21], INGENIOUS [22], INTELLIOT [23], VEDLIOT [24] and TERMINET [25] projects are presented.

3.7.1 IoT-NGIN

H2020 IoT-NGIN project has defined a patterns-based meta-architecture [26] [27] for next-generation IoT systems. The meta-architecture is aimed to act as an architectural map for IoT platforms and services, accommodating both existing, legacy IoT architectures, as well as next-generation IoT architectures.
The meta-architecture is designed around four key artifacts, namely the IoT Architectural Pattern Vertical, the Domain Horizontal, the Quality Vertical, and the Element View. The Elements view of the meta-architecture view is depicted in Figure 13.

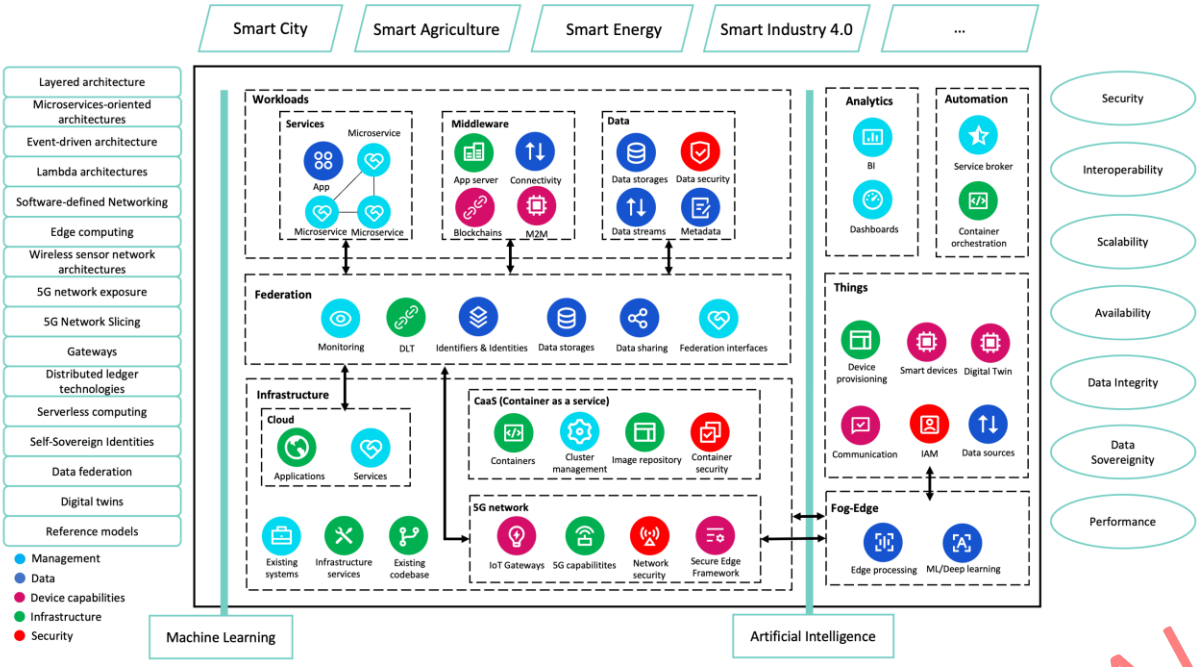

Figure 13: IoT-NGIN Meta-architecture

The IoT-NGIN meta-architecture is organized in the following elements, each of which groups a set of functionalities for IoT systems.

The *Things* functional group contains the elements related to the management, orchestration and proper application support of the far-edge IoT devices.

The *Fog-Edge* functional group refers to the ability of a platform to support, at edge or fog level, the execution of computationally (CPU- or GPU-wise) expensive applications on behalf of the platform-interfacing IoT devices.

The *Analytics* functional group is meant to support the ingested data refinement and transformation into valuable information insights through data analytics processes and assistive technologies.

The elements of the *Automation* functional group are related to enabling the automation perspective of a next-generation, edge-friendly IoT platform, catering on one hand on the proper operated services exposure and, on the other, on the management of the platform, per se, and of its hosted applications. These elements aim to support automation of infrastructure and application provisioning, integration, and management.

The *Infrastructure* functional group contains elements related to the entirety of the infrastructure supporting the IoT platform bootstrapping, configuration, management and, in general, operation.

The *Cloud* subgroup refers to the seamless integration of cloud and edge resources, enabling tasks offloading to the cloud.

The *Container as a Service* subgroup elements ensure that effective infrastructure resources management and coordination is possible, including Cluster management per se, Container operations (hosting and deploying containers), Container security and image repositories, actively supporting the Container Orchestration element of the Automation functional group. This subgroup comprises functionalities that are common in cloud-native environments as well.

The *5G Network* subgroup addresses networking needs existent at the core of edge-oriented platforms, interconnecting the physical world (devices, things) with the digital one (edge infrastructures), ensuring real-time data handling in a secure and trusted manner.

The *Federation* functional group targets at scalability, availability, and stability of the next generation of IoT services as well as sovereignty and transparent control of data and data streams. This group relates to ensuring controlled but also effective data access. The term “Federation” implies platform consistency across the entirety of the operations of the platform at computational level, also known as workloads.

The *Workloads* functional group addresses consistency in the offered web services and the data management lifecycle towards effective interoperability. Under this group, the exposed *Services* group is identified, building up the core of the IoT platform computational environment, including the core platform services and the hosted (user-oriented) ones. In addition, the *Workloads/Middleware* subgroup elements focus on the interoperability potential of the platform, considering both classical client-server, HTTP-based protocols and other friendlier to IoT scopes technologies, relevant to self-organization and data security and sovereignty. Last, but not least, the *Data* subgroup enables low-level data operations, per se, both from an infrastructural point of view but also from the data and metadata management perspective.

The meta-architecture of IoT-NGIN collects key quality requirements, architectural patterns and high-level system components aiming to provide an overall framework for individual system implementations. The motivation is to enable reuse of existing IoT technologies and solutions to new domains and assist structured, informed IoT platform design.

As IoT-NGIN has contributed the meta-architecture for designing IoT ecosystems, it can be related to the NEMO meta-OS meta-architecture, mainly through the infrastructure part, which on the IoT/edge/cloud communication side is similar, while most of IoT-NGIN processes can be considered as services running on top of the meta-OS. Figure 14 depicts the relation among the two meta-architectures.

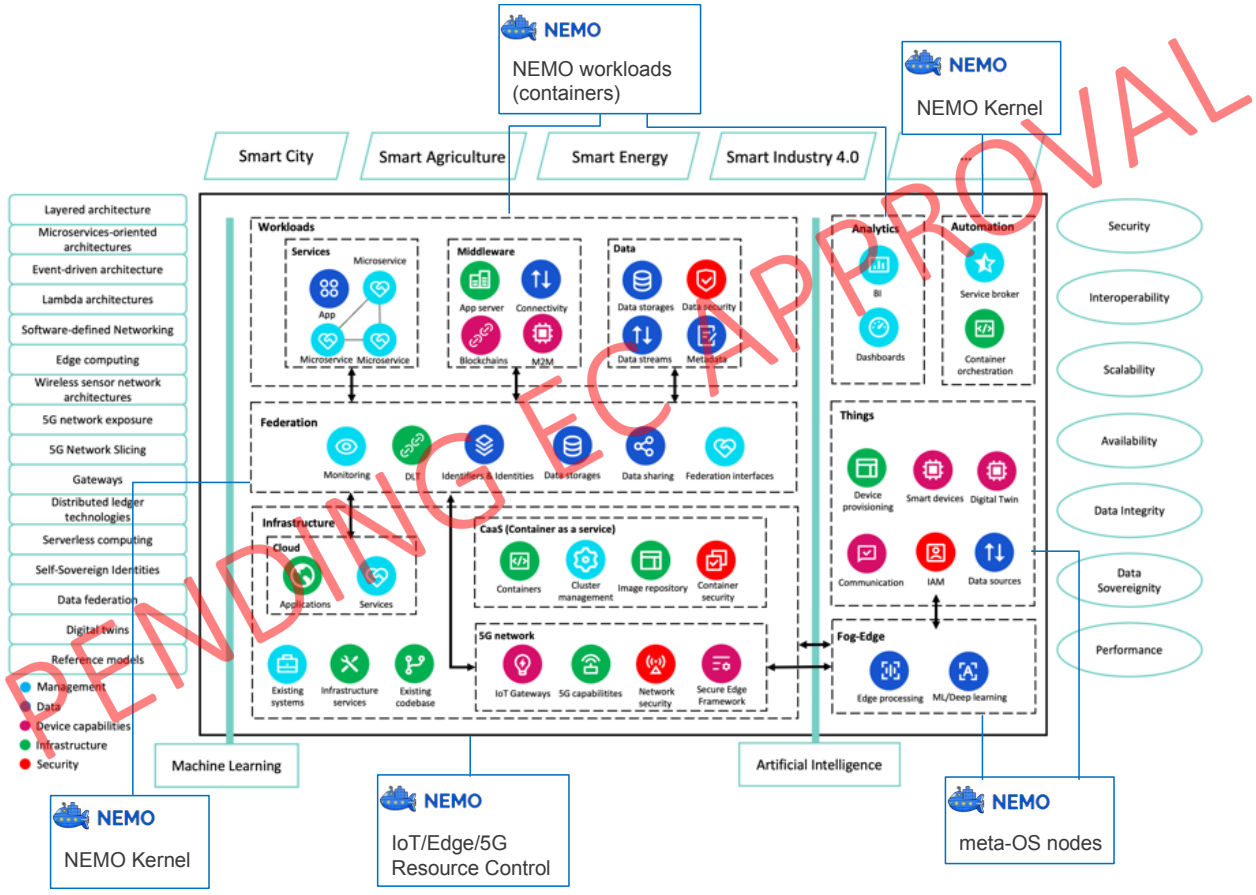

Figure 14: Alignment of IoT-NGIN and NEMO meta-architectures

As shown in the figure, the *Things* and *Fog-Edge* elements of the IoT-NGIN meta-architecture can be seen as meta-OS nodes in NEMO, while *Infrastructure* element includes the NEMO functionalities related to infrastructure, communication and network management. The *Workloads* and *Analytics* parts offer candidate workloads to run on top of NEMO. The *Automation* and *Federation* elements are covered by the NEMO Kernel, supporting the functions included here and additional ones which offer flexible orchestration, configuration, automation and lifecycle management through DevZeroOps methods.

3.7.2 ASSIST-IoT

The ASSIST-IoT architecture was designed considering multiple inputs including (i) the current trends integrating IoT technologies with complementary ones like Edge Computing, Artificial Intelligence and SDN/NFV paradigms; (ii) the expertise of the consortium partners; (iii) the outcomes of previous and concurrent projects as well as of Standards Developing Organisations (SDOs); (iv) extensive research of innovative concepts to improve current performance and scalability of IoT architectures. This architecture was deeply influenced by three architectures, namely the ones provided by the IoT European Large-Scale Pilots (LSP) programme [4], the OpenFog consortium [5], and AIOTI HLA [6].

In the initial stages of developing the ASSIST-IoT, one of the primary decisions was to adopt a layered approach for its Next-Generation IoT (NGIoT) blueprint. This choice was primarily driven by the desire to represent its functionalities and properties in a straightforward manner. Specifically, the conceptual architecture is based on a multidimensional strategy, where horizontal Planes intersect with Verticals, allowing for increased modularity. Planes serve as collections of functions that can be logically stacked upon each other. For instance, when sensor-generated observation data is involved, it needs to traverse through the Smart network and control plane before being processed on the Data management plane. Eventually, it is presented to end-users through a graphical interface on the Applications and services plane. It is important to note that not all information is required to pass through all Planes. In fact, edge devices belonging to the Device and edge plane often perform functions like filtering out necessary data or aggregating it, ensuring that only relevant information is forwarded. It's worth emphasizing that the concept of Planes in ASSIST-IoT should not be confused with the traditional protocol stack approach (similar to OSI model). Instead, it should be seen as an intelligent categorization of logical functions that fall within various plane domains.

In contrast, Verticals encompass essential system properties or aspects that intersect with the overall architecture, along with functions that address specific Next-Generation IoT properties. For instance, even if a comprehensive identity and authorization stack is implemented, security measures should be extended across all Planes, ranging from the network to application code.

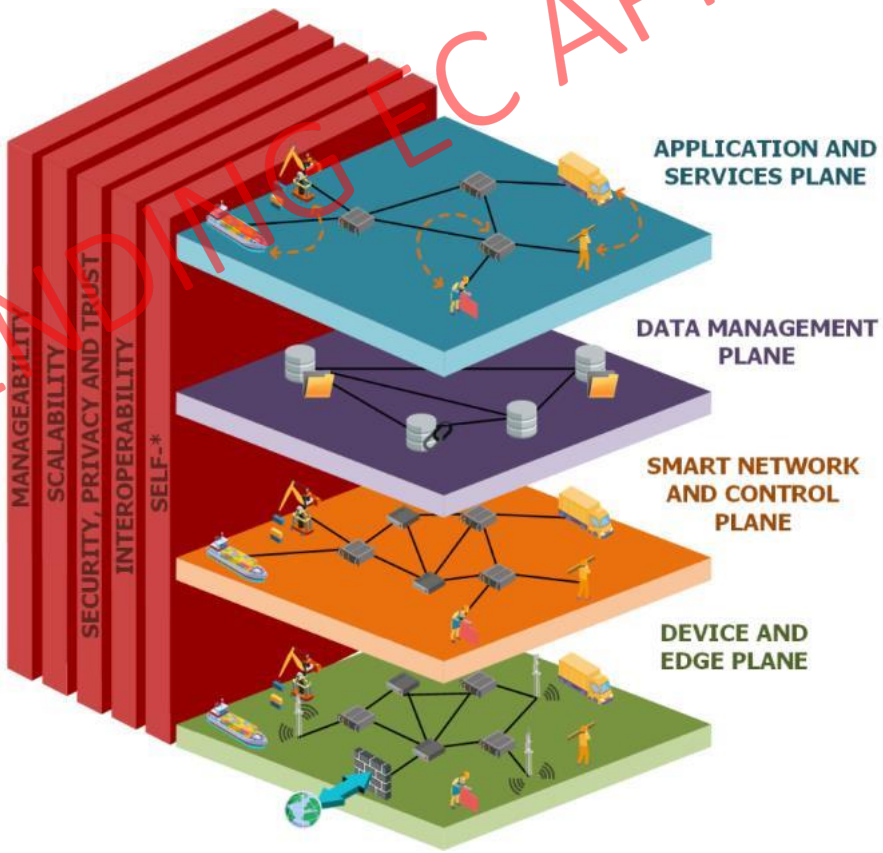

Figure 15: ASSIST-IoT conceptual architecture

The ASSIST-IoT conceptual architecture is presented in Figure 15, while the horizontal planes and verticals are described in more detail as follows.

Horizontal Planes:

- **Device and edge plane** refers to the grouping of functions that can be logically assigned to physical components within the realm of IoT. This encompasses various entities, such as smart devices, sensors, actuators, wearables, edge nodes, and network hardware like hubs, switches, and routers. Therefore, in addition to incorporating the physical elements responsible for the computing and networking infrastructure, this plane also encompasses the functionalities necessary for conducting local intelligent analysis, executing actions, and either pre-processing or elevating data to services in the upper Planes.
- **Smart network and control plane** oversees the virtual and wireless components of network connectivity. It encompasses essential functions that involve technologies facilitating software-based and virtualized networks, such as SD-WAN (Software-Defined Wide Area Network), NFV (Network Function Virtualization), and MANO (Management and Orchestration). This plane follows an access-network-agnostic approach, offering a high level of flexibility in network connections. It provides features like dynamic network configuration, routing, tunnelling, and intelligent firewalling at an advanced level.
- **Data management plane** is responsible for overseeing all functions associated with a virtual shared data ecosystem. Within this ecosystem, data is acquired, delivered, and processed to facilitate crucial data-related operations. This encompasses various mechanisms, such as data routing (moving data between computation nodes and/or services), ensuring interoperability (semantic compatibility), and storage, among other relevant functions.
- **Application and services plane** represents the culmination of the Planes, encompassing end-user and administrative functions as well as various services. It serves as an abstraction layer that leverages the capabilities provided by the underlying Planes, combining them to deliver synergistic value for the entire system. This plane offers valuable insights through user-centric and tactile interfaces, granting users and third-party systems access to the system's functionalities.

Verticals:

- **Self-*** refers to a collection of features that offer autonomous or semi-autonomous capabilities across various dimensions. Specifically, the self-* vertical encompasses different capabilities, including self-diagnosis and self-healing, which enable the autonomous detection and resolution of faulty elements. Additionally, it includes self-configuration and self-provisioning, allowing for the autonomous configuration and provisioning of resources in anticipation of potential increased demand based on statistical predictions, among other capabilities.
- **Interoperability** is a system's ability to work well internally and with external entities. It involves different levels: technical (making it technologically possible), syntactic (allowing data exchange despite different interfaces and languages), and semantic (ensuring shared understanding with precise meanings).
- **Security, Privacy, and Trust** vertical aims to provide important functionalities within the architecture. This includes authorized device registration, secure data sharing, protected storage, and mechanisms to address cyber threats. It's crucial to carefully analyze these aspects, as any flaws can hinder the system's adoption.
- **Scalability** is crucial for an NGIoT deployment to adjust to different workloads, performance levels, costs, and business needs. It requires flexibility in hardware, software, and communications to accommodate changing requirements and support diverse options. This adaptability ensures the system can meet evolving business needs effectively.
- **Manageability** involves configuring system elements, such as computation nodes, and deploying, configuring, and terminating functionalities across the Planes and Verticals. Ease of use is important for adoption. It also ensures proper interfacing of features, enabling the creation of complex services for specific use cases.

Additionally, the Functional View (or Logical View), depicted in Figure 16, allows to expose the functionalities required to fulfill the user needs and address the stakeholders' concerns. It describes the main system's functional elements, their responsibilities, interfaces, and primary interactions.

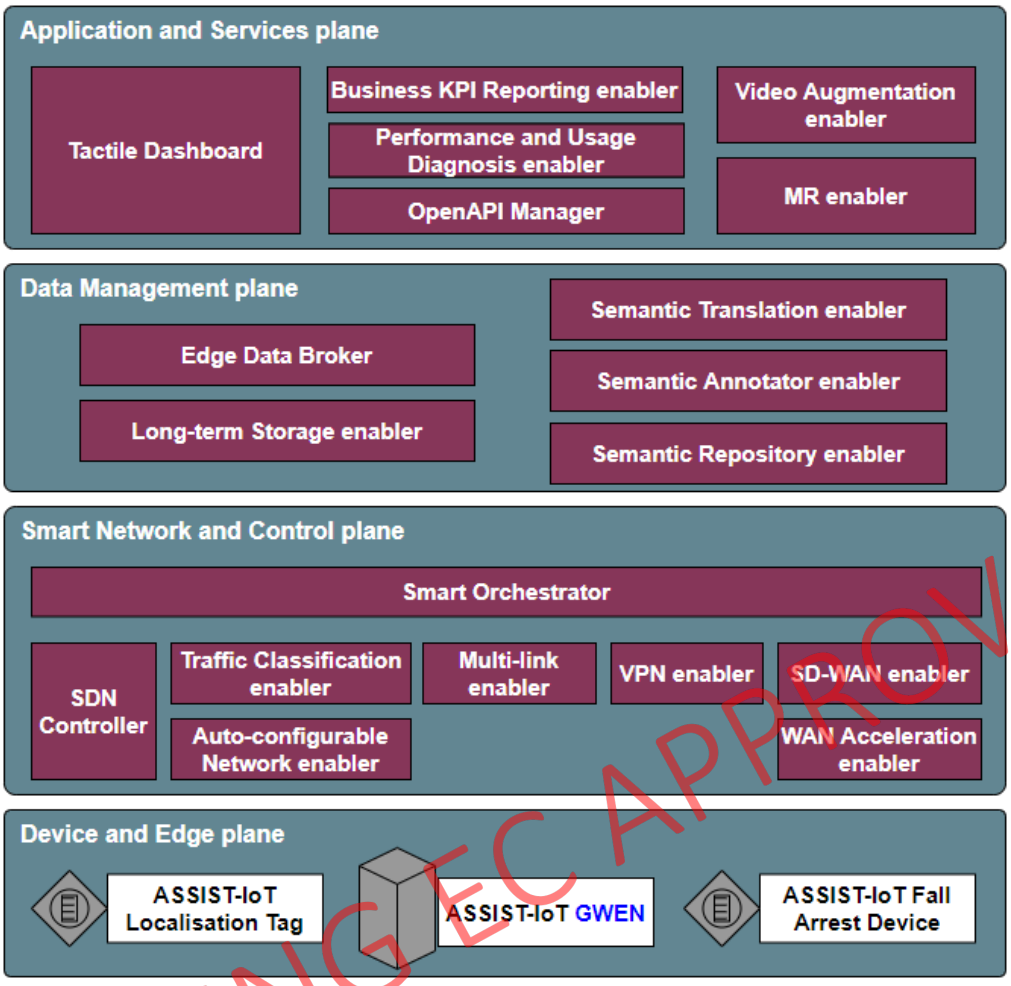

Figure 16: ASSIST-IoT functional view

3.7.3 INGENIOUS

The iNGENIOUS architecture is depicted in Figure 17. It comprises four layers. The initial three layers encompass hardware, heterogeneous networks, and data management and analytics services. On top of these, the fourth layer accommodates applications that rely on the underlying iNGENIOUS components for their functionalities.

The bottom layer of the iNGENIOUS architecture is known as the "things" layer. It encompasses various IoT devices, including sensors and actuators. These devices interact with the physical world in both static and mobile conditions, such as being part of a vehicle or attached to a shipping container. To function as IoT devices, sensors and actuators require embedded computers and network communication hardware. These information-technology components are also part of the bottom layer, with items like wireless modems located at the boundary to the network layer.

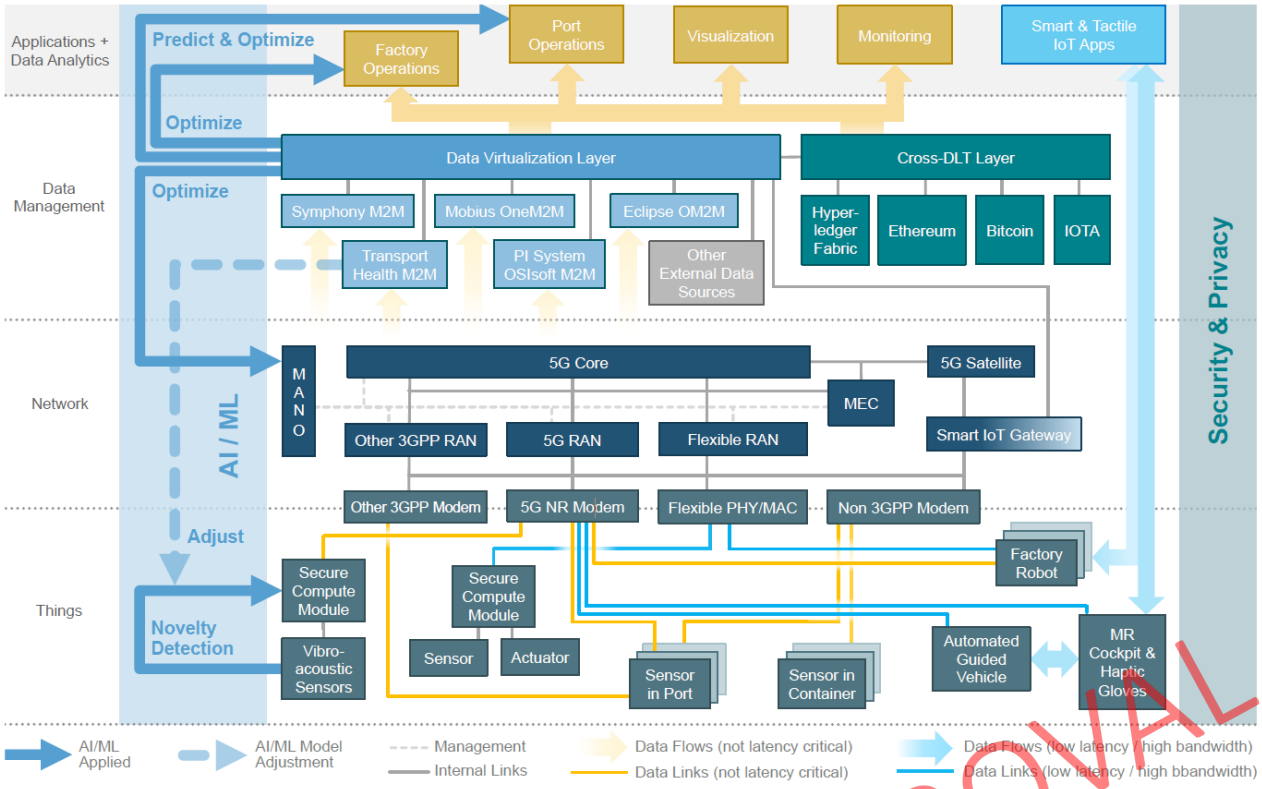

Figure 17: INGENIOUS architecture

Data Management & Application Layers: IoT devices vary not only in terms of their Radio Access Network (RAN), but they are also connected through multiple, incompatible machine-to-machine (M2M) platforms that cater to the diverse stakeholders in the heterogeneous supply chain. Data generated by the IoT devices flows from the network layer into these distinct M2M silos. The iNGENIOUS Data Virtualization Layer (DVL) makes data from all M2M platforms accessible using one common interface. Therefore, the DVL facilitates complete tracking and monitoring of all supply chain assets, along with data-driven predictions and optimizations. The DVL also plays a crucial role in maintaining the integrity of supply chain data by recording it in distributed ledger networks. The iNGENIOUS architecture supports multiple Distributed Ledger Technology (DLT) systems. This responsibility is carried out by a Cross-DLT Layer, which leverages Telefonica's TrustOS to virtualize the DLTs and securely record all transactions passing through the DVL.

IoT Network Layer: Due to the varied and diverse purposes served by IoT devices, there is no universal solution for connecting them to a network. While wired connections may be suitable for devices operating in fixed locations, wireless connections are primarily required, especially in logistics scenarios. Depending on factors like device type, energy constraints, and operating environments, different radio technologies must be utilized. Therefore, the iNGENIOUS architecture needs to support heterogeneous networks that can address the multi-dimensional requirements of bandwidth, latency, range, reliability, and energy efficiency. In addition to 3GPP networks, the bottom layer of the iNGENIOUS architecture incorporates non-3GPP networks, which are connected through a smart IoT gateway. The iNGENIOUS partners also contribute Radio Access Technology (RAN) that allows for adaptable PHY/MAC implementations. To support use cases like transportation-platform health monitoring and container shipping, satellite connectivity is an essential component of the iNGENIOUS network layer.

IoT Things layer: This layer encompasses IoT devices, including sensors and actuators. These devices interact with the physical world in both static and mobile conditions, such as when they are integrated into vehicles or attached to shipping containers. To function as IoT devices, sensors and actuators require embedded computers and network communication hardware. These information-technology

components are also part of the bottom layer, with items like wireless modems positioned at the boundary between the "things" layer and the network layer. Most “thing” components are only needed for specific use cases, but representatives of generic classes such as sensors or actuators are always present.

3.7.4 INTELLIOT

IntellIoT is a European Research and Innovation project that fosters development of humanized IoT and AI devices and systems. The initiative aims to facilitate a competitive ecosystem and to strengthen the European market in finding solutions applicable in healthcare, agriculture and manufacturing. Enabling technologies such as 5G, cybersecurity, distributed technology, Augmented Reality and tactile internet, the project champions end-user trust, adequate security and privacy by design.
The description of the IntellIoT’s architecture is based on the 4+1 Architectural View Model, which aligns with the approach that is followed by NEMO for describing its meta-OS architecture. IntellIoT’s architecture is established up three core pillars that are key to for the project’s concept namely, Collaborative IoT, Human-in-the-loop and Trustworthiness. In addition, five (5) core component groups have been identified, with individual components falling into one of them. The groups are described below:

- **Collaborative IoT enablers:** This group contains the components that realize IntellIoT’s Collaborative IoT pillar, focusing on the cooperation of various semi-autonomous entities (tractors, robots, healthcare devices, etc.) to execute multiple IoT applications.
- **Human-in-the-Loop enablers**: This group contains the components involved in IntellIoT’s Human-in-the-Loop (HIL) pillar, which focuses on involving the human in the process, when necessary, in order to solve complex situations that the system does not yet know how to handle.
- **Trust enablers**: This group involves components that are part of IntellIoT’s Trust pillar. This pillar focuses on privacy, security, and ultimately building trust into the IntellIoT framework.
- **Infrastructure management**: This group is comprised of the computation and communication infrastructure and its management capabilities, which enable the deployment and management of edge applications.
- **Use-Case deployment**: This group involves all components which are Use-Case specific, (i.e., pertaining to the use case environment deployment), such as edge devices and their hardware, edge apps, and edge AI models.

The first 4 groups are comprised from use-case agnostic enablers that constitute the core IntellIoT framework, and which are potentially usable in NG-IoT use cases. Figure 18, illustrates the aforementioned thematic entities that concern project’s implemented enablers.

