INTERNATIONAL TELECOMMUNICATION UNION

TELECOMMUNICATION STANDARDIZATION SECTOR

STUDY PERIOD 2022-2024

SG13-TD412/WP1

STUDY GROUP 13

Original: English

Question(s): 20/13

Geneva, 26 July 2023

TD

Source: Editors

Title: Draft new Recommendation ITU-T Y.AN-Arch-fw: "Architecture framework for Autonomous Networks" (output of e- meeting, 27-30 June 2023)

Contact: Paul Harvey
University of Glasgow
United Kingdom
E-mail: paul.harvey@glasgow.ac.uk

Contact: Leon Wong
Rakuten Mobile
Japan
E-mail: leon.wong@rakuten.com

Contact: Vishnu Ram
Independent Expert
India
Email: vishnu.n@ieee.org

Contact: Xi Cao
China Mobile
P.R. China
Email: caoxi@chinamobile.com

Contact: Xiaojia Song
China Mobile
P.R. China
Email: songxiaojia@chinamobile.com

Contact: Liya Yuan
ZTE
P.R. China
Email: yuan.liya@zte.com.cn

Contact: Yamamoto Hideki
Oki Electric Industry Co., Ltd. (OKI)
Japan
E-mail: yamamoto436@oki.com

Abstract: This is the output of draft new Recommendation ITU-T Y.AN-Arch-fw "Architecture framework for Autonomous Networks" as agreed in Q20/13 meeting, virtual, 27-30 June 2023.

This document is based on SG13-TD390/WP1, output of 22-25 May 2023 Q20/13 meeting's discussion, and amendments according to the 27-30 June 2023 Q20/13 meeting's discussion and results on the following contributions:

No.	Source	Title	Qs	Discussion
Q20-June202	Orange	Comments to clause 6 of draft Rec. Y.AN-Arch-fw	Q20/13	Discussed and accepted with some modifications.

3-C1				
Q20-June2023-C9	University of Glasgow, Rakuten Mobile	Y.AN-Arch-fw: Proposal to update requirements	Q20/13	Discussed and accepted with some modifications.

Work plan for completion of this Recommendation (target for consent: 23 Oct – 3 Nov 2023 SG13 meeting)

- *Final editorial and technical checking, including addressing remaining editor's notes and highlights (if any) and considering the authors' guidelines*

Work schedule towards the completion

- *Q20/13 interim meeting (28-30 August 2023)*
- *SG13 meeting (23 Oct-3 Nov 2023)*

Draft new Recommendation ITU-T Y.AN-Arch-fw

Architecture framework for Autonomous Networks

Summary

This Recommendation provides requirements, architecture, components and related sequence diagrams which together comprise an architecture framework for autonomous networks.

The scope of this Recommendation includes:

- Requirements for the architecture;
- Description of the architecture and its components;
- Sequence diagrams explaining the interactions between the architecture components.

Keywords

Architecture framework, autonomous networks, components, dynamic adaptation, experimentation, exploratory evolution, requirements, sequence diagram

Draft new Recommendation ITU-T Y.AN-Arch-fw

Architecture Framework for Autonomous Networks

1. Scope

This Recommendation provides requirements, architecture components and related sequence diagrams which together comprise an architecture framework for autonomous networks.

The scope of this Recommendation includes:

- Requirements for the architecture
- Description of the architecture and its components
- Sequence diagrams explaining the interactions between the architecture components

2. References

[ITU-T Y.3172] ITU-T Recommendation Y.3172 (2019), *Architectural framework for machine learning in future networks including IMT-2020*

[ITU-T Y.3177] ITU-T Recommendation Y.3177 (2021), *Architectural framework for artificial intelligence-based network automation and fault management in future networks including IMT-2020*

[ITU-T Y.3320] ITU-T Recommendation Y.3320 (2014), *Global information infrastructure, internet protocol aspects and next-generation networks*

3. Definitions

3.1. Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1. Knowledge [b-ETSI GS ENI 005]: analysis of data and information, resulting in an understanding of what the data and information mean.

NOTE - Knowledge represents a set of patterns that are used to explain, as well as predict, what has happened, is happening, or is possible to happen in the future; it is based on acquisition of data, information, and skills through experience and education.

3.1.2. Machine learning (ML) [ITU-T Y.3172]: Processes that enable computational systems to understand data and gain knowledge from it without necessarily being explicitly programmed.

NOTE 1 – This definition is adapted from [b-ETSI GR ENI 004].

NOTE 2 – Supervised machine learning and unsupervised machine learning are two examples of machine learning types.

3.1.3. Machine learning model [ITU-T Y.3172]: Model created by applying machine learning techniques to data to learn from.

NOTE 1 – A machine learning model is used to generate predictions (e.g., regression, classification, clustering) on new (untrained) data.

NOTE 2 – A machine learning model may be encapsulated in a deployable fashion in the form of a software (e.g., virtual machine, container) or hardware component (e.g., IoT device).

NOTE 3 – Machine learning techniques include learning algorithms (e.g., learning the function that maps input data attributes to output data).

3.1.4. Closed loop [ITU-T Y.3115]: A type of control mechanism in which the outputs and behaviour of a system are monitored and analysed, and the behaviour of the system is adjusted so that improvements may be achieved towards definable goals.

NOTE 1 – Observe, Orient, Decide and Act (OODA) [b-OODA], MAPE-K [b-MAPE-K] are examples of closed loop mechanism.

NOTE 2 – Examples of definable goal types are optimization of network resources' utilization and automated service fulfilment and assurance. Goals may be defined using declarative mechanisms.

NOTE 3 – The system may consist of a set of managed entities, workflows and/or processes in a network.

3.2. Terms defined in this document

This Recommendation defines the following terms:

3.2.1. adaptation controller: A controller responsible for selecting candidate controllers ready for integration and for executing their integration in the underlay network.

3.2.2. AN sandbox: An environment in which controllers can be deployed, experimentally validated with the help of models of underlay networks, and their effects upon an underlay network evaluated, without affecting the underlay network.

NOTE - Domain specific models, if available, may be used in experimental validation of controllers. Examples of domain specific models are packet flow models for various types of applications such as video, chat, etc., and radio channel propagation models for various channel conditions.

3.2.3. autonomy engine: An environment where new controllers are autonomously generated and validated.

3.2.4. controller: A workflow, open loop or closed loop of a system under control in an autonomous network, composed of modules, integrated in a specific sequence, using interfaces exposed by the modules, to solve a specific problem or satisfy a given requirement.

NOTE 1 – Modules composing the controller may be workflows, open loops, or closed loops.

NOTE 2 - Modules can be developed independently of the system under control before being integrated into the system under control.

NOTE 3- Examples of system under control are managed entities, workflows and/or processes in an IMT-2020 network.

NOTE 4 – Exploratory evolution and real-time responsive online experimentation are examples of processes independent of the development of modules.

3.2.5. controller design: A low-level, non-executable representation of a controller containing modules, their configurations, and their parameter values which is used to instantiate a controller.

3.2.6. controller specification: A high-level, non-executable representation of a controller with the metadata corresponding to the mandatory functionality of the controller and a utility function to be achieved.

3.2.7. evolution controller: A controller responsible for the evolution of controllers by manipulating the module instance used within a controller, the structure or topology of connections between modules in a controller and/or the values chosen for the module(s) parameters.

3.2.8. experimentation: The process of executing the generated potential scenarios and trials upon the controllers, within the parameters of the scenarios and trials and then collecting the results.

NOTE - Example of experimentation is validating a traffic optimization controller against selected scenarios in a simulation tool, to find the controller performance with respect to a set of pre-defined service level agreements.

3.2.9. experimentation controller: A controller which generates potential scenarios of experimentations based on controller specifications and additional information as provided by the knowledge base, executes the scenarios in the AN Sandbox, collates and validates the results of the experimentation.

3.2.10. knowledge base: An environment which manages storage, querying, export, import, optimization and update of knowledge.

3.2.11. managed entity: A resource, service or controller which is managed

NOTE - An example of a controller as a managed entity, is a function tasked with traffic optimization in the user plane. In this case, the managed entity (controller) exposes interfaces or APIs to enable the collection of information, configuration and execution of the controller.

3.2.12. open loop: A type of control mechanism in which the outputs of the system under control are not used to adjust the behaviour of the system.

3.2.12 workflow: sequence of activities to describe and/or realize a given task executed by a system.

4. Abbreviations and acronyms

This document uses the following abbreviations and acronyms:

AI	Artificial Intelligence
AN	Autonomous Networks
API	Application Programming Interface
AR	Augmented Reality
CDN	Content Distribution Network
CI/CD	Continuous Integration and Continuous Delivery
CL	Closed loop
CN	Core Network

DNN	Deep Neural Network
DNS	Domain Name Service
E2E	End to End
FPGA	Field Programmable Gate Array
KB	Knowledge Base
KPI	Key Performance Indicator
MANO	Management and Orchestration
MEC	Multi access Edge Computing
ML	Machine Learning
NF	Network Function
NFV	Network Function virtualisation
ONAP	Open Networking Automation Platform
OSM	Open Source Management and Orchestration
PNF	Physical Network Function
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RIC	RAN Intelligence Controller
RCA	Root Cause Analysis
SDK	Software Development Kit
SLA	Service Level Agreement
TOSCA	Topology and Orchestration Specification for Cloud Applications
UE	User Equipment
VNF	Virtualized Network Function
VR	Virtual Reality
YAML	Yet Another Meta Language
ZSM	Zero Touch Service Management

5. Conventions

In this Recommendation:

The keywords "is required to" indicate a requirement which must be strictly followed and from which no deviation is permitted, if conformance to this Recommendation is to be claimed.

The keywords "is recommended" indicate a requirement which is recommended but which is not absolutely required. Thus, this requirement need not be present to claim conformance.

The keywords "can optionally" indicate an optional requirement which is permissible, without implying any sense of being recommended. This term is not intended to imply that the vendor's implementation must provide the option, and the feature can be optionally enabled by the network

operator/service provider. Rather, it means the vendor may optionally provide the feature and still claim conformance with this Recommendation.

6. Introduction

Autonomous networks are networks that possess the capabilities to monitor, operate, recover, heal, protect, optimize, and reconfigure themselves; these are commonly known as the self-* properties [b-Kephart 2003].

The application of various machine learning (ML) approaches to a single or a set of targeted use cases aims to automate the operation or management, reduce cost, optimise the resources used, or automatically detect or predict unusual situations or circumstances [b-ITU-T Y.Supp 55].

One common problem in the application of ML to these use cases is the problem of model drift. Model drift is the phenomena whereby either the goal of the ML model changes overtime (conceptual drift) or when the available data no longer enables the model to form the same relationships (data drift). This problem can be seen most obviously in financial markets, where market predictions must be frequently revisited to address the reality that the operating environment (the market) has changed compared to when the model was made. Several tools and frameworks have been proposed to help address these issues [b-capacity-allocation] , [b-evolution] , [b-bayesian-radio], [b-RL], [b-AUTOML], [b-AUTOML-ZERO].

The reality is that ML is one consideration that is required to achieve the autonomous operation of the network. Other considerations include emergence of new software and hardware technologies; introduction of new services to the network and new ways of using the networks [b-NGMN-5G]; definitions change – what is good today is not necessarily good tomorrow.

As the operational environment and context of our networks change, so too must the processes of control that we use to operate them.

Closed-loop (CL) software control has become an increasingly popular way to enable the automatic operation of the network. There are a range of different CL approaches in different domains: efficient and simple, strategic, tactical, centralised, distributed, intelligent, adaptative, hierarchical [b-Rossi 2020], [b-ITU-JFET]. Irrespective of the approach or purpose, the logical concept of a CL [b-Kephart 2003], [b-OODA] is a self-contained entity with the ability to operate or monitor one or more managed entities. In this context, a CL suffers the same limitation in achieving autonomous network operation since being bound by the purpose for which it was designed, even in the cases when its design includes the support of some dynamic adaptive capabilities.

The conclusion of the above is that no matter the domain of operation, technology, algorithm, intelligence, or data set used, an autonomous network will require the ability to adapt beyond pre-defined operational bounds not only in logic deployed to operate and manage of the network but also in the process that it uses to generate such deployable logic, so called “design-time procedures” [ITU Y.3177].

The key purpose and goal of the architecture described in this Recommendation is to support the continuous evolutionary-driven creation, validation, and application of a set of controllers to a network and its services such that the network and its services may become autonomous. A controller is a workflow, closed loop or open loop of a system under control. It is composed of modules, integrated in a specific sequence, and using interfaces exposed by the modules, to solve a specific problem or satisfy a given requirement. Modules composing the controller may be workflows, open loops, or closed loops and can be developed independently of the system under control before being integrated into the system under control.
The continuous evolutionary-driven creation, validation and application of controllers is used in the use cases to realize autonomous networks [b-ITU-T Y.Supp 71] and the key concepts (see below) required to enable them.

In this way, the traditional autonomic self-* principles [b-Kephart 2003] are attributed to the controllers which are applied to the network and its services. The responsibility for adaptation of controllers themselves over time is the responsibility of the architecture specified in this Recommendation. The separation of the adaptation of a controller from the application of a controller to the network and its services enables the complimentary efforts in standards and research for CL, ML, as well as the general area of network management in the pursuit of autonomous networks and directly addresses need for automatic design-time procedures.

The main concepts behind autonomous networks which are elaborated in this Recommendation are exploratory evolution, real-time responsive online experimentation and dynamic adaptation.

The concept of exploratory evolution introduces the mechanisms and processes of exploration and evolution to adapt controllers in response to changes in the underlay network. These processes generate new controllers or update (evolve) existing controllers to respond to such changes and solve the situation or task at hand more appropriately.

The continuous process, based on monitoring and optimization of deployed controllers in the underlay network, is called real-time responsive online experimentation.

NOTE - Real-time responsive online experimentation is also called “experimentation” in this Recommendation.

Dynamic adaptation is the final concept in equipping the network with autonomy and the ability to handle new and hitherto unseen changes in network scenarios.

With consideration of the above concepts, an autonomous network is a network which can generate, adapt, and integrate controllers at run-time using network-specific information and can realize exploratory evolution, real-time responsive online experimentation and dynamic adaptation.

In addition, the requirements for the architecture below also consider the following concepts: knowledge in autonomous networks and orchestration in autonomous networks.

The analysis of data and information from the network, resulting in an understanding of what the data and information mean, is referred to as knowledge. Knowledge is used in autonomous networks for supporting the continuous exploratory evolution, realtime online experimental validation, and dynamic adaptation.

Orchestration involves managing workflows and processes in the AN and steps in the lifecycle of controllers. This also requires coordination with various other functions in the AN as well as outside the AN.

7. Requirements for the architecture

This clause describes the requirements for the AN architecture.

Editor’s note: for possible consideration, ordering of reqts in each clause 7.X by "required” followed by “recommended” and then “optional”

7.1. Requirements for Exploratory Evolution

The following are requirements with respect to exploratory evolution in autonomous networks.

