ITUPublications

International Telecommunication Union

Standardization Sector

ITU Focus Group Technical Specification

(09/2022)

Focus Group on Autonomous Networks

(FG-AN)

Architecture framework for autonomous networks

Technical Specification ITU FG-AN

Architecture framework for autonomous networks

Summary

This Technical Specification provides an architecture framework for autonomous networks. The scope of this document includes:

– Requirements for the architecture.

– Description of the architecture components and subsystems.

– Description of the architecture.

– Sequence diagrams explaining the interactions between the architecture components.

Keywords

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

Note

This is an informative ITU-T publication. Mandatory provisions, such as those found in ITU-T Recommendations, are outside the scope of this publication. This publication should only be referenced bibliographically in ITU-T Recommendations.

Contributors:

Paul HARVEY
University of Glasgow
United Kingdom
Email: paul.harvey@glasgow.ac.uk

Leon WONG
Rakuten Mobile
Japan
Email: leon.wong@rakuten.com

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

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

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

This Technical Specification has been published as approved by the focus group, without any subsequent editorial review.

© ITU 2026

Some rights reserved. This publication is available under the Creative Commons Attribution-Non Commercial-Share Alike 3.0 IGO licence (CC BY-NC-SA 3.0 IGO; https://creativecommons.org/licenses/by-nc-sa/3.0/igo).

If you wish to reuse material from this publication that is attributed to a third party, such as tables, figures or images, it is your responsibility to determine whether permission is needed for that reuse and to obtain permission from the copyright holder. The risk of claims resulting from infringement of any third-party owned material in the publication rests solely with the user.

Table of Contents

Technical Specification ITU FG-AN

Architecture Framework for autonomous networks

1 Scope

This Technical Specification provides an architecture framework for autonomous networks. The scope of this Technical Specification includes:

– Requirements for the architecture.

– Description of the architecture components and subsystems.

– Description of the architecture.

– Sequence diagrams explaining the interactions between the architecture components.

2 References

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

[ITU-T Y.3115] Recommendation ITU-T Y.3115 (2022), *AI enabled cross-domain network architectural requirements and framework for future networks including IMT-2020.*

[ITU-T Y.3176] Recommendation ITU-T Y.3176 (2020), *Machine learning marketplace integration in future networks including IMT-2020.*

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

[ITU-T Y.3320] Recommendation ITU-T Y.3320 (2014), *Requirements for applying formal methods to software-defined networking.*

[ITU-T Y.3525] Recommendation ITU-T Y.3525 (2020), *Cloud computing – Requirements for cloud service development and operation management.*

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

3 Definitions

3.1 Terms defined elsewhere

This Technical Specification uses the following term 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.

3.2 Terms defined in this Technical Specification

This Technical Specification defines the following terms:

3.2.1 adaptation controller: A controller responsible for selecting candidate controllers from a set of generated controller configurations which are ready for integration, executes the integration to the underlay.

NOTE – 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).

3.2.2 AN sandbox: 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.

3.2.3 autonomy engine: A collection of subsystems where trial and error process is applied on controllers to generate new candidate controllers and validate them.

NOTE – For example, the grouping of the evolutionary exploration subsystem and the real-time responsive online experimentation subsystem together forms Autonomy Engine.

3.2.4 controller: 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 – Examples of system under control are managed entities, workflows and/or processes in an IMT-2020 network.

NOTE 2 – Exploratory evolution and real-time responsive online experimentation are examples of processes in independent development of modules or closed loops.

NOTE 3 – Modules may themselves be workflows, open loops, or closed loops.

3.2.5 controller design: A low-level, non-executable representation of 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 necessary functionality of the controller and a utility function to be achieved.

3.2.7 evolution controller: A controller responsible for 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 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.

3.2.9 knowledge base: 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.

NOTE 2 – 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 3 – 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.

3.2.10 managed entity: A resource, service, closed loop or controller which is managed.