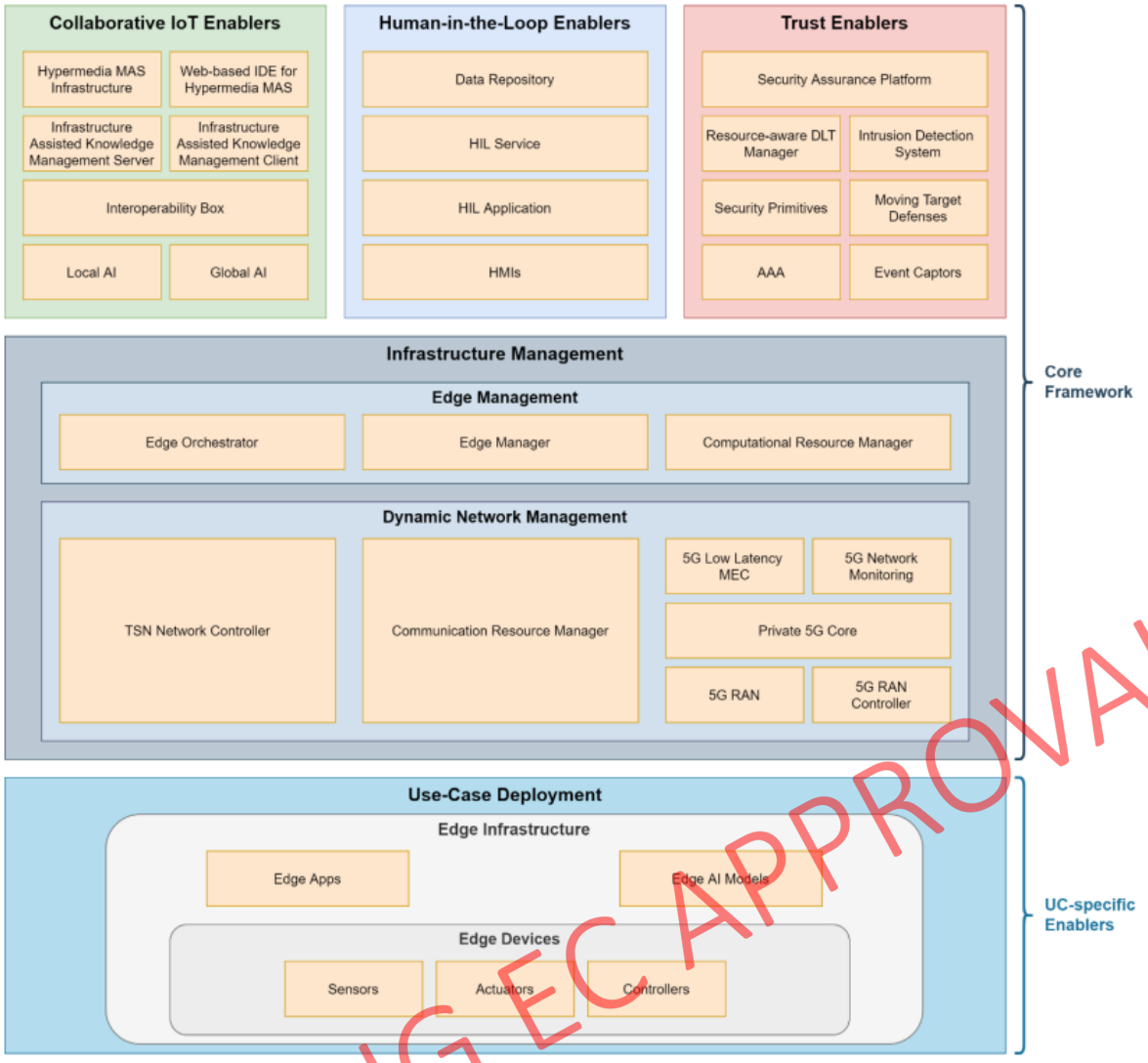

Figure 18: High-level view of IntellIoT's logical architecture

Elaborating on the high-level alignment between NEMO and IntellIoT and in an attempt to identify high-level similarities, where applicable, it is evident that the distribution of AI components across the system and within the system's components is one of the common characteristics of both projects' architecture. In IntellIoT, the kernel of the system-wide AI components is the Hypermedia Multi-Agent System (MAS) infrastructure.

The Infrastructure-assisted Knowledge Management (IAKM) which is located in the IntellIoT infrastructure (private network or private 5G MEC), acts as an AI/ML broker and storage according to W3C-defined semantics. In addition, the Global AI Component in IntellIoT acts as the centralized entity that supports individual devices to share the training results and it manages the entire federated learning process. The Local AI Component is responsible of training AI models using the local datasets at the IoT device. Similarly, NEMO CF-DRL component realizes federated learning mechanisms and aims to deliver AI-logic to the NEMO components (e.g. NEMO meta-orchestrator).

Moreover, similarly to the NEMO' meta-orchestrator, IntellIoT's MAS contains an orchestrator that facilitates the communication between the system and the edge app. More specifically, the communication and computation infrastructure builds the foundation of the IntellIoT framework and allows the deployment and dynamic management of edge applications (Edge Apps). The central point for triggering the deployment of Edge Apps is the Edge Orchestrator that works together with one / multiple Edge Manager(s) to orchestrate the efficient deployment of an Edge App onto one / multiple edge devices.

In IntellIoT, the Communication Resource Manager is responsible for dynamically reconfiguring the 5G network (RAN and CN) for optimal performance of the 5G infrastructure, i.e., to open, configure, maintain and monitor dedicated 5G resources. In NEMO the abovementioned functionality is facilitated by the mNCC.
With respect to security, in IntellIoT the Agents within Hypermedia MAS infrastructure, Edge Apps and Interoperability Box interact with the DLT Manager enabling decentralized journaling of the system's operation. On the other hand, NEMO also incorporates DLT technology that safeguards the business-logic of the communications that adhere to the resource provisioning activities.
Lastly, an additional common architectural feature of both projects is the common underlying 5G infrastructure. In NEMO the 5G core network and resources manager and in IntellIoT the Communications Resource Manager are both responsible for dynamically reconfiguring the 5G network (RAN and CN) for optimal performance of the 5G infrastructure.
To conclude, NEMO has the opportunity to capitalize on the work that has been conducted in projects such as IntellIoT and optimize the operational effectiveness of their implemented technologies.

3.7.5 VEDLIOT

The EU-funded VEDLIoT project develops an IoT platform that uses deep learning algorithms distributed throughout the IoT continuum. The proposed new platform with innovative IoT architecture is expected to bring significant benefits to a large number of applications, including industrial robots, self-driving cars, and smart homes. The project offers an Open Call at project midterm, incorporating additional VEDLIoT-related industrial use-cases in the project, increasing the market readiness of the VEDLIoT solutions.
In terms of hardware, VEDLIoT offers a platform, the Cognitive IoT platform, leveraging European technology, which can be easily configured to be placed at any level of the compute continuum starting from the sensor nodes and then edge to cloud. Driven by use cases in the key sectors of automotive, industrial, and smart homes, the platform is supported by cross-cutting aspects satisfying security and robustness. Overall, VEDLIoT offers a framework for the Next Generation Internet based on IoT devices required for collaboratively solving complex DL applications across a distributed system.
The VedlIoT project capitalizes on architecture frameworks that organize architectural descriptions and associated requirements into distinct architectural views. These different views are necessary to describe the diverse use cases and concerns associated with the VEDLIoT platform. An architectural view expresses "the architecture of a system from the perspective of specific system concern". This concept is similar to the NEMO project's meta-architecture, where again the NEMO framework architecture instantiations reflect its Living Lab Use Case particular requirements and logic.
The VedlIoT project concerns three (3) pilots namely, the Automotive m Industrial IoT and Smart Home ones. The NEMO project is partially aligned as it realizes similar Living Lab Use Cases. However the VedlIoT ones are oriented towards energy efficiency.
VedlIoT architecture overview is presented in Figure 19. VEDLIoT aims at enabling the use of DL algorithms in IoT by accelerating and optimizing applications with energy efficiency in mind. Compared to the NEMO metaOS framework, energy efficiency on deployed applications is a common objective for both of the projects.

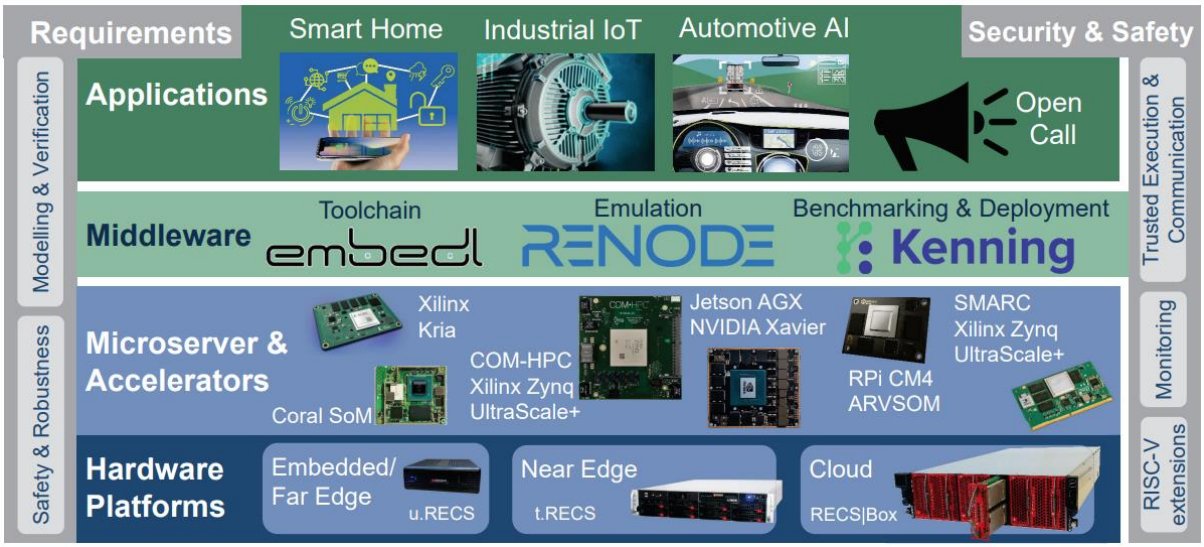

Figure 19: VEDLIoT architecture overview

To conclude, the VedlIoT AIoT hardware platform offers tailored hardware components and supplementary accelerators for AIoT applications, ranging from embedded systems to edge computing and cloud platforms. Although NEMO metaOS has a broader scope, both projects utilize hardware platforms and infrastructures that cover the Far Edge, Near Edge and Cloud continuum. In addition they seem to be fairly aligned on the concept of bringing AI logic in the IoT applications and devices, where both projects they are incorporating AI/ML models based decision making assistance.

3.7.6 TERMINET

The TERMINET project aims to provide a novel next generation reference architecture based on cutting-edge technologies such as SDN, multiple-access edge computing, and virtualization for next generation IoT, while introducing new, intelligent IoT devices for low-latency, market-oriented use cases.
TERMINET suggests an NG-IoT architecture [28], taking full advantage of a variety of technologies such as SDN, Multiple-access Edge Computing (MEC), Federated Learning and Digital Twins. In particular, as illustrated in Figure 20, the TERMINET architecture is composed of six layers: (a) Physical Layer, (b) Middleware Layer, (c) Intelligence Layer, (d) Platform Layer and (e) Application Layer.

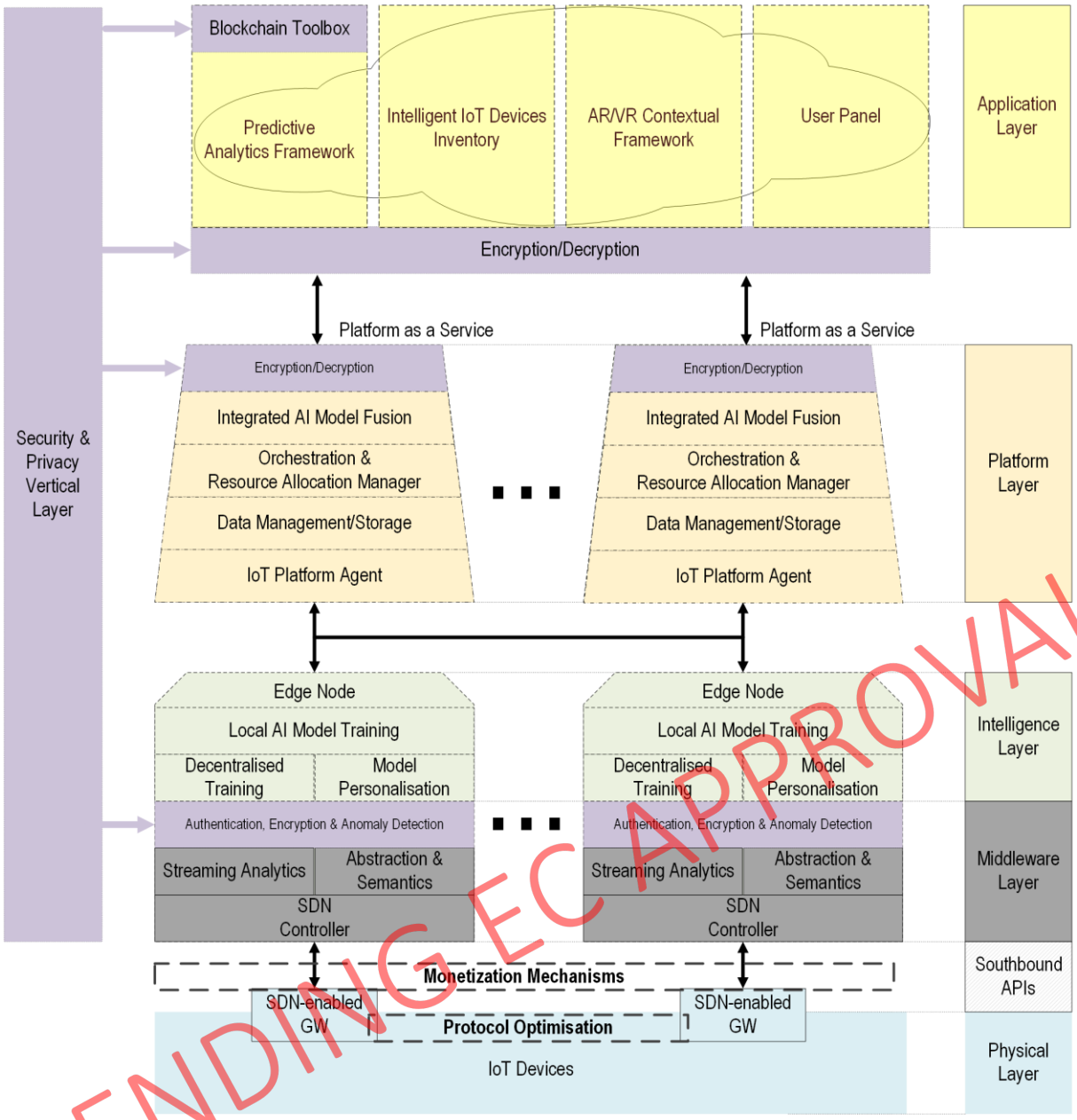

Figure 20: The TERMINET Architecture

The Physical Layer includes physical and virtualised IoT devices that are connected to the hardware-based and virtual SDN switches. Next, the Middleware Layer is devoted to the network monitoring and controlling activities managed by the SDN controller. This layer focuses on the southbound interfaces and includes streaming analytics as a pre-processing step before the activities of the Intelligence Layer. This layer is devoted to federated learning activities for a variety of purposes, including smart farming, personalised healthcare and predictive maintenance. Subsequently, the Platform Layer pays special attention to the orchestration services carried out by the Vertical Application Orchestration. Finally, the Application Layer refers to cloud computing sources and applications offered either by TERMINET itself or the TERMINET end users.

Similarly to rest projects investigated, TERMINET provides candidate workloads for the NEMO meta-OS, which may come from the Middleware Layer and above. In addition, the project considers both IoT and edge nodes, which could be integrated into the meta-OS infrastructure.

4 Convergence to the meta-OS meta-architecture

4.1 NEMO Meta-architecture

ISO/IEC/IEEE 42010 defines *architecture framework* as "conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders" [11]. In this document, we adopt the slightly modified term **"meta-Architecture framework", as "conventions, principles and practices for the description of meta-architectures established within an ecosystem of various domains of application and/or community of stakeholders"**. The NEMO meta-architecture aims to serve as a basis for building meta-OS reference architectures, facilitating entities' integration in the meta-OS world.

NEMO has defined a meta-OS meta-Architecture Framework (MAF), which includes the following elements, as depicted in Figure 21:

- Rationale
- Entity of interest
- Stakeholders
- Stakeholders' perspective
- Concerns
- Viewpoints
- Cross-cutting functions

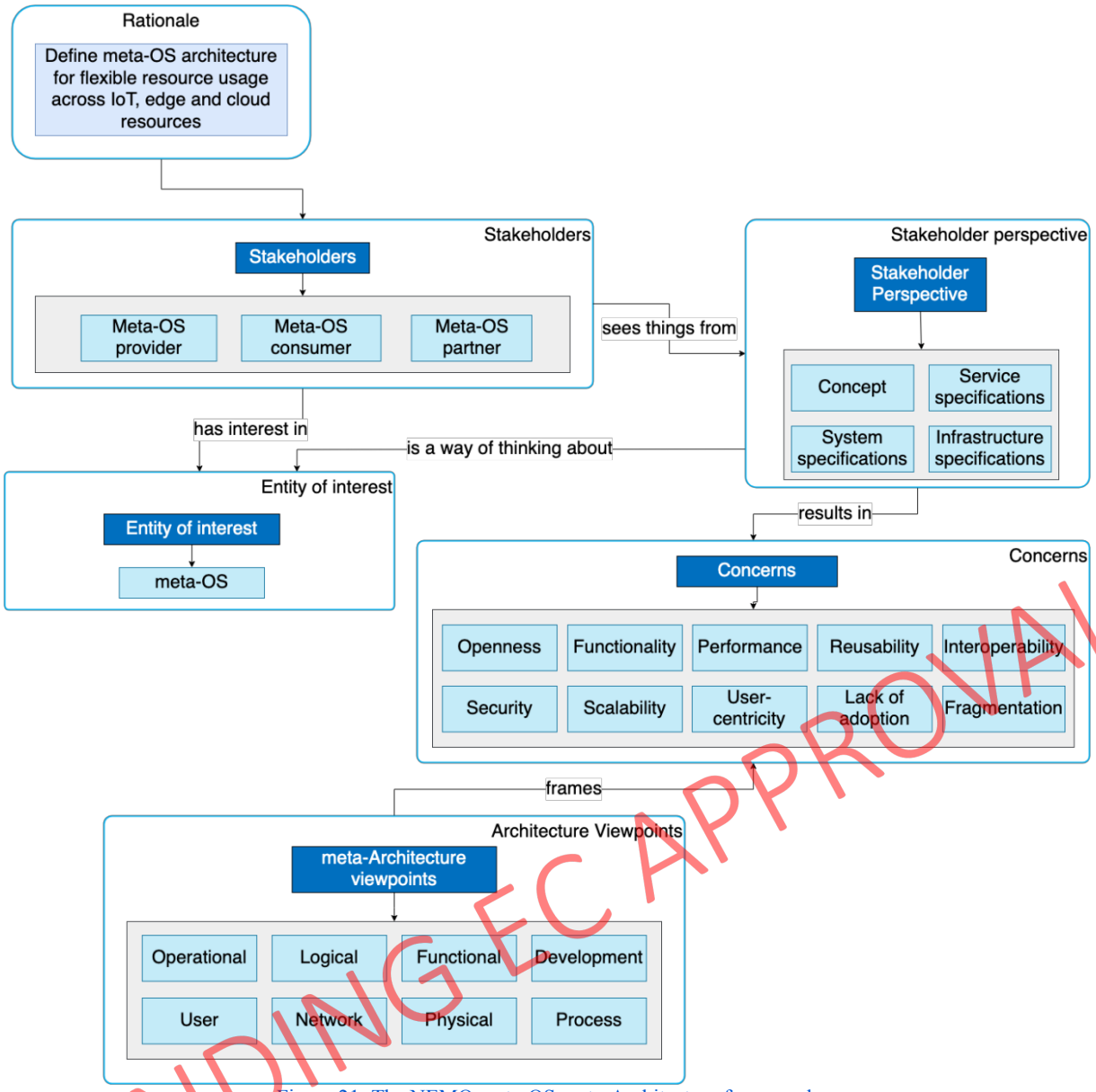

Figure 21: The NEMO meta-OS meta-Architecture framework

The NEMO meta-OS MAF elements are described in the following subsections.

4.2 Rationale

The NEMO meta-OS MAF aims to facilitate the design of meta-OS ecosystems, in a way that they will be scalable, extensible, modular and interoperable. Thus, it provides specifications of elements that would be of interest for a meta-OS architecture designer. The selection of the NEMO MAF elements has been made on the premise to provide a consistent domain agnostic way of describing any specific meta-OS architecture.

4.3 Entity of interest

The *Entity of interest* refers to the meta-Operating system for the IoT, edge and cloud continuum.
The term meta-OS has been coined from Microsoft [29] (MetaOS/TAOS), possibly referring to "*a platform on top of that foundation [cc SharePoint, the Office 365 substrate, Azure, Microsoft's machine-learning infrastructure and more] - one oriented around people and the work they want to do rather*

than our devices, apps, and technologies." This can be understood as a layer that could leverage its AI technology to harness user data and make user applications coherently smarter and more user-centric, rather than oriented to devices, applications and services.
Moreover, ROS (Robot Operating System) [30] is defined as a meta-operating system for robots, clarifying that: *"It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers."* So, the meta-OS additionally allows communication of processes running at different nodes to communicate at runtime.
In NEMO, the meta-OS is considered to enable abstraction of hardware and software management across IoT, edge and cloud resources, providing an interface between humans (application) and any computing device. In the meta-OS, intelligence may move closer to the point of decision, supporting every activity, process and decision that ranges from ad-hoc micro-cloud cluster self-organization to micro-services migration and intent based programming, while ensuring interoperability, trust, cybersecurity and privacy policies enforcement.

4.4 Stakeholders

Stakeholders are parties (individuals, groups or organization) who have some interest in the meta-OS. These could include parties who may pay for the meta-OS or provide or even sustain it, as these activities would require that they should have a clear idea of the meta-OS architecture.
Each stakeholder has at least one *Concern* over the meta-OS. Indicative stakeholders' groups include end users, operators, acquirers, owners, suppliers, developers, maintainers, markets., etc.
The following stakeholders are identified:

- Meta-OS provider, referring to parties that may host, provide and/or manage the meta-OS.
- Meta-OS consumer, referring to consumers of the meta-OS services, basically referring to application and service owners wishing to run their applications on the continuum.
- Meta-OS partner, referring to parties that may create value on top of the meta-OS, which may result from integration of own resources, development on top of the meta-OS, service brokerage and enablement, but also auditing.

4.5 Stakeholders' perspective

The *Stakeholder perspective* is used to group concerns for each of the identified *Stakeholders*, considering the way the interest on the meta-OS is perceived by each *Stakeholder*. In NEMO, four perspectives have been identified for the three Stakeholder groups, namely:

- Concept: It represents concerns related to the meta-OS strategic intent, as expressed by the capabilities envisioned for the meta-OS. The concerns of this perspective aim to support the analysis and optimization of the meta-OS capabilities, supporting their description, interaction and operation.
- Meta-OS specifications: This perspective expresses concerns on the description of the meta-OS, which will cater for the delivery of capabilities, activities, as well as resource and data exchanges.
- Service specifications: This perspective expresses concerns on the description of services running on top of the meta-OS. The concerns address the identification and description of services supporting the IoT-edge-cloud native paradigm.
- Infrastructure specifications: This perspective groups concerns around the infrastructural elements supporting the delivery of the meta-OS capabilities. These could refer to physical or virtual computational, storage and network resources. The concerns may refer to description of the structure, connectivity and behaviour of the various types of infrastructural elements.

Table 2 classifies the identified *Stakeholder perspectives* per *Stakeholder* of the meta-OS architecture.

Table 2: The Stakeholders' perspectives for each Stakeholder

	Meta-OS provider	Meta-OS consumer	Meta-OS partner
Concept	✓	✓	✓
Meta-OS specifications	✓		
Service specifications	✓	✓	✓
Infrastructure specifications	✓		✓

4.6 Concerns

According to ISO/IEC/IEEE 42010, a *Concern* is any interest in the meta-OS, as perceived by a *Stakeholder*. The identification of concerns in the meta-architecture is fundamental in the architecture description, as the meta-OS is multi-disciplinary in the sense that it has different types of *Stakeholders*. So, the *Concerns* may lead to different forms of presentation which may target different Stakeholders. Ideally, the traceability of stakeholders' concerns and their form of presentation (i.e., viewpoint in the meta-OS meta-architecture) is key to deriving an interest-driven meta-OS architecture.
The identified concerns of the meta-OS meta-architecture are described in Table 3.

Table 3: The Concerns in the meta-OS meta-architecture

Concern	Description
User-centricity	The ability of the meta-OS to deliver capabilities according to users' needs and desires, rather than according to services/applications needs
Functionality	The ability of the meta-OS to deliver capabilities fully or partly as and when required and allow people and applications/services which interact with the meta-OS to work effectively
Security	The ability of the system to reliably control, monitor and audit who can execute which types of activities on the meta-OS resources and data, as well as the ability to detect and mitigate security incidents.
Scalability	The ability of the meta-OS to handle increasing workload or infrastructural resources.
Interoperability	The ability of the meta-OS to incorporate different types of physical or virtual infrastructure.
Performance	The ability of the meta-OS to deliver the expected level of capabilities under the mandated profile
Openness	The modularity of the meta-OS and conformance to open interfaces.
Fragmentation	The ability of the meta-OS to deliver its capabilities coherently across IoT, edge and cloud resources
Sustainability	The ability of the meta-OS to adapt capabilities' delivery according to energy consumption goals.

4.7 Viewpoints

A *Viewpoint* in the meta-architecture includes a set of conventions used to develop an Architecture View. The Viewpoint aims to frame a set of *Concerns*.
NEMO has identified the following Viewpoints for the meta-OS meta-architecture:

- Network
- User
- Logical

- Operational
- Functional
- Process
- Development
- Physical

The viewpoints are analysed in the following tables, providing the objectives aimed to be covered and the *concerns* addressed by each, as well as the usage and presentation form of the corresponding views.

Table 4: The Network Viewpoint

Network Viewpoint	
Description	This viewpoint aims to • identify infrastructure elements • identify different classes of physical devices, based on their capabilities and placement on the IoT, edge and cloud continuum. • identify the artefacts upon which resources are deployed and can show the nodes that the resources realize
Concerns addressed	Fragmentation
Usage	Definition of meta-OS concepts Definition of meta-OS options Meta-OS deployment Operational planning
Representation	Tabular, Topological

Table 5: The User Viewpoint

User Viewpoint	
Description	This viewpoint aims to • identify different users, roles and subroles in the meta-OS. • identify the activities enabled by the meta-OS capabilities for each user.
Concerns addressed	User centricity
Usage	Definition of meta-OS concepts Definition of meta-OS options Definition of stakeholders' interests Definition of security and privacy rules
Representation	Structured Text, Entity-Relationship diagram

Table 6: The Logical Viewpoint

Logical Viewpoint	
Description	This viewpoint aims to • identify the entities constructing or involved in the meta-OS • identify the conceptual inter-relations among those entities • identify entities in the meta-OS which relate to monitoring or imposing sustainability goals
Concerns addressed	Functionality Sustainability
Usage	Definition of meta-OS concepts Source for elicitation of functional requirements
Representation	Entity-Relationship diagram

Table 7: The Operational Viewpoint

Operational Viewpoint	
Description	This viewpoint aims to • identify use cases of application of the meta-OS • identify user operational activities through specific scenarios • present added value through the meta-OS in business cases • identify assumptions for the realization of these business cases • identify desired outcomes and measurable benefits associated with the defined use cases
Concerns addressed	User centricity Functionality
Usage	Definition of strategic vision for the meta-OS Analysis of expected results and effects Definition of operational activities enabled for each user Meta-OS capabilities' planning (requirements), matching the definition of operational activities
Representation	Structured text (usage scenarios)

Table 8: The Functional Viewpoint

Functional Viewpoint	
Description	This viewpoint aims to • identify meta-OS capabilities which may support the defined operational activities per stakeholder • group the meta-OS capabilities into sets that constitute functional layers • identify the technical components which may deliver the meta-OS capabilities • identify cross-cutting functionalities
Concerns addressed	Functionality Security Scalability Interoperability Openness Sustainability
Usage	Feedback to meta-OS product management; capability planning
Representation	Structured text; Black diagram

Table 9: The Process Viewpoint

Process Viewpoint	
Description	This viewpoint aims to • Identify representative use cases delivering the combined capabilities of the meta-OS • Identify the components delivering those use cases • Identify the interactions among these components for delivering the use cases • Identify data flows within these interactions
Concerns addressed	Functionality Security Interoperability

Process Viewpoint	
Usage	Highlight potential integration requirements Implement/enable interactions among meta-OS components
Representation	Sequence diagram

Table 10: The Development Viewpoint

Development Viewpoint	
Description	This viewpoint aims to • identify the implementation details for components delivering meta-OS capabilities • identify measurable targets for capabilities' verification
Concerns addressed	Functionality Security Interoperability Performance
Usage	Implementation Setting Capability Requirements
Representation	Class diagram (components) Structured text (metrics)

Table 11: The Physical Viewpoint

Physical Viewpoint	
Description	This viewpoint aims to • identify the deployment setups for the components delivering meta-OS capabilities • guide the integration of those components • identify hardware and software requirements for the installation of those components • provide installation and user guides for the integrated components delivering NEMO capabilities
Concerns addressed	Security Performance
Usage	Integration Deployment
Representation	Topology diagram (deployment setup, components) Structured text (requirements, guides)

Figure 22 presents the NEMO MAF viewpoints and their relations and interactions for designing a meta-OS architecture.

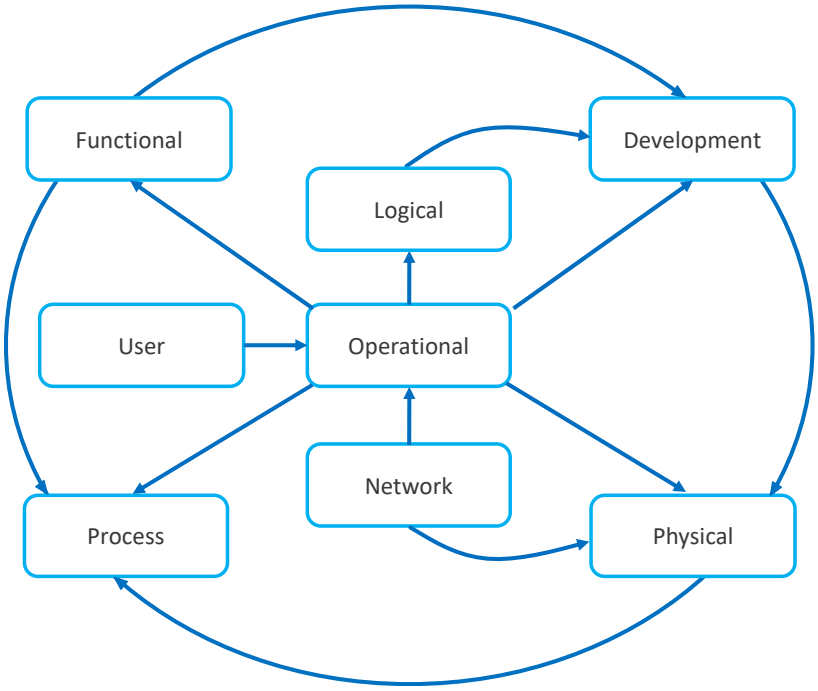

Figure 22: The NEMO meta-architecture viewpoints

It has to be noted that the viewpoints address stakeholders' concerns. The *stakeholder perspectives* are mapped into the defined concerns, and these are meant to be addressed through the development of the defined viewpoints. Figure 23 presents indicative the way the Stakeholder Perspectives are perceived as Concerns and their transition to Viewpoints in the NEMO MAF.