Requirement	Description
AN-arch-evo-req-001	The AN architecture is recommended to enable the integration and the plugin of algorithms into controllers. NOTE – Algorithms may be third party provided algorithms, machine learning or artificial intelligence models
AN-arch-evo-req-002	The AN architecture is required to have the ability to generate and update controller descriptions.
AN-arch-evo-req-003	The AN architecture is required to have the ability to generate potential scenarios of exploratory evolution, taking the controller descriptions as input. NOTE - Specific mechanisms or algorithms used for evolution are out of scope of this Recommendation.
AN-arch-evo-req-004	The AN architecture is required to have the ability to execute the potential scenarios of exploratory evolution, taking the controller descriptions as input and to collate the output in the form of evolvable controller descriptions. NOTE - Several rounds of evolutions may be applied on the same set of controller descriptions.
AN-arch-evo-req-005	The AN architecture is required to support the generation of potential configurations for the integration of controllers to specific underlay networks. NOTE 1 – Configurations may include reference points, API formats, data models, etc. This generation of configurations may take the controller descriptions as input along with the description or metadata related to the underlay networks. NOTE 2 – While the specific configurations for the integration of controllers for specific use cases is out of scope of this Recommendation, the definition of acceptable formats for representing such configurations is for further study. NOTE 3 - Examples of underlay networks are edge networks, core networks, management plane, CI/CD pipelines.
AN-arch-evo-req-006	The AN architecture is required to enable the management of points of metadata exchange in the AN workflow, with a peer entity. NOTE 1 – Examples of metadata regarding the AN workflow include current capabilities, status and context of the AN components, including knowledge base, controllers, orchestrators, simulators, etc. Managing may include identifying actors and points in the AN workflow, capturing metadata regarding the workflow which can then be exchanged. NOTE 2 - Examples of points of metadata exchange in the AN workflow are the different stages of experimentation and dynamic adaptation.

	NOTE 3 - The format used for the metadata exchange is out of scope of this Recommendation. NOTE 4 – Examples of AN workflows include the generation of potential experimentation scenarios and the generation of potential evolution scenarios. NOTE 5 - Peers may include humans and machines.
AN-arch-evo-req-007	The AN architecture is required to have the ability to integrate the impacts of metadata exchange with peer entities involved in AN workflows. NOTE – Examples of impacts of the metadata exchange are updates of knowledge base and selection of API versions to use for adaptation.
AN-arch-evo-req-008	The AN architecture is required to support the optimization of controllers. NOTE – Examples of optimization of controllers are optimization of adaptation mechanisms like data collection, data quality and frequency. Other examples are optimization of closed loop implementations like Root Cause Analysis (RCA) mechanisms and recommendations on better algorithms for achieving the same objective. Other examples are formation or evolution of new controllers to address new or unforeseen problems in the underlay network.
AN-arch-evo-req-009	The AN architecture is required to support the ability of discovering the characteristics of controllers which are relevant for enabling evolution. NOTE - Examples of characteristics of controllers which are relevant for evolution are: capabilities exposed by the controllers, and requirements to be satisfied for the controllers.
AN-arch-evo-req-010	The AN architecture is required to support the capability of recommending modules which can satisfy the characteristics of controllers.
AN-arch-evo-req-011	The AN architecture is required to enable the integration of controllers from different domains to achieve complex use cases. NOTE - For example, AN may integrate controllers in different domains of the network, like RAN and core network domains.
AN-arch-evo-req-012	The AN architecture is required to enable the utilization of declarative specifications of use cases while deciding the design, deployment and management of controllers.
AN-arch-evo-req-013	The AN architecture is required to allow for the utilization of declarative specifications of use cases to capture both use cases' requirements from applications and deployment requirements from underlay networks.
AN-arch-evo-req-014	The AN architecture is required to support the capability of choosing the compatible set of interfaces to integrate with underlay network services. NOTE – Example of interfaces that may be used include interfaces to monitor the services and controllers in the underlay networks.
AN-arch-evo-req-015	The AN architecture is required to support the creation and/or recommendation of candidate designs for potential network services and

	interfaces in the underlay networks, which may possibly satisfy new use cases. NOTE – Design creation may also be triggered in response to an observed fault in the underlay networks.
AN-arch-evo-req-016	The AN architecture is required to consider the use case specific requirements (including operator preferences) while deciding the operator preferences for design, deployment, management of controllers, including connectivity options between various domains. NOTE - For example, in rural areas, end-users may have usage patterns with characteristics depending on applications (e.g. low mobility, high bandwidth). The choice of last mile connectivity options may be influenced by such preferences.
AN-arch-evo-req-017	The AN architecture is required to support capabilities enabling the evolution of inter-domain connectivity among controllers deployed in various domains of the underlay network.
AN-arch-evo-req-018	The AN architecture is required to support the adaptation to the evolution of applications at run time. NOTE – An example of adapting to the evolution of applications at run time is onboarding new applications or changes in existing applications deployed by the service providers in the underlay network.
AN-arch-evo-req-019	The AN architecture is required to support the adaptation to changes in the external systems that the AN interfaces with. NOTE - Examples of external systems are various management and orchestration systems, policies and corresponding management systems, workflow management systems, user management systems. External systems deployed by the operator may be provided by multiple vendors.
AN-arch-evo-req-020	The AN architecture is required to support the ability of recommending changes in capabilities of monitoring, configuring and analysis of parameters from underlay networks.
AN-arch-evo-req-021	The AN architecture can optionally support the capability to recommend new capabilities and requirements for the NFs deployed in underlay networks.
AN-arch-evo-req-022	The AN architecture is required to support learning of metrics' derivations from collected parameters and measurements, where such learnings may change. NOTE – Derivation of metrics may use AI/ML techniques. Derivation mechanisms may change over a period of time or events. In such cases, mechanisms such as re-training may be applied to update the derivation models.
AN-arch-evo-req-023	The AN architecture is required to support the derivation of requirements for service life cycle management in underlay networks based on analysis of requirements from different domains in the network. NOTE 1 - For example, closed loops for optimization in RAN may be configured based on analysis of requirements from the CN.

	NOTE 2 - Service life cycle management includes resource allocation, scaling and optimization.
AN-arch-evo-req-024	The AN architecture is required to support optimization of intents based on monitoring of the performance of derived controllers deployed in various domains of the underlay network and their life cycles. NOTE - For example, derivation of intents in the CN may be optimized based on the feedback obtained from monitoring the (derived) closed loops deployed in the RAN.
AN-arch-evo-req-025	The AN architecture is required to support application development capabilities to automate the design and instantiation of underlay network services. NOTE - For example, platforms such as Kubernetes [b-kubernetes], MEC or O-RAN may expose SDKs or APIs for automation of design and deployment of network services.
AN-arch-evo-req-026	The AN architecture is required to support application development capabilities to automate the design and/or instantiation of controllers in various levels of the underlay network. NOTE - For example, platforms such as ONF [b-ONF], O-RAN and ONAP allow design and deployment of xApps and AI/ML models respectively. In some cases, 3rd party repositories may be accessed to select and deploy xApps and/or AI/ML models.
AN-arch-evo-req-027	The AN architecture is required to enable designing, developing and deploying applications by the AN at various levels of the underlay network. NOTE 1- For example, in coordination with the edge network orchestrating function, AN may provide a design for vertical applications to be deployed at a specific edge location. NOTE 2 - The design and development of applications may be achieved in coordination with CI/CD pipelines.
AN-arch-evo-req-028	The AN architecture is required to support capabilities for monitoring of the feedback from controllers deployed at various domains of the underlay network in order to optimize the design, development and deployment of controllers in general. NOTE - For example, in coordination with the edge network orchestrating function, AN may optimize the existing design for vertical applications to be deployed at a specific edge location.
AN-arch-evo-req-029	The AN architecture is recommended to enable AN discovery of service level trade-offs. NOTE – An example of service level trade-offs isgreater accuracy of inference versus larger resource for training of AI/ML models.
AN-arch-evo-req-030	The AN architecture is required to support the ability of triggering re-design of network services based on monitoring of network services in underlay networks.
AN-arch-evo-req-031	The AN architecture is required to provide means for the AN to trigger an evolution of network services based on the monitoring of their lifecycle in underlay networks.

AN-arch-evo-req-032	The AN architecture is required to AN support usage of various types of controllers for problem discovery in various domains of the underlay network. NOTE - For example, data collection agents may be designed separately from the analysis. Data collection agents may be deployed in the edge network whereas analysis may be performed at the core cloud.
AN-arch-evo-req-033	The AN architecture is recommended to support adaptive design of controllers using hardware adaptation techniques. NOTE 1 – Examples of hardware adaptation techniques include detection of hardware capabilities and adaptive design. NOTE 2 - Examples of hardware adaptation techniques include optimization of AI/ML models to FPGA architectures. Adaptive design may involve considerations of design trade-offs such as energy efficiency, accuracy, etc.
AN-arch-evo-req-034	The AN architecture is recommended to provide capabilities for the AN to support feedback and optimization of hardware adaption process for controllers. NOTE - For example, the hardware adaptation process for controllers may involve (1) translation of high-level description to an intermediate representation amenable to optimization, (2) optimization considering the design trade-offs, and (3) hardware implementation and integration. The translation and optimization steps above may themselves be tuned based on monitoring and feedback from the controllers integrated in the hardware.
AN-arch-evo-req-035	The AN architecture is required to allow the AN providing evolution of controllers as a service to underlay networks. NOTE – Underlay networks may have different types of controllers deployed including from 3rd parties. AN discovers the characteristics of different types of controllers and provides evolution as a service which results in evolved controller candidates for deployment in the underlay networks.
AN-arch-evo-req-036	The AN architecture is required to support capabilities for the AN to discover and utilize different evolution services for controllers. NOTE – Underlay networks may need evolution of controllers deployed including from 3rd parties. AN discovers the characteristics of different types of evolution services for controllers and utilizes them. Utilizing the services of evolution as a service may include providing intents as inputs, providing evolution algorithms as inputs, providing module and/or controller repositories as input and accepting evolved controller candidates as outputs.
AN-arch-evo-req-037	The AN architecture can optionally support the ability for the AN to monitor, optimize and create new inter-controller coordination strategies, along with the design of new controllers. NOTE - Optimization may use AI/ML mechanisms.

AN-arch-evo-req-038	The AN architecture is required to support the ability for the AN to customize controllers which can be deployed in various types of underlay networks, based on the characteristics of these underlay networks.
AN-arch-evo-req-039	The AN architecture is required to support decomposition of controller designs into parts, which can be mapped to various parts of the underlay network based on the capabilities and requirements. NOTE 1 - Example of decomposition is splitting of user plane programs into modules which can be hosted in various user plane functions in the underlay network. NOTE 2 - Example of considerations while decomposition of controller designs is resource requirements of the controllers and capabilities of the underlay network.
AN-arch-evo-req-040	The AN architecture is recommended to support monitoring and optimization of decomposition, design, placement or deployment of controllers in various parts of the underlay network.
AN-arch-evo-req-041	The AN architecture is required to support composition of controllers deployed in various parts of the underlay network to form complex controllers.

7.2. Requirements for Online Experimentation

The following are requirements with respect to online experimentation in autonomous networks.

Requirement	Description
AN-arch-exp-req-001	The AN architecture is required to support the validation and processing of controllers' descriptions, so that exploratory evolution can be applied on these controller descriptions. NOTE 1 – Exploratory evolution may result, among others, in interconnecting of descriptions together to form complex controller descriptions or in a list of controllers. NOTE 2 – Exploratory evolution may be a triggered, or periodic process
AN-arch-exp-req-002	The AN architecture is required to support the ability to generate the potential scenarios of experimentation, taking the controller descriptions as input. NOTE 1- Specific configurations and "limits" of experiments may be specified in the "metadata" and "constraints" related to the controller descriptions. NOTE 2- Specific mechanisms for arriving at the scenarios for experimentation may use AI/ML analytics or other forms of analytics and are out of scope of this Recommendation.

AN-arch-exp-req-003	The AN architecture is required to have the ability of executing experimentations, collating and validating the results of the experimentations, considering the metadata and constraints and corresponding controller descriptions. NOTE 1- Experimentations may have several phases, for example, simulation driven, testbed driven or canary test driven. The phases of an experimentation may be configurable and automated, for example, as per a workflow. NOTE 2- The specific success and failure criteria for the experiments are out of scope of this Recommendation. Acceptable formats for representing metadata and constraints related to potential success and failure as related to a use case are for further study.
AN-arch-exp-req-004	The AN architecture is recommended to enable the integration of various forms of testing components including simulators and data generators, including those provided by third party providers.
AN-arch-exp-req-005	The AN architecture is required to support the ability of, verifying, before the actual integration of controllers into the underlay networks, that the proposed evolution of the controllers is compatible with the underlay networks.
AN-arch-exp-req-006	The AN architecture is required to support the automation and abstraction of the experimentation of underlay network services,
AN-arch-exp-req-007	The AN architecture is required to provide capabilities for the creation of experimentation strategies regarding the deployment of experimentation scenarios, testing and validation of controllers in sandbox environment.
AN-arch-exp-req-008	The AN architecture is required to support capabilities allowing the AN to deploy, test and validate controllers in sandbox environment.
AN-arch-exp-req-009	The AN architecture is required to provide capabilities allowing the AN to analyse the results from experiments in sandbox environment and to use those results to update the knowledge base, optimize deployed controllers in underlay networks as well as optimize the experimentation strategies in sandbox. NOTE – An example of controller optimization is the selection of new controllers or modules. An example of experimentation strategies' optimization is the selection of new test scenarios.
AN-arch-exp-req-010	The AN architecture is required to support the ability for the AN to experiment the generation of changes to user specific models which may help ease of experience for users in hitherto unforeseen circumstances. NOTE - AN experimentation may be done in a sandbox using simulators.
AN-arch-exp-req-011	The AN architecture is required to support capabilities for the AN to provide experimentation of controllers as a service to underlay networks. NOTE – Underlay networks may have different types of controllers deployed including from 3rd parties. AN discovers the characteristics of different types of controllers, providing different experimentation

	services and results as output for various types of experimentation scenarios.
AN-arch-exp-req-012	The AN architecture is required to support means for the AN to import and export configurations for simulators. NOTE - Examples of configurations for simulators are simulated network topologies, simulated number of devices, simulated traffic settings and closed loop interfaces.
AN-arch-exp-req-013	The AN architecture is required to enable the AN to asynchronously trigger experimentations. NOTE - Examples of asynchronous triggering are evolution of new set of controllers which need to be validated, updated network configurations by operator, provisioning or update of network functions in the underlay network.
AN-arch-exp-req-014	The AN architecture is required to enable the validation of experimentation results by the AN. NOTE - Examples of validation include sanity checks, functional and non-functional tests.
AN-arch-exp-req-015	The AN architecture is required to support the ability of the AN to provide feedback on the design of controllers based on the results of experimentation. NOTE - Examples of feedback are experimentation logs, test scenarios along with detailed results.
AN-arch-exp-req-016	The AN architecture is required to support the ability of the AN to design experimentation scenarios based on use case descriptions. NOTE - Example of design representation is TOSCA definitions, derived from use case representations.
AN-arch-exp-req-017	The AN architecture is required to provide the ability for the AN to discover and utilize other experimentation services for controllers. NOTE – Underlay networks may have experimentation of controllers deployed including from 3rd parties. AN discovers the characteristics of different types of experimentation services for controllers and utilizes them. Utilizing the services of experimentation as a service may include providing intents as inputs, providing experimentation algorithms as inputs, providing module and/or controller repositories as input and accepting experimentation results as outputs.
AN-arch-exp-req-018	The AN architecture is required to support the AN ability to select the experimentation scenarios, data generation and simulators, based on the selected reference points in the underlay network where the controllers could be deployed.
AN-arch-exp-req-019	The AN architecture is recommended to enable the AN creating a virtual model of real environment for experimentation. NOTE – Creating a virtual model may use visualization and perception mechanisms like AR/VR, simulation engines and data generation mechanisms.

AN-arch-exp-req-020	The AN architecture is recommended to support experimentation and derivation of inter-controller coordination strategies. NOTE - Examples of inter-controller interactions are resolution of conflicting goals for various use cases, like power consumption vs. coverage optimization. Example of a relevant strategy is game theory based cooperative and non-cooperative mechanisms.
AN-arch-exp-req-021	The AN architecture can optionally support experimentation in domain specific sandbox and optimization of domain specific intents. NOTE – ML sandbox [ITU-T Y.3172] is an example of domain specific sandbox.

7.3. Requirements for Dynamic Adaptation

The following are requirements with respect to dynamic adaptation in autonomous networks.