NOTE – This term differs from other definitions [b-ETSI GS ZSM 009-1, b-Huebscher 2008] as a controller may also be managed, similarly, to closed loops, managed resources e.g., VNFs, PNFs) and managed services (e.g., cloud services).

4 Abbreviations and acronyms

This Technical Specification 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
RCA	Root Cause Analysis
RIC	RAN Intelligence Controller
SDK	Software Development Kit
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 Technical Specification:

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

6 Introduction

With the strong progression of digital transformation in telecommunication networks enabling the practical and industrial realisation of software mechanisms to manage the network [b-WEF-DTI], recent focus has turned towards the processes of control that will drive these mechanisms.

The ultimate form of such control is known as autonomous networks. These 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]. Indeed, the interest in autonomous networks is currently so strong that 3GPP [b-3GPP TR 28.810], ETSI [b-ETSI GS ENI 002, b-ETSI-AN], TMForum [b-TMFORUM-AN-WP], and the IRTF [b-NMRG] all have historic or ongoing initiatives under the heading of autonomy.

Many of these efforts manifest themselves in the application of various machine learning (ML) approaches to a single or a set of targeted use cases with the aim of automating the operation or management, reducing cost, optimising the resources used, or automatically detecting or predicting unusual situations or circumstances.

One common problem in the application of ML to these use cases is the problem of model drift. Model drift is the phenomenon whereby either the goal of the ML model changes over time (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 enabling technology amongst many that are required to achieve the autonomous operation of the network: new software and hardware technology is created, updated, deprecated [b-Acumos-DCAE]; new services are introduced to the network and new ways of using the networks are emerging all the time [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 a de facto standard in the approach taken to 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, adaptive, hierarchical [b-Rossi 2020, b-ITU-JFET].

Irrespective of the grouping 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 of being bound by the purpose for which it was designed, even in the cases when that purpose includes some dynamic reflective 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 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 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.

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 this architecture. This partition of concerns 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" [ITU-T Y.3177].

The main concepts behind autonomous networks which are elaborated here are exploratory evolution, real-time responsive online experimentation and dynamic adaptation. To study and analyse use cases along these concepts in networks, a basic building block called "controller" is introduced. Controllers are used in the use cases to further elaborate autonomous networks and the key concepts required to enable them.

The concept of exploratory evolution introduces the mechanisms and processes of exploration and evolution to adapt a controller 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.

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.

NOTE – Real-time responsive online experimentation is called "experimentation" in this document.

7 Requirements for the architecture

This clause describes the requirements for the architecture framework.

NOTE 1 – See Appendix I for a mapping of these requirements to the components in clause 8 and use cases in [ITU-T Y.Supp 71].

Requirement	Description
AN-arch-req-001	It is required that AN parses, validates and translates an abstracted use case description, with high level objectives of a controller (use case) into a controller description. NOTE 2 – The abstracted use case description may be hand-crafted as unstructured text or derived from "software modules specifications". NOTE 3 – Controller description may use structured languages formats e.g., TOSCA and may have a structure which facilitates downstream exploration, experimentation, and adaptation. NOTE 4 – Controller description may use/enable properties derived from various domains in the network e.g., properties to describe physical layer, network layer and application layer use cases. NOTE 5 – Examples of use cases are: a) Root cause analysis and diagnosis of network elements based on real-time analysis of data. 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. c) Optimal adjustment of antenna parameters with AI capabilities of multi-dimensional analysis and prediction. d) Management of industry vertical applications and related services in the network.
AN-arch-req-002	It is required that AN validates and processes the controller description, so that exploratory evolution can be applied on the controller descriptions. NOTE 6 – Exploratory evolution may include the following: interconnecting of descriptions together to form complex controller descriptions. NOTE 7 – Exploratory evolution may result in a list of evolvable controllers. NOTE 8 – Exploratory evolution may be a continuous process.
AN-arch-req-003	It is required that AN manages knowledge related to autonomous networks. NOTE 9 – Managing knowledge includes storing, querying, export, import and optimize the knowledge.
AN-arch-req-004	It is required that AN enables integration and/or plugin of various algorithms including 3rd party provided algorithms, machine learning or artificial intelligence models, into controllers.
AN-arch-req-005	It is optional that AN generates and/or updates the controller description.
AN-arch-req-011	It is required that AN enables update of knowledge based on various procedures in the AN.
AN-arch-req-006	It is required that AN generate the potential scenarios of evolution, taking the controller description as input. NOTE 10 – Specific mechanisms or algorithms used for evolution may be out of scope of this document.