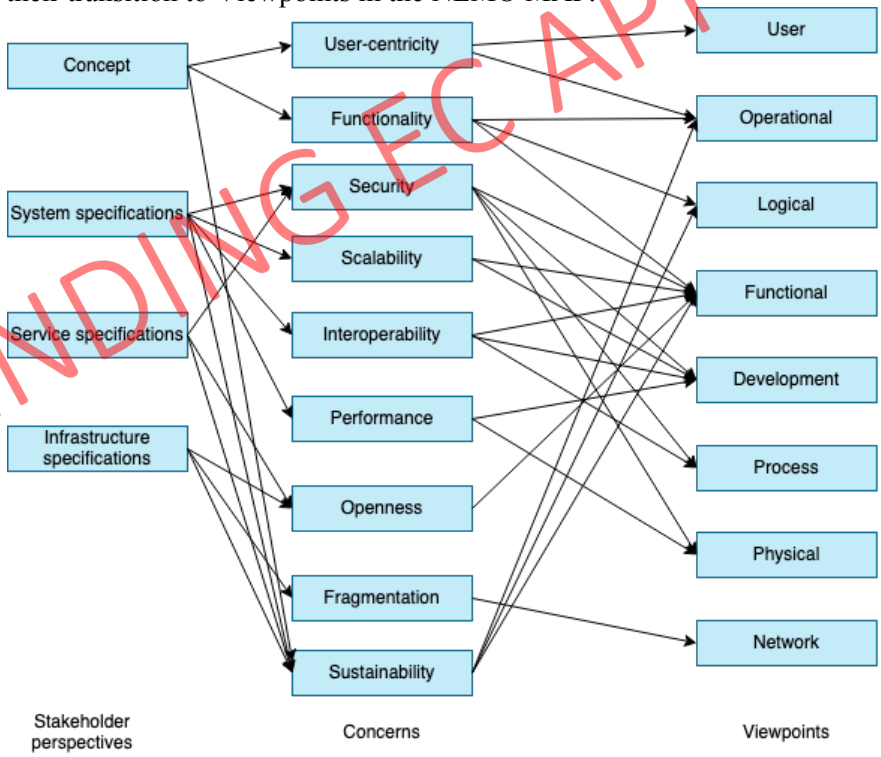

Figure 23: Transition from Stakeholder Perspectives to Concerns and Viewpoints in the meta-OS MAF

4.8 Cross-cutting functions

The NEMO identifies three cross-cutting functions for the meta-OS:

- Cybersecurity & Unified/Federated Access Control, which denotes that cybersecurity should be present at any point in the meta-OS continuum and access to data and controls must be by design managed uniformly;
- Cybersecure Federated Deep Reinforcement Learning, which provides a meta-OS native MLOps platform, supporting advanced Machine Learning (ML) capabilities across the meta-OS. AI and ML should be present in all meta-OS components and this function aims to support this function across them;
- PRESS & Policy Enforcement Framework, which ensures that Privacy, data pRotection, Ethics, Security & Societal (PRESS) and user-driven policies must be respected at all levels and by any component in the continuum.

5 NEMO Architecture

5.1 Network view

The Network view provides the physical and conceptual classification and hierarchy of the NEMO meta-OS computing resources. The network elements are described in tabular format in Table 12.

Table 12: Network elements and concepts in the NEMO meta-OS

Network element	Description
Node	A node is a hardware computing element of highest granularity in the meta-OS continuum. As such, it may represent any physical computing device in the continuum, including IoT, Edge and Cloud Server Devices, or even a virtual machine provided by a cloud provider. As nodes are aimed to provide computing abstraction for executing containers of workloads on them, the distinction in node types is only aimed to help the selection of proper distribution of the containers' orchestration framework, which would be compliant to the computing capabilities of the node in terms of CPU, RAM and storage.
Cluster	A Cluster is defined as a grouping abstraction of nodes which are managed by a single entity. A cluster could include one or more nodes, which altogether form the resource pool upon which different workloads' execution is orchestrated. Nodes can be added or removed from the cluster, without this affecting the execution of workloads. In NEMO, clusters are K8s clusters.
Cluster Set	A Cluster Set provides a grouping abstraction for clusters participating in the meta-OS. It allows to restrict the workload execution among the clusters participating in the cluster set.
Cluster Shell	A Cluster Shell is the complete set of clusters participating in the meta-OS. A Cluster Shell is scoped at meta-OS level and implies that clusters are managed in parallel in a coherent way. The Cluster Shell allows performing cluster-aware administration, being aware and managing groups of clusters (i.e. Cluster Sets). It is usually composed of multiple clusters, allowing high-availability, fault tolerance and flexible use of available infrastructure across them. Unless otherwise defined, a workload's execution should be able to be orchestrated in a coherent manner across all the clusters in a single meta-OS instance.
Admin domain	The Admin domain represents the IP space and infrastructure managed by a single entity, usually an enterprise, even if they do not belong to the entity. Workload orchestration in the meta-OS should be coherent across admin domains, respecting the security and privacy rules of the owning enterprise.

The topology of the defined network elements is depicted in Figure 24. The figure illustrates how the concepts described in Table 12 are conceived in NEMO, in particular the Node, the Cluster, the Cluster Set, the Cluster Shell and the Admin Domain.

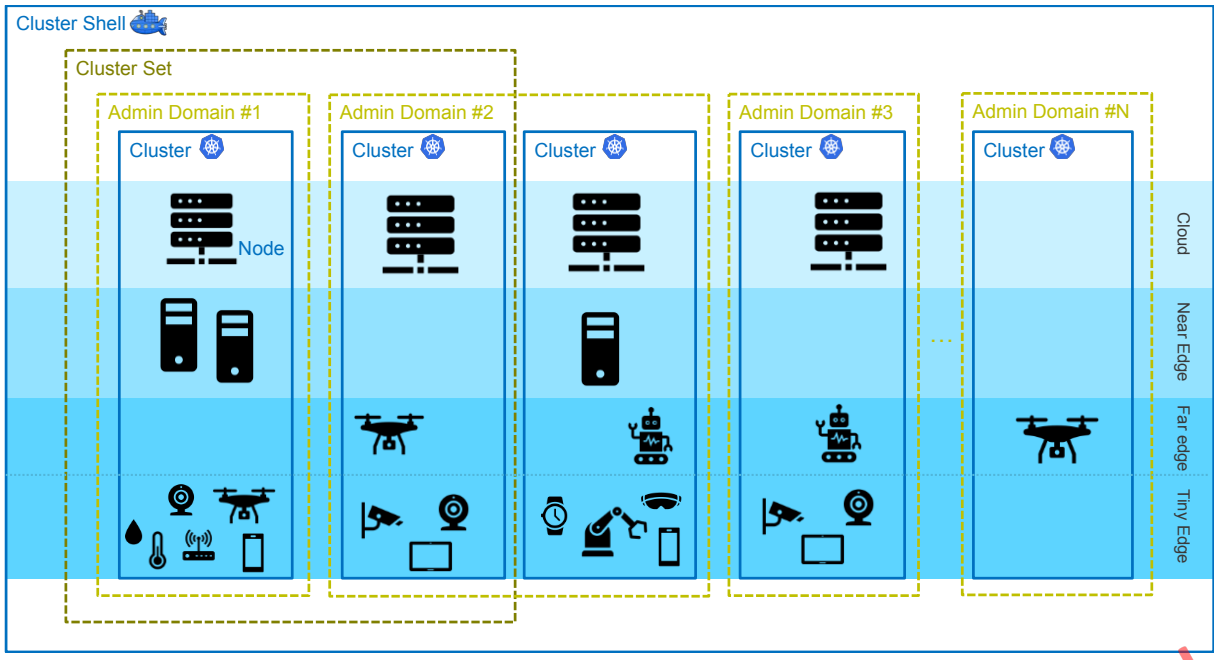

Figure 24: Network topology for the NEMO meta-OS

Moreover, the node placement in the continuum is highlighted. It is worth noting that for the conceptual classification of nodes across the continuum, NEMO adopts the near, far and tiny edge computing definitions suggested by Cloud Native Infrastructure VP, SUSE [31].

The *near edge* includes devices which are *nearest* to the centralized services provided by cloud servers. Nodes in the near edge typically belong to the infrastructure hardware and IP space of communications service providers.

The *far edge* includes nodes which are typically farthest from the cloud servers and belong to the IP space and infrastructure owned and managed at enterprise level.

The *tiny edge* includes fixed-function devices, such as sensors, actuators and IP cameras within the enterprise network and infrastructure. As a result, it is a subgroup within the far edge, but the tine edge devices have very constrained computing and storage resources. This category embraces the introduction of the Industrial Internet of Things (IIoT) into the cloud native management world.

Last, *cloud servers* represent server in micro-/datacentres with significant computing capacity, which could belong to any type of cloud, i.e. public, private, hybrid, etc.

The figure includes some indicative cases which aim to clarify and differentiate between the NEMO network concepts. For example, an Admin Domain may include one or more clusters, which may be formed by nodes on the same premises or in remote ones. The Admin Domain relates to the management of the clusters, i.e. a single Admin Domain may be administered by a single entity (K8s operator). Also, the Cluster Set denotes a group of clusters which may be belong in the same or different admin domains and its role is purely to restrict workload execution in those.

Furthermore, a cluster may include only one (single-node cluster) or more nodes, while the nodes participating in a cluster could belong to any level at the continuum. An implication of this setting is that nodes of highly diverse capabilities and resources (from HPC cloud servers to constrained IoT sensors, for example) should be able to be treated uniformly from the orchestration point of view. This is attempted to be addressed via the various K8s distributions, either vanilla K8s compatible or managed ones, some of which offered as lightweight K8s distributions specifically appropriate for edge computing cases, targeting low or very constrained devices. Some indicative examples include MicroK8s [2], K3s [3], K0s [4], minikube [5], KubeEdge [6]. Moreover, on top of that, NEMO aspires to provide the cloud-native meta-OS that will be able to effectively manage the lifecycle of workloads across the participating nodes and clusters, addressing security and privacy concerns, offering orchestration at scale and in a user-centric manner.

5.2 User view

The user view aims to identify the users, in terms of roles and subroles, along with their activities within the meta-OS. The NEMO users represent the stakeholders of the meta-OS and are described in a similar manner to ISO/IEC 17789:2014, describing the users in the Cloud Computing Reference Architecture. Aligned to that, Figure 25 depicts the NEMO MAF User View elements and their relations. A User is described by a set of one or more Roles. Then, each Role is further composed on Subroles, each of which may apply a set of Activities. An *aspect* refers to a cross-cutting function.

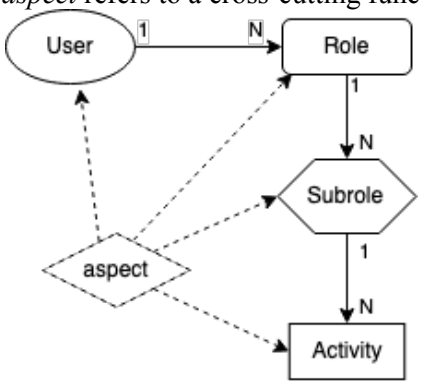

Figure 25: User view entities and their relations

The NEMO meta-OS users include the following roles:

- Meta-OS provider: This group represents parties that may host, provide and/or manage the meta-OS.
- Meta-OS consumer: This group represents consumers of the meta-OS services, basically referring to application and service owners wishing to run their applications on the continuum.
- Meta-OS partner: This group includes parties that may create value on top of the meta-OS, which may result from integration of own resources, development on top of the meta-OS, service brokerage and enablement, but also auditing.

The subroles and activities of each user role are provided in the following subsections.

5.2.1 Meta-OS provider

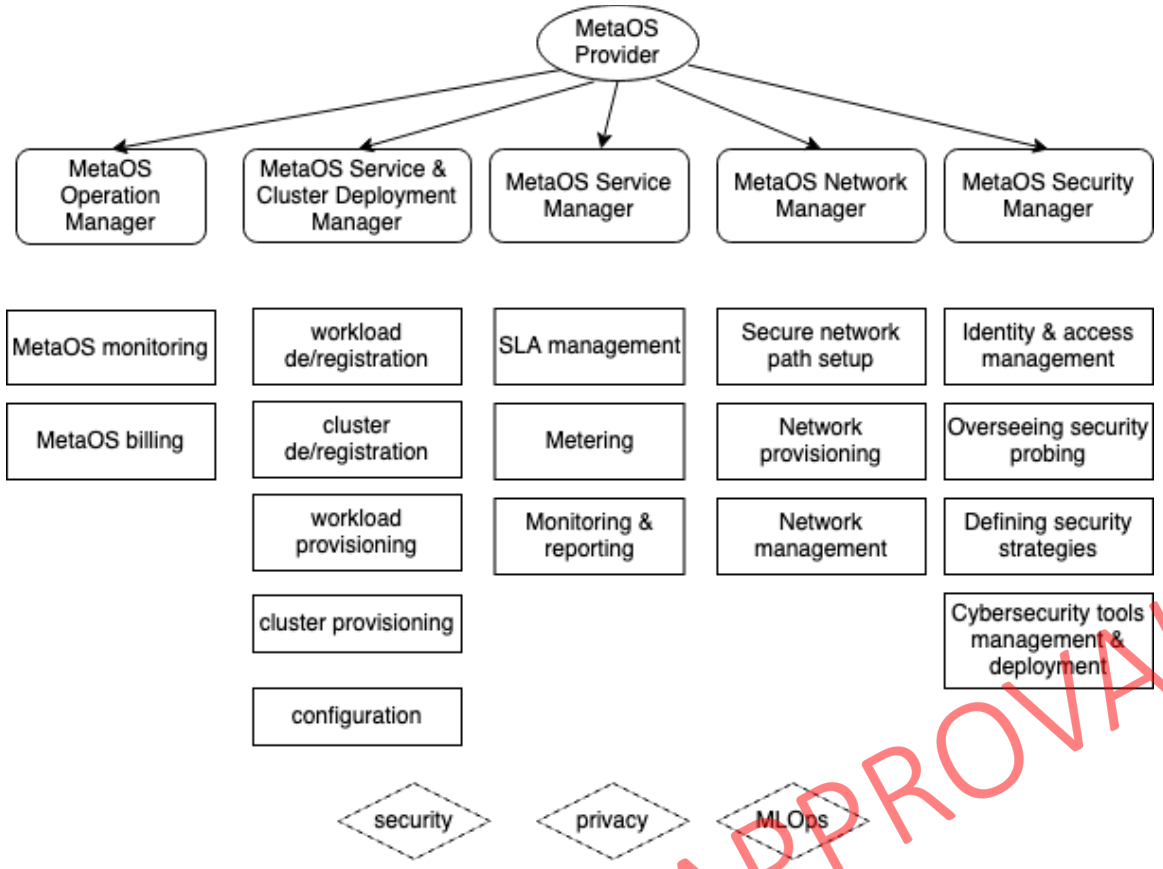

Figure 26: Meta-OS provider subroles, activities and aspects

The Meta-OS provider role is further analyzed in subroles with relevant activities and aspects as shown in Figure 26. The activities of each subrole are drawn below the relevant subrole. Also, the aspects apply to all depicted elements. Hence, the Meta-OS provider includes the following subroles:

- Meta-OS operations manager, whose main goal is to ensure that the meta-OS is functioning properly. The activities of this subrole include:
  - Meta-OS monitoring, including metering capability and monitoring of the meta-OS usage and performance, as well as relevant report generation
  - Meta-OS billing including the definition of billing strategies for meta-OS consumers
- Meta-OS service & cluster deployment manager, who aims to ensure that consumers' workloads or clusters will be properly deployed and provisioned in the meta-OS. The activities of the Meta-OS service & cluster deployment manager include:
  - workload de/registration, realized as workload initialization on the meta-OS after consumers' relevant request for registration, as well as removal of the workload from the meta-OS repos, again after relevant request of the owner (consumer) or decision of this subrole, as a result of e.g. breaking meta-OS terms of use.
  - cluster de/registration, similar to workload de/registration, but for clusters.
  - workload provisioning, ensuring that deployed workloads are accessible by the eligible users and/or roles.
  - cluster provisioning, which includes ensuring that registered clusters are automatically available in the meta-OS and defining rules for the orchestration of workloads on top of them, e.g. as part of a restricted set of clusters or as part of the whole meta-OS.
  - configuration, which refers to configuration options for meta-OS service updates, upgrades and onboarding of new clusters into the meta-OS.
- Meta-OS service manager, who is responsible for the meta-OS workload's lifecycle. Their activities include:

- SLA management, including policies' definition, monitoring and enforcement
- PRESS compliance, including enforcement of GDPR and potential customer-driven privacy policies' definition and enforcement
- Metering, i.e., providing a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts).
- Monitoring & reporting, referring to discovery, tracing, monitoring and usage or performance report generation.

- Meta-OS network manager, whose main goal is to ensure that secure and reliable network paths are established for communication of microservices or other workloads across the meta-OS. The activities of this subrole include:
  - Secure network path setup, e.g., by defining the rules for the construction of network paths, which should fulfil customer-driven requirements and ensuring that requested network connections can be automatically set up
  - Network provisioning, which includes the configuration and delivery of network services, e.g., load balancing.
  - Managing network elements and capabilities provided in the meta-OS by meta-OS partners, including e.g., time sensitive networking (TSN) capabilities' integration, private 5G networks.
- Meta-OS security manager, whose main aim is to ensure end-to-end security in the meta-OS ecosystem. Their activities include:
  - Managing identity and access management solution
  - Overseeing security probing, monitoring and reporting about the meta-OS workloads at runtime
  - Defining strategies, rules and tools for inducing security picks during the meta-OS DevOps and ZeroOps lifecycle
  - Managing and deploying tools that ensure cybersecure AI operations of the meta-OS.

Moreover, *security*, *privacy* and *MLOps* are identified as crosscutting aspects, i.e., behaviors or functionalities which need to be coordinated by the *MetaOS Provider* and must be supported in the meta-OS.

5.2.2 Meta-OS consumer

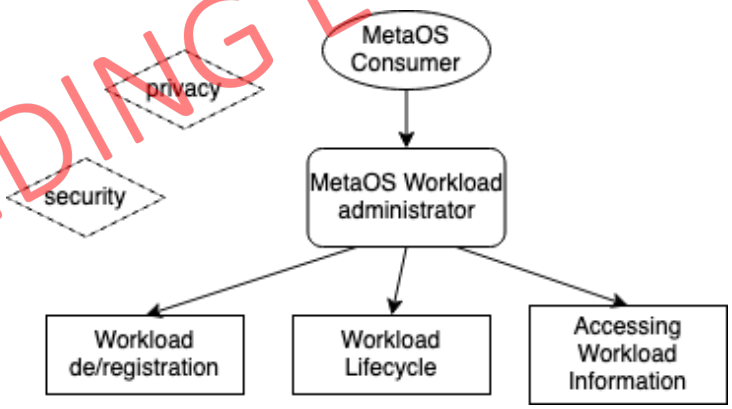

Figure 27: Meta-OS Consumer subroles, activities and aspects

The Meta-OS consumer refers to users of the meta-OS. The definition of its subroles and aspects are depicted in Figure 27. This role includes the following subroles:

- Meta-OS workload administrator, who is the entity owning or managing a workload that is desired to be executed on the meta-OS continuum. An example of this role would be an application vendor, wishing to provide their application to their customers (application end-users) via the Software-as-a-Service (SaaS) model. The activities of this role include:
  - Verify and de/register workloads in the meta-OS, i.e., making the relevant request to the meta-OS provider

- Selecting and using meta-OS services, related to managing their workload's lifecycle, definition of their workload requirements, etc.
- Accessing monitoring information about their workloads' usage and performance
- Accessing accounting information about their workloads.

The crosscutting aspects for this role include *security* and *privacy*.

5.2.3 Meta-OS partner

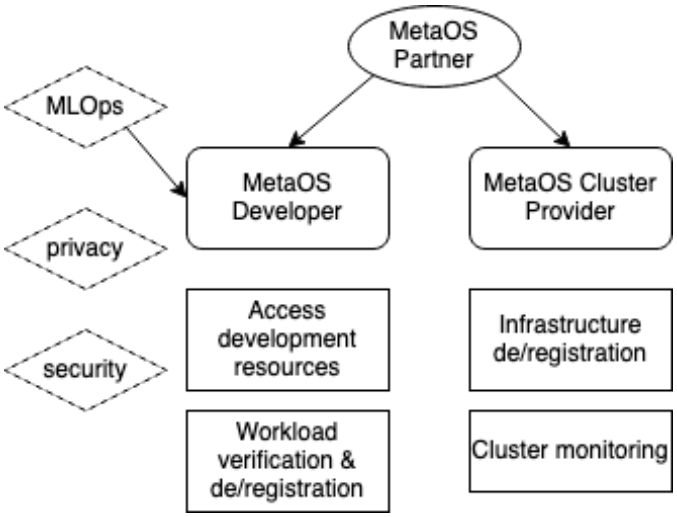

Figure 28: MetaOS Partner subroles, activities and aspects

The Meta-OS partner refers to users of the meta-OS. Its subroles, activities and aspects are depicted in Figure 28. This role includes the following subroles:

- Meta-OS developer, who builds software or "workloads" using the NEMO API, SDK and tools available through the meta-OS. The workload could refer to horizontal services, which are domain-independent services and are aimed to provide some basic and common functions extending NEMO capabilities. These are considered as "plugins" in the NEMO meta-OS and could be used for developing applications on top of NEMO or enhancing/extending user's experience in NEMO. In addition, the workload could refer to vertical services, i.e., domain-specific applications. The activities of this subrole include:
  - Request and access NEMO development resources, such as API, SDK and tools
  - Verify and de/register workloads, after their development has been completed.
- Meta-OS cluster provider, who refers to infrastructure owners who wish to make their resources available as nodes/clusters of the meta-OS. The infrastructure is conceived as physical or virtual compute resources, that will be made available as one or more clusters in the NEMO meta-OS. The activities of this role include:
  - Requesting de/registration of own infrastructure into the meta-OS
  - Monitoring of the cluster activity, e.g., in terms of usage or performance with respect to defined policies and agreements between the meta-OS partner and the meta-OS provider.

The crosscutting aspects for this role include *security*, *privacy* and *MLOps*.

5.3 Logical view

The first version of the logical view is depicted in Figure 29. Updates to it are expected as the components' roles and interactions get more mature and concrete. The figure depicts the main logical entities identified so far and high-level dependencies.

In particular, the metaOS, *ClusterShell*, *Cluster*, *ClusterSet* and *Node* entities represent the elements defined in the network view, while the *User* entity represents the three user types defined in the user view. Moreover, the *Workload* entity represents an application, (NEMO core) component, plugin or microservice, whose functionalities can be described in a *WorkloadDocument* and be exposed as

Resources. The *Resource* entity represents the programmatic representation for such functionality exposure. A *Workload* may bear a *WorkloadOperation*, i.e., a registration, deployment or migration operation, as envisaged so far. A *WorkloadOperation*, such as deployment or migration, may be triggered as a result of an *OperationDecision*. The deployment and execution of the Workload should be governed by defined Policies. A *Policy* may consist of a set of Service-Level Agreements (*SLAs*), which are described by a set of Service-Level Objectives (*SLOs*). The *SLO* includes a property and a target that should be met. So, in case this target is not met or achieved, a *WorkloadEvent* occurs. This could be, for example, an SLA, security or operation-related event. The performance against defined SLOs is monitored through *Metering* objects (such as metrics). Last, but not least, the *Accounting* entity provides accounting information for given workloads, clusters and users.

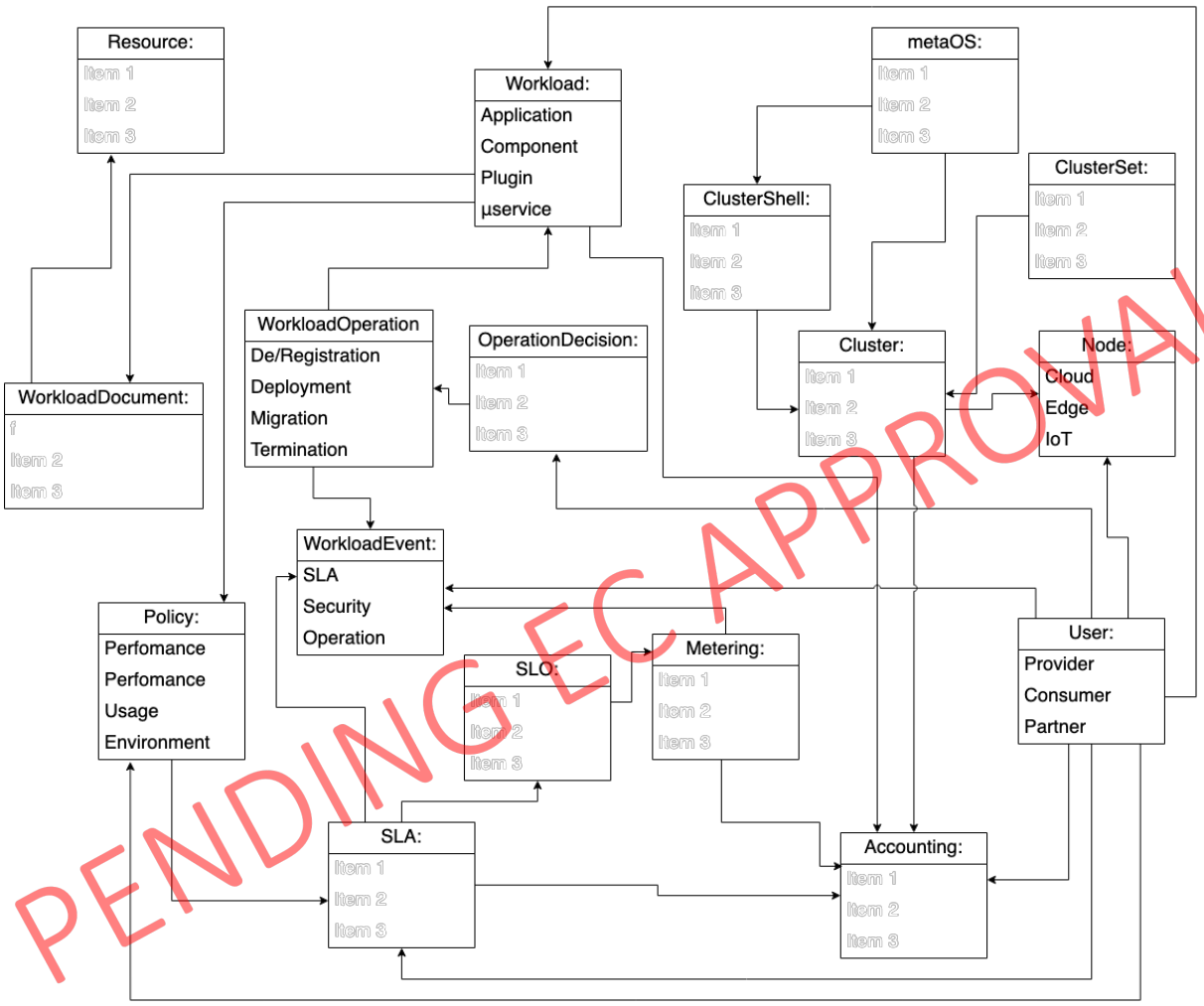

Figure 29: Logical view of the NEMO metaOS architecture

5.4 Operational view

The Operational View of the NEMO architecture is provided in the form of the NEMO Use Case scenarios and descriptions. Thus, it has been provided in D1.1 [32].

5.5 Functional view

Following the NEMO functional stack vision, the functional view of the NEMO metaOS architecture is depicted in Figure 30.

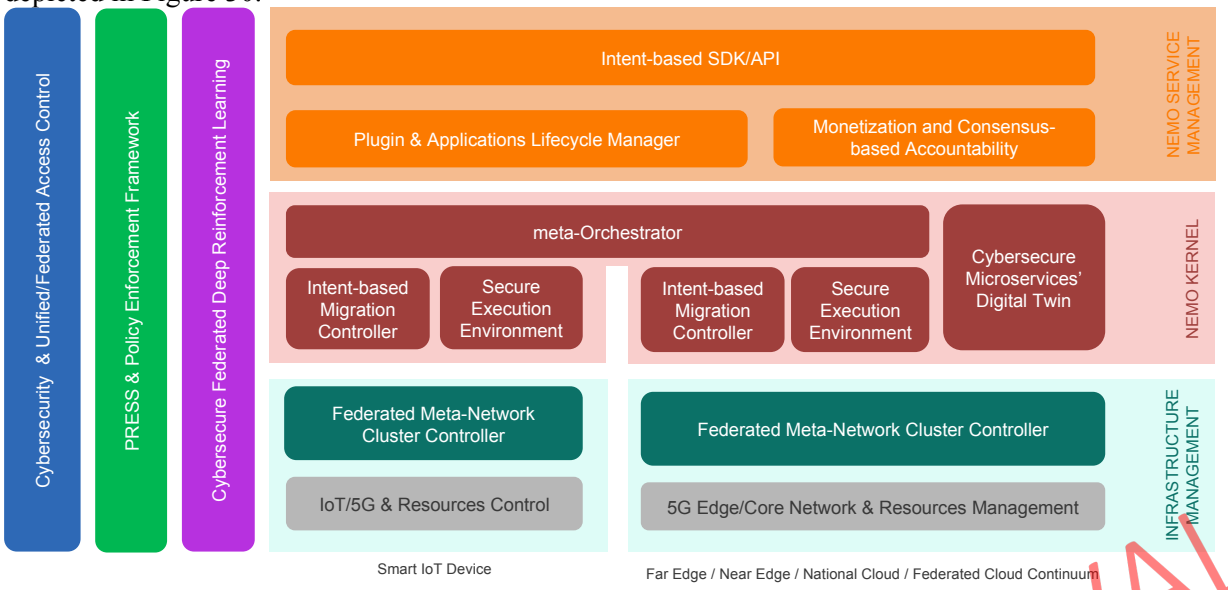

Figure 30: The functional view of the NEMO metaOS architecture.

The functional view identifies three horizontal layers:

- The *infrastructure management* layer, which refers to federated management and orchestration of network resources. This layer integrates third party solutions for network and resource management. The NEMO contribution in this layer is realized through the Federated Meta-Network Cluster Controller (mNCC), which aims to provide transparent network connectivity within the metaOS, supporting application-driven requirements for latency, multi-path connectivity and service isolation through network micro-slices. The mNCC integrates Time Sensitive Network (TSN) to support deterministic communication between wireless and fixed devices in the context of private 5G networks.
- The *NEMO kernel*, which includes the core NEMO components supporting workload scheduling and execution across the continuum. In this layer, the meta-Orchestrator has a critical role, undertaking the workload scheduling and placement across the federated clusters, guaranteed that agreed Service Level Objectives (SLOs) are met. The meta-orchestrator (MO) provides a meta control plane on top of the available clusters of nodes and is assisted by the Intent-based Migration Controller (IMC) in the execution of the service operation decision, such deployment, migration, scaling, etc. Secure workload execution via unikernels is supported by the Secure Execution Environment (SEE). Moreover, the Cybersecure Microservices- Digital Twin (CMDT) caters for traceability of workloads during their metaOS lifecycle.
- The *NEMO Service Management*, which acts as a DevZeroOps layer offering full-stack automated operations, greatest flexibility, improved developers' productivity and direct monetization and sustainability. The NEMO Plugin & Applications Life-Cycle Manager (LCM) aims to enable deployment of workloads, applications or plugins, onto NEMO, through the user's space. The Intent-based API/SDK (IAS) enables and facilitates third parties to develop and deploy on top of NEMO, exposing NEMO functionalities as programmatic APIs and providing software libraries for facilitating NEMO-compliant development. Moreover, Monetization and Consensus-based Accountability (MOCA) aims to support monetization and accountability for both the applications and plugins running on NEMO, but also for network resources integrated into the NEMO infrastructure. Overall, this layer supports ZeroOps deployment, providing service provisioning, resource configuration, applications life cycle

management and automated response to infrastructure issues in a multi-user, multi-operator, multi-tenant environment.