Requirement	Description
AN-arch-adp-req-001	The AN architecture is required to support the ability to select candidate controllers from a set of controllers ready for integration and to execute their integration to specific underlay networks, taking as input the generated configurations for integration. NOTE 1– The specific criteria for selecting controllers for integration are out of scope of this Recommendation. Formats for representing such criteria are for further study. NOTE 2 – The specific mechanisms used for the integration of controllers to underlay networks are out of scope of this Recommendation. Examples of such mechanisms are service based architectures [b-ETSI TS 129 500] and continuous integration mechanisms [b-ITU-T Y.3525].
AN-arch-adp-req-002	The AN architecture is required to allow AN decisions about new opportunities for the deployment of controllers in underlay networks.
AN-arch-adp-req-003	The AN architecture is required to allow AN decisions about the configuration of controllers which are to be deployed in underlay networks.
AN-arch-adp-req-004	The AN architecture is recommended to enable the monitoring of controllers which are already deployed in underlay networks.
AN-arch-adp-req-005	The AN architecture is required to enable the reporting and monitoring of AN components and procedures by humans and/or other automation mechanisms.
AN-arch-adp-req-006	The AN architecture is required to support the discovery of deployed controllers in the underlay networks, including those deployed by third party providers.
AN-arch-adp-req-007	The AN architecture is required to support the discovery and consumption of services provided by service management frameworks.

	NOTE - Examples of service management frameworks are ONAP [b-ONAP] and OSM [b-OSM].
AN-arch-adp-req-008	The AN architecture can optionally provide the ability to influence the services provided by service management frameworks. NOTE - Examples of AN influence upon services provided by service management frameworks are passing policies and intents to service management frameworks. Service management frameworks may use them to design, deploy new and/or modified services.
AN-arch-adp-req-009	The AN architecture is required to support the discovery of the interfaces with underlay networks used for integration. NOTE - Interfaces with underlay networks may use specific APIs, e.g., APIs for data collection, for configuration of network functions, etc. Discovery of interfaces may involve API metadata including parameters, versions, range of parameters.

AN-arch-adp-req-010	The AN architecture is required to enable, on a per use case basis, the discovery of the underlay networks' specific parameters candidate for optimization, data points for collection in the underlay network and the relevant KPIs for tracking. NOTE - For example, in edge deployments, underlay networks may use multi access edge computing (MEC) APIs [b-ETSI GS MEC 012].
AN-arch-adp-req-011	The AN architecture is required to support the ability of customizing the integration of controllers in underlay networks, considering the integration options exposed by the underlay networks. NOTE – Examples of integration options are interfaces, parameters and configurations exposed by the underlay networks. Examples of underlay networks are networks for industry vertical applications.
AN-arch-adp-req-012	The AN architecture is required to support the automation and abstraction of the evolution of underlay network services. NOTE 1 – Examples of evolution of underlay network services include updates to support new features, migration to new service platforms and technologies. NOTE 2 – Service provider may monitor the evolution but do not manually execute the evolution of underlay network services.
AN-arch-adp-req-013	The AN architecture is required to support the discovery of service management frameworks used by underlay networks. NOTE - This may help in managing and automating the lifecycle of underlay network services by the AN.
AN-arch-adp-req-014	The AN architecture is required to support the end-to-end integration of controllers, taking into account the evolution of connectivity options in the underlay networks. NOTE - For example, access network may be using various different types of connectivity technologies which may evolve over a period of time.

AN-arch-adp-req-015	The AN architecture is required to support the monitoring of the dynamic changes in capabilities of monitoring, configuration and analysis of parameters from underlay networks. NOTE - Given the independent evolution of controllers, of the underlay networks and of the applications deployed, mechanisms such as those for discovery, publishing and subscription may be used to provide flexibility in monitoring parameters.
AN-arch-adp-req-016	The AN architecture is required to support the utilization of the dynamic changes in capabilities of monitoring, configuring and analysis of parameters from underlay networks.
AN-arch-adp-req-017	The AN architecture is required to support capabilities enabling the deployment of controllers which utilize both simulated and real networks as underlay networks.
AN-arch-adp-req-018	The AN architecture can optionally support capabilities enabling the correlation of declarative specifications of network services with those of controllers and the use of that correlation to integrate controllers in the same or different domains of the network. NOTE - Examples of correlation of declarative specifications of controllers with those of network services include mapping of interfaces, capabilities and requirements. Other examples are identifying opportunities for deriving specifications, e.g. using substitution mechanisms in TOSCA [b-OASIS TOSCA-v1.3].
AN-arch-adp-req-019	The AN architecture is required to support capabilities for the discovery of topology and architecture connectivity/split options and capabilities in the underlay networks and to consider such options while integrating controllers in underlay networks. NOTE - For example, 3GPP networks may have various architecture split options [b-3GPP TS 38.801].
AN-arch-adp-req-020	The AN architecture is required to have capabilities to integrate intelligent controllers at various levels of the underlay network. NOTE - Examples of intelligent controllers are controllers integrating AI/ML models.
AN-arch-adp-req-021	The AN architecture is required to enable the integration of controllers to network management and application management at various levels of the underlay network. NOTE - Examples of functionalities of such controllers are placement of functions and choice of architecture splits.
AN-arch-adp-req-022	The AN architecture is required to enable AN run-time discovery of new use cases for optimization in underlay networks.
AN-arch-adp-req-023	The AN architecture is required to AN support usage of various types of controllers for problem isolation in various domains of the underlay network. NOTE - Collaborative communication between controllers may be used to isolate the problem.

AN-arch-adp-req-024	The AN architecture is required to AN support design and/or selection of controllers based on the problem isolated in the underlay network. NOTE - For example, 3rd party controllers from repositories may be selected to address the problem.
AN-arch-adp-req-025	The AN architecture is required to AN support deployment of new controllers with new capabilities to address the problems detected in the underlay network.
AN-arch-adp-req-026	The AN architecture is required to support the AN ability to select reference points in the underlay network where controllers could be deployed. NOTE - Examples of considerations for AN while selecting the reference points are trade-offs in terms of benefits (e.g. spectral efficiency, latency, etc) as against the computational overheads of training of models or communication overheads.
AN-arch-adp-req-027	The AN architecture is required to support the AN ability to select the controllers for integration in the underlay network based on the results of the experimentations which are in turn based on the reference points in the underlay network where the controllers could be deployed.
AN-arch-adp-req-028	The AN architecture is required to support the AN ability to integrate the controllers, selected based upon the experimentation results, at the reference points in the underlay network where the controllers could be deployed. NOTE - This may involve control and data flow modifications according to the integration methods for controllers in the underlay network.
AN-arch-adp-req-029	The AN architecture can optionally enable optimal placement of controllers by the AN based on the application requirements and capabilities at various domains of the underlay network.
AN-arch-adp-req-030	The AN architecture is required to enable AN continuous monitoring of capabilities at various levels of the underlay networks and trigger update of controllers based on any changes to the capabilities. NOTE - Examples of changes to capabilities of underlay networks are addition, deletion of network functions, updates to software versions.
AN-arch-adp-req-031	The AN architecture is required to support the placement or deployment of controllers in various parts of the underlay network. NOTE - Deployment may consider resource availability and other capabilities of the underlay network along with the requirements of the controller.
AN-arch-adp-req-032	The AN architecture is required to support a tightly coupled and loosely coupled integration with underlay networks. NOTE 1 - The capability and preference of the underlay network to perform a tightly coupled or loosely coupled AN integration may be discovered at the time of integration with the underlay network.

	NOTE 2 - In tightly coupled integration, the underlay network may utilize the components provided by AN provider. Whereas in loosely coupled integration, the underlay network may deploy and use the underlay network providers' components for AN.
AN-arch-adp-req-033	The AN architecture is required to enable controllers to use domain specific mechanisms to manage resource sharing and service integrations, across domains in the underlay networks. NOTE - Examples of domain specific mechanisms include dynamic service agreements and distributed ledger technologies.
AN-arch-adp-req-034	The AN architecture is required to support the discovery by the AN of the need for new controllers based on monitoring of the underlay network. NOTE – The need for new controllers may be discovered based on monitoring of various parameters or KPIs.
AN-arch-adp-req-035	The AN architecture is required to support, based on the discovery of the need for new controllers, the selection of new controllers from controller repositories, as candidates to be deployed in the underlay network. NOTE - Controller repositories may be external or internal to the AN provider.
AN-arch-adp-req-036	The AN architecture is required to support the evaluation of new candidate controllers to be deployed in the underlay network from the selected ones from repositories' candidates. NOTE – Evaluation may be done based on use case specific metrics.
AN-arch-adp-req-037	The AN architecture is required to support the run-time deployment of new controllers in the underlay network based on the candidates evaluated from repositories.
AN-arch-adp-req-038	The AN architecture can optionally support peer to peer interaction between controllers without the intervention of a centralized coordinating function. NOTE – Example of peer interaction is exchange of metadata for resource allocation and load balancing.

7.4. Requirements for Knowledge

The following are requirements with respect to knowledge in autonomous networks.

Requirement	Description
AN-arch-knw-req-001	The AN architecture is required to provide capabilities for the management of knowledge related to autonomous networks. NOTE – Managing knowledge includes storing, querying, exporting, importing, and optimizing knowledge.

AN-arch-knw-req-002	The AN architecture is required to enable the update of knowledge based on the various processes involved in the AN.
AN-arch-knw-req-003	The AN architecture is required to support the utilization of components like stored controllers and knowledge to deploy and manage controllers in underlay networks. NOTE – Examples of components include stored controllers and knowledge.
AN-arch-knw-req-004	The AN architecture is required to enable the storage and management of supporting artifacts for the lifecycle management of controllers. NOTE 1 – Examples of supporting artifacts are knowledge, AI/ML or other types of models, workflow representations, policies which need to be applied while managing the lifecycle of controllers, etc. NOTE 2 – Examples of management of supporting artifacts are storage of knowledge in knowledge base, creation, modification, deletion, and storage of AI/ML models in ML model repository [ITU-T Y.3176] and of policies, query and discovery of various artifacts.
AN-arch-knw-req-005	The AN architecture is required to support the production of human and machine-readable reports of periodic or aperiodic nature.
AN-arch-knw-req-006	The AN architecture is recommended to enable the analysis and correlation of domain specific, unstructured data in natural languages from the underlay networks. NOTE 1 – Examples of domain specific unstructured data in natural languages are logs from NFs. NOTE 2 – Advances in analysis of natural language text may be exploited from third party models and repositories.
AN-arch-knw-req-007	The AN architecture is recommended to support capabilities for deriving knowledge from the analysis and correlation of domain specific unstructured data in natural languages from the underlay networks.
AN-arch-knw-req-008	The AN architecture is required to support capabilities for importing and exporting of controller specifications at various stages of their management. NOTE - Examples of various stages of management of controller specifications are before and after exploratory evolution.
AN-arch-knw-req-009	The AN architecture is required to support the integration of third party provided derivation mechanisms for metrics from collected parameters and measurements. NOTE – An example of derived metrics is QoE while an example of collected measurements is QoS parameters.
AN-arch-knw-req-010	The AN architecture is required to support the capturing of both service KPI requirements as well as deployment preferences and considerations in the intent.
AN-arch-knw-req-011	The AN architecture is required to provide capabilities for the AN to capture domain specificities.

	NOTE 1 - Examples of domain specificities are latency criteria, location information, data privacy requirements, etc. NOTE 2 - Examples of capturing domain specificities are TOSCA service definitions. The design of controllers may also be represented using TOSCA declarative definitions.
AN-arch-knw-req-012	The AN architecture is required to provide the ability for the AN to capture service specificities. NOTE - Examples of service specificities include service level requirements for QoS.
AN-arch-knw-req-013	The AN architecture is required to support the AN capability of using inputs from external environment and user specific models to design as well as apply controller outputs to underlay networks. NOTE - Example of inputs from external environments is mobility prediction models for users with assistive needs or groups of users.
AN-arch-knw-req-014	The AN architecture is required to support the AN capability of using user preferences while designing as well as applying controller outputs to underlay networks. NOTE - Standard representations of user profiles or preferences or user models with assistive needs are examples of user preferences.
AN-arch-knw-req-015	The AN architecture is recommended to enable the transfer by the AN of user specific models between different domains in the underlay network. NOTE - This may help in update of models in other domains, update of simulators.
AN-arch-knw-req-016	The AN architecture is required to provide means for the AN to integrate data collection mechanisms. NOTE - Data collection mechanisms may include AR/VR glasses or other types of sensors. Data collection mechanisms may be provided by 3rd party providers.
AN-arch-knw-req-017	The AN architecture is recommended to provide means for the AN to use virtual models along with the real-world inputs to analyse and optimize the underlay network and to provide feedback to operators. NOTE - Analysis may use AI/ML models. Optimization may involve configurations in the underlay network. Feedback to operators may be generated in AR/VR formats.
AN-arch-knw-req-018	The AN architecture can optionally support the ability to integrate in the AN third party modules or applications for collection, analysis, or feedback. NOTE - For example a software development kit (SDK) may be exposed to third party developers who may develop new applications to analyse the AR-collected data.
AN-arch-knw-req-019	The AN architecture is required to enable the AN discovery of capabilities available at the various domains of underlay networks.

	NOTE - Capabilities of the underlay networks may differ based on their resource availability, e.g., compute, memory, already deployed controllers.
AN-arch-knw-req-020	The AN architecture can optionally enable the AN creation of use case descriptions which can then be decomposed to controllers which can be deployed at various domains of the underlay network, based on the capabilities at those levels of the underlay network. NOTE – use case descriptions may be in the form of intents.
AN-arch-knw-req-021	The AN architecture is required to support means for the AN to use interoperable format for storing controllers. NOTE - Various components in AN may read and write from the stored controllers, e.g., an evolution controller may read existing controllers (even from third parties) and utilize them for composing new controllers, which are in turn written in the storage.
AN-arch-knw-req-022	The AN architecture can optionally support the ability for the AN to discover the characteristics of underlay networks at runtime.
AN-arch-knw-req-023	The AN architecture is required to support description of use cases in a declarative format.
AN-arch-knw-req-024	The AN architecture is required to support derivation of domain specific intents from the use case description. NOTE – ML intent [ITU-T Y.3172] is an example of domain specific intent.

7.5. Requirements for Autonomous Network Orchestration

The following are requirements with respect to orchestration in autonomous networks.

Requirement	Description
AN-arch-ano-req-001	The AN architecture is required to support the ability of parsing, validating and translating abstracted use case descriptions, with high level objectives of a controller into controller descriptions. NOTE 1 – The abstracted use case descriptions may be hand-crafted as unstructured text or derived from controller specifications. NOTE 2 – Controller descriptions may be provided using structured languages formats (e.g. TOSCA [b-OASIS TOSCA-v1.3]) and may be structured in a way which facilitates downstream exploration, experimentation, and adaptation. NOTE 3– Controller descriptions may use and enable properties derived from various domains in the network, e.g., properties allowing to describe use cases of physical layer, network layer and application layer. NOTE 4– Examples of use cases are:

	(a) Root cause analysis and diagnosis of network elements based on real time analysis of data - see FG-AN-usecase-006 in [ITU-T Y.Supp 71] (b) Intelligent energy saving solution based on automatic data acquisition, AI-based energy consumption modelling and inference, facilities parameters control policies decision, facilities adjustment actions implementation, energy saving result evaluation and control policies continuous optimization – see FG-AN-usecase-007 in [ITU-T Y.Supp 71] (c) Optimal adjustment of antenna parameters with AI enabled multi-dimensional analysis and prediction - seeFG-AN-usecase-008 in [ITU-T Y.Supp 71] (d) Management of 3rd party vertical applications and related services in the network – see FG-AN-usecase-010 in [ITU-T Y.Supp 71]
AN-arch-ano-req-002	The AN architecture is required to have the ability to manage the lifecycle of controllers. NOTE 1 - Examples of management tasks of a controller's lifecycle include creating a configuration for the controller (based on the capabilities of the underlay network), creating an instance of the controller in the underlay network, monitoring the execution of the controller in the underlay network and subsequently optimize the controllers or related parameters. NOTE 2 – Examples of subsequent optimization are recommendations on new AI/ML analysis techniques, data collection techniques, evolution of controllers to move up the intelligence level [ITU-T Y.3173].
AN-arch-ano-req-003	The AN architecture is required to enable the management of lifecycle of controllers based on output of controllers. NOTE – Examples are management of lifecycle of controllers in underlay network (such as RAN), by controllers in underlay network (e.g. CN), creation and optimal positioning of controllers in the RAN by controllers in a higher domain, as well as evolution, experimentation and deployment of controllers.
AN-arch-ano-req-004	The AN architecture is required to enable the consumption of services exposed by service management frameworks used by underlay networks. NOTE 1 - Services exposed by service management frameworks include configuration, lifecycle management and customization. NOTE 2 - Examples of service management frameworks include ETSI ZSM framework [b-ETSI GS ZSM 009-1].
AN-arch-ano-req-005	The AN architecture is required to support capabilities enabling the AN adaptation to the evolution of underlay network services, which may take place independently to the evolution of AN. NOTE - The modification of underlay network services may be managed by an t entity different than the one managing the evolution of AN.