Requirement	Description
AN-arch-req-007	It is required that AN executes the potential scenarios of evolution, taking the controller description as input and collates the output in the form of evolvable controller descriptions. NOTE 11 – Several epochs of evolutions may be applied on the same set of controller descriptions.
AN-arch-req-008	It is required that AN generate the potential scenarios of experimentation, taking the controller description as input. NOTE 12 – Specific configurations and "limits" of experiments may be specified in the "metadata" and "constraints" related to the controller description. NOTE 13 – Specific mechanisms for arriving at the scenarios for experimentation may use AI/ML or other forms of analytics and are out of scope of this document.
AN-arch-req-009	It is required that AN executes the experimentation, collates and validates the results of the experimentation, considering the metadata and constraints and corresponding controller descriptions. NOTE 14 – Experimentation may have several phases for example, simulation driven, testbed driven or canary test driven. The phases of experimentation may be configurable and automated, for example, as per a workflow. NOTE 15 – The specific success/failure criteria for the experiments may be out of scope of this document, however, an acceptable format for representing such metadata and constraints related to potential success/failure as related to a use case may be studied.
AN-arch-req-010	It is required that AN enables integration of various forms of testing components including simulator and data generators, including those provided by third party providers.
AN-arch-req-012	It is required that AN generate the potential configurations for integration of controllers to specific underlays. NOTE 16 – Configurations may include reference points, API formats, data models, etc. This generation of configurations may take the controller description as input along with the description or metadata related to the underlays. NOTE 17 – While the specific configurations for integration of controllers for specific use cases may be out of scope of this document, an acceptable format for representing such configurations may be studied. NOTE 18 – Example of underlays are edge networks, core networks, management plane, CI/CD pipelines, etc.
AN-arch-req-013	It is required that, before integration of controllers into underlays, AN "just in time" verifies that the evolution (applied so far to the controller) has not made it incompatible for integration with the underlay.
AN-arch-req-014	It is required that AN selects the candidate controllers from a set of controllers which are ready for integration and executes the integration of controllers to specific underlays, taking as input the generated configurations for integration. NOTE 19 – The specific criteria for selecting the controllers for integration may be out of scope of this document, however an accepted format for representing such criteria may be studied.

Requirement	Description
	NOTE 20 – The specific mechanisms used for integration of controllers to underlays are out of scope. Examples of such mechanisms are service based architectures [b-ETSI TS 129 500] and continuous integration mechanisms [ITU-T Y.3525]. NOTE 21 – Example of ways of integration of controllers to specific underlays are (a) autonomous network components act as a ZSM service consumer to achieve some AN use cases or (b) mapping of AN components to ZSM subsystems. Or (c) add new components to ZSM subsystems to achieve AN use cases. (d) ZSM acts as an underlay, AN components act upon ZSM subsystems. NOTE 22 – Examples of specific underlays are various domains in the network.
AN-arch-req-015	It is required that AN manages the points of exchange of metadata in the AN workflow, with a peer entity. NOTE 23 – Examples of metadata regarding the 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 workflow during the AN, to capture metadata regarding the workflow which can then be exchanged. NOTE 24 – Examples of points of exchange of metadata in the AN workflow are different stages of experimentation, adaptation. NOTE 25 – The format of exchange is out of scope of this document. NOTE 26 – Examples of workflow during the AN include generation of the potential scenarios of experimentation and the generation of potential scenarios of evolution. NOTE 27 – Peers may include humans and machines.
AN-arch-req-016	It is required that AN, based on the exchange of metadata in the AN workflow, with a peer entity, integrate the impacts of such exchanges into the AN. NOTE 28 – Examples of impacts of the exchange of metadata are update of knowledge base and selection of API versions to use for adaptation.
AN-arch-req-017	It is required that AN manages the lifecycle of controllers. NOTE 29 – Examples of managing the lifecycle of controllers include creating a configuration for the controller (based on the capabilities of the underlay), creating an instance of the controller in the underlay, monitoring the execution of the controller in the underlay and subsequent optimization of the controllers or related parameters. NOTE 30 – Examples of subsequent optimization are recommendations on new AI/ML analysis techniques, data collection techniques, evolution to move up the intelligence level [b-ITU-T Y.3173].
AN-arch-req-018	It is required that AN decides new opportunities for deployment of controllers in underlays.
AN-arch-req-019	It is required that AN decides configuration of controllers which are to be deployed in underlays.
AN-arch-req-020	It is required that AN enables monitoring of controllers which are already deployed in underlays.
AN-arch-req-021	It is required that AN enables reporting and monitoring of AN components and procedures by humans and/or other automation mechanisms.