The NEMO cross-cutting functions include:

- Cybersecurity and unified/federated access control, which ensures the security of metaOS operations across the metaOS layers, in the context of cloud native cybersecurity, federated access and identity management across the metaOS components, as well as secure and encrypted inter-process communication.
- Data & Services Policy Compliance Enforcement via multi-faced, policies able to cope with the different aspects of the applications life cycle (security, privacy, costs, environmental impact, etc). These functions ensure that PRESS rules and GDPR, as well as user-defined rules, are respected across the metaOS layers and components.
- Cybersecure Federated MLOps, which provides inherent integration of AI operations and services into the metaOS, yielding AI-based decisions and or controls alongside the metaOS. This function aims to support the complete Machine Learning (ML) lifecycle, e.g., from ML development and training to serving and infeperformed within metaOS components, ensuring AI cybersecurity.

The following subsections provide detailed description of the role of and functionality supported by each NEMO component for the functional layers and cross-cutting functions identified. Moreover, the interactions with other NEMO components or external users, as well as the requirements (as of D1.1) covered by each of the NEMO components are also presented.

5.5.1 NEMO Infrastructure Management

5.5.1.1 Federated Meta-Network Cluster Controller

5.5.1.2 Description

Meta Network Cluster Controller (mNCC) is an automated, self-organizing entity conceived to facilitate the dynamic creation and self-healing of fog IoT network clusters in the edge-cloud continuum connectivity. This module is closely linked to the Cybersecure Federated Deep Reinforcement Learning (CF-DRL) module and the meta-Orchestrator, which provides information on the Cybersecure Microservices Digital Twin (CMDT) and the PRESS & Policy Enforcement Framework (PPEF) modules.

The mNCC module is designed to address various use cases (UCs) related to connectivity and Cloud/Edge/Fog IoT deployments. The following Living Labs' use cases are specifically defined in the project's Description of Action (DoA) to benefit from mNCC:

- NEMO Integration Infrastructure Technology Lab
- Smart Farming Use Case & Living Lab
- Smart Media/ City & XR Use Cases & Living Lab
- NEMO multi-Living Labs Federation

Furthermore, other use cases, such as UC4 (Smart Manufacturing & Industry 4.0 Use Cases & Living Lab), may also require the deployment of additional features supported by this module.

To fulfill the aforementioned functionalities, the mNCC comprises seven subcomponents, as depicted in Figure 31. The core component of mNCC is the *Connectivity Controller*, which establishes point-to-point or point-to-multipoint overlays to connect the various clusters that form the NEMO's network functional core. This controller is intended to be based on the L2S-M solution [33], which provides additional networking functionalities to the standard Kubernetes Container Network Interface (CNI) approach. L2S-M enables the management of virtual networks in Kubernetes using Software-Defined Networking (SDN) and facilitates the attachment of workloads (pods) to "OpenStack-like" virtual networks.

The *Connectivity Controller* receives information from three other modules: the *Network Domain*, the *Compute Domain*, and the *Communication Endpoints*. The *Network Domain* module collects and integrates network topology information, creating an internal map of network resources and their

availability. Similarly, the *Compute Domain* module focuses on the computing capabilities of the pods and identifies potential new requirements or situations of over-deployment. The *Communication Endpoints* module checks the various endpoints and provides information on their properties, if available.
These three modules are also interconnected with the *Technology Adaptors* module, which serves as a southbound interface to interact with different network and compute technologies, providing the final connectivity substrate.
To establish different types of connectivity services (e.g., micro-slices, multipath connections, etc.), the *Connectivity Controller* relies on network requests. The *Intent-Based System* module translates management requests received in the form of intents into service data models that can be understood by the controller.
Finally, to expose its results and state to the rest of the NEMO system, this ecosystem incorporates the *Network Exposure* module. This module utilizes the IETF ALTO technology [RFC7285], which gathers transport and link-level information from various routing protocols (agnostic to the specific transport protocol used to retrieve the information). ALTO integrates the network topology obtained from lower levels and exposes it to service layer applications, providing an updated view of the network state.
Figure 31 illustrates the various subcomponents and interactions within the mNCC module.

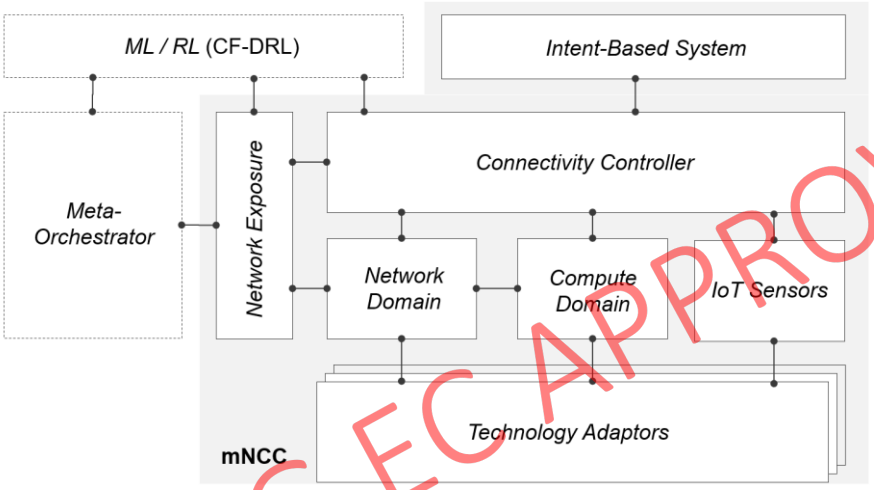

Figure 31: Initial design of mNCC

5.5.1.3 Main functionalities

Taking as basis the needs of use cases and living labs, we have performed an initial / preliminary analysis of expected features for mNCC. Further interactions could be required with WP2 for the refinement of features and availability needs.
In our pursuit of network component specifications, it is crucial to consider the specific needs of real-world use cases. The following list outlines the requirements for each use case, which will serve as key factors in informing the development of the network component.
For the Smart Industry use case, high-speed and ultra-low latency communication capabilities, particularly for Time Sensitive Networks (TSN), are essential to support critical industrial processes. Additionally, the network component should have the capacity to handle massive data uploads to the edge or cloud, enabling efficient storage, processing, and analysis.
In the context of Smart Farming, real-time video analysis capabilities are necessary to facilitate timely monitoring and decision-making in agricultural settings. The network component should also offer flexibility in deploying training jobs across edge and cloud resources to support machine learning tasks. Furthermore, the ability to migrate services within and between clusters is crucial for seamless operations and resource optimization.

For the Smart Energy use case, the network component should provide reliable edge and cloud connectivity. This connectivity is necessary for applications such as monitoring CCTV cameras and conducting machine learning training for energy-related systems.
In the realm of Smart Media and Extended Reality (XR), the network component must ensure assured high bandwidth to meet the demands of media streaming and XR applications, thus delivering an immersive user experience. Support for multipath communication is also crucial to enhance network reliability and performance. Furthermore, the network component should facilitate microservice migration to enable dynamic deployment and scalability of media and XR services.
In order to satisfy these requirements, the following functionalities have been identified; Support of (micro-)slices as connectivity service between far-edge/edge/cloud, multipath, service migration, monitoring and accounting, capability exposure and micro-slice intents.
The mNCC will provide support for (micro-)slices as a connectivity service, enabling seamless communication between far-edge, edge, and cloud environments. It will ensure the fulfilment of assured Service Level Objectives (SLOs) for performance metrics such as latency, throughput, and reliability. The specific values for these SLOs will be determined during the specification process. Additionally, the mNCC will address the unique challenges associated with far-edge connectivity to ensure efficient and reliable communication.
The mNCC should support multipath communication, allowing for the efficient utilization of network resources and enhanced reliability. The main complexity of this functionality lies in handling replicas and in the elimination of data copies, ensuring that redundant information is managed appropriately within the network. This functionality will contribute to improved data transmission efficiency, fault tolerance, and load balancing.
The mNCC will also enable seamless service migration to support dynamic changes in network requirements. It will employ a "make before break" approach for network adaptation, ensuring a smooth transition without interrupting ongoing services. The mNCC will facilitate mobility and traffic redirection, allowing services to be relocated without disruption and enabling efficient resource allocation and optimization.
The incorporation of robust monitoring capabilities is considered in mNCC, in order to gather network performance data and enable efficient accounting and resource management. This functionality will provide valuable insights into network behaviour and facilitate accurate resource allocation and utilization analysis. Exposing the capabilities and functionalities of the managed network to other NEMO management modules will facilitate interoperability and enable effective coordination and collaboration within the network ecosystem. And last, but not least, the mNCC will support the definition and enforcement of (micro-)slice intents, which specify the desired behaviour and requirements for a particular network slice. This functionality will enable fine-grained control and customization of network behaviour, ensuring that each slice meets its specific needs and objectives.
By incorporating these functionalities into the mNCC, we aim to develop a powerful and versatile network cluster controller that addresses the complex demands of the NEMO management systems. The mNCC will provide enhanced connectivity services, support critical time-sensitive applications, enable efficient multipath communication, facilitate seamless service migration, and offer monitoring, accounting, capability exposure, and (micro-)slice intent management capabilities.

5.5.1.4 Interactions

This is an initial understanding on interactions of mNCC with other components of the NEMO architecture. Further refinements on architecture design or use case definition can motivate changes in this initial approach.

Interacting with	Interaction type	Description of interaction
Cybersecure Federated Deep Reinforcement Learning (CF-DRL)	Input	The CF-DRL module serves to enhance the intelligence of the mNCC (more concretely, with the Connectivity controller) by enabling network decision-making based on anticipated scenarios and "what-if" analyses. The primary objective of this interaction is to proactively

Interacting with	Interaction type	Description of interaction
		anticipate network changes before they become critical. By doing so, it enables zero-latency network modifications, enhancing capabilities in advance and reducing unnecessary escalation and associated costs over prolonged periods. This integration between CF-DRL and mNCC enables dynamic network adjustments to optimize performance and resource allocation, ensuring efficient utilization of network resources and avoiding potential bottlenecks.
Meta-Orchestrator	Input/Output	The meta-Network Cluster Controller (mNCC) interacts with the meta-orchestrator in both input and output capacities. In terms of input, the mNCC receives network-related information from the meta-orchestrator, including data on network availability, performance metrics, and constraints. This information is crucial for the mNCC to make informed decisions regarding workload placement, resource allocation, and optimization of orchestration processes. Also, it receives instructions and requests that pertain to network configurations and adjustments necessary to meet the requirements of orchestrated workflows. The mNCC executes these instructions, ensuring that the networking environment efficiently supports the orchestration and operation of distributed computing workflows. On the other hand, as output, the mNCC exposes the different network performance metrics and the network context to allow the meta-Orchestrator to have an update view of the network managed.
PRESS & Policy Enforcement Framework	Output	The mNCC also exposes the network context to the PRESS & PEF module, so this component could evaluate during runtime if any of the connections violates the policies defined and, in that case, it communicates this to the meta-Orchestrator in order to taking corrective actions.
Transport network (5G network, MANO, RAN)	Input/Output	The mNCC must also interact with the underly network through the different technology adaptors. The mNCC will export the slice control and the network decision requested, using the network programmability to archive the connectivity goals for NEMO. As an input, the mNCC will receive the topology and compute context from different routing or state protocols. This information allows the mNCC to have an update and realistic network view, that will be used to taking the different decisions.
VIM	Input/Output	In a same way as with the transport networks, the mNCC will interact with the NEMO's VIM to request for network updates. In this case the different requests will depend on the best places to instantiate the networking resources on the fog or to ask for cluster modifications. As an input, the mNCC will receive a view of how the clusters are distributed and how much capacity they have.

Interacting with	Interaction type	Description of interaction
		This information will be used to evaluate the best way to deploy the different services.

5.5.1.5 Requirements

The mNCC contributes to addressing a set of the defined NEMO functional and non-functional requirements, as defined in D1.1 [32]. Table 13 and list those functional and non-functional requirements, respectively, following the numbering and description adopted in D1.1 and justify how the mNCC contributes to their satisfaction.

Table 13: NEMO functional requirements addressed through the mNCC

Requirement ID	Requirement description	Requirement satisfaction
NEMO_FR01	The platform must provide access to measurements.	NEMO is able to grant access to the users to the different metrics that has been recollected.
NEMO_FR02	The platform must provide options to manage/view sensors/devices.	mNCC counts with a way to extract and configure the parameters of the network sensors/devices monitored.
NEMO_FR07	The platform should support monitoring of SLOs, e.g., related to energy consumption or CO2 emissions.	mNCC has a defined process and protocol to extract SLOs metrics and a way to integrate them to check the different SLOs.
NEMO_FR08	The platform must respect data sovereignty and privacy requirements.	mNCC do not use personal data or third-party information that it is not strictly necessary.
NEMO_FR09	The platform must support collection of monitoring data, such as the weather and plant conditions.	mNCC is able to export data from the sensors and so on to collect and integrate them.
NEMO_FR011	The monitoring devices must support network connectivity.	All NEMO devices (both network transport devices and edge-devices) support network connectivity.
NEMO_FR013	The platform should be able to perform alternative scheduling or geographical distribution of smart farming services based on user goals.	mNCC redistributes services and load-balances on the smart farming use case to adapt it to the different needs presented by each timing and geography situation.
NEMO_FR23	The platform must provide access to collected data.	NEMO is able to grant access to the users to the different data that has been recollected.
NEMO_FR27	The platform has the capability to monitor the real-time data from the sensors deployed in the grid.	mNCC has a way to extract and share the parameters of the sensors/devices monitored in real-time.
NEMO_FR29	High-tech power sensors should be useful to elaborate on new strategies, in order to improve the power quality in a secure way.	mNCC is able to integrate and share the information extracted from the high-tech power sensors to make it useful.

Requirement ID	Requirement description	Requirement satisfaction
NEMO_FR30	Based on sampled data, phasors are calculated with high precision and the synchronization process must be very fast. Indeed, innovative reconfiguration and self-healing schemas should rely on appropriate measurements.	mNCC is able to synchronize and reconfigure network devices in real-time, using the collected parameters to accelerate the process.
NEMO_FR55	The broadcaster must be able to monitor the signal quality and QoE parameters of the transmission to ensure streaming quality.	NEMO broadcaster obtains QoS and QoE metrics and is capable to use them to ensure the streaming quality.
NEMO_FR57	Control signals (voice and data) and audio/video return channels are to be transferred between the technical director location and the venue via the cloud network.	mNCC has a defined process to transfer control signals and audio/video between the technical location to the desired destiny by using the cloud network.
NEMO_FR58	NEMO must provide the adequate resources to the service provider to map these requirements onto the cloud network and perform accordingly.	NEMO system has the capability to provide a map with a performance's metrics representation to service providers.
NEMO_FR60	NEMO will be able to allocate and launch the required services/VNFs on a location basis.	NEMO is able to allocate and launch the required services/VNFs on a location basis.
NEMO_FR61	The service provider must be able to chain services/VNFs with the help of a service orchestrator.	The service provider is able to chain services/VNFs with the help of a service orchestrator.
NEMO_FR67	Max. end-to-end network latency (RTT) - It comprises the latency of the whole network path excluding end devices on-site (like the network gateway or HW video coder) <= 50 ms.	Nemo can guarantee a maximum end to end latency for all communications.
NEMO_FR68	Max latency of end-to-end signal transport (video, audio and control data) - it comprises the latency of the whole signal path including converting of end devices on-site and media-specific VNFs). Maximum E2E latency one way for video and audio: <= 500 ms Max. E2E latency for return video (one way): <= 500 ms (Typically uses less bandwidth because of low-resolution proxy transfer) Max. end-to-end latency for intercom (if needed): <= 100 ms (according to ITU G.114).	NEMO system provides the required data (both for media and communications) with a maximum latency guaranteed for the
NEMO_FR75	Network will support diverse devices (wearables, AR/VR headsets) with different performance (e.g., high throughput, low latency and massive connection densities).	mNCC is able to interact with different devices being the system technological-agnostic. These devices may have different network requirements or performances, but that do not

Requirement ID	Requirement description	Requirement satisfaction
		affect to the capability to work with them.
NEMO_FR76	The platform must ensure the interoperability with external systems (i.e., multi sensorial stimuli system).	mNCC is able to interact with different external systems and devices using well-known and standardized protocols.
NEMO_FR77	The platform components involving direct interaction with the end-users should be quick to respond to the users' actions.	mNCC is able to interact with the end-users in near real-time, answering user's request with a maximum latency guarantied.
NEMO_FR83	Network must support diverse devices (wearables, AR/VR headsets) with different performance (e.g., high throughput, low latency and massive connection densities).	mNCC can interact with different devices being the system technological-agnostic. These devices may have different network requirements or performances, but that must not affect to the capability to work with them.

Table 14: NEMO non-functional requirements addressed through the mNCC

Requirement ID	Requirement description	Requirement satisfaction
NEMO_NFR01	The NEMO platform must respect security and privacy requirements.	NEMO platform communications and processes follow required security standards and good practices.
NEMO_NFR02	NEMO should support High Availability features.	NEMO platform has a High Availability extension or at least has the APIs to interact with a potential HA extension.
NEMO_NFR04	The NEMO platform should be flexible and scalable in the sense of exploiting available resources according to set goals. It should be scalable in the sense of providing additional resources when computationally heavy tasks are initiated.	NEMO platform scales with the traffic and the connected devices providing enough connectivity to all of the network clients but also avoiding an over-deployment when the requested capability is low.
NEMO_NFR05	Secure communication of sensitive data related to the infrastructure should be provided.	mNCC does not provide sensitive data by insecure channels nor to unknown users.
NEMO_NFR06	The 5G availability should allow achieving better performances in data transmission.	mNCC 5G integration optimizes data transmission improving it compared to traditional network paradigms.
NEMO_NFR12	Data shall be consistent, reliable, transparent and accessible only to authorized users.	mNCC does not provide inconsistent or unknown data and it sends it just to authorized users.
NEMO_NFR13	Store data in a safe and tamperproof manner.	mNCC counts with a safe way to store data in order to ensure consistency reliability and privacy.
NEMO_NFR14	The platform must ensure the traceability for the operator.	mNCC has a defined process to check the traceability for the different changes realized and it is also able to export it to the network operator when it is required.

Requirement ID	Requirement description	Requirement satisfaction
NEMO_NFR15	The platform must have capabilities of a monitoring system.	mNCC has the capability to monitoring network services and systems, recollecting performance metrics and creating alerts if some network segment does not work as expected.
NEMO_NFR18	The platform must provide mechanisms for security and data privacy.	mNCC counts with a safe way to exchange data and a defined process to ensure the privacy to the network platform.
NEMO_NFR19	The platform should support high availability deployments.	mNCC has the ability to work with high availability modules and requirements.

5.5.2 NEMO Kernel

5.5.2.1 Meta-Orchestrator

5.5.2.1.1 Description

The meta-orchestrator component is designed as a highly advanced and intelligent open-source software system. Its primary goal is to enable the decentralization and distribution of computing workflows across the IoT to Edge to Cloud Continuum. By acting as a central orchestrator, it manages the coordination and execution of complex distributed systems while addressing the challenges posed by their increasing complexity and heterogeneity.

The meta-orchestrator provides a comprehensive and holistic approach to orchestration by considering various aspects of distributed computing workflows. It conducts an in-depth investigation of the structure and application programming interfaces (APIs) of various micro-schedulers and local orchestrators. This analysis allows the meta-orchestrator to seamlessly integrate and coordinate with different components within the distributed system architecture.

The intelligence capabilities of the meta-orchestrator are at the core of its decision-making process. It considers crucial parameters such as migration time, downtime, and overhead time when orchestrating computing workflows. By evaluating these parameters, the meta-orchestrator ensures that workflows are orchestrated in a manner that minimizes disruption and maximizes efficiency.

In addition to the fundamental parameters, the meta-orchestrator also evaluates a wide range of functional and non-functional requirements. Functional parameters include network and resource availability via Intents, which are essential considerations for successful workflow execution. By assessing these factors, the meta-orchestrator ensures that computing resources are allocated appropriately and optimally.

Non-functional requirements play a crucial role in the decision-making process of the meta-orchestrator as well. It considers policies, energy efficiency, CO2 footprint, and FinOps requirements such as networking and hosting costs. By considering these non-functional requirements, the meta-orchestrator enables the optimization of workflows based on multiple criteria, including environmental impact and cost-effectiveness.

The meta-orchestrator acts as a central hub for managing distributed systems across the IoT to Edge to Cloud Continuum. It provides a high-level view of the system, allowing for efficient coordination and resource management. By leveraging the capabilities of micro-schedulers and local orchestrators, the meta-orchestrator ensures that computing workflows are distributed effectively and executed on the most suitable resources available within the continuum.

The meta-orchestrator's intelligent decision-making capabilities not only optimize resource utilization but also contribute to the scalability and adaptability of the distributed system. It can dynamically adjust the allocation of resources based on changing conditions and requirements. This adaptability is especially valuable in scenarios where the system experiences fluctuations in workload, availability of resources, or environmental conditions.

Moreover, the meta-orchestrator fosters interoperability and compatibility across different components and systems within the distributed architecture. It provides standardized interface that allow for seamless integration and communication with other components. This interoperability ensures that the distributed system operates cohesively and efficiently, even when composed of heterogeneous and diverse components.

The high-level design of the NEMO meta-Orchestrator is depicted in Figure 32, identifying the main subcomponents. The role and internal interactions among them are provided in Table 15.

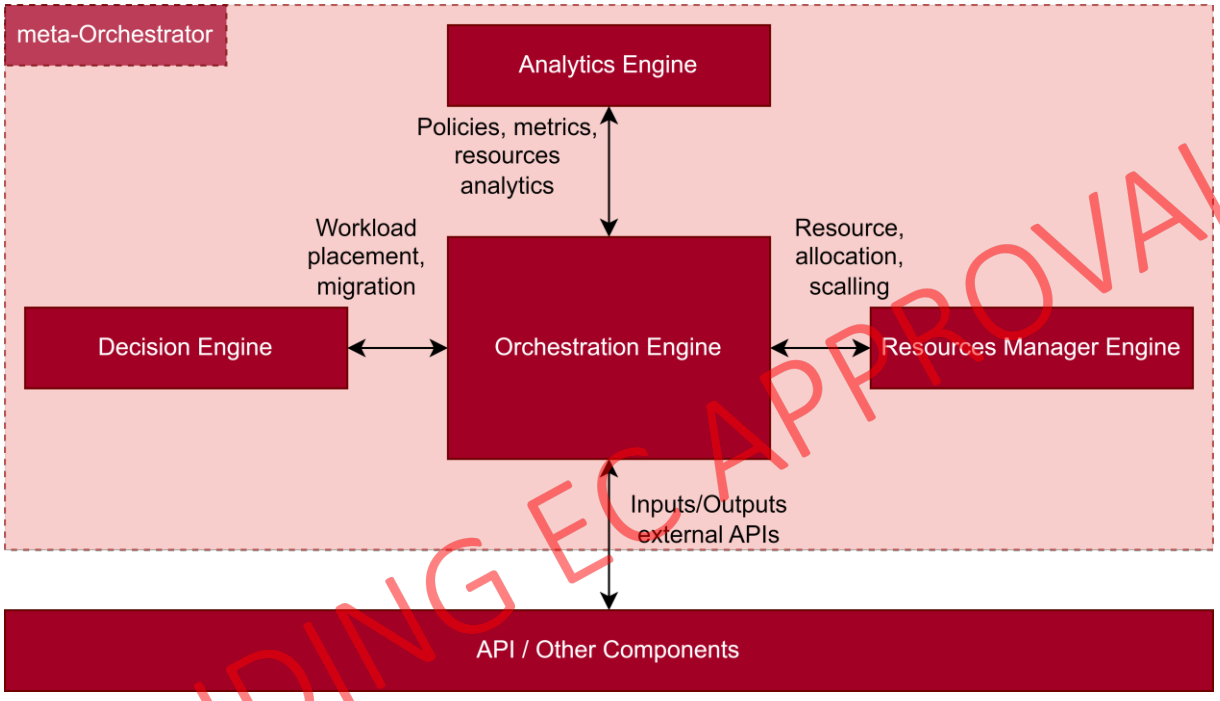

Figure 32: High-level design of the NEMO meta-orchestrator

Table 15: Analysis of the meta-orchestrator elements

Meta-orchestrator sub-component	Role and internal interactions
Orchestration Engine	The orchestration engine interacts with other internal components both as an input and output. As an input, it receives information from the resource manager, decision engine, analytics engine, and integration component. This information includes resource availability, workload characteristics, policies, and data related to resource utilization and performance. As an output, the orchestration engine provides instructions and requests to other components for resource allocation, workload placement, and integration with external tools and frameworks. It serves as the central component within NEMO, abstracting complexities and ensuring efficient management and control of the distributed computing workflow.

Meta-orchestrator sub-component	Role and internal interactions
Resource Manager	The resource manager interacts with the orchestration engine as an output by providing information about the complete lifecycle of resources, including provisioning, scaling, monitoring, and deprovisioning. It also receives instructions and requests from the orchestration engine as an input for resource allocation, scaling, and deprovisioning based on the orchestrated workflows' requirements. The resource manager effectively manages the allocation and deallocation of resources to ensure optimal utilization and availability for the computing workflows orchestrated by NEMO.
Decision Engine	The decision engine interacts with the orchestration engine as an output by providing policies for policy enforcement, cost optimization, and workload placement. It receives instructions and requests from the orchestration engine as an input, considering workload characteristics, resource availability, and performance metrics. The decision engine plays a vital role in providing intelligent decision-making capabilities, ensuring efficient orchestration of computing workflows while adhering to specified policies and minimizing operational costs.
Analytics Engine	The analytics engine interacts with the orchestration engine as an output by providing insights and recommendations generated from collected and analyzed data related to resource utilization, performance, and other relevant metrics. It receives feedback and updates from the orchestration engine as an input, regarding the effectiveness and impact of the suggested optimizations. The analytics engine focuses on optimizing the overall orchestration process by identifying potential bottlenecks, improving resource allocation strategies, and enhancing the performance of the distributed computing environment.
Integration Component (API/Other components)	The integration component interacts with the orchestration engine as an output by providing connectors and APIs for seamless integration with different orchestration tools and frameworks. It receives information and requirements from the orchestration engine as an input for compatibility and interoperability with various tools and frameworks. The integration component enables the meta-orchestrator to leverage existing infrastructure and frameworks efficiently, simplifying the adoption and deployment of NEMO within diverse computing ecosystems.

5.5.2.1.2 Main functionalities

The following functionalities enable the meta-orchestrator to effectively coordinate, manage, and optimize computing workflows across the distributed system, promoting decentralization and efficient resource utilization.

- Workflow Orchestration: The meta-orchestrator is responsible for coordinating and managing the execution of computing workflows across the IoT to Edge to Cloud Continuum. It ensures that tasks and components within the workflow are executed in the most efficient and effective manner possible.
- Intelligent Decision-Making: Leveraging advanced intelligence capabilities, the meta-orchestrator makes intelligent decisions when orchestrating computing workflows. It considers parameters such as migration time, downtime, and overhead time to optimize the orchestration process.
- Parameter Evaluation: The meta-orchestrator evaluates various parameters to optimize workflow orchestration. This includes assessing network and resource availability to allocate

resources effectively. It also considers non-functional requirements such as policies, energy efficiency, CO2 footprint, and FinOps requirements like networking and hosting costs.

- Integration with Micro-Schedulers and Local Orchestrators: The meta-orchestrator conducts an in-depth investigation of the structure and APIs of micro-schedulers and local orchestrators. It integrates with these components to seamlessly coordinate and distribute computing workflows across the continuum.
- Resource Management: The meta-orchestrator efficiently manages and allocates computing resources across the distributed system. It ensures that resources are utilized optimally, considering factors such as workload, availability, and non-functional requirements.
- Scalability and Adaptability: The meta-orchestrator supports the scalability and adaptability of the distributed system. It dynamically adjusts resource allocation based on changing conditions and workload fluctuations, enabling the system to respond to varying demands effectively.
- Interoperability: The meta-orchestrator fosters interoperability and compatibility across different components within the distributed architecture. It provides standardized interfaces for seamless integration and communication with other system components.
- Domain Migration: The meta-orchestrator enables the migration of workflows across different domains within the IoT to Edge to Cloud Continuum. It facilitates the seamless transfer of computing tasks and data between different environments, ensuring continuity and efficiency.

5.5.2.1.3 Interactions

The meta-Orchestrator interacts with other NEMO components, as described in Table 16.

Table 16: Interactions of the meta-Orchestrator with other NEMO components and external entities

Interacting with	Interaction type	Description of interaction
Intent-based Migration Controller	Input/Output	The meta-orchestrator interacts with the Intent-based Migration Controller both as an input and output. As an input, the Intent-based Migration Controller provides migration intents, specifying requirements and constraints for migrating computing workflows across domains. The meta-orchestrator considers these migration intents during the orchestration process, ensuring seamless migration and continuity of workflows. As an output, the meta-orchestrator provides feedback and updates to the Intent-based Migration Controller regarding the status and progress of the workflow migration. This feedback helps the controller track and monitor the migration process and make necessary adjustments if required.
meta–Network Cluster Controller (mNCC)	Input/Output	The meta-orchestrator interacts with the meta-Network Cluster Controller (mNCC) both as an input and output. As an input, the mNCC provides network-related information, such as network availability, performance metrics, and constraints. The meta-orchestrator utilizes this information to make informed decisions regarding workload placement, resource allocation, and orchestration optimization. As an output, the meta-orchestrator provides instructions and requests to the mNCC for network-related configurations and adjustments based on the orchestrated workflows' requirements. The mNCC executes these instructions to ensure the networking environment supports the efficient orchestration and functioning of distributed computing workflows.

Interacting with	Interaction type	Description of interaction
Cybersecure Microservices' Digital Twin (CMDT)	Input/Output	The meta-orchestrator interacts with the Cybersecure Microservices' Digital Twin (CMDT) both as an input and output. As an input, the CMDT provides information related to the security and integrity of microservices within the distributed computing environment. This input helps the meta-orchestrator assess the security risks and vulnerabilities associated with different microservices and incorporate security measures into the orchestration decisions. As an output, the meta-orchestrator provides instructions and requests to the CMDT for security-related configurations, monitoring, and enforcement. The CMDT executes these instructions to ensure the cybersecure operation of microservices throughout the orchestration process.
Cybersecure Federated Deep Reinforcement Learning (CF-DRL)	Input/Output	The meta-orchestrator interacts with the Cybersecure Federated Deep Reinforcement Learning (CF-DRL) component both as an input and output. As an input, the CF-DRL component provides reinforcement learning models and insights related to security and risk management. The meta-orchestrator incorporates these insights into its decision-making process to enhance security measures, risk mitigation, and policy enforcement. As an output, the meta-orchestrator provides feedback and updates to the CF-DRL component regarding the impact and effectiveness of the security and risk management measures taken. This feedback helps the CF-DRL component refine its models and strategies, resulting in improved security and risk management within the orchestration environment.