AN-arch-ano-req-006	The AN architecture is required to utilize the available connectivity options provided by the underlay networks to deploy and integrate controllers in different levels of the underlay networks.
AN-arch-ano-req-007	The AN architecture is required to support the monitoring of changes to underlay network connectivity among controllers deployed in various domains.
AN-arch-ano-req-008	The AN architecture is required to support the exposure of a single point of monitoring and managing of the AN functionalities deployed by the operator.
AN-arch-ano-req-009	The AN architecture is required to support the discovery of changes to network functions in the underlay networks and to include the changed functions in the AN while providing AN functionalities like evolution. NOTE - This enables plug and play of new NFs in the underlay networks.
AN-arch-ano-req-010	The AN architecture is required to support the provision of inputs to external systems regarding potential scenarios and requirements. NOTE - Examples of inputs are reports generated from AN on new use case scenarios, such as experimentation scenarios.
AN-arch-ano-req-011	The AN architecture is required to support the ability for the AN to create, store, customize and export controllers which can be deployed in various types of underlay networks.
AN-arch-ano-req-012	The AN architecture can optionally support the ability for the AN to use third party toolsets for the development and visualization of controllers. NOTE - Graphical user interfaces to edit workflows are examples of third party toolsets.
AN-arch-ano-req-013	The AN architecture can optionally support input of use case design within AN whereas design and deployment of controllers in underlay networks may be done by third parties.
AN-arch-ano-req-014	The AN architecture is required to support interfaces between AN and resource orchestration functions in the underlay network. NOTE 1 - Examples of resource orchestration mechanisms are not only network function virtualization (NFV) management and network orchestrator (MANO) but also domain specific resource managers like multi-user schedulers in RAN. NOTE 2 – The AN interfaces may be used to trigger actions or monitoring.

8. Architecture Framework Description

This clause describes the basic building blocks, components and subsystems. It also describes the list of external functionalities that the architecture framework depends upon.

8.1 High-level Architecture Framework

This clause describes the high-level architecture framework for autonomous networks, as shown in Figure 1. As noted previously, the goal of this architecture is to support the continuous evolutionary-driven creation, validation, and application of a set of controllers to a network and its services such that the network and its services may become autonomous.

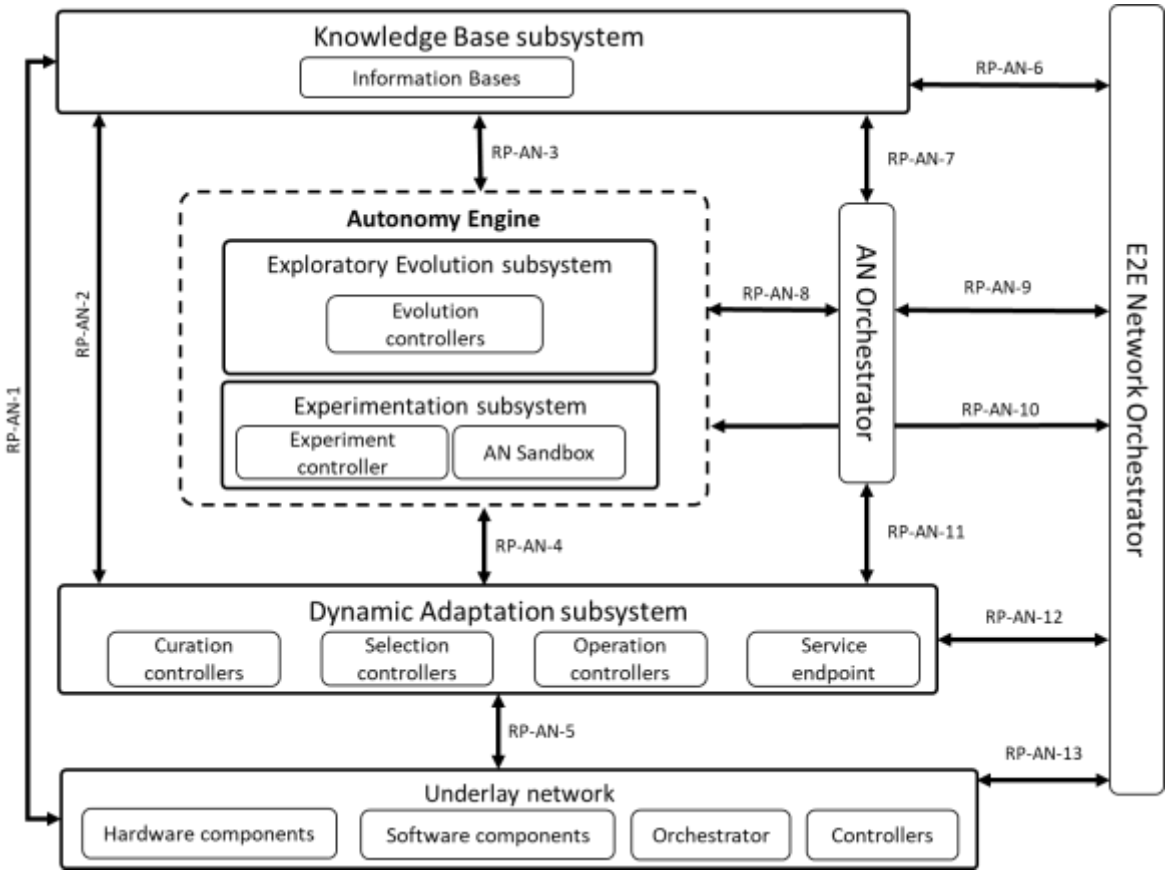

Figure 1: High-Level Framework for Autonomous Network

Editor's note: to consider possible basic introductory description of figure 1 in terms of the 5 subsystems. below,

The reference points shown in Figure 1 are the followings:

RP-AN-1, RP-AN-2, RP-AN-3 and RP-AN-6: Reference points between Knowledge Base Subsystem and underlay network, Dynamic Adaptation Subsystem, Autonomy Engine, E2E Network Orchestrator and AN Orchestrator respectively. As discussed in clause 7.4, these reference points enable access to knowledge base from other subsystems in the architecture framework.

RP-AN-4: Reference point between Autonomy Engine and Dynamic Adaptation Subsystem. This reference point is used for providing evolutionary exploration and experimentation functionalities to the Dynamic Adaptation Subsystem.

RP-AN-5: Reference point between Dynamic Adaptation Subsystem and underlay network. This reference point is used for providing selection and integration of controllers to an underlay, as the underlay undergoes changes at run-time.

RP-AN-7, RP-AN-8 and RP-AN-11: Reference points between AN Orchestrator and Knowledge Base, Autonomy Engine and Dynamic Adaptation Subsystem respectively. These reference points enable the AN orchestrator to manage the workflows and processes in the AN and the lifecycle of controllers.

RP-AN-9, RP-AN-10, RP-AN-12: Reference points between E2E Network Orchestrator and AN Orchestrator, Autonomy Engine, and Dynamic Adaptation Subsystem respectively. These reference

points are used by the E2E Network Orchestrator to manage and orchestrate control network entities in the autonomous network framework. These reference points may use existing procedures as defined in [ITU-T Y.3100].

RP-AN-13: Reference point between E2E Network Orchestrator and underlay network. This reference point is used by the E2E Network Orchestrator to manage and orchestrate control network entities in the underlay network. This reference point may use existing procedures as defined in [ITU-T Y.3100].

NOTE 1 – Detailed description of the reference points shown in Figure 1 is for future study.

An example realisation of the architecture framework for Autonomous Networks can be found in Appendix I (IMT-2020 network underlay).

In addition to the architecture components, there are functionalities external to this architecture framework, which may enhance the AN architecture. See Appendix III for details.

NOTE 2 – The details of the interaction of these external functionalities with the architecture framework through the architecture framework reference points are out of scope of this document.

8.2 Description of controller

In this architecture, we introduce the term “controller”. As introduced in clause 6, a controller is a workflow, open loop or closed loop [ITU-T Y.3115] composed of modules, integrated in a specific sequence, using interfaces exposed by the modules, which can be developed independently of the system under control before integration into the system under control, to solve a specific problem or satisfy a given requirement.

NOTE 1 – Modules may themselves be workflows, open loops, or closed loops. Other examples of modules include aggregation functions, DNS configuration interfaces, functions gathering orchestrator statistics, an entire deep neural network (DNN) model, a single layer of a DNN model, etc.

Exploratory evolution and experimentation are examples of functionalities in the AN which act upon controllers. Exploratory Evolution hosts evolution controllers which provide the functionality that creates and modifies a controller in accordance with the system under control and the real-time changes therein. Experimentation subsystem hosts experimentation controller which provides the functionality that validates controllers using inputs from a combination of underlay network, simulators and/or testbeds. In addition, the Dynamic Adaptation subsystem hosts the curation, selection and operation controllers which provide the functionality of process of continuous integration of controllers to an underlay as the underlay undergoes changes at run-time.

NOTE 2 - Examples of system under control are managed entities, workflows and/or processes in an IMT-2020 network.

The architecture described here enables the design, creation, and adaptation of these controllers.

This architecture inputs modules that are amenable to composition and produces controllers which are in turn modular.

Figure 2 shows the different forms interaction of controllers with the underlay network.

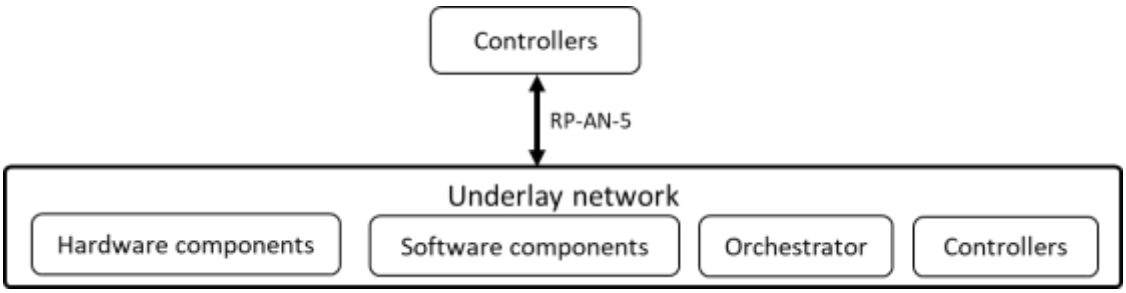

Figure 2: Controllers and Underlay Interaction

The interactions are:

- Controller interacting with hardware components [b-LogicNets].
- Controller interacting with software components.
- Controller interacting with an orchestrator or other software control mechanisms.

  NOTE 3 - Other software control mechanisms, such as workflow tools [b-FRINX], may be considered as orchestrators.

- Controller interacting with other controllers.

NOTE 4- Building upon this simple representation, hierarchies of controllers may be formed.

8.3 Description of the sub-systems and their components

This clause describes the sub-systems of the high-level architecture framework shown in Figure 1, and associated components.

8.3.1 Autonomy Engine

Autonomy engine refers to the grouping of the evolutionary exploration subsystem and the experimentation subsystem described in clauses 8.3.1.1 and 8.3.1.2 respectively. Together, these architectural components enable the more general trial and error process where new candidate controllers are generated in the former and validated by the latter. This grouping directly addresses the need for automatic “design-time procedures” [ITU-T Y.3177].

In addition to controllers, Appendix II describes how the AN architecture can be used to achieve autonomous operation of itself.

8.3.1.1 Exploratory Evolution Subsystem

Exploratory evolution enables exploration and evolution to adapt controllers in response to changes in the underlay network. Knowledge stored in the Knowledge Base subsystem is used in autonomous networks for supporting the continuous exploratory evolution. As explained in 8.1, Reference point RP-AN-3 allows Exploratory evolution subsystem to interface with the Knowledge base. Figure 3 shows an overview of the Exploratory evolution subsystem and its relation to the Knowledge Base subsystem as new controllers are generated or existing controllers are updated (evolved) as part of exploratory evolution.

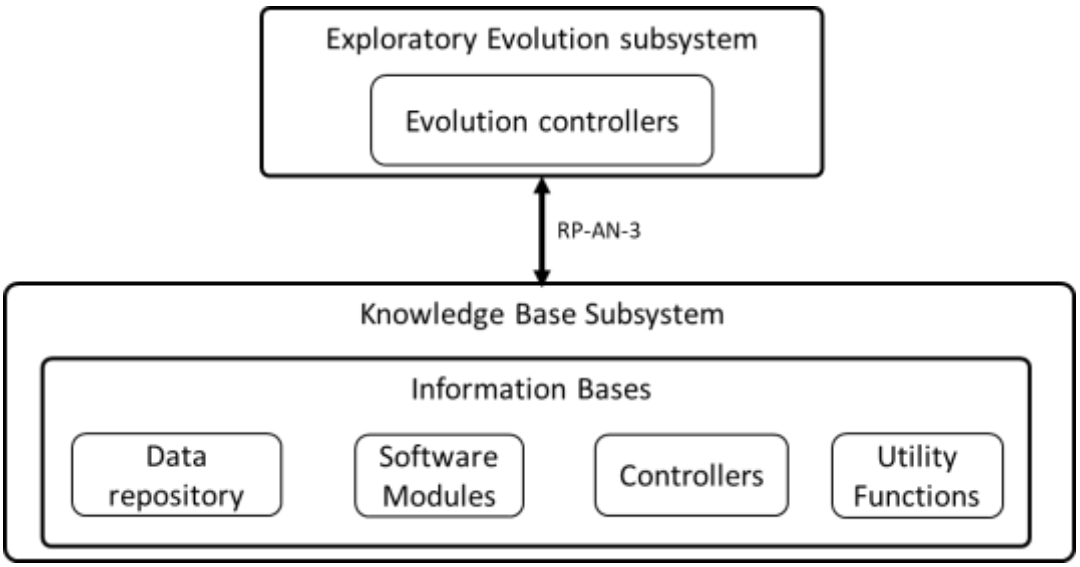

Figure 3: Exploratory Evolution Overview

As stated in clause 6, any approach towards an autonomous network (truly or otherwise), requires the ability to adapt its operation. This adaptation can be motivated by changing operation environments, new technological innovation, faults, human error, the pursuit of contextual optimality, etc. Additionally, based on the requirements in clause 7, this architecture requires the ability to alter the logic which is used to operate autonomous networks (i.e., controllers). Without such functionality, it is not possible to achieve adaptation which is sufficiently flexible across the spectrum of use cases, operational environments, technological innovations, and potential human errors.