Requirement	Description
AN-arch-req-022	It is required that AN utilizes supporting components like stored controllers and knowledge in the AN to deploy and manage controllers in underlays.
AN-arch-req-0xx	It is required that AN discovers and consumes services provided by service management frameworks. NOTE 31 – Examples of service management frameworks are ONAP, OSM.
AN-arch-req-0xx	It is optional that AN influence the services provided by service management frameworks. NOTE 32 – Examples of influence affected by AN upon services provided by service management frameworks are passing policies and intents to service management frameworks. Service management frameworks may be used to design, deploy new and/or modified services.
AN-arch-req-023	It is required that AN optimizes the controllers. NOTE 33 – 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. Other examples are formation or evolution of new controllers to address new or unforeseen problems in the network.
AN-arch-req-024	It is required that AN discovers the interfaces with underlays used for integration. NOTE 34 – Interfaces with underlays may use specific APIs for e.g., APIs for data collection, configuration of underlay functions etc. Discovery of interfaces may include API metadata including parameters, versions, range of parameters etc.
AN-arch-req-025	It is required that, for a given use case, AN discovers the underlay specific parameters open for optimization, data points for collection in the underlay and the relevant KPIs for tracking. NOTE 35 – For example, in edge deployments, underlays may use MEC (multi access edge computing) APIs.
AN-arch-req-026	It is required that AN discovers the characteristics of controllers which are relevant for enabling evolution. NOTE 36 – 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-req-027	It is required that AN recommends modules which can satisfy the characteristics of controllers. NOTE 37 – Controllers may be composed of one or many modules.
AN-arch-req-028	It is required that AN enables the management of lifecycle of controllers based on output of controllers. NOTE 38 – Examples are management of lifecycle of controllers in a lower domain (such as RAN), by controllers in a higher domain (e.g., CN), creation and optimal positioning of controllers in the RAN by controllers in a higher domain, evolution, experimentation, deployment, of controllers. NOTE 39 – Complex AN behaviours may be achieved by using such cross-domain "chaining" of controllers.