5.5.2.1.4 Requirements

The meta-Orchestrator contributes to the NEMO functional and non-functional requirements, listed in Table 17 and Table 18, respectively, following the numbering and description adopted in D1.1.

Table 17: NEMO functional requirements addressed through the meta-Orchestrator

Requirement ID	Requirement description	Requirement satisfaction
NEMO_FR01	The platform must provide access to measurements.	The meta-orchestrator provides access to measurements by considering various parameters for orchestrating workflows. It assesses factors like migration time, downtime, and overhead time for decision-making.
NEMO_FR03	The platform must provide options to manage users.	The meta-orchestrator supports user management by considering functional and non-functional requirements, including network availability and resource allocation.
NEMO_FR07	The platform should support monitoring of SLOs, e. g., related to energy consumption or CO2 emissions.	The meta-orchestrator supports monitoring of SLOs like energy consumption and CO2 emissions, optimizing workflows based on environmental impact and efficiency.

Requirement ID	Requirement description	Requirement satisfaction
NEMO_FR08	The platform must respect data sovereignty and privacy requirements.	The meta-orchestrator respects data sovereignty and privacy by ensuring appropriate allocation of resources and communication while orchestrating workflows
NEMO_FR14	The platform should be able to perform alternative scheduling or geographical distribution of smart farming services based on user goals.	The meta-orchestrator enables alternative scheduling and distribution by intelligently orchestrating workflows based on user-defined strategies and goals.
NEMO_FR15	The Smart Farmer should be able to define strategies for the use of available resources.	The meta-orchestrator allows the end-user to define resource use strategies, guiding its decision-making process for orchestrating workflows.
NEMO_FR23	The platform must provide access to collected data.	The meta-orchestrator provides access to collected data by considering various parameters and factors for orchestrating workflows.
NEMO_FR24	The platform must provide access to the devices.	The meta-orchestrator provides access to devices by integrating and coordinating with different components within the distributed system architecture.

Table 18: NEMO non-functional requirements addressed through the meta-Orchestrator

Requirement ID	Requirement description	Requirement satisfaction
NEMO_NFR01	The NEMO platform must respect security and privacy requirements.	The meta-orchestrator respects security and privacy requirements by ensuring secure allocation of resources and communication while orchestrating workflows.
NEMO_NFR02	NEMO should support High Availability features.	The meta-orchestrator supports High Availability by intelligently allocating resources and adapting to changing conditions to ensure system availability.
NEMO_NFR04	The NEMO platform should be flexible and scalable in the sense of exploiting available resources according to set goals. Should be scalable in the sense of providing additional resources when computationally heavy tasks are initiated.	The meta-orchestrator provides flexibility and scalability by dynamically adjusting resource allocation based on goals and task demands, ensuring optimal resource utilization.
NEMO_NFR05	Secure communication of sensitive data related to the infrastructure should be provided.	The meta-orchestrator ensures secure communication by implementing encryption and secure protocols for sensitive data transmission within the system.

NEMO_NFR08	CPO platform shall be portable. So, moving from one OS to other OS does not create any problem.	The meta-orchestrator ensures portability by providing compatibility with multiple operating systems, enabling seamless migration without issues.
NEMO_NFR13	Store data in a safe and tamperproof manner.	The meta-orchestrator ensures safe and tamperproof data storage by implementing secure data management practices within the system.
NEMO_NFR14	The platform must ensure the traceability for the operator.	The meta-orchestrator ensures traceability by maintaining logs and records of system activities, facilitating accountability and monitoring.
NEMO_NFR15	The platform must have capabilities of a monitoring system.	The meta-orchestrator provides monitoring capabilities by tracking resource usage, performance, and system health for effective management and optimization.
NEMO_NFR16	The platform should offer the possibility to switch from the automated operation to manual operation.	The meta-orchestrator supports manual operation by allowing operators to intervene and adjust resource allocation and workflow orchestration as needed.
NEMO_NFR18	The platform must provide mechanisms for security and data privacy.	The meta-orchestrator ensures security and data privacy by implementing robust security mechanisms and privacy controls within the system.
NEMO_NFR19	The platform should support high availability deployments.	The meta-orchestrator supports high availability by intelligently managing resources and tasks to ensure continuous operation and system resilience.
NEMO_NFR20	Live migration should be done using specific microservices.	The meta-orchestrator supports live migration using specific microservices, ensuring efficient and seamless migration of computing tasks and resources.

5.5.2.2 Intent-based Migration Controller

5.5.2.2.1 Description

The Intent-based Migration Controller (IMC) represents a pivotal component within the overarching framework of the IoT to Edge to Cloud Continuum. Its primary purpose revolves around facilitating seamless and efficient migration of computing workloads across distributed systems, encompassing IoT devices, edge computing infrastructure, and cloud environments. By harnessing the power of intent-based networking principles and capitalizing on the knowledge gained from previous experience with the meta-orchestrator, the IMC aims to optimize resource utilization, enhance scalability, and ensure uninterrupted service delivery throughout the migration process. Figure 33 represents the IMC in the context of the NEMO Kernel and Continuum.

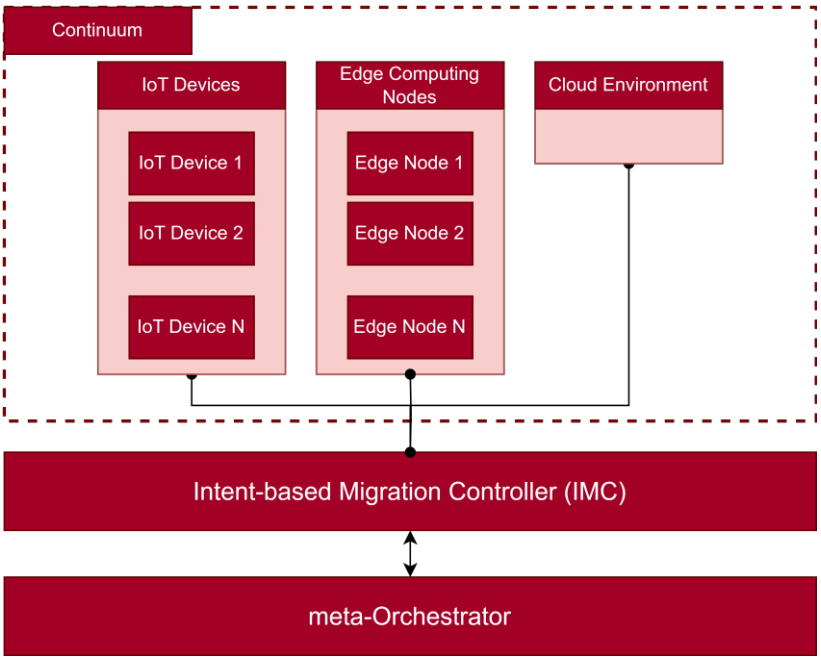

Figure 33: High-level design of the NEMO Intent Based Migration Controller

The scope of the Intent-based Migration Controller spans a wide range of migration scenarios encountered within the IoT to Edge to Cloud Continuum. These scenarios encompass the migration of workloads between edge devices and cloud platforms, migration between different edge computing nodes, and the orchestration of workload migration from legacy systems to modern IoT platforms. By effectively addressing these cases, the IMC provides an intelligent and adaptive framework that streamlines the migration process while considering the unique characteristics, constraints, and requirements of the IoT, edge, and cloud domains.

Within the context of IoT to cloud migration, the Intent-based Migration Controller empowers organizations to seamlessly transfer data and computational tasks from IoT devices to cloud environments. This migration may be driven by various factors such as resource limitations of IoT devices, the need for advanced analytics capabilities in the cloud, or the desire to centralize data storage and processing. By abstracting the underlying complexities and intricacies, the IMC orchestrates the migration process, ensuring data integrity, preserving real-time capabilities when necessary, and optimizing the utilization of cloud resources.

When operating within the realm of edge computing, the Intent-based Migration Controller plays a vital role in managing workload migration between different edge nodes. Such migration becomes necessary due to factors like varying resource availability, changing network conditions, or shifting workload demands. Leveraging intent-based principles, the IMC endeavours to comprehend the desired outcomes of migration, considering factors such as latency requirements, resource utilization, and application dependencies. Through its orchestration capabilities, the IMC empowers organizations to dynamically allocate and balance workloads across edge nodes, optimizing performance and resource utilization.

Drawing from the rich experience and interactions with the meta-orchestrator, the Intent-based Migration Controller seamlessly integrates with this higher-level orchestrator. By providing migration intents, receiving orchestration instructions, and exchanging information, the IMC collaborates with the meta-orchestrator to achieve a holistic and optimized migration process. The IMC harnesses the capabilities of the meta-orchestrator to effectively orchestrate and manage migration activities, ensuring smooth transitions, minimal disruption, and compliance with specified policies and constraints.

5.5.2.2.2 Main functionalities

These functionalities focus on the core capabilities of the Intent-based Migration Controller in facilitating the expression of migration intents, utilizing intent-based networking principles, and

integrating with the meta-Network Cluster Controller to enhance migration capabilities within the IoT to Edge to Cloud Continuum.

- Migration Intent Expression: The IMC provides the capability for users or organizations to express their migration intents at a high level. It allows them to define the desired outcomes, constraints, and requirements of the migration process, such as latency, resource utilization, data integrity, and security protocols.
- Intent-based Networking: Leveraging the principles of intent-based networking, the IMC interprets the migration intents expressed by users and translates them into actionable instructions for the migration process. It considers factors such as workload characteristics, network conditions, resource availability, and latency requirements to optimize the migration decision-making process.
- Integration with mNCC: The IMC integrates with the meta-Network Cluster Controller (mNCC) to enhance its migration capabilities. It leverages the connectivity and communication with mNCC to obtain real-time network information, such as network conditions, topology, and available resources. This integration enables the IMC to make informed migration decisions based on network-awareness and optimize the migration process.

5.5.2.2.3 Interactions

The IMC interacts with other NEMO components, as described in Table 19.

Table 19: Interactions of the IMC with other NEMO components

Interacting with	Interaction type	Description of interaction
meta-Orchestrator	Input/Output	The IMC receives migration intents and high-level requirements from the meta-Orchestrator as input. It interprets and processes these inputs to generate migration plans and recommendations. The IMC provides migration decisions and recommendations to the meta-Orchestrator as output. It communicates the intent-based migration plans, resource requirements, and constraints to the meta-Orchestrator for further coordination and orchestration.
meta-Network Cluster Controller (mNCC)	Input/Output	The IMC receives real-time network information from the mNCC, which includes network conditions, topology, and available resources. This information is used to make informed migration decisions and optimize the migration process. The IMC communicates with the mNCC to gather network information, including network conditions, topology, and available resources. It provides migration requirements, such as bandwidth and latency constraints, to the mNCC for network-aware migration planning.
Secure Execution Environment (SEE)	Input/Output	The IMC receives information and feedback from the SEE regarding the security measures and protocols implemented during the migration process. This feedback helps validate the secure execution of migrated workloads and ensures compliance with security requirements. The IMC provides migration instructions and requirements to the SEE. It communicates the necessary security protocols, data

Interacting with	Interaction type	Description of interaction
		integrity measures, and other migration-related security considerations to ensure secure migration of workloads.

5.5.2.2.4 Requirements

Intent-based Migration Controller contributes to the NEMO requirements, listed in Table 20 and Table 21, following the numbering and description adopted in D1.1.

Table 20: NEMO functional requirements addressed through IMC

Requirement ID	Requirement description	Requirement satisfaction
NEMO_FR01	The platform must provide access to measurements.	The Intent-Based Migration Controller provides access to measurements as it considers factors like resource availability, network conditions, and workload characteristics for migration decisions.
NEMO_FR03	The platform must provide options to manage users.	The Intent-Based Migration Controller supports user management by allowing users or organizations to express migration intents, defining desired outcomes and requirements.
NEMO_FR07	The platform should support monitoring of SLOs, e. g., related to energy consumption or CO2 emissions.	The Intent-Based Migration Controller supports monitoring of SLOs like energy consumption and emissions, considering such factors during migration intent interpretation and decision-making.
NEMO_FR08	The platform must respect data sovereignty and privacy requirements.	The Intent-Based Migration Controller respects data sovereignty and privacy by considering security protocols and requirements expressed in migration intents.
NEMO_FR14	The platform should be able to perform alternative scheduling or geographical distribution of smart farming services based on user goals.	The Intent-Based Migration Controller performs alternative scheduling and distribution by interpreting migration intents to meet user-defined constraints and goals.
NEMO_FR15	The Smart Farmer should be able to define strategies for the use of available resources.	The Intent-Based Migration Controller allows the Smart Farmer to define resource use strategies within migration intents, guiding its decision-making process.
NEMO_FR23	The platform must provide access to collected data.	The Intent-Based Migration Controller provides access to collected data by considering data integrity and communication protocols during migration orchestration.
NEMO_FR24	The platform must provide access to the devices.	The Intent-Based Migration Controller provides access to devices by integrating with network information from meta-Network Cluster Controller for migration decisions.

Table 21: NEMO non-functional requirements addressed through IMC

Requirement ID	Requirement description	Requirement satisfaction
NEMO_NFR01	The NEMO platform must respect security and privacy requirements.	The Intent-Based Migration Controller respects security and privacy requirements by considering secure migration intents and communication protocols.
NEMO_NFR02	NEMO should support High Availability features.	The Intent-Based Migration Controller supports High Availability features by considering migration intents that prioritize availability and redundancy.
NEMO_NFR04	The NEMO platform should be flexible and scalable in the sense of exploiting available resources according to set goals. Should be scalable in the sense of providing additional resources when computationally heavy tasks are initiated.	The Intent-Based Migration Controller is flexible and scalable by interpreting migration intents that define resource utilization goals and additional resource needs for heavy tasks.
NEMO_NFR05	Secure communication of sensitive data related to the infrastructure should be provided.	The Intent-Based Migration Controller provides secure communication by implementing secure protocols and encryption for sensitive data related to migration activities.
NEMO_NFR08	CPO platform shall be portable. So, moving from one OS to other OS does not create any problem.	The Intent-Based Migration Controller ensures portability by allowing migration intents to specify platform preferences and constraints.
NEMO_NFR13	Store data in a safe and tamperproof manner.	The Intent-Based Migration Controller stores data securely by following tamperproof data storage practices within the migration process.
NEMO_NFR14	The platform must ensure the traceability for the operator.	The Intent-Based Migration Controller ensures traceability by logging and tracking migration activities and decisions for operator accountability.
NEMO_NFR15	The platform must have capabilities of a monitoring system.	The Intent-Based Migration Controller has monitoring capabilities by interpreting migration intents that involve monitoring and resource utilization considerations.
NEMO_NFR16	The platform should offer the possibility to switch from the automated operation to manual operation.	The Intent-Based Migration Controller offers the possibility of manual operation by interpreting migration intents that allow for manual intervention and decision-making.
NEMO_NFR18	The platform must provide mechanisms for security and data privacy.	The Intent-Based Migration Controller provides mechanisms for security and data privacy by interpreting migration intents that include security and privacy constraints.
NEMO_NFR19	The platform should support high availability deployments.	The Intent-Based Migration Controller supports high availability deployments by

Requirement ID	Requirement description	Requirement satisfaction
		considering migration intents that prioritize availability and redundancy in the migration process.
NEMO_NFR20	Live migration should be done using specific microservices.	The Intent-Based Migration Controller performs live migration using specific microservices as specified in migration intents, facilitating seamless migration of workloads.

5.5.2.3 Cybersecure Microservices' Digital Twin

5.5.2.3.1 Description

In an ever-evolving digital world the direction taken in the management of complex services consists in considering each one composed of several microservices, and this poses new challenges. It is clear in fact that these microservices need to be organized, monitored, and managed effectively and at the same time we need to ensure data security, providing real-time performance metrics, and maintaining transparency.

The novelty of the CMDT architecture addresses these very challenges and takes into account the necessity for scalability, the need for real-time monitoring and updates, and the demand for secure and transparent operations.

Key components of this architecture, depicted in Figure 34, include:

- AAA: In the diagram shown in Figure 25 there is an Authentication and Authorization Service responsible for handling user access to the CMDT resources. It utilizes a role-based access control method, where different roles are assigned to users, granting them specific permissions based on their roles. This component ensures that only authorized users can access the CMDT resources. These functionalities can be operated by the CMDT or also be provided by an external service depending on the UCs.
- CMDT API: The CMDT service provides APIs that enable the interaction with the micro-services, (which can be in the form of management of packaged applications and editable YAML files). These APIs allow users to access and utilize the services provided by the CMDT.
- Additionally, there is a CMDT Service Layer: This component is responsible for interpreting and manipulating service descriptions for various external components.
- There is a database that stores YAML files describing microservices and what they look like. Finally, a blockchain client is integrated to ensure the security and decentralization of critical portions of the data descriptors that dynamically evolve over time, ultimately contributing to service performance assurance.

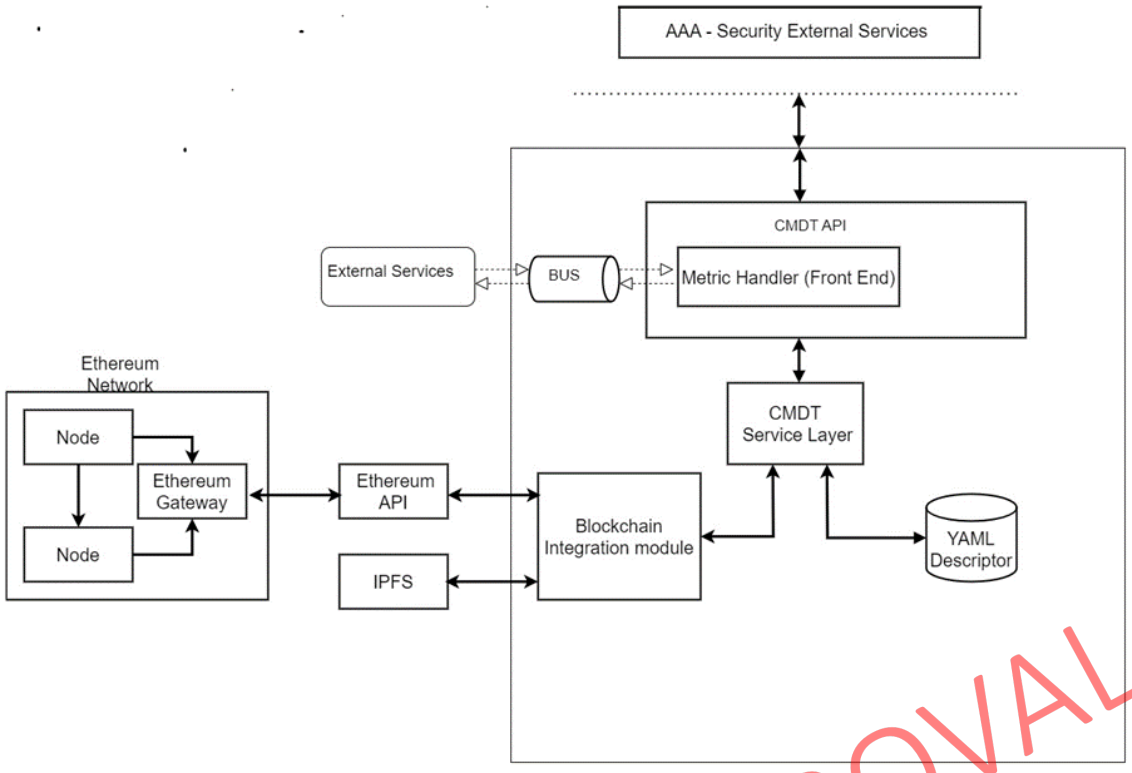

Figure 34: The CMDT high-level design

The CMDT subcomponents interact with each other as listed in Table 22.

Table 22: Analysis of the CMDT elements

CMDT subcomponent	Role and interactions
CMDT Service Layer	The CMDT Service Layer will still parse YAML files from the database, and it will also interact with the Blockchain Integration Module to get data from the blockchain. Similarly, if requested, it will send data to the Blockchain Integration Module when a YAML file needs to be updated on the blockchain.
Blockchain Integration module	The Blockchain Integration Module will interact with the CMDT service layer and service impersonation as described above. It will also communicate with the Blockchain client to do the actual reading and/or writing to the blockchain and/or with the IPFS. Blockchain Integration Module is basically a manager who, depending on the use case, decides which part of the Digital Twins to store either on the blockchain and/or on IPFS
AAA module	This external module will take care of enable authentication, authorization and accountability for external component
CMDT API	These APIs allow users to access and utilize the services provided by the CMDT
Metric Handler	After the performance metrics are modified and communicated by the external component, you will need a component to update the relevant service's YAML file with these new data.
Database YAML Descriptor	The database is where the YAML files will be stored, so both for writing and for reading

5.5.2.3.2 Main functionalities

The main functionalities of CMDT can be summarized and instantiated in the following components:

- Database: The database is where the YAML files will be stored. MongoDB or PostgreSQL would be an appropriate choice due to their strong support for document-oriented data structures.
- CMDT Service Level: You will need a component to parse and validate YAML files. This component should be able to both read and write to the database and save in parallel, if required, also to the Blockchain Integration Module
- The CMDT service provides APIs that enable the interaction with the micro-services, (which can be in the form of management of packaged applications and editable YAML files). These APIs allow users to access and utilize the services provided by the CMDT.
- Metrics Handler: This component collects the performance metrics of a service from different viewpoints (for example, security, runtime, etc.) once the service has run.
- Blockchain Client: this component interacts with the blockchain network. It is responsible for writing to and reading from the blockchain. Depending on which blockchain you choose to use (like Ethereum, Hyperledger Fabric, etc.), the interface for this client will be different.
- Blockchain Integration Module: This component interacts with the CMDT service layer representation and the blockchain and/or IPFS client. Decides which parts of the YAML file should be stored on the blockchain and/or IPFS and coordinates reading and/or writing from the blockchain.

5.5.2.3.3 Interactions

The CMDT interacts with other NEMO components, as described in Table 23.

Table 23: Interactions of the CMDT with other NEMO components

Interacting with	Interaction type	Description of interaction
Meta-Orchestrator	Input/Output	The Cybersecure Microservices Digital Twin (CMDT) interacts with the Meta-Orchestrator as both input and output. As an output, the CMDT provides information related to the security and integrity of microservices within the distributed computing environment. This output helps the meta-orchestrator evaluate the security risks and vulnerabilities associated with different microservices and incorporate security measures into orchestration decisions. As input, the Meta-Orchestrator provides instructions and requests to the CMDT for security-related configurations, monitoring, and enforcement. The CMDT executes these instructions to keep the microservices functioning safely throughout the orchestration process.
Intent-based Migration Controller	Input	The IMC communicates with the CMDT to gather containers' information, network information, including network conditions, topology, and available resources.
PRESS & Policy Enforcement Framework	Input	The PRESS & Policy Enforcement Framework communicates with the CMDT to ensure policy compliance through multifaceted policies that address different aspects of the application lifecycle (security, privacy, cost,

Interacting with	Interaction type	Description of interaction
		environmental impact, etc.), channeling "violations" as input.
Plugin & Applications Lifecycle Manager	Input/Output	The Plug-ins and Application Lifecycle Manager communicates with the CMDT. This is a key component and will be a flexible plug-in and application lifecycle manager, which will enable over-the-air and timely implementation of the required plug-ins. As input to the CMDT, the Plug-ins and Application Lifecycle Manager provides "descriptors", while as output from the CMDT it provides "lifecycle info".
Monetization and Consensus-based Accountability	Input/Output	Monetization and consent-based accountability communicates with the CMDT. As Input and Output, they only exchange information about the payload you insert on the blockchain.

5.5.2.3.4 Requirements

The CMDT contributes to the NEMO functional and non-functional requirements, listed in Table 24 and Table 25, respectively, following the numbering and description adopted in D1.1.

Table 24: NEMO functional requirements addressed through the CMDT

Requirement ID	Requirement description	Requirement satisfaction
NEMO_FR03	The platform must provide options to manage users.	Via the AAA module that will take care of making authentication, authorization and accountability possible for external components
NEMO_FR13	The monitoring devices must be able to communicate data to and receive control commands from the NEMO platform.	CMDT shall be able to communicate data and receive control commands from the NEMO platform.
NEMO_FR08	The platform must respect data sovereignty and privacy requirements.	Access to CMDT is granted to authorised users only
NEMO_FR80	The solution should have an Application Server (Rest API) for communication between system devices and applications.	CMDT API module: The CMDT service provides APIs that enable interaction with microservices (which can come in the form of packaged application management and editable YAML files). These APIs allow users to access and use the services provided by the CMDT.
NEMO_FR82	The solution could provide a UI tool for specifying what to send to subscribers.	The solution could possibly provide a user interface tool for specifying what to send to subscribers.
NEMO_FR76	The platform must ensure the interoperability with external systems (i.e. multi sensorial stimuli system).	CMDT will ensure interoperability with external systems.

Table 25: NEMO non-functional requirements addressed through the CMDT

Requirement ID	Requirement description	Requirement satisfaction
NEMO_NFR01	The NEMO platform must respect security and privacy requirements.	CMDT complies with all privacy and security requirements of NEMO

Requirement ID	Requirement description	Requirement satisfaction
		platform, integrates identity management and access control to access API resources
NEMO_NFR04	The NEMO platform should be flexible and scalable in the sense of exploiting available resources according to set goals. It should be scalable in the sense of providing additional resources when computationally heavy tasks are initiated.	CMDT should also be scalable in the sense of providing additional resources when computationally heavy tasks are started
NEMO_NFR05	Secure communication of sensitive data related to the infrastructure should be provided.	All communications between CMDT and other system components will be established over secure channels. CMDT also integrates identity management and access control to access API resources
NEMO_NFR12	Data shall be consistent, reliable, transparent and accessible only to authorized users.	The CMDT integrates identity management and access control for accessing the API resources, a task likely performed by the external AAA module.
NEMO_NFR13	Store data in a safe and tamperproof manner.	CMDT stores data securely and tamper-proof.
NEMO_NFR14	The platform must ensure the traceability for the operator.	CMDT will expose the services related to the traceability of the running microservices, eventually we will understand if it records the operator's activity, it (logs operator activity) will probably be delegated to the AAA module.
NEMO_NFR15	The platform must have capabilities of a monitoring system.	CMDT will expose service-related monitoring, in particular it will make use of the "Metrics Handler" component which collects the performance metrics of a service from different points of view (for example, security, runtime, etc.) once the service is performed.
NEMO_NFR16	The platform should offer the possibility to switch from the automated operation to manual operation.	The platform should possibly offer the possibility to switch from automated to manual operation. To be checked later and possibly understood where feasible.
NEMO_NFR17	High accuracy of detection and identification.	The CMDT integrates identity management and access control to access API resources, a task likely performed by the external AAA module.
NEMO_NFR18	The platform must provide mechanisms for security and data privacy.	The use of secure execution environments increases the security of the NEMO platform.

5.5.2.4 Secure Execution Environment

5.5.2.4.1 Description

The Secure Execution Environment is an extended version of the commonly used cluster orchestrator Kubernetes. Kubernetes can orchestrate and monitor microservices using containers, however, containers share the host's kernel, so a vulnerability in a container might expose the host and thus all other containers on the node.

The proposed solution enhances Kubernetes with the ability to run classical binary software in unikernels and webassembly (WASM) programs in secure enclaves. Additionally, we investigate the enhancement of the migration capabilities, so that new application scenarios, such as IoT-Device clusters or location aware services are possible.

A Unikernel is a lightweight virtual machine (VM) image, containing only the application and the necessary functions to run in a VM, but removes all other functionalities that a traditional virtual machine has (kernel, drivers, userspace and more). The VM provides an enhanced isolation, and the minimalistic image is more lightweight and often even faster than a container.

Trusted Execution Environments (TEEs) is an environment to run applications within a set of encrypted memory pages to ensure the integrity of that data against modification from the host or other applications and hide the information from these parties. This relies on hardware features and promises only a minimal performance impact. Enarx [34] combines this technology with a WASM runtime to provide a defined platform for applications requiring the increased integrity.
The high-level design of SEE is depicted in Figure 35.

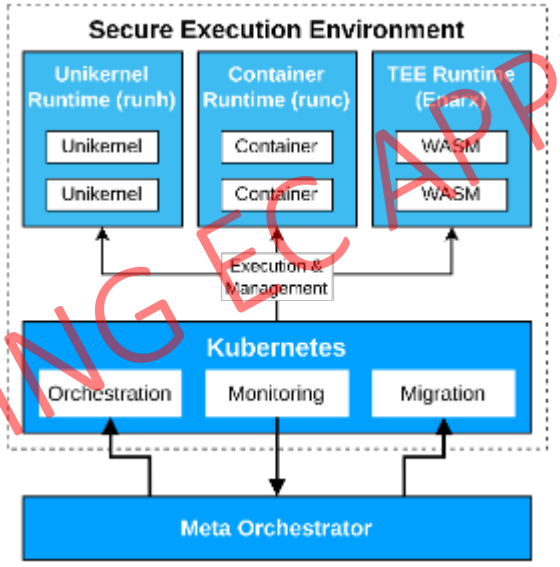

Figure 35: High-level design of the Secure Execution Environment

The components *Unikernel Runtime*, *Container Runtime*, *TEE Runtime* have yet to be developed and/or integrated.

5.5.2.4.2 Main functionalities

The secure execution environment is a drop-in replacement for Kubernetes. It inherits all functionalities, such as orchestration, interfaces or monitoring.
Additionally, it can run unikernels for enhanced isolation and WASM in secure enclaves for increased trust. Currently, it is not possible to control the migration of pods in a kubernetes cluster. The migration component will grant fine grained migration capabilities of containers and unikernels for more advanced use-cases.

5.5.2.4.3 Interactions

The SEE interacts with other NEMO components, as described in Table 26.

Table 26: Interactions of the SEE with other NEMO components

Interacting with	Interaction type	Description of interaction
Meta Orchestrator	Input	Control of the SEE (deployment of services, control etc)
	Output	Monitoring information of running services
	Input	Migration commands
Other Users	Input/Output	Other users may interact with the component manually or potentially from other services

5.5.2.4.4 Requirements

The SEE contributes to the NEMO requirements, listed in Table 27, following the numbering and description adopted in D1.1.

Table 27: NEMO requirements addressed through SEE

Requirement ID	Requirement description	Requirement satisfaction
NEMO_NFR18	The platform must provide mechanisms for security and data privacy.	The use of secure execution environments increases the security of the NEMO platform
NEMO_NFR19	The platform should support high availability deployments.	The SEE component shall not reduce the availability of the platform.
NEMO_NFR20	Live migration should be done using specific microservices.	The SEE component shall support migration.

5.5.3 NEMO Service Management

5.5.3.1 Intent-based SDK/API

NEMO will rely on its Intent-based Application Programming Interface (API) and Software Development Kit (SDK) for maximizing the adoption potential by third party entities, including both the meta-OS consumers and meta-OS partners, as well as external applications and (micro-)services. It is aimed to expose NEMO lower-level functionality to the outside world in an easily accessible format, minimizing the effort needed on their side to adapt applications, services and plugins to NEMO-capable ones, but also introducing minimal distraction compared to common practice for proficient (K8s) cluster users.