NOTE 1 - It is important to remember that controllers may themselves possess the ability to adapt their outputs based on learning or experience – so called *cognitive* controllers [b-Mwanje 2020]. Even in this case, there is a limit to their ability to adapt to the unknown (e.g., a never before seen anomaly), to embrace new technologies (e.g. a new transport protocol), or to handle error (e.g. even the authors of this document have been known to create bugs). In all cases, human intervention is required.

Controller specifications are high-level, non-executable representations of a Controller with the metadata corresponding to necessary functionality of the controller and a utility function to be achieved. Controller designs are low-level, non-executable representations of controller containing modules, their configurations, and their parameter values which is used to instantiate a controller. Controller designs are derived from controller specifications by the evolution controller.

Collections of controllers may be formed with each controller tasked with the same purpose but with different compositions.

Hence, the evolutionary exploration subsystem is responsible for:

1. The automatic generation of controller designs from composable software module specifications.
2. The automatic modification of controller designs based on existing controller and module specifications and/or designs.
3. The automatic generation of controller designs from controller specifications
4. The automatic modification of controller designs based on existing controller specifications.

Exploratory evolution will enable the automated design or modification of controllers and their hierarchies for the purpose of exploring the range of possible controller logics – and hence how the controller will adapt to the operational environment.

NOTE 2 – One approach to achieve such automated design for controllers and controller hierarchies is population-based artificial intelligence (AI) techniques, such as evolutionary computing [b-evolution].

8.3.1.1.1 Evolution controller

Exploratory evolution is the process that creates and modifies a controller in accordance with the system under control and the real-time changes therein.

NOTE 1 – An example of a process that creates a controller is the composition of controllers from modules or other closed loops. This may involve the selection of modules which are used for composition.

NOTE 2 – An example of a process that modifies an existing controller is the dynamic change in the controller's structure by adding new modules, deleting existing modules, replacing existing modules, or rearranging the structure of a controller's modules, in accordance with the real time changes in the system under control.

An evolution controller is the component responsible for managing the application of exploratory evolution on controllers. Exploratory evolution is the ability to modify the structure and configuration of a controller. This assumes that the controllers are composed of modular and configurable elements or "building blocks". Thus, a controller's structure may be modified by:

- The configuration of each software module's parameters
- The selection of which modules are present within a controller
- The relationships between the modules within the controller

The process of exploratory evolution is agnostic to whether the current operational environment is known ahead of time or is completely new and unseen. The process includes generating options for exploratory evolution and based on the characteristics of the controller and the knowledge base, applying such evolutions on various types of controllers. As part of this, controller characteristics may be discovered, new controllers may be composed from modules or other controllers to provide new capabilities in the network. Declarative representation of use case, provided by AN orchestrator, is used as input by the evolution controller. Controller descriptions may be updated by the evolution controller based on the exploratory evolution.

NOTE 3 - Examples of processes to drive the modification of a controller are:

1) biologically inspired artificial evolution, as found in evolutionary computing or genetic programming [b-large-evolution, b-evolution],

2) Bayesian optimisation [b-bayesian-radio],

3) game theoretic approaches [b-game-theory].

Specific algorithms, including those provided by third party solution providers, used for exploratory evolution are out of scope of this document.

NOTE 4 - Examples of application of exploratory evolution in various application contexts are given below:

- 1) A "RAN channel scheduling controller" is an example of a controller used to allocate radio resources to users in a multi-user environment. Exploratory evolution is applied to a RAN channel scheduling controller in response to the change of radio channel feedback from the UE. This may include selecting the most appropriate algorithm from a set of alternatives.
- 2) An "anomaly detection controller" is an example of a controller used to detect abnormal states in the operation of a network service, such as security attacks or peaks in resource usage for network function. In this context, the new approaches of data fusion algorithms [b-

data-fusion] may be applied. Exploratory evolution is applied to “anomaly detection controller” by optionally using and configuring newly provided data fusion algorithms as the input of an “anomaly detection controller”.

- 3) A “time-to-live controller” is an example of a controller used to configure the time duration for which a certain content is cached in a CDN server. In a time-to-live controller in a caching system at the edge, optimisation of the timeout parameter(s) is an example of application of exploratory evolution.
- 4) A “scaling controller” is an example of a controller used to increase or decrease the resource allocation for a network function. In this context, exploratory evolution may be applied by controlling the configuration of the scaling method of deployed controllers in a specific network domain.

NOTE 5 – Optimisation of exploratory evolution, e.g., reducing the time taken for exploratory evolution in previously seen operational environments, is possible by using accumulated knowledge. However, such optimisation scenarios are out of scope of this document.

8.3.1.2 Experimentation Subsystem

Experimentation subsystem is concerned with the validation of controllers**Error! Reference source not found.**. It will design, orchestrate, and execute experimental scenarios. These are supported by Knowledge Base subsystem, AN Orchestrator and E2E Network Orchestrator.

Figure 4 shows an overview of the Experimentation subsystem and its relationship with Knowledge Base subsystem, AN Orchestrator and E2E Network Orchestrator through various Reference points described in clause 8.1.

Reference point RP-AN-3 allows Experimentation subsystem to interface with the Knowledge base as experimentations are designed and updated as part of experimentation process. Reference point RP-AN-8 and RP-AN-10 allow the Experimentation subsystem to orchestrate the experimentations.

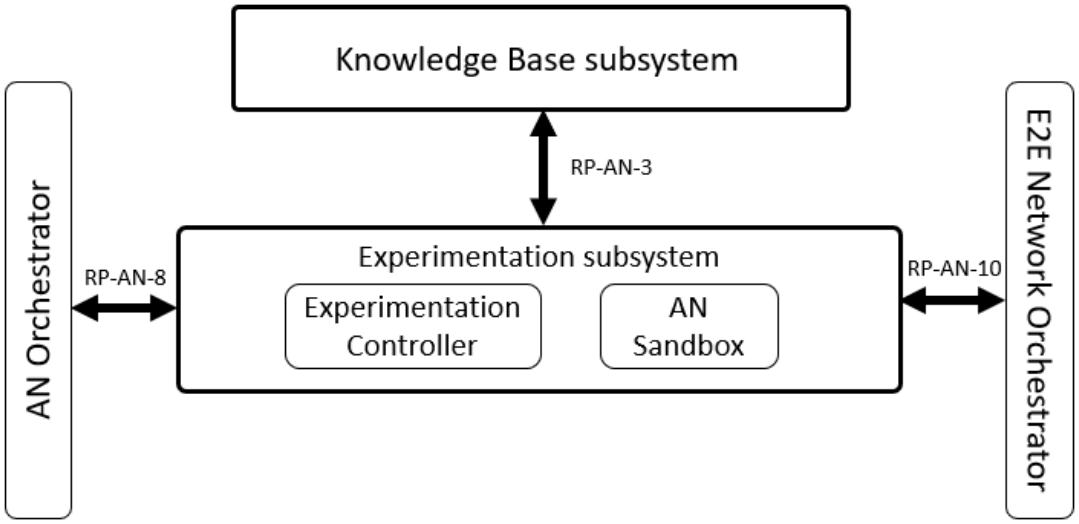

Figure 4: Experimentation Subsystem

Controllers must be validated before being integrated to the underlay to ensure that they are free of errors and meet both functional and non-functional requirements.

Validation is a spectrum of activities that may encompass one or more tasks, including static testing, simulation, testbed deployment and canary testing. In addition, validation can all be used to

assess non-functional properties, such as trust, providing confidence in the "handover of work, duties, or decisions" to the architecture.

To compliment these validation activities, the experimentation controller also requires additional input from about the underlay network and its configuration. As shown in **Error! Reference source not found.**, this information should be stored and made available from the knowledge base. Representative examples of such data are discussed in clause 8.3.3.

8.3.1.2.1 Experimentation Controller

Experimentation is the process that validates controllers using inputs from a combination of underlay network, simulators and/or testbeds. The process of experimentation ensures that the controller under experimentation satisfies the use case requirements and is compatible with deployment in the intended underlay.

An experimentation controller is a component which generates potential scenarios of experimentations based on controller descriptions and representations of the use case. Experimentation controller uses additional information, as provided by the knowledge base and AN orchestrator, in the process of generating scenarios of experimentation.

NOTE 1 - Methods for generating scenarios for experimentation are assisted by additional information including knowledge captured in the knowledge base and/or machine learning. Experimentation controller may exploit the structured representation (e.g., TOSCA YAML [b-OASIS TOSCA-v1.3]) of the controllers to derive scenarios for experimentation. Experimentation scenarios can also be provided by 3rd party providers to be used by the experimentation controller.

In addition to generating scenarios for experimentation, experimentation controller executes the scenarios in the AN Sandbox, collates and validates the results of the experimentation. Reports may be generated by experimentation controller which captures information from the steps of generating scenarios, execution and validation of controllers. These reports may be shared with humans or used for analysis by algorithmic methods. Experimentation scenarios may be optimized as result of analysis of the experiments.

NOTE 2 - Selection of new validation or test scenarios are examples of optimizations applied to experimentation scenarios.

NOTE 3 - In the process of experimentation, experimentation controller can use different types of components such as simulators, data generators hosted in the AN Sandbox, including those provided by 3rd parties. Experimentation may be triggered by various AN workflows which necessitates validation of controllers, e.g. software update of controllers. The process of experimentation may be configurable e.g., it may be triggered periodically, asynchronously.

NOTE 4 - Examples of experimentation in various application contexts are given below:

- The use of static "sanity checking" such as formal methods [ITU-T Y.3320] or model checking to ensure that provided management and orchestration solutions are well-formed against pre-defined rules
- The use of simulators or digital twins in offline validation of controllers. These simulators or digital twins can support the same interfaces as underlays.
- The use of digital twins [b-Digital-twin] in online validation of controllers before deployment

NOTE 5 - online validation involves use of timescales comparable to real underlays e.g., validation of controllers (xApps) [b-ORAN] using digital twins.

- Combinations of the above to achieve broader coverage of validation, from the offline validation to online validations during the operation of the underlay.

8.3.1.2.2 AN Sandbox

AN Sandbox is an environment in which controllers can be deployed, experimentally validated with the help of (domain specific) models of underlays, and their effects upon an underlay evaluated, without affecting the underlay.

NOTE 1 - AN Sandbox generates reports regarding the experimental validation of controllers. These reports are collated by the experimentation controller and the Knowledge Base is updated.

NOTE 2 - The domain specific models of underlays are generated using inputs from underlays. These inputs are used in configuring simulators in AN Sandbox. For example, the packets per second to be used to simulate a real-world scenario. In addition, AN Sandbox simulates scenarios which are rarely or never seen in underlays. For example, a burst of traffic which rarely occurs in real network.

AN Sandbox hosts different types of components such as simulators, data generators, including those provided by 3rd parties. Experimentation controller may trigger experiments of various AN workflows which necessitate validation of controllers, e.g. software update of controllers.

8.3.2 Dynamic Adaptation Subsystem

Dynamic adaptation equips the underlay network with autonomy and the ability to handle new and hitherto unseen changes in network scenarios. Knowledge stored in the Knowledge Base subsystem is used in autonomous networks for supporting the continuous dynamic adaptation. AN Orchestrator and E2E Network Orchestrator support the orchestration needed for dynamic adaptation towards underlay network.

As explained in clause 8.1, Reference point RP-AN-2 allows Dynamic adaptation subsystem to interface with Knowledge base subsystem. Reference points RP-AN-11 and RP-AN-12 allow Dynamic adaptation subsystem to interface with AN orchestrator and E2E Network Orchestrator respectively. RP-AN-5 allows Dynamic adaptation subsystem to interface with underlay network. RP-AN-13 allows E2E network orchestrator to interface with underlay network.

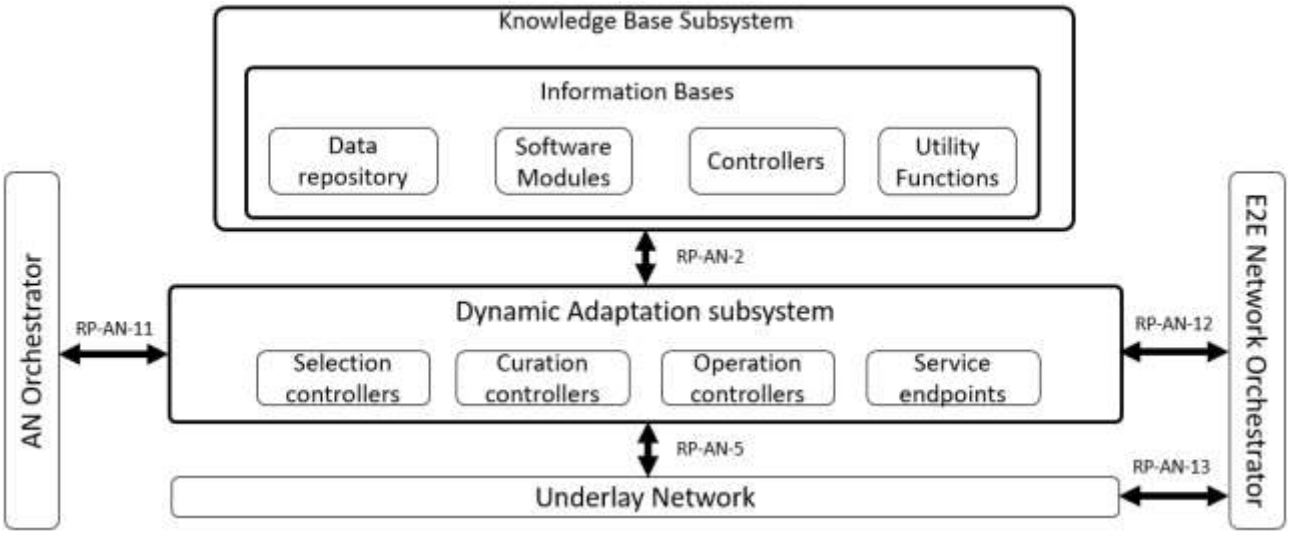

Figure 5: Dynamic Adaptation Subsystem

The dynamic adaptation subsystem is responsible for curating a set of controllers which may be considered as “fit for purpose” or “safe enough to try” and selecting a subset of controllers for integration with the underlay.

NOTE 1 - Here, “fit for purpose” is evaluated based on the fitness function or utility score obtained from the experimental evaluation system (clause **Error! Reference source not found.**.2).

This set of controllers is drawn from the controllers which were validated by the experimental subsystem. Additionally, this subsystem is responsible for which of these curated controllers should be selected for actual deployment in the management of the managed entity. Precisely when, under what conditions, or with what frequency curation or selection happens are configurable properties of the curation and selection processes themselves. This is necessary as each managed entity, as well as the operational and business environments in which they operate, vary from use case to use case. To accommodate this, the curation process is guided by requirements. Examples of such requirements may include:

- The size of the curated controller lists
- The average utility of the curated controller lists
- The diversity of the controllers within the curated controller lists
- The utility threshold required to be considered to enter or remain within the curated controller lists.

NOTE 2 - it is important to remember that metrics such as KPI, QoS, QoE, etc, are expected to be represented within a controller’s utility function.

As shown in Figure 5, the controllers are stored within the network information base. As the evolvable controllers undergo constant evolution, it is the responsibility of the dynamic adaptation to bring stability to the operation of the managed entity by creating a level of separation between evolvable controllers managed by the autonomy engine and the operation controller integrated with the managed entity.

Also shown in Figure 5, the curation and selection processes are realised as controllers. As such, their internal structure may be composed (and subsequently evolved) from different modules as required. Controllers for curation and selection are discussed in clause 8.3.2.1.

NOTE 3 - For example, a selection controller may be composed of modules which send the trend of fluctuation of user populations, network traffic, or resource demands. As discussed in 8.2 such controllers may be implemented using ML pipelines.

Selection and integration of controllers to a managed entity requires a stable set of functioning controllers which can respond correctly in sub-second timescales, depending on the use case in question.