Requirement	Description
AN-arch-req-029	It is required that AN enables integration of controllers from different domains to achieve complex use case NOTE 40 – For example, AN may integrate controllers in different domains and levels of the network, like RAN and Core network domains.
AN-arch-req-030	It is required that AN enables the storage and management of supporting artifacts for the lifecycle management of controllers. NOTE 41 – 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 42 – Examples of management of supporting artifacts are storage of knowledge in knowledge base, create, modify, delete, storage of AI/ML models in ML model repository [ITU-T Y.3176] and policies, query, discovery of various artifacts, etc.
AN-arch-req-031	It is required that AN customizes the integration of controllers in underlays, considering the integration options exposed by the underlay. NOTE 43 – Examples of integration options are interfaces, parameters and configurations exposed by the underlay. Examples of underlays are industry vertical applications and networks.
AN-arch-req-032	It is required that AN automates and abstracts the evolution of underlay network services, from overlay service providers. NOTE 44 – Examples of evolution of underlay network services include updates to support new features, migration to new service platforms and technologies. NOTE 45 – Service provider may monitor the evolution (see AN-UC03-REQ-001-Component-req-00a above) but does not manually execute the evolution of underlay network services.
AN-arch-req-033	It is required that AN utilizes the declarative specification of use case while deciding the design, deployment and management of controllers.
AN-arch-req-034	It is required that AN utilizes the declarative specification of use case to capture both use case requirements from vertical applications and deployment requirements from underlays.
AN-arch-req-035	It is required that AN automates and abstracts the experimentation of underlay network services, from overlay service providers.
AN-arch-req-036	It is required that AN chooses the compatible set of interfaces to integrate with underlay network services. NOTE 46 – E.g., interfaces may be used to monitor the services and controllers in the underlay networks.
AN-arch-req-037	It is required that AN produces human and machine readable reports of periodic or aperiodic nature.
AN-arch-req-038	It is required that AN discovers service automation frameworks used by underlays. NOTE 47 – This may help in managing and automating the lifecycle of underlay services by the AN.
AN-arch-req-039	It is required that AN consumes services exposed by service automation frameworks used by underlays. NOTE 48 – Services exposed by service automation frameworks include configuration, lifecycle management and customizations. NOTE 49 – Examples of service management frameworks include ETSI ZSM framework.

Requirement	Description
AN-arch-req-040	It is required that AN adapts to evolution of underlay network services, which may be done independently to the evolution of AN. NOTE 50 – E.g., the evolution of underlay network services may be managed by different entity than the one managing the evolution of AN.
AN-arch-req-041	It is required that AN creates and/or recommends candidate designs for potential network services and interfaces in the underlay, which may possibly satisfy new use cases. NOTE 51 – This (design creation) may also be in response to an observed fault in the underlay.
AN-arch-req-042	In an end to end service delivery deployment, it is required that AN integrates with evolution of various generations of connectivity options between various subsystems of the underlay. NOTE 52 – E.g., access network may be using various different types of technologies and they may evolve over a period of time.
AN-arch-req-043	It is required that AN utilizes heterogeneous connectivity options provided by the underlay to deploy and integrate controllers in different levels of the underlay.
AN-arch-req-044	It is required that AN considers the use case specific requirements (including operator preferences) while deciding the operator preference for design, deployment, management of controllers, including connectivity options between various domains. NOTE 53 – For example in rural areas, end-users may have usage patterns characteristic to certain applications (e.g., low mobility, high bandwidth). Choice of last mile connectivity options may be influenced by such preferences.
AN-arch-req-045	It is required that AN monitors the changes to connectivity provided by the underlay between controllers deployed in various domains. NOTE 54 – Thus, inter-domain connectivity which connects various controllers, forms another underlay for AN.
AN-arch-req-046	It is required that AN enables evolution of inter-domain connectivity which connects controllers in various domains of the underlay.
AN-arch-req-047	It is required that AN adapts to the evolution of vertical applications at run-time. NOTE 55 – E.g., of adapting to evolution of vertical applications at run-time is onboarding new applications or changes in existing applications deployed by service providers in the network.
AN-arch-req-048	It is required that AN adapts to changes in the external systems that it interfaces with. NOTE 56 – Examples of external systems are various management and orchestration systems, policies and corresponding management systems, workflow management systems, user management systems, which may be deployed from multiple vendors by the operator.
AN-arch-req-049	It is required that AN exposes single point of monitoring and managing AN functionality deployed by the operator.