5.5.3.1.1 Description

The NEMO Intent based API/SDK scope is twofold. First, it aims to expose NEMO functionality through a set of resources of a programmatic interface. This will be realized as both API service description and implementation, which will facilitate external users in accessing the NEMO services, but also the NEMO system to limit access to its resources only to eligible users and roles. Moreover, the API/SDK offer a flexible and modular framework for developing, registering, discovering and provisioning workloads, i.e., applications, services or plugins, through NEMO.

So, the Intent based API/SDK are aimed for meta-OS consumers (workload owners) and meta-OS partners (plugin providers), wishing to make use of the meta-OS continuum. The API/SDK allow them to develop new NEMO-compliant workloads, using the SDK and consuming the API services. In addition, they may easily port their existing workloads into NEMO, making use of the SDK, relieved from the overhead of performing the relevant integrations manually, which would potentially require them to study NEMO documentation and familiarize with internal NEMO concepts. Instead of this, the

API/SDK allow the integration of existing and new services into NEMO, introducing minimal overhead over developers' common everyday operations as cluster users. Of course, similar operations could be triggered by third-party applications, services or plugins. This facilitated integration capability endows NEMO a user-centred flavour as a meta-OS, giving priority to usability goals for the meta-OS.
On the system side, the NEMO Intent based API & SDK aim to automate the processes for workload registration and deployment to the extent possible and thus support ZeroOps deployment, eliminating manual interventions in the NEMO DevOps cycle. The definition and adoption of common NEMO specifications for workloads helps automating their registration, discovery, provisioning in the NEMO meta-OS and allows for their systemic handling in a coherent way. Moreover, the API provides the logic that ensures that desired exposable functionality of any NEMO workloads (components, applications, services or plugins) will be automatically discovered and exposed through the API, with minimal effort from the developer's side (NEMO consumer or partner).

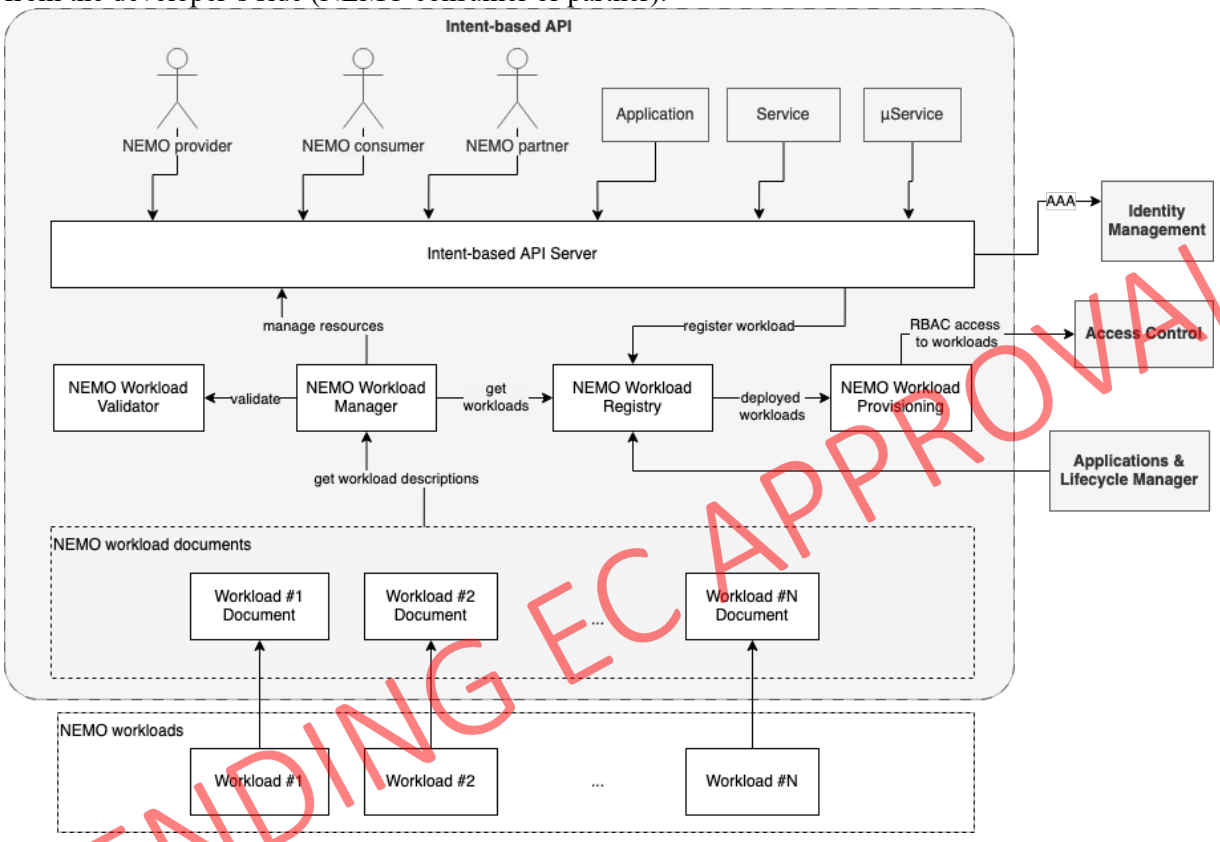

Figure 36: The NEMO Intent-based API

Intent-based API

The high-level architecture of the Intent based API is depicted in Figure 36. The API exposes NEMO functionality in the form of resource based programmatic interfaces. Its goal is to provide a single entry point for third-party entities to the NEMO functionalities, as offered by the NEMO framework components, but also additional software elements (plugins) which extend the NEMO functionality and may be added by the third parties.
Automation is at the core of the API design. So, the API handles workloads ("*NEMO workloads*" in Figure 36), whether they are NEMO framework components, services or plugins, in a consistent manner. The first step for their common handling would be to describe the workloads in a common format, following common specifications for any *workload type*. This is realized through the NEMO *workload documents*. There should be one workload document for each workload to be discoverable in the NEMO meta-OS.
Then, automated workload discovery is made possible, as the exposable services for each registered workload can be identified and exposed as *Resources* in the API Server. Specifically, the NEMO

Workload Manager may check all available workloads (as recorded in the *Workload Registry*) for services that could be exposed. This information could be also available in the workload document. Before exposing those, NEMO should ensure that the named workloads and their documents are compliant to NEMO. The compliance may be defined at communication, container, network or even hardware level. This validity check is performed through the *Workload Validator*. For the workloads that successfully pass the validity check, the exposable API services are automatically discovered and relevant *resources* are automatically created to expose those services. The Intent-based API Server acts then as an API gateway to the NEMO workloads' services, in which the endpoints (resources) are automatically created. In this way, the API, and accordingly the NEMO services' exposure, is fully dynamic and scalable. As long as new workloads are registered or existing ones are updated or removed, the API will be able to automatically update the exposed resources. This dynamicity makes NEMO really flexible and autonomous.
An important aspect of the Intent-based API is the access control over the API endpoints. Access to each workload endpoints should be granted based on both the permissions assigned to each user role and the individual user per se, i.e. based on their credentials. For instance, an application owner, having their application running on NEMO, must be able to make a registration request and access informative data regarding only their own application. On the other hand, they must not be able to apply meta-OS wide administrative actions about their application, such as granting registration to the NEMO meta-OS even for their application. For this, the API relies on the Identity Management and Access Control components of the NEMO Cybersecurity & Unified/Federated Access Control.
Last, but not least, the API supports and facilitates the workload registration into the NEMO platform. Its role in this process would be to ensure that the NEMO consumer/partner wishing to register their workload will be able to make the registration request and receive the required token for requesting a workload deployment. Moreover, once a given workload is deployed, the *Workload Provisioning* component of the API will ensure that access to it is provided according to the defined Role Based Access Control (RBAC) rules.

Intent-based SDK
The SDK aims to facilitate the adoption of NEMO by third parties, mainly referring to workload owners wishing to make their workloads compliant to NEMO and deploy them in the meta-OS continuum. The SDK will provide a set of code libraries which will make the integration with NEMO much faster and more efficient. This can allow workload developers easily adopt the NEMO meta-OS, saving them the time needed to build their code from scratch for NEMO and getting aware of the NEMO specificities.
Indicatively, the Intent-based SDK will support the workload registration as NEMO-capable workload, workload authentication, authorization and accounting into NEMO, etc.

5.5.3.1.2 Main functionalities

The functionality supported by the NEMO Intent-based API/SDK can be summarized as follows.

- Automated NEMO service discovery: The Intent-based API addresses the painful topic of service discovery, responding to how a workload consumer could access the services offered by a workload instance running on NEMO. The API provides a mechanism for registering/deregistering workloads and for identifying and exposing their services as RESTful endpoints. This process automated and simplifies access to NEMO functionalities for both the meta-OS consumer and the meta-OS provider.
- Access control in NEMO services: The exposed NEMO services should be protected by authorized access, as well as access by non-eligible users. The API integrates identity management and access control, which allow restricting access to NEMO resources only to entities that should be able to do so.
- Easy integration into NEMO: A timely topic on the adoption of new platform evolves around how easy it is for newcomers to created platform-compliant tools or adapt their systems or services, in order to integrate them with the platform. Both the API and the SDK aim to simplify this process, by offering software artifacts for workload development, registration/deregistration, validation and provisioning.

5.5.3.1.3 Interactions

The Intent based API/SDK interacts with both external users and workloads, as well as with other NEMO components, as described in Table 28.

Table 28: Interactions of Intent-based API/SDK with other NEMO components and external entities

Interacting with	Interaction type	Description of interaction
Identity management	Input	The Intent based API consumers AAA services from the Identity Management component. Indicatively, it will perform entities' authentication and authorization for API endpoints and will provide meta-OS consumers with tokens for their workload registration/deregistration based on the component. Moreover, the SDK integrates the Identity Management components for applying AAA on the provided functions.
Access Control	Input/Output	The Intent-based API will rely on the Access Control component for controlling access to the API endpoints, based on user's, roles' or other criteria. Moreover, the API will support workload provisioning, i.e. providing access to workload information, as soon as it is up and running. Access to the prominent users and roles will be granted through a request (output) to the Access Control component.
Applications & Lifecycle Manager	Output	The Application & Lifecycle Manager requires access to the Workload Registry in order to receive information about the workloads running on NEMO.
All NEMO components and plugins	Input/Output	The API acts as a gateway for the NEMO components' and plugins' services. As such, it communicates with the relevant components (both as input/output requests) in order to expose their services.
Meta-OS Provider	Input/Output	The meta-OS Provider may use the API in order to grant workload de/registration requests, initialize the registration within NEMO, etc.
Meta-OS Consumer	Input/Output	The meta-OS Consumer will use the API and SDK in order to develop new workloads or integrate existing ones into NEMO. Also, they will use the API in order to de/register workloads into the system.
Meta-OS Partner	Input/Output	Meta-OS partners, such as plugin developers/owners, may use the API and SDK in order to develop, integrate or de/register plugins into NEMO.

5.5.3.1.4 Requirements

The Intent-based API/SDK contribute to addressing a set of the defined NEMO functional and non-functional requirements. Table 29 lists those requirements, following the numbering and description adopted in D1.1 [32] and justifies how the Intent-based API/SDK contributes to their satisfaction.

Table 29: NEMO use case requirements addressed through the Intent-based API/SDK

Requirement ID	Requirement description	Requirement satisfaction
NEMO_FR01	The platform must provide access to measurements.	The Intent-based API/SDK will expose services related to the meta-OS continuum monitoring, originally supported in PPEF.
NEMO_FR02	The platform must provide options to manage/view sensors/devices.	The Intent-based API/SDK will expose services related to the meta-OS continuum cluster and nodes, originally supported in PPEF or the meta-orchestrator, which may support these operations.
NEMO_FR03	The platform must provide options to manage users.	The Intent-based API/SDK will expose services related to user, identity and access management, originally supported by the NEMO Cybersecurity & Unified/Federated Access Control.
NEMO_FR04	The platform should support ML/FL training and ML model sharing/serving.	The Intent-based API/SDK will expose services related to these operations, originally offered by the NEMO Cybersecure Federated Deep Reinforcement Learning (CFDRL).
NEMO_FR05	The platform should provide ML classification accuracy probability.	The Intent-based API/SDK will expose services related to ML inference via model serving, originally offered by the NEMO Cybersecure Federated Deep Reinforcement Learning (CFDRL).
NEMO_FR07	The platform should support monitoring of SLOs, e.g., related to energy consumption or CO2 emissions.	The Intent-based API/SDK will expose services related to PRESS policies and SLAs/SLOs monitoring, originally supported in PPEF.
NEMO_FR15	The Smart Farmer should be able to define strategies for the use of available resources.	The Intent-based API/SDK will expose services related to the definition of intents desired to be supported during workload execution, which may include requirements for the resource usage.
NEMO_FR24	The platform must provide access to the devices.	The Intent-based API/SDK will expose services related to the meta-OS continuum cluster and nodes, originally supported in PPEF, which covers IoT, edge and cloud devices in the continuum.
NEMO_FR25	The platform must provide options to manage users.	The Intent-based API/SDK will expose services related to user, identity and access management, originally supported by the NEMO Cybersecurity & Unified/Federated Access Control.
NEMO_FR26	The Smart Farmer should be able to define strategies for the use of available resources.	The Intent-based API/SDK will expose services related to the definition of intents desired to be supported during workload execution, which may include requirements for the resource usage.
NEMO_FR56	Several video streams are to be transferred through the	The Intent-based API/SDK will expose services related to the definition of intents desired to be

Requirement ID	Requirement description	Requirement satisfaction
	cloud/network. Bandwidth requirements must be met accordingly.	supported during workload execution, which may include requirements on the bandwidth.
NEMO_FR58	NEMO must provide the adequate resources to the service provider to map these requirements onto the cloud network and perform accordingly.	The Intent-based API/SDK will expose services related to the definition of intents desired to be supported during workload execution. The allocation of resources and monitoring of performance is then part of the NEMO kernel activities.
NEMO_FR60	NEMO will be able to allocate and launch the required services/VNFs on a location basis.	The Intent-based API/SDK will expose services related to the definition of intents desired to be supported during workload execution, which may include requirements on the location. The allocation of resources and deployment of workloads is under the NEMO kernel responsibility.
NEMO_FR61	The service provider must be able to chain services/VNFs with the help of a service orchestrator.	The Intent-based API/SDK will expose services related to the definition of workflows desired to be executed by the NEMO meta-OS, which covers service chaining definitions.
NEMO_FR62	NEMO applies a central control unit (Cognitive Network Optimization) that is used by the service provider to adjust/adapt the network dynamically according to the specific requirements and conditions.	The Intent-based API/SDK will expose services related to the definition of intents desired to be supported during workload execution, which may include specific requirements and conditions.
NEMO_FR63	NEMO must be able to monitor and control the network and ensure adherence to QoS levels (bandwidth, average bit rate, round trip delay).	The Intent-based API/SDK will expose services related to the definition of intents desired to be supported during workload execution, which may relate to network and QoS requirements. The monitoring an d adjustment of workload execution is under the NEMO kernel responsibility.
NEMO_FR67	Max. end-to-end network latency (RTT) - It comprises the latency of the whole network path excluding end devices on-site (like the network gateway or HW video coder) <= 50 ms.	The Intent-based API/SDK will expose services related to the definition of intents desired to be supported during workload execution, which may include requirements on the network latency.
NEMO_FR68	Max latency of end-to-end signal transport (video, audio and control data) - it comprises the latency of the whole signal path including converting of end devices on-site and media-specific VNFs). Maximum E2E latency one way for video and audio: <= 500 ms Max. E2E latency for return video (one way): <= 500 ms (Typically	The Intent-based API/SDK will expose services related to the definition of intents desired to be supported during workload execution, which may include requirements on the network latency.

Requirement ID	Requirement description	Requirement satisfaction
	uses less bandwidth because of low-resolution proxy transfer) Max. end-to-end latency for intercom (if needed): <= 100 ms (according to ITU G.114).	
NEMO_FR70	The MEC platform and underlying NFVI is required to deploy and run all the needed VNFs. The estimated use of resources is: • High CPU power, preferably new processor generation (>= 96 cores). • 128 GB RAM • 1 TB Storage SSD Multiple 10 Gbit/s and 1 Gbit/s interfaces. GPU processing capability.	The Intent-based API/SDK will expose services related to the definition of workflows desired to be executed by the NEMO meta-OS and of intents desired to be supported during workload execution, which may include requirements on the computing and network resources.
NEMO_FR75	Network will support diverse devices (wearables, AR/VR headsets) with different performance (e.g., high throughput, low latency and massive connection densities).	The Intent-based API/SDK will expose services related to the definition of intents desired to be supported during workload execution, which may include execution on different device capabilities.
NEMO_FR 77	The platform components involving direct interaction with the end-users should be quick to respond to the users' actions.	The Intent-based API/SDK will expose services to end users and will introduce negligible or no overhead in control messages' communication.
NEMO_FR83	Network must support diverse devices (wearables, AR/VR headsets) with different performance (e.g., high throughput, low latency and massive connection densities).	The Intent-based API/SDK will expose services related to the definition of intents desired to be supported during workload execution, which may include requirements on execution on different device capabilities.
NEMO_NFR01	The NEMO platform must respect security and privacy requirements.	The Intent-based API/SDK integrates identity management and access control for accessing the API/SDK resources.
NEMO_NFR05	Secure communication of sensitive data related to the infrastructure should be provided.	The Intent-based API/SDK will incorporate identity management and access control for accessing the API/SDK resources. Moreover, expose services related to the definition of intents desired to be supported during workload execution, which may include requirements on secure execution or communication.
NEMO_NFR09	CPO platform login shall be processed by 3 seconds.	The Intent-based API/SDK will expose services related to the definition of intents desired to be supported during workload execution, which may include relevant network requirements.

Requirement ID	Requirement description	Requirement satisfaction
NEMO_NFR10	Charging station ping shall be under 200 ms.	The Intent-based API/SDK will expose services related to the definition of intents desired to be supported during workload execution, which may include network requirements.
NEMO_NFR11	Electric vehicle ping shall be under 200 ms.	The Intent-based API/SDK will expose services related to the definition of intents desired to be supported during workload execution, which may include network requirements.
NEMO_NFR12	Data shall be consistent, reliable, transparent and accessible only to authorized users.	The Intent-based API/SDK integrates identity management and access control for accessing the API/SDK resources and thus data.
NEMO_NFR14	The platform must ensure the traceability for the operator.	The Intent-based API/SDK will expose services related to traceability of microservices running on the meta-OS continuum, originally supported in the Cybersecure Microservices' Digital Twin (CMDT).
NEMO_NFR15	The platform must have capabilities of a monitoring system.	The Intent-based API/SDK will expose services related to the meta-OS continuum monitoring, originally supported in PPEF.
NEMO_NFR18	The platform must provide mechanisms for security and data privacy.	The Intent-based API/SDK integrates identity management and access control for accessing the API/SDK resources.

5.5.3.2 Plugin & Applications Lifecycle Manager

The Plugin & Applications Lifecycle Manager (LCM) is flexible mechanism for unified, just-in-time plugins and applications life cycle management across the NEMO ecosystem. The Lifecycle Manager (LCM) will be the interface between the NEMO ecosystem and the NEMO users, providing an interface for seamless deployment of services and applications in the NEMO ecosystem.

5.5.3.2.1 Description

The NEMO LCM allows meta-OS consumers and meta-OS partners to install and deploy registered applications, services, or plugins to NEMO meta-OS automated but totally transparent to the user. Based on applications' requirements/manifest, the NEMO LCM will gain ingress access rights, download the necessary plugins and associated dependencies on demand, and install them on the devices while checking for security warnings.

The Intent based API/SDK allows meta-OS consumers and meta-OS partners to develop NEMO-compliant workloads and register them into NEMO framework. The LCM offers an interface to NEMO users to deploy registered workloads or manage already deployed workloads while providing information on the running services owned by the user.

Providing a seamless interface to deploy and run services in the NEMO ecosystem, the LCM interacts with the meta-Orchestrator to communicate requested workload operations while at the same time informs PPEF about the requested SLOs and SLA, sends accounting data for the workload in MOCA component and registers service descriptor in CMDT to ensure traceability of the deployment activities. While a service is running in NEMO meta-OS an event-based mechanism monitors critical events related to the performance of the service. Moreover, a security controller monitors security related events, alerts the user for detected abnormalities, and applies mitigation actions based on specified cyber threats. Finally, LCM will check for available updates/bug fixing and install them over the air.

Figure 37 shows the positioning of LCM in NEMO architecture and interactions with other NEMO components.

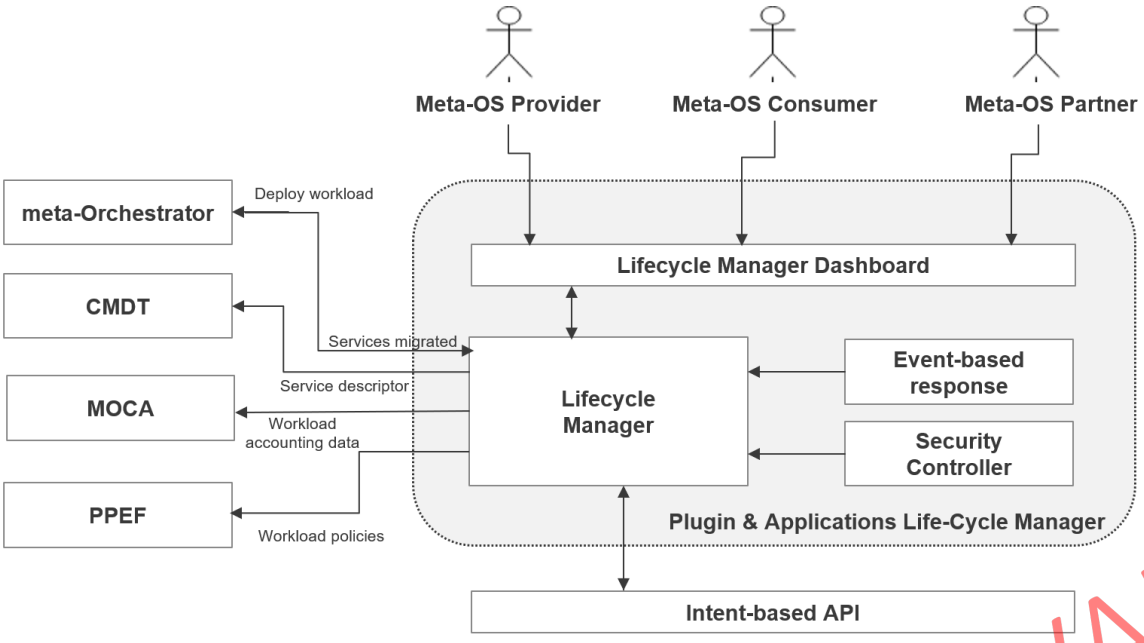

Figure 37: NEMO Plugin & Applications Lifecycle Manager

5.5.3.2.2 Main functionalities

NEMO LCM provides an interface to NEMO users to deploy and monitor plugins, applications, and services deployed in NEMO ecosystem. Its main functionalities include:

- Deployment of registered workloads in NEMO meta-OS considering also dependencies, service policies and workload accounting data.
- Manage running workloads, monitor their performance, provide security related services and check for new versions of the deployed service.

5.5.3.2.3 Interactions

The Plugin & Applications Lifecycle Manager interacts with other NEMO components, as described in Table 30.

Table 30: Interactions of the Plugin & Applications Lifecycle Manager with other NEMO components

Interacting with	Interaction type	Description of interaction
Identity management	Input	The LCM consumes AAA services from the Identity Management component. Indicatively, it will perform entities' authentication and authorization to provide meta-OS consumers access to their workload data.
Access Control	Input/Output	The LCM will rely on the Access Control component for controlling access based on user's, roles.
Intent based API	Input	The LCM requires access to the Workload Registry in order to receive information about the workloads registered or running on NEMO.
meta-Orchestrator	Input/Output	The LCM interacts with the meta-Orchestrator both as an input and output. As an output, the LCM provides installation and deployment commands such us (install/uninstall, start/stop). As an input, the meta-orchestrator provides feedback and updates regarding

Interacting with	Interaction type	Description of interaction
		the status and progress of the workflow migration in order to track and monitor the migration process.
MOCA	Output	Provides accounting data related to the deployed service and its requirements.
PPEF	Output	LCM provides the SLA definitions that concern the plugins that will be deployed in NEMO meta-OS.
Event-based response	Input	LCM is getting information for workload performance data based on detected events.
Security Controller	Input	LCM is getting security related notifications while automated mitigation actions are applied when possible.

5.5.3.2.4 Requirements

LCM contributes to the NEMO requirements, listed in Table 31, following the numbering and description adopted in D1.1.

Table 31: NEMO requirements addressed through LCM

Requirement ID	Requirement description	Requirement satisfaction
NEMO_FR01	The platform must provide access to measurements.	The LCM will provide information on services running in the NEMO meta-OS framework.
NEMO_FR03	The platform must provide options to manage users	The LCM will provide services depending on user's, identity and access management
NEMO_FR08	The platform must respect data sovereignty and privacy requirements.	Access to LCM is granted to authorised users only
NEMO_NFR01	The NEMO platform must respect security and privacy requirements.	LMC is respecting all privacy and security requirements of NEMO platform while it adds to security by monitoring security issues on running containers
NEMO_NFR05	Secure communication of sensitive data related to the infrastructure should be provided.	All communications between LMC and other system components will be established over secure channels
NEMO_NFR14	The platform must ensure the traceability for the operator.	LMC registers operator activity

5.5.3.3 Monetization and Consensus-based Accountability

MOCA supports the pre-commercial exploitation of the NEMO platform across the multi-operator/multi-tenant IoT/5G continuum. This mechanism implements a consensus-based distributed architecture for sharing networking, computing, and storage resources from various end-users and (competing) telecom and cloud providers. This approach enables the creation of new business models allowing volunteers and professionals to adopt the NEMO platform and offer hosting and migration services according to the resources as a service (RaaS) paradigm. MOCA offers a traceable way to build future business trade-offs between providers sharing bundles of computing, memory, storage resources and I/O resources for a short period of time based on DLT-based smart contracts. To achieve its goals, MOCA collaborates with the meta-Orchestrator, the CMDT and the monitoring framework, as illustrated in Figure 38.

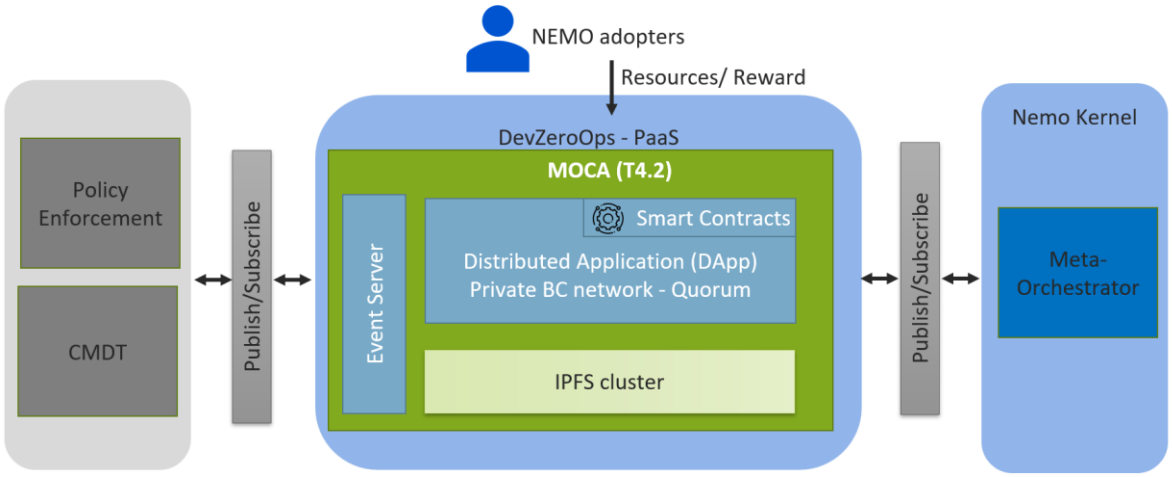

Figure 38: The MOCA component and interactions

The MOCA component provides a secure way to engage end-users and infrastructure providers to adopt NEMO platform for deploying their services or/and offering resources. Despite the fact the MOCA will not support financial transactions it proposes a monetization approach that can be used in the future for financial transactions between the stakeholders. The main idea is based on the implementation of the RaaS model, through which the shareholders can "sell" individual resources for a few seconds at a time so that end users can benefit from a wide and dynamic continuum of heterogeneous types of resources. Following this approach, the providers can exploit their idle resources and the end users can utilize these resources in order to achieve their services SLOs at best prices. The system will trace all transactions using DLT technology and Smart Contracts and will provide accounting services based on the utilization of the resources.

5.5.3.3.1 Main functionalities

MOCA realizes specific technical functionalities to support the provided services to the NEMO stakeholders, in terms of monetization resources, and transactions between NEMO end-users and infrastructure providers. The main functionalities are listed below:

- **Support secure resources allocation transactions**. NEMO adopters offer resources to the platform in order to be used for a specific period of time for service deployment. The exchange process between the NEMO platform and the adopters is done via a secure and traceable way that ensures data privacy between different infrastructure providers. These transactions can be performed on private or public blockchain networks.
- **Business models based on DLT smart contracts**. The pre-commercial exploitation will be supported by the introduction of new business models through DLT smart contracts. Each infrastructure provider will get a specific number of "reward grades" for the resources, that has offered to the platform. These grades can be used for service deployments of the provider's clients in the NEMO continuum. The business modes will calculate the number of "reward grades" based on several aspects like the amount of offered to NEMO resources, the demand from other users, etc.
- **End-to-end consensus-based accounting mechanism**. Each running service in the NEMO platform consumes computational, networking, and storage resources from several types of multi-tenant, multi-operator IoT/5G cloud continuum. MOCA provides an accounting mechanism based on DLT technology that collects information regarding the life cycle of each running service in order to provide accounting information for each user. Based on this mechanism future business models will be defined in order to support the sustainability of the project.

- **Resource utilization in infrastructure levels.** MOCA interacts with the monitoring framework of PRESS in order to determine the operational status along the IoT/5G continuum in terms of CPU, memory, and storage usability. This information is very helpful to the meta-orchestrator in order to decide the optimal location for the service deployment and for the accounting mechanism to calculate the amounts of "reward grades" per provider.

5.5.3.3.2 Interactions

MOCA interacts with other NEMO components, as described in Table 32.

Table 32: Interactions of MOCA with other NEMO components

Interacting with	Interaction type	Description of interaction
Meta-orchestrator	output	MOCA provides the list of available infrastructures in the continuum for service deployment, according to their current resource availability in terms of CPU, memory, and storage usage.
CMDT	input	MOCA receives information about the owner, the deployment location, time, and scaling and migration actions that took place during the life cycle of each service.
PRESS	input	MOCA receives monitoring information about the performance status of infrastructure and service levels.

5.5.3.3.3 Requirements

MOCA contributes to the NEMO requirements, listed in Table 33, following the numbering and description adopted in D1.1.