Accordingly, the autonomy engine and dynamic adaptation subsystem corresponds to the *design-time* and *run-time* concepts, respectively, as expressed in [ITU-T Y.3177].

8.3.2.1 Adaptation Controller

Dynamic adaptation is the process of continuous integration of controllers to an underlay, as the underlay undergoes changes at run-time. Integration of controllers may involve multiple domains of the underlay.

NOTE 1 – Examples of underlays are edge networks, core networks, management plane, CI/CD pipelines, etc. Integration of controllers into the underlay involves usage of underlay specific APIs, customization of interfaces, configurations and interface elements used for integration. Examples of changes undergone by the underlay are updates in software or hardware components, failures in software or hardware components, configuration changes, or other external dependencies (including those provided by 3rd parties). Continuous integration includes updating the controllers in the underlay to handle the changes undergone by the underlay.

Examples can include:

- Scaling via the routing of traffic to different processing nodes in either the use of control plane via DNS updates or SDN configurations, refer FG-AN-usecase-040 in [ITU-T Y.Supp 71]
- Scaling via the number, replication, and distribution of bare metal, virtual machine, or container resources attached to network services refer FG-AN-usecase-018 in [ITU-T Y.Supp 71]
- Scaling via the priority or relative bandwidth allocation of radio spectrum to different user or use case categories in RAN in FG-AN-usecase-032 in [ITU-T Y.Supp 71]

NOTE 2 – Specific configurations for integration of controllers for specific use cases may be out of scope of this document.

Adaptation controller is the component in AN responsible for selecting candidate controllers from a set of generated controller configurations which are ready for integration and executes the integration to the underlay. An adaptation controller will monitor deployed controllers and the underlay, deciding opportunities for new controller integrations to the underlay. In monitoring a deployed controller, an adaptation controller will discover underlay specific parameters (including those provided by 3rd parties) for optimisation, data points of collection and KPIs for tracking and may update such knowledge to the knowledge base.

Adaptation controller has two parts: Curation controller (responsible for selection and maintenance of the controllers within the curated controller lists from the evolvable controllers) and Selection Controller (responsible for the selection of a services' operational controller from the curated controller lists).

Generation of configurations for adaptation may take the controller description as input along with the description or metadata related to the underlays. In the process of adaptation, the adaptation controller may utilize the services provided by service management frameworks such as ONAP [b-ONAP].

Reports may be generated by adaptation controller which captures information from the process of adaptation. These reports may be shared with humans or used for analysis by algorithmic methods. Adaptation process may be optimized as result of analysis of the reports.

NOTE 3 - Examples of adaptation in various application contexts are given below:

- The need to use different traffic shaping algorithms for various geographical contexts, such as urban vs rural
- Business priorities may change over a period of time, e.g. prioritization of performance KPIs over energy efficiency or prioritisation of internal applications over third party applications. These changes in business priorities may necessitate the use of different virtual machine or container scheduling controllers.
- There could be a need to deploy new technology in order to improve or optimise operation, including adding new capabilities that previously did not exist. E.g., new AI/ML algorithms or new data fusion approaches to blend the increasing number of data sources.
- There could be a need to deploy new technology in order to address errors or faults. E.g., data acquisition or actuation software for new hardware devices or adaptation software to account for incompatibilities in deployed technology.

Selection of candidate controllers from a set of generated controller configurations is followed by the processes used to drive adaptation in the underlay. Examples of processes used to drive adaptation are:

- Simple threshold-based replacement of one deployed controller with another, where threshold is defined against a controller's performance
- PID controller [b-PID]-based replacement of one deployed controller with another

- The use of AI/ML model in the prediction of future operation and response to pre-emptively exchange a deployed controller [ITU-T Y.Supp 71]
- Combinations of the above in concert with knowledge stored within the knowledge base

8.3.2.2 Operation Controller

An operation controller is a controller responsible for the operation of a managed entity. Operation may include analysing the data (e.g., throughput or latency) related to the managed entity and applying actions (e.g., scale in/out or migration) to the managed entity. An operation controller is selected and applied to the managed entity by selection controller. After application of Operation Controller to a managed entity, the controller is continuously monitored by the selection controller for the purpose of providing the most effective operation of the managed entity.

While evolution controllers, experimentation controllers, and adaptation controllers are responsible for activities within the high-level architecture framework, operation controllers are responsible for activities out with the high-level architecture framework.

8.3.2.3 Service Endpoints

As discussed in [ITU-T Y.3104], the network functions interact with each other to provide the IMT-2020 network services specified in [ITU-T Y.3102]. The network functions within the core network (CN) control plane (CP) interact with each other using service interfaces.

The service interfaces used for interaction between the Dynamic Adaptation subsystem and the underlay network are referred to as service endpoints.

8.3.3 Knowledge Base subsystem

Autonomous networks require the collection, description, usage, storage, and analysis of data. The analysis of data and information, resulting in an understanding of what the data and information mean is often referred to as knowledge.

Data, information, and knowledge which is required for the controllers to operate the managed entity and in its goal of supporting the continuous exploratory evolution, realtime online experimental validation, and dynamic adaptation.

Knowledge base is a subsystem which manages storage, querying, export, import and optimization and update knowledge, including that derived from different sources including structured or unstructured data from various components or other subsystems.

NOTE 1 - Knowledge includes metadata which is derived from the capabilities, status of AN components. This knowledge is stored and exchanged as part of interactions of AN components with knowledge base. Knowledge can be derived from different sources including structured or unstructured data from various actors involved in a use case and/or various experiments in AN Sandbox.

Managing knowledge includes storing, querying, export, import and optimize the knowledge. AN workflows, including exchange of knowledge between AN components, may in turn result in update of knowledge base.

NOTE 2 – Uses of knowledge stored in knowledge base by other components include to facilitate the deployment and management of controllers in underlays, and selection and optimization of experimentation strategies in the experimentation stage.

Examples of knowledge stored in a KB are:

1) relevant descriptions of modules and controller meta data taxonomies and ontologies.

2) An underlay network configuration represents the various arrangement, relationships, contents, and settings of the elements of an underlay network as may be required by the online realtime experimental evaluation subsystem to build and configure an experimental underlay network, or

other architectural components. E.g., network topology, host configuration and location related parameters, Types of services and application requirements. The configurations may be represented using OASIS TOSCA YAML [b-OASIS TOSCA-v1.3]

3) Metrics: Metrics are the data related to the status and performance of the different components of the architecture, controllers, operating environment, underlay network, and managed entities. e.g., resource usage such as CPU usage, workload such as packet rate, performance metrics such as QoS.

8.3.4 AN Orchestrator

AN orchestrator is the component responsible for managing workflows and processes in the AN and steps in the lifecycle of controllers. To manage the workflows and processes in AN, AN orchestrator coordinates with various other functions in the AN as well as outside the AN.

NOTE 1 - Examples of workflows and processes in the AN are interactions with E2E network orchestrators, interactions with knowledge base, and AN component repositories. Examples of controller instances are:

- A set of Java objects to be executed on the JVM
- A workflow of tasks as represented in the FRINX machine [b-FRINX]
- A CL in the ZSM framework [b-ETSI GS ZSM 009-1]
- A controller in the ONAP framework [b-Acumos-DCAE]
- An ML pipeline [ITU-T Y.3172]

NOTE 2 - Steps in the lifecycle of controllers are creation or instantiation of controllers from controller designs, storage, validation, update, deletion, discovery, configuration, deployment, monitoring of controllers.

NOTE 3 - Some steps in other functions applied to controllers are outside the scope of lifecycle of controllers, e.g., optimization of controllers may be achieved with the help of functions external to the AN.

Being part of the management plane, AN orchestrator provides interface to human operators in the form of reports regarding the functioning of the AN and human interfaces for configuring the AN, where applicable.

8.3.5 Underlay network

An underlay network is a telecommunication network and its related network functions. An underlay network may provide interfaces which facilitates application of controllers.

The underlay network may consist of hardware, software components, in addition to management components such as orchestrators.

NOTE – An IMT-2020 network is an example of an underlay network.

8.3.6 E2E Network Orchestrator

As defined in [ITU-T Y.3100] in the context of IMT-2020, orchestration is the set of processes aiming at the automated arrangement, coordination, instantiation and use of network functions and resources for both physical and virtual infrastructures by usage of optimization criteria.

Based on the orchestration, E2E network orchestrator is a collection of functions, interfacing with the subsystems in the autonomous network architecture framework, to manage and orchestrate control network entities in the autonomous network including the underlay network.

9. Sequence diagrams

This clause gives the sequence diagrams showing the interaction between the architecture framework components.

Editor's note: to consider a NOTE here to indicate this is a high level presentation of the diagrams without reference to RPs.

9.1 Exploratory Evolution of Controllers

Exploratory Evolution of Controllers involves creation and modification of controllers in accordance with the underlay network and the real-time changes therein. Below is an example scenario where controllers are created. In this example, AN operator provides a new use case specification from which new controller specification controllers are derived. There are additional example scenarios where the Evolution controller reuses an existing controller specification and applies the Exploratory evolution process.

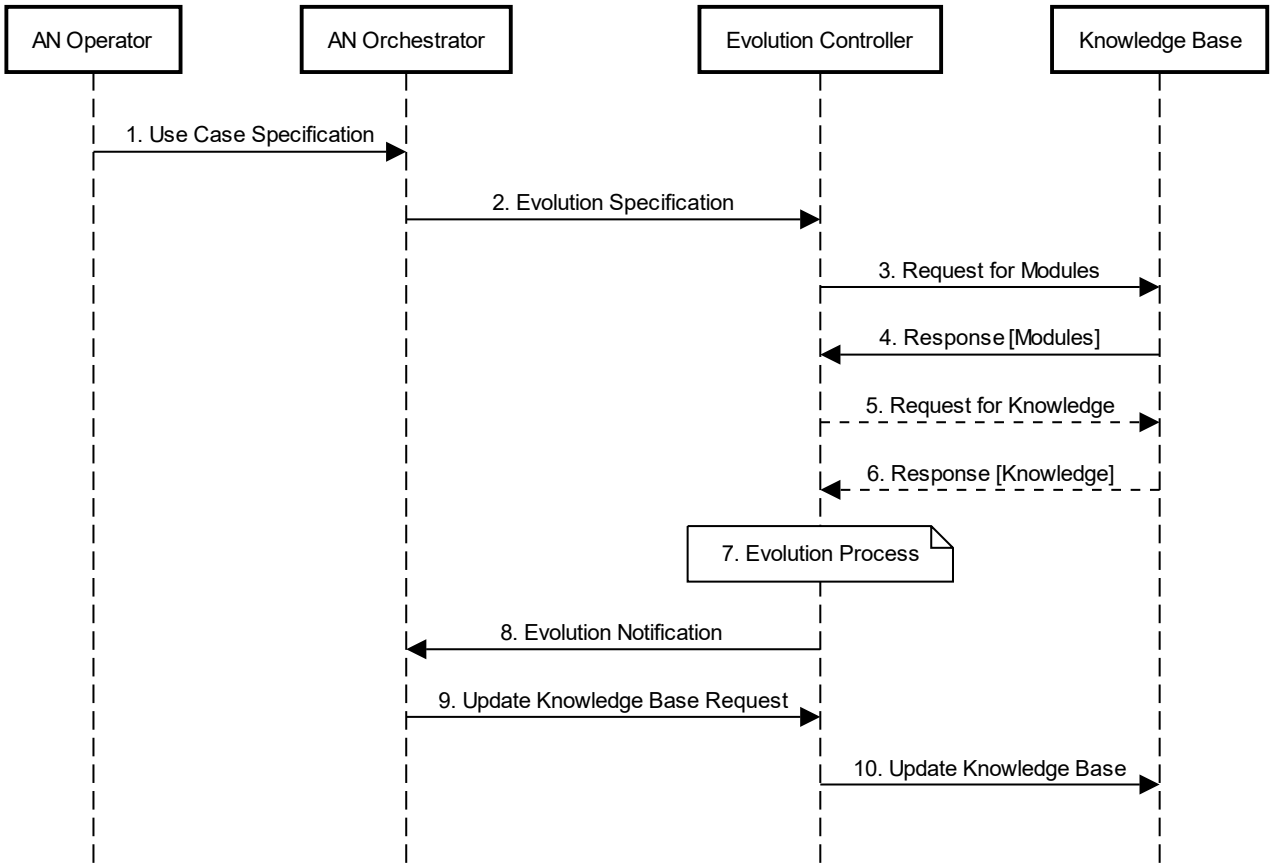

Figure 6: Creation of Controllers

The steps involved in the scenario described in Figure 6 are:

1. AN operator provides a use case specification to the AN orchestrator. Use case specification includes the actors, their relationships and utility functions corresponding to the use case.
2. An orchestrator derives an evolution specification from the use case specification. Evolution specification has a Controller specification with the metadata corresponding to necessary functionality of the controller and a utility function to be achieved (after the exploratory evolution process).
3. evolution controller queries the KB for modules corresponding to the controller specification.
4. The knowledge base replies to the evolution controller with the available modules corresponding to the request.
5. This is an optional step where the evolution controller requests knowledge from the knowledge base relevant to the use case specification or the exploratory evolution process.

6. Corresponding to the optional step 5, the knowledge base responds with requested knowledge
7. Evolution controller applies the exploratory evolution process to create new controller(s). This includes composition of controllers from modules or other closed loops as described in clause 8.2
8. The evolution controller updates the KB. This includes storing the generated controllers to the knowledge base.

NOTE - Discussion of the logic used to drive the exploratory evolution process is beyond the scope of this document. Examples of such processes can be found in clause 8.3.1.1.

9.2 Experimentation for Controllers

Experimentation for Controllers involves validation of controllers using inputs from a combination of underlay network, simulators and/or testbeds. Below is an example scenario where evolvable controllers are validated. In this example, a new experiment specification, which has controller specifications to be validated, is provided by AN orchestrator. Experimentation controller derives scenarios for experimentation based on the experiment specifications. Based on these scenarios, Experimentation controller interacts with the KB to gather additional supporting specifications

(experiments and/or controllers) and relevant knowledge to design an experiment to validate the controllers included in the Experiment specification.

NOTE 1 - There are additional example scenarios where the Experimentation controller reuses an existing experiment specification (stored in the KB) and designs the experiments to validate the controller included in the experiment specification provided by the AN orchestrator.