Table 33: NEMO requirements addressed through MOCA

Requirement ID	Requirement description	Requirement satisfaction
NEMO_FR01	The platform must provide access to measurements.	MOCA will communicate directly with the monitoring system and will retrieve performance information related to the usability levels of the continuum.
NEMO_FR08	The platform must respect data sovereignty and privacy requirements.	MOCA will use DLT technology to ensure data security, privacy, and traceability.
NEMO_FR25	The platform must provide options to manage users.	MOCA supports several user roles.
NEMO_FR58	NEMO must provide the adequate resources to the service provider to map these requirements onto the cloud network and perform accordingly.	MOCA will keep track of all the available resources in the NEMO continuum and will retrieve their current status from the monitoring system.
NEMO_FR63	NEMO must be able to monitor and control the network and ensure adherence to QoS levels (bandwidth, average bit rate, round trip delay).	MOCA will keep track of all the available resources in the NEMO continuum and will retrieve their current status from the monitoring system.
NEMO_FR70	The MEC platform and underlying NFVI is required to deploy and run all the needed VNFs.	MOCA will keep track of all the available resources in the NEMO continuum and will retrieve their current status from the monitoring system.
NEMO_NFR01	The NEMO platform must respect security and privacy requirements.	MOCA will use DLT technology to ensure data security, privacy, and traceability.

Requirement ID	Requirement description	Requirement satisfaction
NEMO_NFR05	Secure communication of sensitive data related to the infrastructure should be provided.	MOCA will use DLT technology to ensure data security, privacy, and traceability.
NEMO_NFR13	Store data in a safe and tamperproof manner.	MOCA will use DLT technology to ensure data security, privacy, and traceability.
NEMO_NFR14	The platform must ensure the traceability for the operator.	MOCA will use DLT technology to ensure data security, privacy, and traceability.

5.5.4 NEMO PRESS & Policy Enforcement

The PRESS & Policy Enforcement Framework (PPEF) associated procedures in NEMO will be delivered as a framework that aims to address two main objectives. On one hand the project envisages to analyse the *PRESS* (Privacy, data pRotection, Ethics, Security & Societal) concerns associated with the next generation AIoT, especially related with personalized sensing and potential privacy and ethical intervention to the human life. On the other hand, the abovementioned framework projects to materialize a by-design police enforcement set of technical solutions which will enforce compliance of the NEMO-hosted micro-services to the policies defined by the service and the application providers. The policies will be multi-faced, able to cope with the different aspects of the applications life cycle (security, privacy, costs, environmental impact, etc.).

The PPEF design and development will capitalize on a thorough research on Cloud Native Cloud Foundation (CNCF) [35] policy definition and enforcement tools. The monitoring of the underlying NEMO infrastructures' resources will be orchestrated by Prometheus [36], which is a proven CNCF accepted, systems monitoring toolkit. Additional CNCF approaches and/or tools might be selectively adopted. Moreover, AI/ML solution, provided through the CF-DRL component will be incorporated into the PRESS & Policy Enforcement framework to enhance the quality of the policy related decision making towards NEMO hosted micro-services.

5.5.4.1 Main functionalities

The definition of the high-level architecture of the PRESS & Policy Enforcement framework is driven by the 4+1 architectural model. The logical view of the framework underlines the key functionalities that the framework aims to deliver as presented in Figure 39. In addition, the main actors of the framework, namely the "*Service & Application provider*" and "*Policy Makers*" are also illustrated.

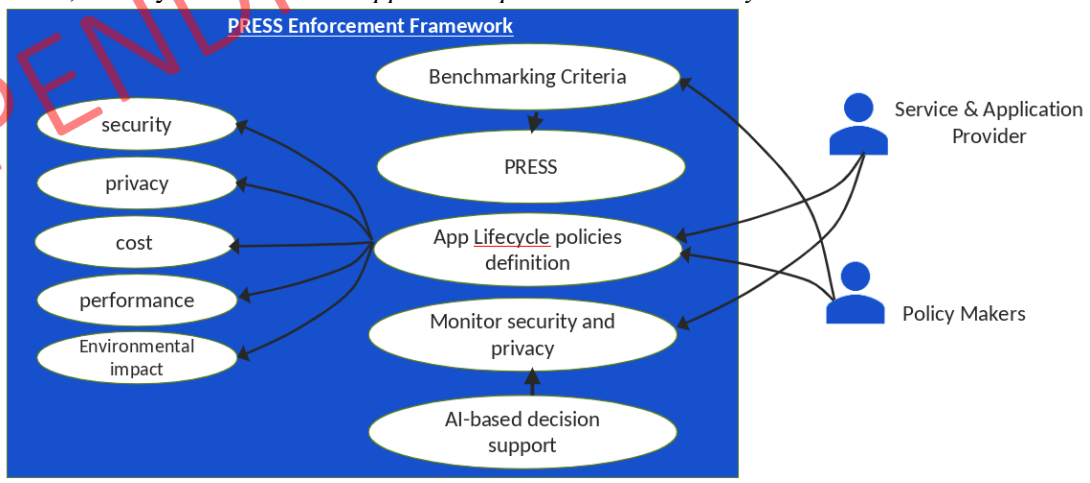

Figure 39: PRESS & Policy Enforcement framework logical view

More specifically, starting with PRESS, internal or external policy makers will provide a set of benchmarking criteria and processes that aim to assess the impact of NEMO-hosted services limiting

the exposure of their ethics, privacy, societal and privacy aspects to third parties. The objective here is to design and implement a toolkit that will safeguard the aforementioned PRESS concerns.
Then, with respect to the Policy Enforcement aspect of the framework the NEMO Policy Makers will define the context of the policies that will be monitored. The policies that will be defined will cover security (e.g HTTPS), privacy (e.g. GDPR), cost, performance (e.g. availability, latency, bandwidth) and environmental impact (e.g. CO2, energy) related properties. In addition, these policies will structure a set of Service Level Objectives (SLOs) that will be monitored by the NEMO adopted monitoring framework, Prometheus.
Then, the NEMO Service and Application provider (or NEMO adopter) will define the preferred policies boundaries in a Service Level Agreement (SLA). The NEMO meta-OS, through the PRESS & Policy Enforcement framework will monitor the associated services or application and will safeguard the compliance against the agreed SLA. Finally, the policy enforcement process will be enhanced by AI-based decision support feedback.
The interactions of the PRESS & Policy Enforcement framework are presented in tabulated format below.

5.5.4.2 Interactions

PPEF interacts with other NEMO components, as described in Table 34.

Table 34: Interactions of the PPEF with other NEMO components

Interacting with	Interaction type	Description of interaction
Plugins Engine	Input	PRESS & Policy Enforcement framework will take as an input the SLA definitions that concern the plugins that will be deployed in NEMO meta-OS.
MOCA	input, output	PRESS & Policy Enforcement framework will receive policy requirements by 3rd parties and will provide to the MOCA infrastructure resource utilization metrics that concern the resource that were made available by 3rd parties through MOCA.
Migration as a Service (MaaS)	output	The PRESS & Policy Enforcement framework will provide as an output migration requests. The decision for these requests will be driven by the framework policy enforcement procedures.
Meta-Orchestrator	Input	The PRESS & Policy Enforcement framework is tightly interconnected with the NEMO kernel and meta-orchestrator. Through the established interfaces the meta-orchestrator will receive input that describes the underlying infrastructure resources that host NEMO micro-services and applications.
CF-DRL	Input	The PRESS & Policy Enforcement framework will incorporate a trained AI/ML model that will enhance the decision-making process. More specifically, the model will receive as input monitoring data of the underlying infrastructure that are collected via Prometheus and given on the specific SLA that is defined by the NEMO adopter will assist the actions that need to be taken in order to ensure the conformance towards the defined SLA.

5.5.4.3 Requirements

Table 35 below lists the requirements that are related to the NEMO pilots' use cases and are addressed by the PRESS & Policy Enforcement framework.

Table 35: NEMO pilots requirements correlation with PPEF

Requirement ID	Requirement description	Requirement satisfaction
SF_01_FR01	The platform must provide access to measurements.	The PPEF will capitalize on the measurements collected by the Prometheus deployments.
SF_01_FR07	The platform should support monitoring of SLOs, e.g., related to energy consumption or CO2 emissions.	The SLOs that will be monitored by the PPEF will include energy related metrics. Kepler and/or Scaphandre will be utilized for that reason.
SF_01_FR08	The platform must respect data sovereignty and privacy requirements.	The PPEF will verify and validate data sovereignty and privacy requirements. If required the deployment of a service will be conducted in SEE.
SF_01_FR013	The platform should be able to perform alternative scheduling or geographical distribution of smart farming services based on user goals.	The PPEF will dictate the deployment requirements based on the agreed SLA. In view of that, the deployment of a service will take into account the user goals including energy consumption related SLOs.
SF_01_FR014	The Smart Farmer should be able to define strategies for the use of available resources.	The PPEF through the description of the services' SLAs will take into account optimal strategies defined by the user.
SF_02_FR05	The platform must respect data sovereignty and integrity.	The PPEF will verify and validate data sovereignty and privacy requirements. If required the deployment of a service will be conducted in SEE.
SF_02_FR06	The platform must provide access to collected data.	The PPEF will capitalize on the measurements collected by the Prometheus deployments.
SF_02_FR012	The Smart Farmer should be able to define strategies for the use of available resources.	The PPEF through the description of the services' SLAs will take into account optimal strategies defined by the user.
SM_01_NFR02	The platform must have capabilities of a monitoring system	The PPEF will capitalize on the measurements collected by the Prometheus deployments.
SC_01_FR08	NEMO will be able to allocate and launch the required services/VNFs on a location basis.	The PPEF will dictate the deployment requirements based on the agreed SLA. In view of that, the deployment of a service will take into account the user goals including energy consumption related SLOs and optimize the deployment of the service on a location basis.
SC_01_FR11	NEMO must be able to monitor and control the network and ensure adherence to QoS levels (bandwidth, average bit rate, round trip delay).	The PPEF will capitalize on the measurements collected by the Prometheus deployments, ensuring the adherence to QoS levels agreed with the user.

5.5.5 NEMO Federated MLOps

5.5.5.1 NEMO Cybersecure Federated Deep Reinforcement Learning

5.5.5.1.1 Description

The CFDRL component is a component that learns a decision-making model. The learning is collaborative and distributed between multiple entities that learn their own local models. The entities also share their experiences in order to build together a common open model. Each entity will use a Reinforcement Learning (RL) algorithm to learn its model. The learning aims to preserve the privacy of the data gathered by each entity. For that, the entities coordinate using Federated Learning routines. We also address several security concerns. External attackers need to be prevented from interfering with the learning by protecting the data stored and the communications. Ill-intentioned entities need to be guarded from poisoning the model. To this end, encryption, anomalies detection, as well as their associated counter measures, will be implemented.

The CFDRL is connected to the Meta orchestrator for decision making. Given a state of orchestration characterized by a description of the micro-services, actions would be decided by CFDRL and would include migration, placement and scaling and reward take into account the migration time, downtime and overhead time.

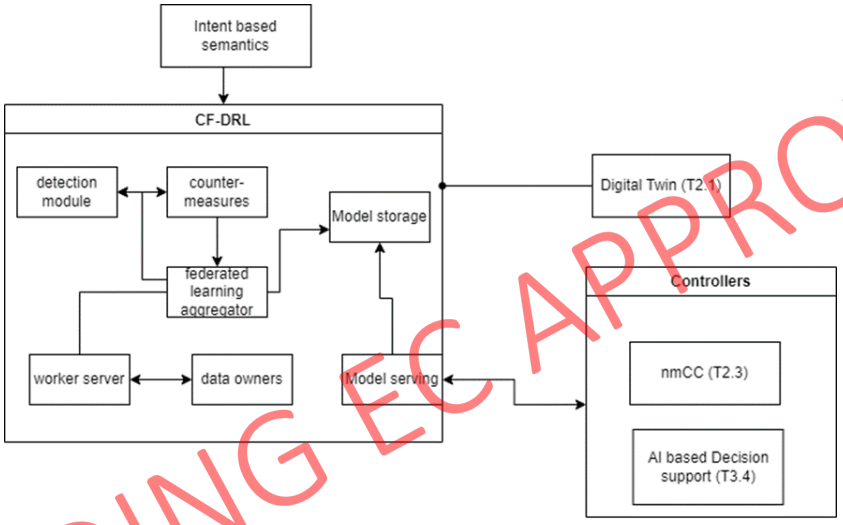

Figure 40: The NEMO Cybersecure Federated Deep Reinforcement Learning component

5.5.5.1.2 Main functionalities

The main functionalities of the CFDRL component are:

- Learning from distributed data: Multiple entities build their local model from interacting with the environment. They store the history of interactions in a replay buffer that is collection of episodes where each episode is composed of a list of (state, action, reward) triplets describing the state of the environment, the action taken by the agent and the reward feedback generated. The learning will be in charge of a RL Trainer. The entities are either capable of generating their own replay buffer from an interaction with the environment (or a simulator of it) or are provided with off-line data logs (the replay buffer of another agent).
- Aggregating the models: The models learned by each entity are aggregated to build a final model. The simpler way to do this is to have a central server in charge of collecting the models of each entity and merge them into one model.
- Ensuring privacy of the data: By design all entities do not communicate their private data, following the Federated Learning they only communicate what is necessary to update the global model. The privacy can be also ensured by encrypting the communication between the server and the entities.

- Serving the model for sequential decision making. The model, once learned, is stored so that it could be shared or served to other components.
- Detecting attacks on the model and data: The CFDRL is composed of a detection module that is able to scan the data and the communication of the models to see if attacks have occurred. If the diagnostic of the detected attack requires it, countermeasure will be implemented.
- Generating attack scenario: To test the robustness of the learning procedure, attacks are generated. Standard scenario includes poisoning the data or serving wrong models to the server.

5.5.5.1.3 Interactions

Interactions of CFDRL have been defined so far only with the meta-Orchestrator, as described in Table 36.

Table 36: Interactions of the CFDRL with other NEMO components

Interacting with	Interaction type	Description of interaction
Meta orchestrator	Input/Output	For learning, inputs are historical data of nodes' and services' activity. For deployment, the inputs are the current state. The output is a model in model sharing or actions in model serving.

5.5.5.1.4 Requirements

CFDRL contributes to the NEMO requirements, listed in Table 37, following the numbering and description adopted in D1.1.

Table 37: NEMO requirements addressed through CFDRL

Requirement ID	Requirement description	Requirement satisfaction
NEMO_FR04	The platform should support ML/FL training and ML model sharing/serving.	CFDRL will leverage the use of federated learning for collaborative ML training and model sharing.
NEMO_FR08	The platform must respect data sovereignty and privacy requirements.	CFDRL will guarantee data privacy through the use of privacy preserving Machine learning.
NEMO_NFR05	Secure communication of sensitive data related to the infrastructure should be provided.	CFDRL will implement secure and private communications through the use of efficient secure mechanisms.
NEMO_NFR18	The platform must provide mechanisms for security and data privacy.	CFDRL will provide secure innovative mechanisms to detect attacks and guarantee data privacy through, for instance, privacy preserving machine learning, and generative adversarial networks.

5.5.6 NEMO Cybersecurity & Unified/Federated Access Control

The high-level conceptual view of the Cybersecurity & Unified/Federated Access Control module of NEMO is given in Figure 41.

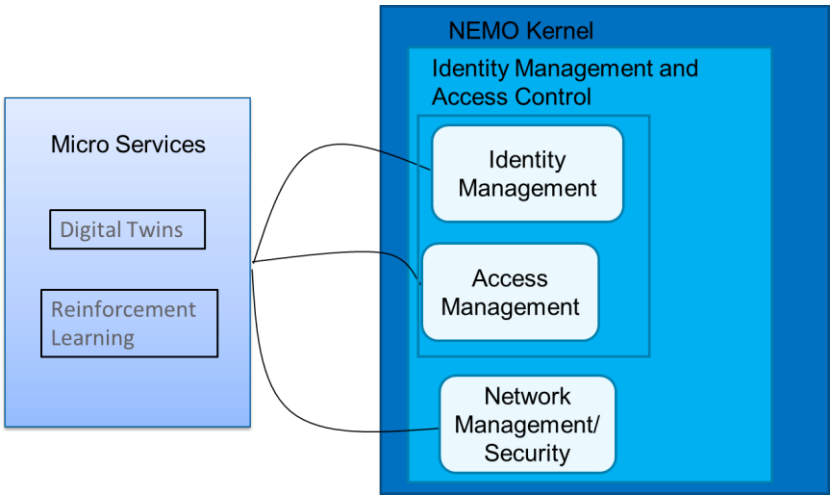

Figure 41: The NEMO Cybersecurity & Unified/Federated Access Control

All the sub-modules of the cybersecurity module are analytically described in the next sub-sections.

5.5.6.1 Identity Management

5.5.6.1.1 Description

The access control of NEMO will implement an Identity and Access Management's (IAM) system which role is to ensure that the right individual has access to the right resources. This is achieved with the combination of the two main components of the IAM system: Identity Management focuses on the provisioning and de-provisioning of identities and Access Management targets authentication, authorization, and policy management.

5.5.6.1.2 Main functionalities

The NEMO IAM system will offer the following services:

- Authentication Services: Authentication involves verifying the user's credentials to permit access to protected resources. Apart from the traditional authentication method of using usernames and passwords for user verification, IAM offers multi factor authentication such as hardware tokens, OTPs and more things that we are going to analyze later on.
- Authorization Management services: Authorization policies guarantee that users can only access the resources and services they are entitled to. According to the role assigned by the organization, a user is given certain privileges and levels of access.
- Identity Management: Identity provisioning describes the procedure of assigning unique credentials to the user such as digital ID or account. De-provisioning is the opposite of provisioning, where the user's account is revoked. LDAP and Active directories are used to manage this process.
- Federated Identity: Federated Identity Management is the process of linking a user's digital identity and attributes between multiple applications through a third-party provider. The identity provider saves user data and login credentials and enables single sign-on without requiring a password. This process is achieved through the exchange of tokens between the identity provider and the service provider using standard identity protocols which we are going to analyze later in this paper.
- Compliance Management: Every system needs to be monitored and reviewed to ensure it's working properly. That also applies to IAM systems, to ensure that it complies with the desired security standards and policies.

The IAM architectures provide a collection of policies, standards, and procedures to ensure that the users are authentic.
The NEMO authentication process (including the services and the management) cover the steps and procedures taken to verify an identity or characteristics claimed by an entity. It can be described as the process that lets us confirm who the user is. There are many policies and standards used to achieve this goal.
The NEMO identity management is based on LDAP which stands for Lightweight Directory Access Protocol, and it is used to manage access credentials stored in Directory Services. In LDAP the information is stored in a tree data structure with different hierarchical levels, commonly at the top level is the root node, below are more nodes representing the groups of objects and at the lower levels are unique objects. An object stored in this structure is called an entry. An entry is composed of data and a unique DN (Distinguished Name). DN is the full address of an entry in the LDAP tree, for example a DN can be composed of an ID which is unique to the entry, the group unit which refers to the group the entry is part of and all the other groups above this subgroup in the hierarchy. We can view the DN as a path to access a specific object. The data referring to an entry are called attributes; attributes are of different types and values depending on the information stored. The client server interaction goes like this:

- A user sends a request to access information stored in an LDAP Server
- The server requires the user to provide the necessary credentials for authentication.
- If the authentication is successful, the server responds with an answer pointing to the location of the information the client requested. Otherwise, access id denied to the client.

LDAP can be a useful addition to an IAM system as it offers a way to manage data centrally and securely.

The federated identity subsystem of NEMO is based on a Single Sign-On (SSO) mechanism and is an authentication method that grants access to the user in different applications with a single set of credentials. SSO as a service helps users get rid of the hassle of entering their credentials every time, they want to use a different application, resulting in an improved experience and in better time management. There are three forms of SSO systems.
In local SSO, the authentication starts when the user logs on to their system. After the user provides the necessary credentials, the system creates a cookie or a token that contains authentication and authorization information. The token is stored in the system, so later when the user attempts to access an application/microservice, the user's token is provided t to the application. If the token is valid the user gain access to the service without having to enter his username and password
Moreover, in NEMO's Federated SSO the authentication process is done through a third-party identity provider (IdP). When a user wants to access a resource of a service provider (SP), the SP sends an authentication request, the authentication request is forwarded to the IdP, which shows a login page to the user, after the user enters the correct login credentials the IdP returns a token containing the necessary authentication and authorization information. The generated token is forwarded to the SP, which validates the information and gives the user permission to access their resources. SAML and OpenID Connect are the protocols mostly used for SSO implementations. Analysing these two protocols will provide us with a better understanding of the SSO process.

5.5.6.1.3 Interactions

The IAM component interacts with other NEMO components, as described in Table 38.

Table 38: Interactions of the IAM component with other NEMO components

Interacting with	Interaction type	Description of interaction
Digital Twins Services	Input	NEMO users' credentials
Reinforcement Learning Services	Input	NEMO users' credentials

Interacting with	Interaction type	Description of interaction
Privileges/Primitives Control	Output	Metadata/Tokens for authenticating NEMO users

5.5.6.1.4 Requirements

IAM contributes to the NEMO requirements, listed in Table 39, following the numbering and description adopted in D1.1.

Table 39: NEMO requirements addressed through IAM

Requirement ID	Requirement description	Requirement satisfaction
NEMO_FR03	The platform must provide options to manage users.	The NEMO identity management system will allow for the handling of several users with different privileges/access rights/etc
NEMO_FR08	The platform must respect data sovereignty and privacy requirements.	Through the NEMO identity management and access control modules each user will have access only to the specified resources/data and no user will be able to access other's users resources and/or data
NEMO_FR23	The platform must provide access to collected data.	Through the identity management and access control modules each user will be able to access the collected data allowed
NEMO_FR24	The platform must provide access to the devices.	Through the identity management and access control modules each user will be able to access the allowed devices
NEMO_FR25	The platform must provide options to manage users.	The NEMO identity management system will allow for the handling of several users with different privileges/access rights/etc
NEMO_NFR18	The platform must provide mechanisms for security and data privacy.	Through the NEMO identity management and access control modules each user will have access only to the specified resources/data and no user will be able to access other's users resources and/or data

5.5.6.2 Access Control

5.5.6.2.1 Description

The NEMO Access control management system will be based upon control policies. They define the rules and conditions that determine how access rights and permissions are granted or denied to users, resources, or functionalities within a system. These policies play a crucial role in maintaining the security, confidentiality, integrity, and availability of sensitive information and resources.
Within NEMO we will investigate the following set of access control policies:

- **Policy Definition**: Access control policies are typically defined by administrators or security experts within an organization. They are documented and implemented as a set of rules that dictate the behavior of the access management system. Policies can be expressed in a formal language or represented through graphical interfaces, depending on the complexity and requirements of the system.
- **Access Control Models**: Access control policies are based on different access control models, which provide a framework for enforcing access control decisions. Common access control models include:

- **Discretionary Access Control (DAC)**: In DAC, the owner of a resource has control over granting or revoking access rights to other users. Each resource can have different access rules defined by its owner.
- **Mandatory Access Control (MAC)**: MAC enforces access control based on a set of predefined rules and labels assigned to users and resources. It is commonly used in high-security environments, where access decisions are determined by security classifications and clearances.
- **Role-Based Access Control (RBAC)**: RBAC assigns permissions to roles and then assigns roles to users. Access decisions are based on the roles assigned to users rather than their individual identities. This model simplifies administration and improves scalability.
- **Attribute-Based Access Control (ABAC)**: ABAC considers various attributes, such as user attributes, resource attributes, and environmental attributes, to make access control decisions. It provides fine-grained control over access based on multiple factors.
- **Rule-Based Access Control (RuBAC)**: RuBAC uses a set of rules or conditions to determine access rights. These rules can be based on various factors, including user attributes, time of day, location, or any other contextual information.

- **Conditions and Factors**: Access control policies take into account various conditions and factors to determine access rights. These factors may include:
  - **User Identity**: Policies can consider the user's identity, such as username, role, group membership, or user attributes, to grant or deny access.
  - **Resource Identity**: Policies can define access rights based on the identity or attributes of the resource being accessed.
  - **Contextual Information**: Policies can consider contextual information, such as time of day, location, device used, network characteristics, or any other relevant information, to make access control decisions.
  - **Relationships**: Policies can define access rights based on relationships between users, resources, or entities within the system. For example, granting access to a resource based on a user's manager or team membership.
  - **Security Classifications**: In some cases, access control policies may consider security classifications or labels assigned to resources and users to enforce stricter access control in high-security environments.
- **Policy Enforcement**: Access control policies are enforced by the access management system. When a user requests access to a resource or functionality, the system evaluates the relevant policies and determines whether access should be granted or denied. This evaluation process involves matching the user's attributes and contextual information against the defined policies.
- **Policy Management**: Access control policies require ongoing management and maintenance. Administrators are responsible for reviewing, updating, and refining policies as the system's requirements evolve. They may need to adapt policies to new regulatory standards, organizational changes, or security threats.
- **Auditing and Compliance**: Access control policies play a crucial role in auditing and compliance efforts. By enforcing policies and logging access.

5.5.6.2.2 Main functionalities

The designed NEMO Access control management system will support the following functionalities.

- RBAC: RBAC will allow for the definition of roles with specific permissions and assign users to those roles. It simplifies access control by managing permissions at the role level rather than assigning them individually to each user.
- Authorization and Access Policies: The system will support the definition and enforcement of a number of other access policies; the list of potential candidates includes Attribute-based access control (ABAC), Context-based access control (CBAC), Rule-based access control. Together

with the identity management system the NEMO security module will be able to specify granular access rules, conditions, and restrictions based on user attributes, roles, or other factors.

- Audit Logs and Reporting: The system will maintain comprehensive audit logs of user activities, authentication attempts, and access control events. These logs are used for security monitoring, compliance audits, and forensic analysis.
- API Security and Authorization: The system will provide security mechanism for securing APIs (Application Programming Interfaces) and enforcing authorization policies for API calls. This will ensure that only authorized applications and users can access and interact with APIs.

5.5.6.2.3 Interactions

The Access Control component interacts with other NEMO components, as described in Table 40.

Table 40: Interactions of the Access Control component with other NEMO components

Interacting with	Interaction type	Description of interaction
Identity Management	Input	Metadata/Tokens for authenticating NEMO users
NEMO Resources	Output	Allowing/Denying access to resources
Network Management	Output	Network packets coupled with user metadata

5.5.6.2.4 Requirements

The Access Control component contributes to the NEMO requirements, listed in Table 41, following the numbering and description adopted in D1.1.

Table 41: NEMO requirements addressed through the Access Control component

Requirement ID	Requirement description	Requirement satisfaction
NEMO_FR08	The platform must respect data sovereignty and privacy requirements.	Through the NEMO identity management and the access control modules each user will have access only to the specified resources/data and no user will be able to access other users' resources and/or data
NEMO_FR23	The platform must provide access to collected data.	Through the identity management and access control modules each user will be able to access the collected data allowed .
NEMO_FR24	The platform must provide access to the devices.	Through the identity management and access control modules each user will be able to access the allowed devices
NEMO_NFR18	The platform must provide mechanisms for security and data privacy.	Through the identity management and access control modules, each user will have access only to the specified resources/data and no user will be able to access other users' resources and/or data.

5.5.6.3 Network management and Security

5.5.6.3.1 Description

The network security module will be based on a message broker, which will be the NEMO kernel component which will play a crucial role in facilitating communication and coordination among distributed systems or applications. The message broker serves as an intermediary for message exchange, providing features such as message routing, queuing, and transformation. By employing

asynchronous communication patterns, message brokers enable loose coupling between senders and receivers, allowing them to operate independently and asynchronously.
Message brokers implement advanced queuing mechanisms to ensure reliable and efficient message delivery. They store messages in queues, providing persistent storage that can withstand system failures, network interruptions, or temporary unavailability of recipients. This ensures message durability and enables fault-tolerant communication.
In addition to queuing, message brokers provide sophisticated message routing capabilities. They employ predefined rules, often based on content-based or header-based filtering, to determine the appropriate destination for each message. These rules allow for flexible and dynamic message distribution, ensuring that messages reach their intended recipients or are processed by specific components within the system.
Message brokers also offer message transformation capabilities to address the heterogeneity of communication protocols, data formats, or message structures. They facilitate seamless interoperability by converting messages from one format to another as they traverse the broker. This enables integration between disparate systems and enhances the flexibility and adaptability of the overall messaging infrastructure.
Security is a critical aspect of message brokers. They employ authentication mechanisms to verify the identity of message senders and recipients, ensuring that only authorized entities participate in message exchange. Encryption techniques may be utilized to protect the confidentiality and integrity of messages during transmission. Access control mechanisms are enforced to govern the permissions and privileges associated with sending, receiving, or processing messages, bolstering the overall security posture.
Message brokers often offer management and monitoring capabilities, providing insights into message flows, performance metrics, and system health. Administrators can track message activity, monitor queue depths, and diagnose potential issues, enabling efficient troubleshooting and system optimization.
Scalability features, such as clustering, load balancing, or replication, are often incorporated to handle increasing message volumes, distribute the load across multiple broker instances, and ensure high availability.
Overall, message brokers serve as a critical middleware infrastructure for achieving robust, scalable, and reliable communication in distributed systems. They enable loose coupling, asynchronous communication, and seamless integration, while addressing challenges related to message persistence, routing, transformation, security, and management. Researchers in the field leverage message brokers to design and implement distributed architectures, enabling efficient communication among diverse components and systems.

5.5.6.3.2 Main functionalities

The main functionalities of the NEMO message broker (NMB) which will also handle the network security part will be the following:

- Message Routing: The NMB will allow messages to be sent from a sender to one or more receivers based on predefined rules or routing criteria. It will provide flexible routing mechanisms to direct messages to the appropriate destinations, ensuring that they reach the intended recipients.
- Message Transformation: The NMB will perform message transformation or enrichment of tasks allowing messages to be translated from one data format to another, ensuring compatibility between different microservices that may use different message formats or protocols.
- Message Queuing: The NMB will provide queuing capabilities, enabling asynchronous communication between sender and receiver.
- Message Filtering and Content-Based Routing: The NMB will support filtering and routing of messages based on their content or attributes. It will be able to examine message properties or payload contents and selectively route them to different destinations based on predefined rules or conditions.

- Message Acknowledgment and Delivery Guarantees: The NMB will support acknowledgment mechanisms to ensure reliable message delivery. It will track whether messages have been successfully received and processed by the intended recipients and can provide guarantees regarding the delivery status.
- Scalability and Load Balancing: The NMB will be designed in such a way so as to handle high message volumes and support the NEMO highly distributed end-platform. It will provide mechanisms for load balancing, distributing message processing across multiple nodes or instances to achieve scalability and performance.
- Security and Authentication: The NMB will enforce security measures such as authentication and authorization, using the identity management and access control units to ensure that only authorized systems or components can send or receive messages as well as encryption and decryption of messages. This will help protect against unauthorized access or tampering of messages.
- Monitoring and Management: The NMB will include monitoring and management functionalities, providing insights into message flow, performance metrics, and system health which will allow for identifying anomalies which can be cause either by performance issues and/or by security attacks.

5.5.6.3.3 Interactions

NMB interacts with other NEMO components, as described in Table 42.

Table 42: Interactions of the NMB with other NEMO components

Interacting with	Interaction type	Description of interaction
Network Management	Input/Output	Network packets sent/received by Network Management systems running on other nodes
Access Control	Input	Network packets coupled with user metadata

5.5.6.3.4 Requirements

NMB contributes to the NEMO requirements, listed in Table 43, following the numbering and description adopted in D1.1.

Table 43: NEMO requirements addressed through NMB

Requirement ID	Requirement description	Requirement satisfaction
NEMO_FR08	The platform must respect data sovereignty and privacy requirements.	The NMB in collaboration with the NEMO identity management and the access control modules will allow each user to have access only to the specified resources/data while no user will be able to access other's users resources and/or data
NEMO_FR23	The platform must provide access to collected data.	The NMB in collaboration with the identity management and access control modules will allow each user to access the corresponding collected data
NEMO_FR24	The platform must provide access to the devices.	The NMB in collaboration with the identity management and access control modules will allow each user to access the corresponding devices

Requirement ID	Requirement description	Requirement satisfaction
NEMO_NFR18	The platform must provide mechanisms for security and data privacy.	The NMB in collaboration with the identity management and access control modules will allow NEMO users to have access only to the specified resources/data while no user will be able to access other's users resources and/or data

5.6 Process View

The process view in the NEMO meta-OS architecture is realized through sequence diagrams, which are presented in the following subsections.

5.6.1 Workload Deployment

The first interaction of third-party users with NEMO, when referring to metaOS consumers and partners, is the registration of workloads, which are candidate applications or plugins, which may be later executed in the metaOS.

The sequence diagram in Figure 42 depicts the workflow suggested for NEMO components in order to deliver these capabilities.

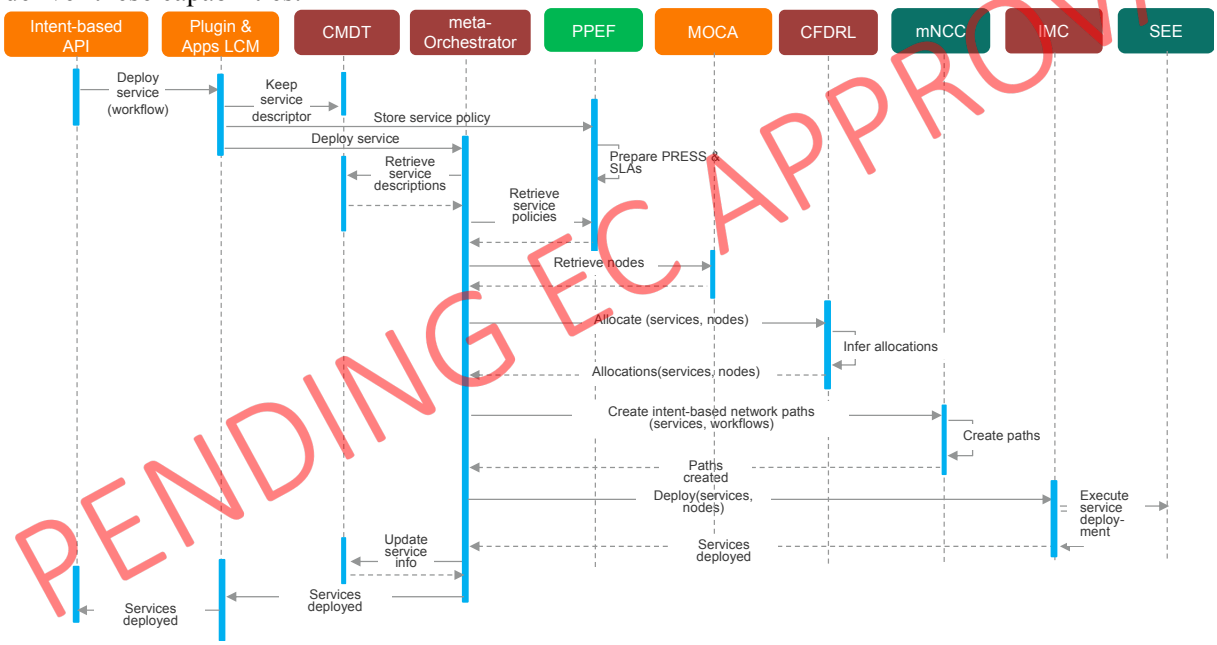

Figure 42: Sequence diagram for workload deployment

As a first step, the user should communicate with the Intend-based API, in order to receive a token and place the workload registration request. This follows internal processes within the API, including the verification from the meta-OS provider, validation of compliance to NEMO rules and specifications and, in case these are successful, registration in NEMO's registry for workloads. Then, a deployment request may be issued (e.g. through kubectl CLI), which reaches the Plugin & Applications LCM. As part of its admission controller functionality, LCM checks the admissibility of the request and communicates workload description details to the CMDT. Moreover, LCM communicates with the PPEF about the workload policies, reflecting user-defined or workload -specific requirements, which may relate to performance, PRESS, environmental or other objectives. PPEF ensures that policies, SLOs and probes are in place, in order to support metering of those objectives. In parallel, the LCM has communicated the workload deployment request to the meta-Orchestrator, which retrieves the workload

descriptor via CMDT and the workload policies' information via the PPEF. Moreover, the meta-Orchestrator retrieves nodes' information via MOCA, in order to take nodes' and clusters' availability into account during the workload placement decision. Then, the meta-Orchestrator consults CF-DRL in order to take the placement decision, as CF-DRL implements the intelligence for such a decision, through an appropriate ML model development, training and serving. Upon receiving the placement suggestions from CF-DRL, the meta-Orchestrator communicates with the mNCC in order to create appropriate network paths or micro-slices, ensuring that the network-related aspects of the workload execution will be appropriately addressed. The mNCC creates the paths that will provide the required network connectivity and notifies the meta-Orchestrator accordingly. Then, the meta-Orchestrator asks the IMC to execute the deployment request on the selected cluster. If it requires secure execution, this is done through SEE. Upon the workloads get deployed, both the meta-Orchestrator, the CMDT, the LCM and, eventually, the user get notified.

5.6.2 Workload Migration

The sequence diagram in Figure 43 describes the flow of interactions among NEMO components for delivering the workload migration capability. In this case, workloads are already running, and a migration need might arise as a result of service or resource monitoring.

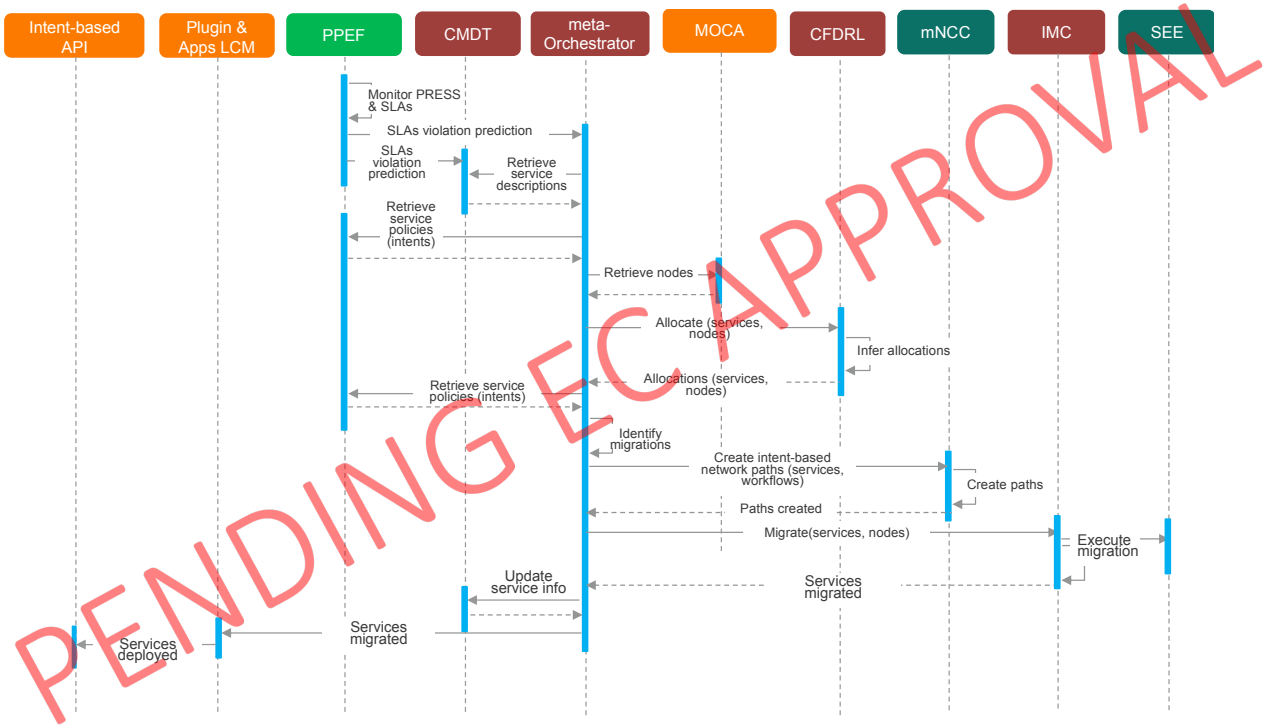

Figure 43: Sequence diagram for workload migration

During workload execution, PPEF continually monitors the metrics, quantifying the performance, usage, PRESS, environmental or other objectives set for the execution of the workloads. Once a potential SLA break is forecasted, PPEF notifies the meta-Orchestrator and the CMDT accordingly. The latter will record the event, while the meta-Orchestrator will take action in order to not let the SLA break event happen. It retrieves the required information, i.e., service descriptions, policies and nodes' information via the CMDT, PPEF and MOCA, respectively. Then, it triggers a new suggestion on workload placement from CF-DRL. Based on CF-DRL's outcome, the meta-Orchestrator will identify the migrations that need to be make and will ask from mNCC to create the relevant network paths. As soon as they are ready, the migration request is forwarded to the IMC, in order to execute it, i.e., migrate a running workload from one cluster to another. Once the migration is completed, the meta-Orchestrator,

the CMDT and the LCM are informed accordingly, including whether the migration has been successful or not.

5.7 Development view

The development view in NEMO provides technical implementation details for the NEMO components. It will be provided in forthcoming deliverables of WP2, WP3 and WP4, which will detail the design options and development activities for the individual components.

5.8 Physical view

The physical view represents a topology map, guiding the deployment of the NEMO meta-OS. It will be part of WP4 deliverables.

6 NEMO Validation & Verification Benchmarking Framework

Validation & Verification (V&V) is an intrinsic component to establish a software model's simulation and prediction capacities with set parameters for the planned use case. Validation is not limited to success/failure exercise for the software simulation rather assesses the uncertainty in the prediction capacities after crunching the data. Then it can be manually judged regarding the suitability and sustainability for the given software application of the NEMO use cases. Ultimately, it provides a structured, documented, transparent approach to integrating the use cases across scales.
In this section, we provide the appropriate guidelines which are going to be used for the implementation of the V&V framework in the context of Task 4.4. The goal of the V&V is two-flowed (i) to support developers in evaluating the performance of their services and (ii) to address the concerns that services operators have in hosting 3rd party services upon their infrastructure. For these reasons, NEMO V&V will provide a well-structured framework, as part of the DevOps approach, that facilitates several tests per each new service from the development, integration, and deployment phases ensuring on one hand that the new service addresses the requirements and Key Performance Indicators (KPIs) and on the other hand that is compatible with the innovative features that NEMO platform offers (i.e. resource scaling, high availability, full-stack automated operations, etc.). One of the essential characteristics of the NEMO V&V framework is modularity meaning that the system should be flexible enough to integrate new services and test tools easily. Considering the heterogeneity of modern network services, it is obvious that each service requires different testing approaches and tools. So, the V&V should provide some common tests applicable to all services (i.e., NEMO platform compatibility tests) but should also support the integration of services' specific tests that verifies specific aspects of each service.

6.1 Overall Verification and Validation strategy

6.1.1 User Service Validation

In modern times, each service is a collection of multiple microservices that are attached to the same virtual network and collaborate with each other to provide specific services to the end users. In the NEMO context, each application should be able to be verified in both service and microservice levels in such a way that ensures it is able to cope with changes in the traffic load of infrastructure. The most common approach to achieve this is to create specific tests per service investigating the behavior under stress by creating artificial load. This approach aims to stress each service in a sandbox environment and validate that the SUT (System Under Test) performs as expected under stress and provides guaranteed level of QoS under any situation taking advantage of the innovative features that NEMO meta-orchestrator provides (i.e., high availability, intent based migration, resource scaling, etc). Without this, it is not possible for a developer and a service provider to know if a service is safe to be deployed. So, the service validation should include tests for service compliance with NEMO orchestrator, functionality validation (Data in/Data out), security tests, and performance testing based on KPIs.

6.1.2 Testing Results

The V&V benchmarking is not intended as a debugging solution for the developers, this activity requires a plethora of information from low-level metrics, logging, system reporting, system load when the issue occurred, and captured input and output information. Then all of this data needs to be presented in an easy-to-use human readable format. It quite simply is a huge task, one worthy of detailed analysis in its own right and it is beyond the scope of NEMO.
On the other hand, the V&V benchmark is expected that when a service fails on one or more tests a brief report is provided and stored in the system for future reference and further analysis. The full details and root cause of the failure will be useful for the developer to understand via their own development and

debugging environment. It is clear that the developer plays a critical role in this approach as he needs to design and provide specific test scripts tailored to the features and characteristics of his application. Furthermore, generic tests will be provided that can be applied to all services. The test results are going to be used to generate the benchmark certificate for each service, which is mandatory for the safe deployment of the service to the NEMO platform. The execution of the tests will be fully integrated with the CI/CD/CP pipelines which are going to automatically invoke each time the developer pushes new code or manually before each new service release.

6.2 Verification and Validation methodology

NEMO V&V framework is essential to ensure software quality, version control, automated testing and V&V process administration. In this direction, there are some open-source tools that support automated scheduling actions and can be easily integrated into DevOps approaches. Some of these tools are mentioned below and they can be part of the core components of the V&V scheduling process.

Git [37] can be considered the best version control system that records changes to a file or even a set of files over time in case it is needed to recall any specific version of the code. GIT strongly supports non-linear development and is compatible with multiple protocols such as HTTP, FTP, and SSH. It has been adopted from the most known DevOps platforms (i.e., Github, Gitlab, Bitbucket, etc.) that provide a distributed cloud repository model with cryptographic and user authentication. The toolkit-based design allows pluggable merge strategies with periodic explicit object packing.

Selenium [38] testing framework allows web application testing across multiple web browser platforms and supports multiple modern programming languages. it is an umbrella project for a range of tools and libraries that enable and support the automation of web browsers by providing extensions to emulate user interaction with browsers, a distribution server for scaling browser allocation, and the infrastructure for implementations of the W3C WebDriver specification that lets you write interchangeable code for all major web browsers. Selenium brings together browser vendors, engineers, and enthusiasts into an ecosystem for the automation of web application testing and development.

Jira [39] is a software application developed by the Australian software company Atlassian that allows teams to track issues, manage projects, and automate workflows. It can be considered as a process administration and work management tool to support various use cases in NEMO which has varied requirements and can act as test case management in an agile software development scenario.

Jenkins [40] offers a simple methodology to set up a CI/CD environment which can have any combination of languages with different source code repositories and automates the routine software development tasks using pipelines. When the software change management type is decided upon, Jenkins can also be distributed as a docker image.

Ansible [41] sets itself apart from other tools as other than being automation platform, it is also an orchestration and deployment tool offering CI/CD with zero downtime. The IT ecosystem can have standardised configurations which reduces operational overhead while implementing DevOps strategy.

V&V systematic approach is imperative for ensuring that NEMO technology is cost-effective and provides risk free credible results. The acceptability criteria revolve around the decision to validate the use case industry's needs and evaluation of the components in the software development paradigm.

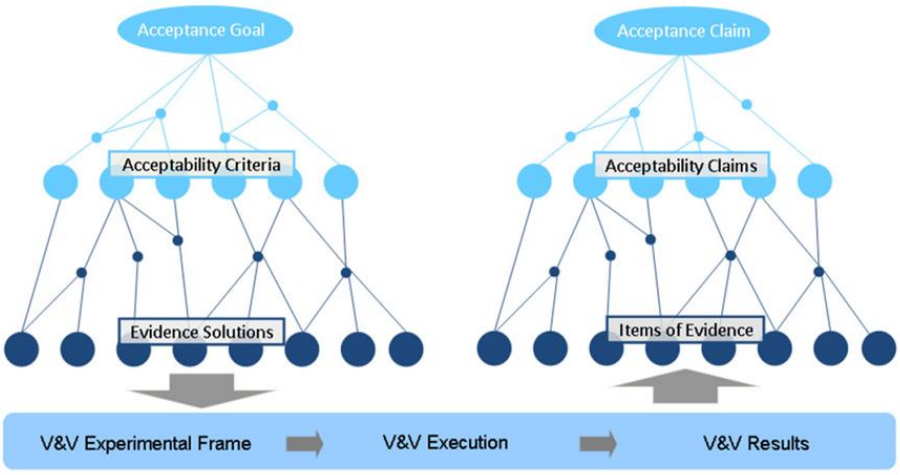

Figure 44: V&V phases

V&V Experimental framework should contain the underlying framework to evaluate the evidence solutions provided by the components, where the reasoning is transparent, traceable, and reproducible. Starting with an acceptance goal in mind, the evaluation criteria are set in accordance with the evidence solutions in the experimental framework. This undergoes the testing framework where the use case requirements are mapped to acceptable test results in the execution stage. The traceability matrix provides a more complete picture as it encapsulates and consolidates the evidence available to develop appropriate and valid results which can be claimed to have acceptable test results.

6.2.1 Testing Approaches

Software testing is a most often used technique for verifying and validating the quality of software and plays a significant role of the software development life cycle (SDLC). The main objective of software testing is to affirm the quality of software system by systematically testing the software in carefully controlled circumstances, another objective is to identify the completeness and correctness of the software, and finally it uncovers undiscovered errors. The most important techniques that are used for finding errors are:

- Blackbox testing refers to examining the System Under Test (SUT) regarding its capabilities without the knowledge of its internal structure, which means that given an input following some specification, blackbox testing verifies whether the SUT behaves correctly and emits the expected output. Therefore, the blackbox testing is most suitable for interface conformance testing as the specification of an interface is exactly a description of expected input and output without considering the implementation that realizes such behavior. As long as the SUT returns the correct output upon a given input regarding to the specification, the tester concludes that the test is successful no matter how the SUT implements this behavior. Functional and non-functional tests are both possible using the blackbox approach.
- White-box testing consists of testing the internal structure of the SUT. It requires a good knowledge of the internal design or code of the SUT but it can give more insight into the SUT's behavior or performance by knowing how the behavior is implemented or how the performance is achieved. In NEMO, the white-box testing will mostly be applied to V&V which refers to testing and analyzing the performance of a service knowing its internal graph. We consider a network service (graph) definition that follows a microservices-based approach that can be functionally decomposed into a set of loosely coupled collaborating functions that interact through well-defined interfaces and possibly depend on themselves. The whitebox testing takes into account monitoring data from the decomposed functions and analyze them to identify how

graph relationships affect the overall performance, and where is the bottleneck deteriorates the performance of the overall service.

It is obvious that each testing approach services different purposes. Table 44 presents the most common testing types with the appropriate approach.

Table 44: Common testing approaches

Testing type	Objective	Testing Approach
Functional Testing	Test functions of the software	Blackbox testing
Performance Testing	Testing software responsiveness and stability under a particular workload	Blackbox testing
Security Testing	Protect data and maintain software functionality	Whitebox testing
Usability Testing	Check ease of use of software	Blackbox testing

6.2.2 Testing Categories

We can distinguish different types or categories of tests. The following is an indicative list of categories that can be facilitated by the V&V platform.

- **Requirement Analysis.** It includes a thorough understanding of the requirements and specifications of the SUT and the NEMO platform. Next, the functional and non-functional requirements should be documented and prioritized, ensuring they are clear, measurable, and testable.
- **Functional testing.** It consists of testing a slice of functionality in a system. The slice of functionality can be a unit of system behaviour, a complex behaviour composed by many unit micro-services, and also can be the whole system's behaviour. It aims to test whether the expected behaviour is successfully done by the system rather than how the behaviour is done with which quality.
- **Performance testing**. It is a non-functional testing technique performed to determine the system parameters in terms of responsiveness and stability under various workload. Performance testing measures the quality attributes of the system, such as scalability, reliability, latency and resource usage.
- **Syntax testing**. It is a static testing which means it does not test the behaviour of the SUT during runtime. It tests the static information associated to the system such as the description files, the metadata, etc.
- **API testing.** It is part of the functional testing that aims to test the implemented API behaves as expected in the specifications.
- **Acceptance Testing.** Involves stakeholders and end-users in defining the acceptance testing which validates that the SUT meets the requirements and expectations of NEMO platform. The acceptance test reflects real-world usage scenarios.
- **System Testing.** It is a comprehensive system testing to evaluate the overall functionality of the SUT. This type of testing verifies that the system meets all the specified requirements and performs as expected in terms of robustness and reliability.
- **Security testing.** This testing technique consists of determining if an information system protects data and maintains functionality as intended. It can involve the above testing techniques to test for example: 1) if an authentication functionality is correctly implemented, 2) if the security policy description is correctly written, 3) if the consumption of resource faced to an attack is controlled and isolated,4) if the SUT vulnerable to penetration attacks, etc.

6.2.3 Test Execution phases

The best way to describe the V&V framework is to define the mechanism that supports the execution of each test by identifying the steps that the framework goes through as it executes it. The tests can be triggered manually by the users, or they can be part of an automated sequence of actions that can be triggered automatically in a periodic way after a specific action i.e. push of new code, release of new component version, etc. All tests will be fully integrated with the CI/CD framework and implemented as executable scripts by well-known automation tools like GitLab runners, Jenkins servers, Ansible, etc. In its general form, a test will consist of the following phases:

- Test Preparation. In test preparation the test environment must be setup, this involves deploying an instance of the SUT in the test environment (i.e., stage, sandbox, etc) and loading all the necessary additional test libraries and tools. Once this is complete the V&V framework is ready to start executing the tests.
- Test Execution. Once the environment is configured and setup the tests are executed in a serial way in order to ensure that the generated results are not affected by parallel test executions.
- Documentation and Reporting. All test plans, test cases, test results, and any issues encountered during testing are documented. Comprehensive reports that provide an overview of the V&V activities and their outcomes are generated and provided to relevant stakeholders.
- Test Shutdown. Once the tests are completed the V&V shutdown the SUT and stores the reports for future reference. Finally, it releases all the allocated resources (i.e., containers, K8s pods, instances of testing and benchmarking tools, memory, etc) and prepares the system for the execution of the next test.

6.2.4 Certification and Labeling

Certification processes are used all over the world in mostly all industrial domains either for regulators or for organizations on a voluntary basis. The aim of a certification process is to ascertain conformity which is defined as the fact that a product, system, body, or even a person meets specified requirements, and which can improve the business interests with regard to products, goods and services. The NEMO V&V framework aims to deliver a base mechanism that can be used for service certification. If the under-test services successfully pass all the predefined tests, then it can be considered a certified service and it can be safely deployed in the NEMO continuum.

A group of stakeholders, including NEMO infrastructure providers and operators, is responsible for defining the specifications and requirements that need to be met by services in order to ensure the compatibility of each 3rd party service with the NEMO platform. The V&V framework will be responsible to execute the tests and collect/store the test results in the V&V for further processing. If the results are satisfactory to conclude the conformity of the services, the services are labelled as “passed”, and should be made available for deployment by the m-orchestrator.

7 Conclusions

The present document has provided the NEMO metaOS architectural specifications. The document presents both the NEMO metaOS meta-architecture and its instantiation for the NEMO metaOS.
Through the proposed metaOS meta-architecture, NEMO aims to facilitate design and development of higher-level (meta) operating systems for the smart Internet of Things with strong computing capacity at the smart device, system and edge-level, embedded in a compute continuum from IoT-to-edge-to-cloud. The NEMO meta-architecture is proposed as a Reference Architecture on top of existing reference architectures, which aims to provide guidance on evolving or creating new meta-OS architectures. In order to achieve this, NEMO in this document presents the Meta-Architecture Framework (MAF), following the conceptual model defined by ISO/IEC/IEEE 42010 for architecture descriptions.
Then, the proposed MAF is instantiated for the description of the NEMO metaOS architecture. The document provides the Network, User, Logical, Functional and Process views of the architecture, while the Operational views is provided as use case scenarios' descriptions, which have been provided in D1.1. Moreover, the Development and Physical views refer to future work of the project and will be reported in future deliverables of WP2, WP3 and WP4.
Moreover, the document presents the NEMO Validation & Verification (V&V), presenting the general strategy and methodology in terms of testing approaches, categories and phases, as well as certification & labelling, to be considered during the project's verification and validation activities.
An updated version of this deliverable is expected on M24, within the tentative deliverable D1.3 "NEMO meta-architecture, components and benchmarking. Final version".

8 References

[1] Kubernetes, 2023. [Online]. Available: https://kubernetes.io/.

[2] Canonical, "MicroK8s - The lightweight Kubernetes," 2023. [Online]. Available: https://microk8s.io/.

[3] Rancher Labs, "K3s - Lightweight Kubernetes," 2023. [Online]. Available: https://k3s.io/.

[4] Mirantis, "K0s," 2023. [Online]. Available: https://k0sproject.io/.

[5] The Kubernetes Authors, "minikube," 2023. [Online]. Available: https://minikube.sigs.k8s.io/docs/.

[6] The KubeEdge Project Authors, "KubeEdge," 2023. [Online]. Available: https://kubeedge.io/.

[7] Akri, "Akri website," 2023. [Online]. Available: https://docs.akri.sh/.

[8] Red Hat, "Where the kernel fits within the OS," 2023. [Online]. Available: https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel.

[9] R. Cloutier, G. Muller, D. Verma, R. Nilchiani, E. Hole and M. Bone, "The Concept of Reference Architectures," *Systems Engineering,* vol. 13, no. 1, pp. 14-27, 2010.

[10] ISO, "ISO/IEC/IEEE 42010:2011 Systems and software engineering — Architecture description," 2011. [Online]. Available: https://www.iso.org/standard/50508.html. [Accessed 2023].

[11] ISO, "ISO/IEC/IEEE 42010:2022 Software, systems and enterprise — Architecture description," 2022. [Online]. Available: https://www.iso.org/standard/74393.html. [Accessed 2023].

[12] ISO/IEC, "JTC 1 SC7 Software and Systems Engineering Brochure," 2020. [Online]. Available: https://www.iso.org/files/live/sites/isoorg/files/developing_standards/who_develops_standards/docs/ISO_IEC_JTC%201_SC%207%20Brochure.pdf.

[13] ISO/IEC/IEEE, "A Conceptual Model of Architecture Description," [Online]. Available: http://www.iso-architecture.org/42010/cm/. [Accessed 2023].

[14] Gaia-X, "Gaia-X website," [Online]. Available: https://gaia-x.eu/. [Accessed 28 08 2023].

[15] IDSA, "International Data Spaces," [Online]. Available: https://internationaldataspaces.org/. [Accessed 29 08 2023].

[16] BDVA, "Big Data Value Association," [Online]. Available: https://www.bdva.eu/. [Accessed 29 08 2023].

[17] OpenDEI, "Open DEI website," [Online]. Available: https://www.opendei.eu/. [Accessed 29 08 2023].

[18] AIOTI, "Alliance for IoT and Edge Computing Innovation," [Online]. Available: https://aioti.eu/. [Accessed 29 08 2023].

[19] FIWARE, [Online]. Available: https://www.fiware.org. [Accessed 2022].

[20] IoT-NGIN, "IoT-NGIN project website," H2020 957246, [Online]. Available: https://iot-ngin.eu/. [Accessed 29 08 2023].

[21] ASSIST-IoT, "ASSIST-IoT project website," H2020 957258, [Online]. Available: https://assist-iot.eu/. [Accessed 29 08 2023].

[22] INGENIOUS, "INGENIOUS project website," H2020 957216, [Online]. Available: https://ingenious-iot.eu/web/. [Accessed 29 08 2023].

[23] INTELLIOT, "INTELLIOT project website," H2020 957218, [Online]. Available: https://intelliot.eu/. [Accessed 29 08 2023].

[24] VEDLIOT , "VEDLIOT project website," H2020 957197, [Online]. Available: https://vedliot.eu/. [Accessed 29 08 2023].

[25] TERMINET , "TERMINET project website," H2020 957406, [Online]. Available: https://terminet-h2020.eu/. [Accessed 29 08 2023].

[26] IoT-NGIN, "D1.2 - IoT meta-architecture, components, and benchmarking," H2020 957246 - IoT-NGIN Deliverable Report, 2021.

[27] IoT-NGIN, "D1.3 - IoT meta-architecture alignment and continuous technology watch," H2020 957246 - IoT-NGIN Deliverable Report, 2022.

[28] P. Radoglou-Grammatikis, T. Lagkas and P. Sarigiannidis, "Next Generation IoT Reference Solution: The TERMINET Project," Adjacent Digital Politics Ltd., 2021. [Online]. Available: https://www.openaccessgovernment.org/terminet/126273/. [Accessed 29 08 2023].

[29] "ZDNET," 2020. [Online]. Available: https://www.zdnet.com/article/what-is-microsofts-metaos/. [Accessed 2023].

[30] ROS, "Robot Operating System," [Online]. Available: https://www.ros.org. [Accessed 2022].

[31] B. Keith , "Near, Far or Tiny: Defining and Managing Edge Computing in a Cloud Native World," 2021. [Online]. Available: https://vmblog.com/archive/2021/04/27/near-far-or-tiny-defining-and-managing-edge-computing-in-a-cloud-native-world.aspx.

[32] NEMO, "D1.1 - Definition and analysis of use cases and GDPR compliance," HORIZON - 101070118 - NEMO Deliverable Report, 2023.

[33] UC3M, "L2S-M," 2023. [Online]. Available: https://github.com/Networks-it-uc3m/L2S-M.

[34] Enarx, "Enarx," [Online]. Available: https://enarx.dev/. [Accessed 29 08 2023].

[35] The Linux Foundation, "Cloud native computing foundation," [Online]. Available: https://www.cncf.io/. [Accessed 29 08 2023].

[36] The Prometheus Authors, "Prometheus," [Online]. Available: https://prometheus.io/. [Accessed 29 08 2023].

[37] git, " Git --distributed-is-the-new-centralized," [Online]. Available: https://git-scm.com/. [Accessed 29 08 2023].

[38] Software Freedom Conservancy, "selenium," [Online]. Available: https://www.selenium.dev/. [Accessed 29 08 2023].

[39] Atlassian, "Jira," [Online]. Available: https://www.atlassian.com/software/jira. [Accessed 29 08 2023].

[40] Jenkins, "Jenkins," [Online]. Available: https://www.jenkins.io/. [Accessed 29 08 2023].

[41] RedHat, "Ansible," [Online]. Available: https://www.ansible.com/. [Accessed 29 08 2023].