Editor's note: to align terminology to "Experimentation controller" in all diagrams

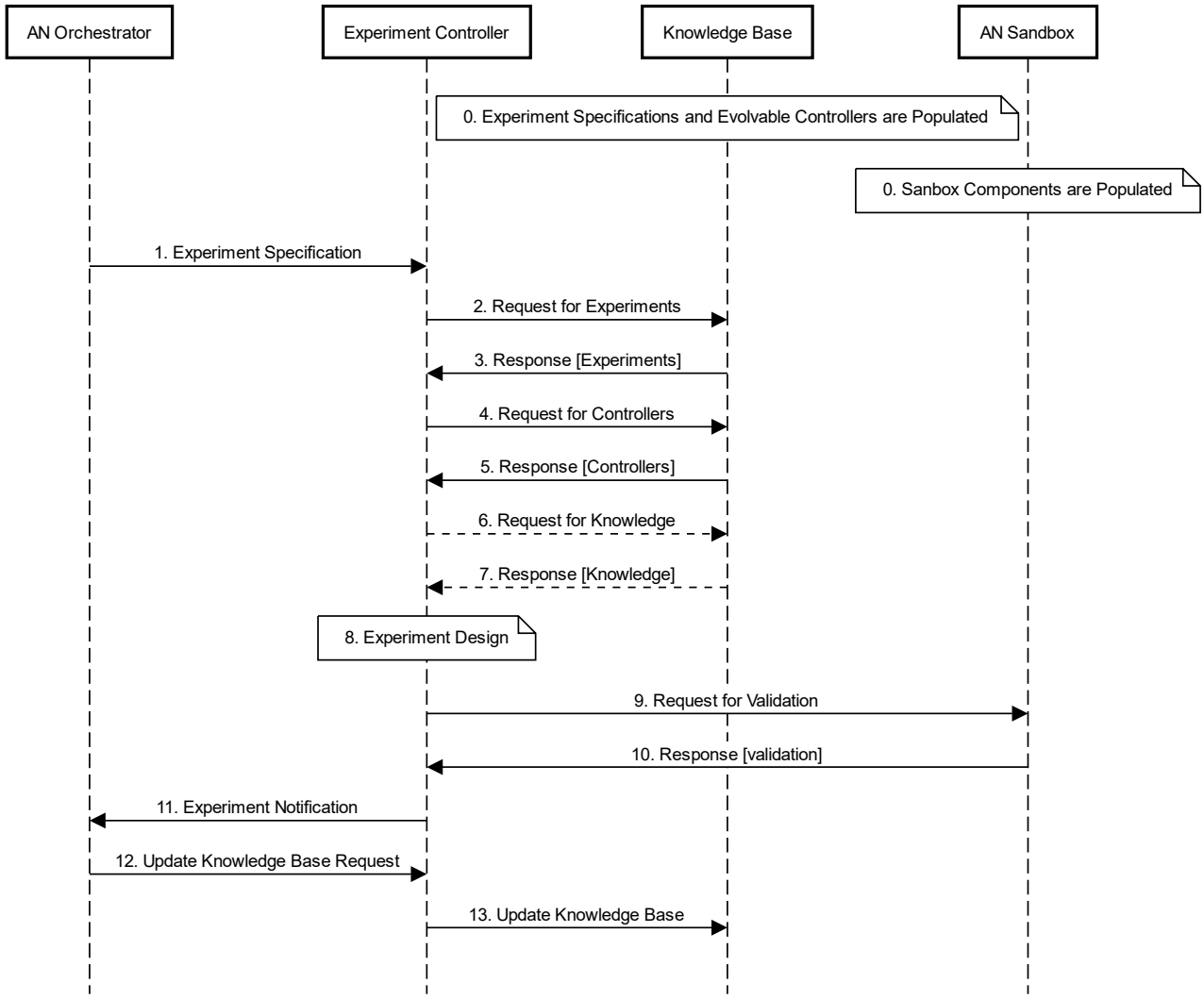

Figure 7: Validation of Controllers

The steps involved in the scenario described in Figure 7 are:

Pre-requisites:

0. Experiment specifications and evolvable controllers are populated in the KB. This may be done based on Creation of controllers in Fig 8 or based on offline provisioning by AN operator. In addition, the AN Sandbox is populated with components which are ready for instantiation and execution to validate the controllers.
1. AN orchestrator provides experiment specification which has the controller specification for the controller to be validated.
2. Experimentation controller requests the current list of experiments from the KB
3. The knowledge base replies with the requested data, if any
4. Experimentation controller requests the current list of controllers from the KB
5. The knowledge base replies with the requested data, if any

6. Experimentation controller requests additional knowledge from the KB, needed to support the experiment design.

NOTE 2 - Examples of additional knowledge may include operational data from the underlay, such as user traffic behaviour, user density in a geographical area, previous security attacks, known bad configurations of base station tilt angles, or mean time between failures for certain hardware models.

7. The knowledge base replies with the requested data, if any
8. The experimentation controller will design potential experimentation scenarios
9. For each experimentation scenarios, the experimentation controller will request the AN sandbox to perform the validation
10. The AN sandbox will report the results to the experimentation controller
11. The experimentation controller will perform any necessary analysis of the results, and notifies the AN orchestrator.
12. AN orchestrator triggers the experimentation controller to update the knowledge base.
13. The Experimentation controller updates the KB. This includes storing the experiment results for the validated controllers to the knowledge base.

NOTE 3 - Discussion of the logic used to drive the experiment design is beyond the scope of this document. Examples of such processes can be found in clause 8.3.1.2.

9.3 Dynamic adaptation of Controllers

Dynamic adaptation is the process of continuous integration of controllers to an underlay, as the underlay undergoes changes at run-time. Below is an example scenario where validated controllers are curated, selected and deployed to the underlay.

In this example, the AN Orchestrator will provide the curation controller with an adaptation specification (which has the controller specifications), to drive the curation process. Curation controller queries the KB for validated controllers and relevant knowledge. Followed by this, the

AN Orchestrator will provide the selection controller with an adaptation specification (which has the controller specification) to drive the selection process.

NOTE 1 – There are additional example scenarios where the Curation controller reuses existing controller specifications rather than deriving them from the Adaptation specification. Similarly, there are additional example scenarios where the selection controller reuses existing controller specifications rather than deriving them from the Adaptation specification.

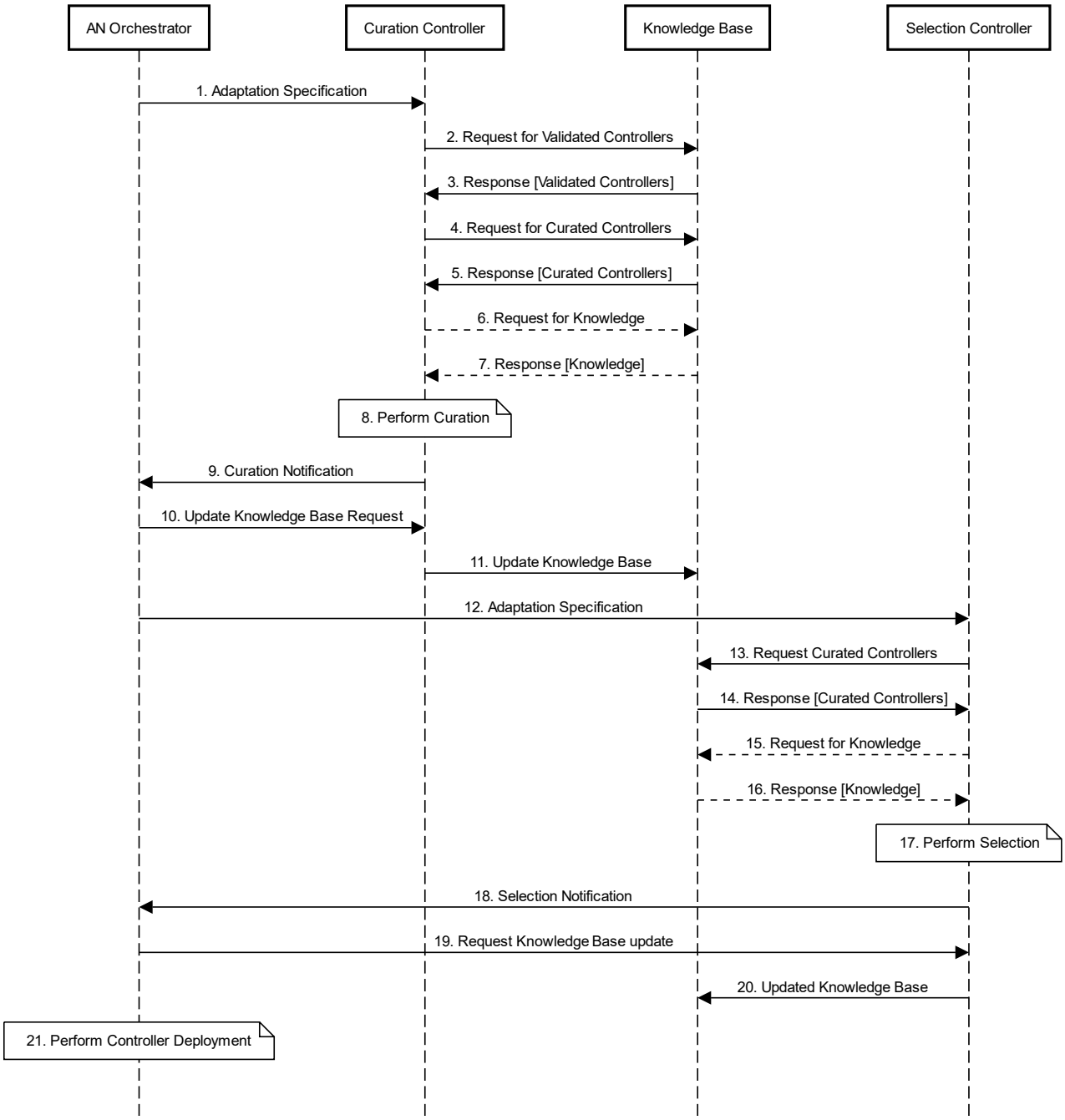

Figure 8: Dynamic Adaptation of Controllers

The steps involved in the scenario described in Figure 8 are:

1. The AN Orchestrator will provide the curation controller with an adaptation specification (which has the controller specifications), to drive the curation process.

2. The curation controller derives the controller specifications from the adaptation specification and requests validated controllers from the Knowledge Base
3. The Knowledge Base replies with the requested data, if any
4. The curation controller requests the list of curated controllers for the use case from the Knowledge Base
5. The Knowledge Base replies with the requested data, if any
6. The curation controller requests additional knowledge from the Knowledge Base needed to support the curation process

   NOTE 2 – Examples of additional knowledge may include controller utility scores, current traffic load, computational resource consumption, common modules used in the composition of controllers, or semantic relationships to currently deployed controllers for other use cases.

7. The Knowledge Base replies with the requested data, if any
8. The Curation Controller performs the curation process which decides the validated controllers that will be added to the curated list, if any, and which controllers in the curated list should be removed, if any.
   NOTE 3 - Discussion of the logic used to drive the curation process is beyond the scope of this document. Examples of such processes can be found in clause 8.3.2.1.

9. The Curation Controller notifies the AN Orchestrator that it has completed the curation process
10. The AN Orchestrator request the Curation Controller to update the Knowledge Base with the curated list of controllers
11. The Curation Controller updates the Knowledge Base with the list of curated controllers for the use case
12. The AN Orchestrator provides the adaptation specification to the Selection Controller
13. The Selection Controller derives the controller specifications from the adaptation specifications and requests the list of curated controllers from the Knowledge Base
14. The Knowledge Base replies with the list of requested data, if any
15. The curation controller requests additional from the Knowledge Base needed to support the curation process

    NOTE 4 – Examples of addition knowledge may include controller utility scores, current traffic load, computational resource consumption, common modules used in the composition of controllers, or semantic relationships to currently deployed controllers for other use cases.

16. The Knowledge Base replies with the requested data, if any
17. The Selection Controller will perform the selection process which decides the Curated Controller that should be deployed to the underlay. The deployed controllers are known as Operational Controllers for the specified use case, if any.

    NOTE 5 - Discussion of the logic used to drive the selection process is beyond the scope of this document. Examples of such processes can be found in clause 8.3.2.1.

18. The Selection Controller notify the AN Orchestrator that it has completed the selection process
19. The AN Orchestrator request the Selection Controller to update the Knowledge Base with the controller to be deployed as the Operational Controller
20. The Selection Controller will update the Knowledge Base
21. The AN Orchestrator will perform the necessary lifecycle actions to deploy the Operational Controller for the use case.

10. Security Considerations

This Recommendation describes the Autonomous Networks architectural framework which is expected to be applied in future networks including IMT-2020: therefore, general network security requirements and mechanisms in IP-based networks should be applied [ITU-T Y.2701] [ITU-T Y.3101].

Sensitive information should be protected as a high priority in order to avoid leaking and unauthorized access.

Additional specific security considerations concern autonomous networks security evaluation (e.g., analyzing the characteristics of autonomous networks to evaluate risk of evasion attack). Moreover, to ensure robust autonomous networks, the reliability of the exploratory evolution, real-time online experimentation and dynamic adaptation and the generated controllers needs to be assessed before applying the controllers to the network.

Appendix I: An example realisation of the architecture framework for Autonomous Networks with technology specific underlays

Figure I.1 gives an example of the Architecture Framework for Autonomous Networks (AN architecture framework) with an IMT-2020 network [ITU-T Y.3111] [ITU-T Y.3104] underlay.

NOTE – A simplified version of Figure 1 is shown in Figure I.1, however, all functionalities described in clause 8 are applicable.

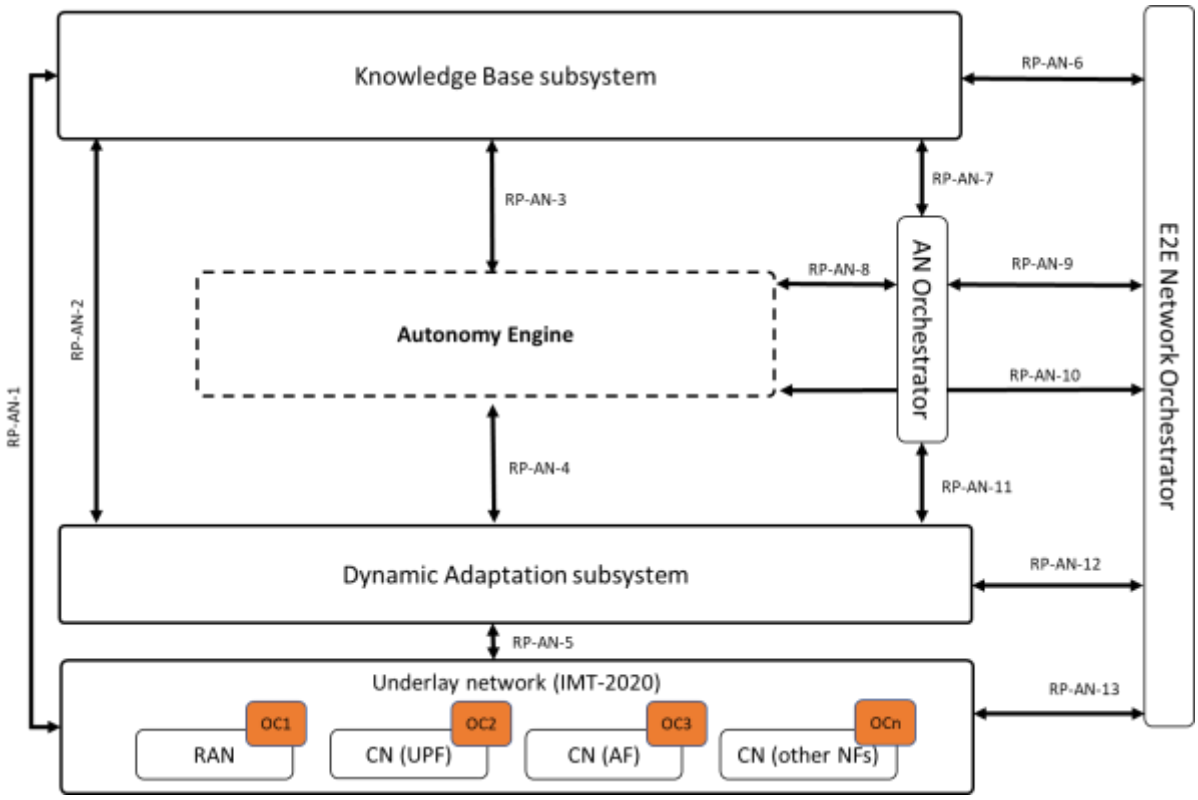

Figure II.1 Example of a realisation of the AN architecture framework with an IMT-2020 network underlay

In this example realisation, operation controllers (OC) are instantiated in the IMT-2020 underlay network. As described in clause 8.3.2.2, OCs are responsible for the operation of a managed entity, including the analysis of data from the managed entity. Additionally, OCs are continuously monitored by the dynamic adaptation subsystem to ensure the most effective operation of the managed entity.

OCs are integrated with various IMT-2020 network functions and/or application functions. The AN architecture framework acts as an overlay providing the subsystems such as evolutionary exploration (clause 8.3.1.1), online experimentation (clause 8.3.1.2), dynamic adaptation (clause 8.3.2) and knowledge base subsystem (clause 8.3.3).

I.1. Examples of deployment locations of controllers

As shown in Figure I.1, various OCs may be deployed in various parts of the IMT-2020 underlay network. Examples are the following:

- OC1: This controller is deployed in the access network and uses inputs from the access network to make determinations about capacity coverage optimization and then applies them to the management functions.
- OC2: This controller is deployed in the CN (UPF) and uses inputs from the CN to make determinations about packet forwarding and then applies them to the management functions.
- OC3: This controller is deployed in the CN (AF) and uses inputs from the CN to make determinations about AR/VR and then reports findings to a vertical application;

- OCn: This controller is deployed in the CN (other NFs) and uses inputs from the CN to make determinations about traffic load balancing and then applies them to the management functions.

For all examples, the controller deployment locations may be specified in the controller specification.

The dynamic adaptation subsystem interfaces with the AN orchestrator to deploy the OCs to the underlay via the E2E Orchestrator. This can be realised via reuse of reference points defined in [ITU Y.3111].

NOTE 1 – The dynamic adaptation subsystem applies the criteria for dynamic adaptation of an OC (e.g., instantiation or replacement) as discussed in clause 8.3.2.

NOTE 2 – While the deployment of an operational controller on a UE [ITU-T Y.3104] may be technically possible, the associated operational challenges, such as data privacy or security concerns, are out with the scope of this Recommendation. Examples of such an operation controller may include one tasked with monitoring the state of a service in the UE as input to end to end decision making (e.g., QoE measurements for media streaming services).

I.2. Example realisation of Exploratory Evolution

An example of a process to apply exploratory evolution to OCs deployed in various parts of the IMT-2020 underlay network is the following:

as shown in steps 5 to 10 of Figure 6 (clause 9.1.1), based on data stored in the knowledge base collected from the underlay network, deployed controllers or experimentation, the Evolution Controller (clause 8.3.1.1) will generate new controller designs for OC1, OC2, OC3, or OCn. These controller designs are then stored in the knowledge base.

I.3. Example realisation of Online Experimentation

An example of a process to apply online experimentation to OCs deployed in various parts of the IMT-2020 underlay network is the following:

as shown in steps 6 to 9 of Figure 7 (clause 9.1.2), based on data stored in the knowledge base regarding the status and configuration of the access network in the underlay (steps 6 & 7 of Figure 8), the Experimentation Controller (clause 8.3.1.2) will generate possible experimental scenarios for OC1 (step 8 of Figure 7).

I.4. Example realisation of Dynamic Adaptation

An example of a process to apply dynamic adaptation to OCs deployed in various parts of the IMT-2020 underlay network is the following:

as shown in steps 15 to 21 of Figure 8 (clause 9.1.3), based on data stored in the knowledge base regarding the status and configuration of the core network in the underlay, the status and configuration of currently deployed OC2, and information regarding other relevant curated controllers (steps 15 & 16 of Figure 8), the Adaptation Controller (clause 8.3.2.1) will decide which curated controller to be deployed in the underlay (step 17 of Figure 8) as the new revision of OC2 (step 21 of Figure 8).

Appendix II: Self-reflective use of the AN architecture

The AN architecture framework shown in Figure 1 is used for creating/adapting controllers, validating controllers, and applying controllers to a managed entity. Despite having different roles, from a compositional perspective, the exploratory evolution, experimentation and dynamic adaptation can be applied to these controllers.

The architecture is self-reflective in its operation i.e. architecture has the ability to modify its own operation to more effectively adapt to the current operational situation without the involvement of the human using the same processes as managed entities, as shown in Fig II.1. Thus, the architecture itself becomes a collection of managed entities.

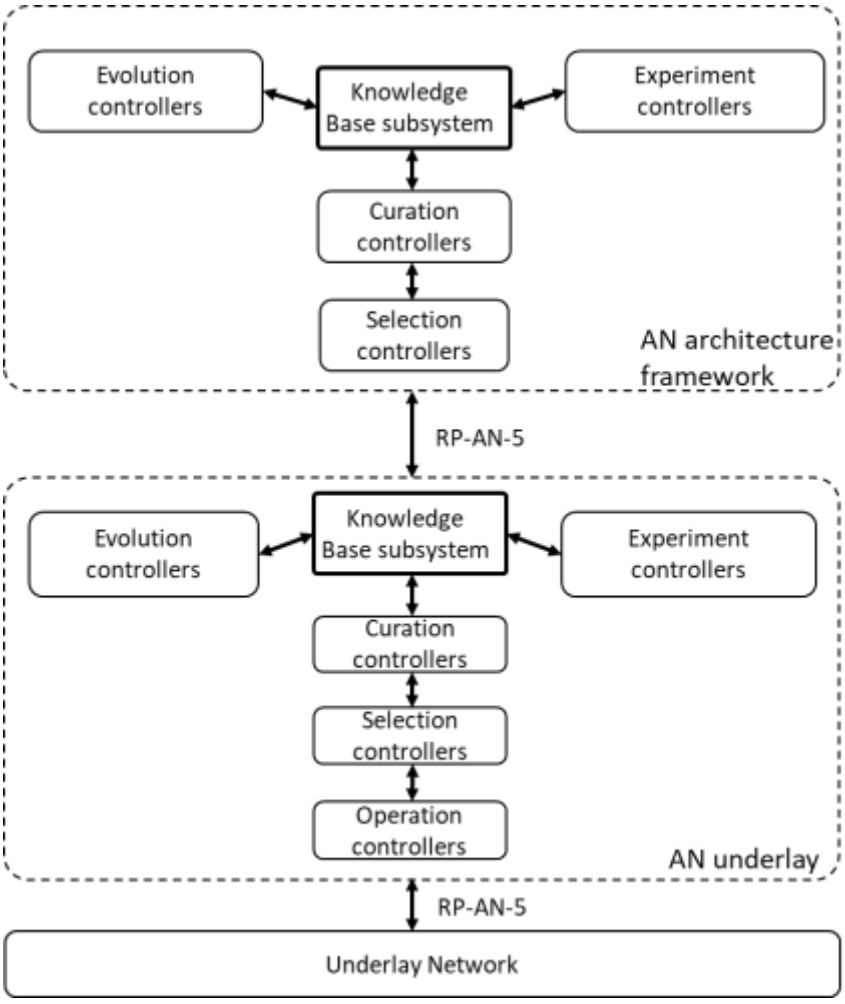

Figure II.1: Self-Reflective use of the AN architecture framework

NOTE 1 - Even though Fig II.1 shows only a general application of the AN architecture framework to itself, specific instances are possible where an operational controller, an evolution controller, an experimentation controller, a selection controller, and/or a curation controller are the managed entity.

NOTE 2 - Fig II.1 refers to AN underlay. This concept refers to an instance of AN architecture framework (or a subset of it) used, in turn, as an underlay of another instance of AN architecture framework.

Appendix III: External functionalities

In addition to the architecture components, there are functionalities external to this architecture framework, which may enhance the AN architecture. These external functionalities are provided by existing implementations.

Examples may include:

- AN Controller repositories [b-dagsthul]: These are repositories which contain controllers and modules. AN components (e.g. evolution controller) may access these repositories to implement extended functionalities, e.g. composing new controllers.
- External knowledge repositories: In addition to knowledge bases implemented within the AN architecture, there are external knowledge repositories used by AN architecture to access such knowledge.
- Domain orchestrators: These may be implemented by third parties and may provide specific functions associated with specific technologies such [b-ETSI GS ZSM 009-1, b-FRINX, b-ONAP].
- Development pipelines (CI/CD pipelines): They provide continuous development environment for components, modules and controllers.
- Model repositories [ITU-T Y.3176]: They store the specifications of models used in the AN.

NOTE - Examples of types of models which are stored in the model repositories include:

- Models used by a controller: these are models either placed within a controller or is accessed by the controller, via an API exposed by a third party.
- Models used by an AN component: these are models either placed within a AN component such as AN orchestrator, or is accessed by the component, via an API exposed by a third party.

Bibliography

[b-ITU-T Y.3525] ITU-T Recommendation Y.3525, “Cloud computing – Requirements for cloud service development and operation management”

[b-ITU-JFET] ITU Journal on Future and Evolving Technologies, Blessed et al, “Network resource allocation for emergency management based on closed-loop analysis"

[b-ITU-T Y.Supp 55] ITU-T Supplement 55 to ITU-T Y.3170-series Recommendations (2019), Machine learning in future networks including IMT-2020: Use cases

[b-ITU-T Y.Supp 71] ITU-T Supplement 71 to ITU-T Y-3000 series Recommendations (2022), Use cases for Autonomous Networks

[b-Acumos-DCAE] Acumos DCAE Integration, https://wiki.onap.org/display/DW/Acumos+DCAE+Integration

[b-AUTOML] Google’s AutoML tool, https://cloud.google.com/automl

[b-AUTOML-ZERO] Real, E., Liang, C., So, D. and Le, Q., 2020, November. Automl-zero: Evolving machine learning algorithms from scratch. In International Conference on Machine Learning (pp. 8007-8019). PMLR.

[b-bayesian-radio] L. Maggi, A. Valcarce and J. Hoydis, "Bayesian Optimization for Radio Resource Management: Open Loop Power Control," in IEEE Journal on Selected Areas in Communications, vol. 39, no. 7, pp. 1858-1871, July 2021, doi: 10.1109/JSAC.2021.3078490.

[b-capacity-allocation] D. Bega, M. Gramaglia, M. Fiore, A. Banchs and X. Costa-Perez, "AZTEC: Anticipatory Capacity Allocation for Zero-Touch Network Slicing," IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, 2020, pp. 794-803, doi: 10.1109/INFOCOM41043.2020.9155299.

[b-CDNSim] K. Stamos, G. Pallis, A. Vakali: “Integrating Caching Techniques on a Content Distribution Network”. In Proceedings of the 10th East-European Conference on Advances in Databases and Information Systems, LNCS series of Springer Verlag, Thessaloniki, Greece, September 2006

[b-Chaos Engineering] Kazuyuki Aihara and Ryu Katayama. 1995. Chaos engineering in Japan. Commun. ACM 38, 11 (Nov. 1995), 103–107. DOI:https://doi.org/10.1145/219717.219801

[b-data-fusion] Jens Bleiholder and Felix Naumann. 2009. Data fusion. ACM Comput. Surv. 41, 1, Article 1 (January 2009), 41 pages. https://doi.org/10.1145/1456650.1456651

[b-dagsthul] The Dagsthul Artefact Repository, https://drops.dagstuhl.de/opus/institut_darts.php

[b-Digital-twin] P. Almasan et al., "Network Digital Twin: Context, Enabling Technologies and Opportunities," in IEEE Communications Magazine, doi: 10.1109/MCOM.001.2200012.

[b-evolution] Whitley, Darrell. "An overview of evolutionary algorithms: practical issues and common pitfalls." Information and software technology 43.14 (2001): 817-831.

[b-ETSI GS ZSM 009-1] ETSI GS ZSM 009-1 V1.1.1 (2021-06) Zero-touch network and Service Management (ZSM); Closed-Loop Automation; Part 1: Enablers

[b-ETSI-AN] ETSI Whitepaper, Autonomous Networks, supporting tomorrow's ICT business, October 2020

[b-ETSI-GS-ENI-002] Experiential Networked Intelligence (ENI); ENI requirements

[b-ETSI GS ENI 005] ETSI GS ENI 005 V2.1.1 (2021-12) “Experiential Networked Intelligence (ENI); System Architecture”

[b-ETSI GS MEC 012] ETSI GS MEC 012 V2.2.1 (2022-02) **“**Multi-access Edge Computing (MEC); Radio Network Information API”

[b-ETSI TS 129 500] ETSI TS 129 500 V15.0.0 (2018-07) 5G; 5G System; Technical Realization of Service Based Architecture; Stage 3 (3GPP TS 29.500 version 15.0.0 Release 15).

[b-FRINX] FRINX Machine, https://github.com/FRINXio/FRINX-machine

[b-game-theory] Ahmad, I., Kaleem, Z., Narmeen, R., Nguyen, L.D. and Ha, D.B., 2019. Quality-of-service aware game theory-based uplink power control for 5G heterogeneous networks. Mobile Networks and Applications, 24(2), pp.556-563.

[b-Huebscher 2008] Huebscher, C. and McCann, A. (2008) A Survey of Autonomic Computing Degrees, Models, and Applications. ACM Computer Survey, 40, Article No. 7. http://dx.doi.org/10.1145/1380584.1380585

[b-Kephart 2003] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” *Computer (Long. Beach. Calif).*, vol. 36, no. 1, pp. 41–50, 2003.

[b-Knowledge Graph] ITU AI/ML in 5G Challenge —” Applying knowledge graph and digital twin technologies to smart optical network”. Online presentation.

[b-Kubernetes] Kubernetes, Production-Grade Container Orchestration, https://kubernetes.io/docs/concepts/overview/

[b-large-evolution] Damien Anderson, Paul Harvey, Yusaku Kaneta, Petros Papadopoulos, Philip Rodgers, and Marc Roper. 2022. Towards evolution-based autonomy in large-scale systems. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22). Association for Computing Machinery, New York, NY, USA, 1924–1925.

[b-LogicNets] Umuroglu et al. "LogicNets: Co-Designed Neural Networks and Circuits for Extreme-Throughput Applications, https://arxiv.org/abs/2004.03021

[b-Mwanje 2020] Mwanje SS, Mannweiler C, editors. Towards Cognitive Autonomous Networks: Network Management Automation for 5G and Beyond. John Wiley & Sons; 2020 Oct 12.

[b-NGMN-5G] NGMN 5G White Paper

[b-NMRG] The Internet Research Task Force (IRTF), Network Management Research Group NMRG, https://irtf.org/nmrg

[b-ns3] NS-3 Network Simulator, https://www.nsnam.org/

[b-OASIS TOSCA-v1.3] TOSCA Simple Profile in YAML Version 1.3. https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-Profile-YAML-v1.3-os.pdf.

[b-OODA] J. Boyd, G.T. Hammond, and Air University (U.S.). Press. A Discourse on Winning and Losing. Air University Press, Curtis E. LeMay Center for Doctrine Development and Education, 2018.

[b-ONAP] ONAP, Open Network Automation Platform, https://www.onap.org/about

[b-ONF] Open Network Foundation Mobile Network Projects, https://opennetworking.org/onf-mobile-projects/

[b-ORAN] The O-RAN Whitepaper 2022 (RAN Intelligent Controller), Rimedo Labs, https://rimedolabs.com/blog/the-oran-whitepaper-2022-ran-intelligent-controller/

[b-OSM] OSM, Open Source MANO, https://osm.etsi.org/docs/user-guide/latest/01-quickstart.html

[b-PID] S. Bennett, "Development of the PID controller," in *IEEE Control Systems Magazine*, vol. 13, no. 6, pp. 58-62, Dec. 1993, doi: 10.1109/37.248006.

[b-Rossi 2020] F. Rossi, V. Cardellini, and F. Lo Presti, “Hierarchical Scaling of Microservices in Kubernetes,” *2020 IEEE Int. Conf. Auton. Comput. Self-Organizing Syst.*, pp. 28–37, Aug. 2020.

[b-RL] Sutton, R.S. and Barto, A.G., 2018. Reinforcement learning: An introduction. MIT press.

[b-TMFORUM-AN-WP] Autonomous Networks: Empowering Digital Transformation For The Telecoms Industry, whitepaper, May 2019.

[b-WEF-DTI] Digital Transformation Initiative Telecommunications Industry, whitepaper by the World Economic Forum

[b-3GPP TR 28.810] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Study on concept, requirements and solutions for levels of autonomous network (Release 16)

[b-3GPP TS 38.801] 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Study on new radio access technology: Radio access architecture and interfaces (Release 14)

_____