Requirement	Description
AN-arch-req-050	It is required that AN monitors the dynamic changes in capabilities of monitoring, configuration and analysis of parameters from underlay networks. NOTE 57 – Based on the independent evolution of controllers and the underlay and the applications deployed by verticals, mechanisms such as discovery, publishing, subscription mechanisms may be used to provide flexibility in monitoring parameters.
AN-arch-req-051	It is required that AN utilizes the dynamic changes in capabilities of monitoring, configuration and analysis of parameters from underlay networks.
AN-arch-req-052	It is required that AN recommends changes in capabilities of monitoring, configuration and analysis of parameters from underlay networks.
AN-arch-req-053	It is required that AN discovers the changes to network functions in the underlay and includes those changed functions while providing the AN functionalities like evolution. NOTE 58 – This enables plug and play of new NFs in the underlay.
AN-arch-req-054	It is required that AN provides inputs to external systems regarding potential scenarios and requirements. NOTE 59 – Examples of potential inputs are reports generated from AN on new use case scenarios, experimentation scenarios.
AN-arch-req-055	It is optional that AN recommends new capabilities and requirements of NF in underlays.
AN-arch-req-056	It is required that AN discovers deployed controllers in the underlay, including those deployed by third party providers.
AN-arch-req-057	It is required that AN enables deployment of controllers which utilizes both simulated and real networks as underlays. NOTE 60 – In case of controllers which span across network domains, this may involve some modules of the controller integrated with simulated underlays.
AN-arch-req-058	It is required that AN enables analysis and correlation of domain specific, unstructured data in natural languages from the underlays NOTE 61 – Examples of domain specific unstructured data in natural languages are logs from NFs. NOTE 62 – Advances in analysis of natural language text may be exploited from third party models and repositories.
AN-arch-req-059	It is required that AN derives knowledge from analysis and correlation of domain specific, unstructured data in natural languages from the underlays.
AN-arch-req-060	It is optional that AN enables correlation of declarative specification of network services with that of controllers. NOTE 63 – Examples of correlation of specifications of controllers with that of network services include mapping of interfaces, capabilities and requirements. Other examples are identifying opportunities for deriving detailed specifications. E.g., using substitution mechanisms in TOSCA.
AN-arch-req-060	It is optional that AN enables correlation of declarative specification of network services with that of controllers and use that correlation to integrate controllers at different levels of the network.

Requirement	Description
	NOTE 64 – Examples of correlation of specifications of controllers with that of network services include mapping of interfaces, capabilities and requirements. Other examples are identifying opportunities for deriving detailed specifications. E.g., using substitution mechanisms in TOSCA. NOTE 65 – Examples of different levels of the network are RAN, Core Network.
AN-arch-req-061	It is required that AN supports import and export of controller specifications at various stages of their management. NOTE 66 – Examples of various stages of management of controller specifications are before and after exploratory evolution.
AN-arch-req-062	It is required that AN supports flexible learning of derivations of metrics from collected parameters/measurements. NOTE 67 – Derivation of metrics may use AI/ML techniques. Derivation mechanisms/algorithms 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-req-062	It is required that AN supports integration of third party provided derivation mechanisms for metrics from collected parameters/measurements. NOTE 68 – Example of metrics derived are QoE, example of measurements collected are QoS parameters.
AN-arch-req-063	It is required that AN supports capturing both high level service KPI requirements as well as deployment preferences and considerations in the intent.
AN-arch-req-064	It is required that AN supports derivation of requirements for service life cycle management in underlays based on the intent derived from different levels in the network. NOTE 69 – For example, closed loops for optimization in RAN may be configured based on derived intents from the CN. NOTE 70 – Service life cycle management includes resource allocation, scaling, optimization etc.
AN-arch-req-065	It is required that AN supports optimization of intents based on monitoring the performance of derived closed loops deployed in various levels of the network and their life cycles. NOTE 71 – 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-req-066	It is required that AN utilizes the application development pipelines to automate the design and instantiation of underlay network services. NOTE 72 – For example, platforms such as k8s, MEC or O-RAN may expose SDKs or APIs for automation of design, deployment of network services.
AN-arch-req-067	It is required that AN utilizes the application development pipelines to automate the design and/or instantiation of closed loops in various levels of the network. NOTE 73 – For example, platforms such as ONF, O-RAN and ONAP allows design and deployment of xApps and AI/ML models respectively. In some cases, 3rd party respositories may be accessed to select and deploy xApps and/or AI/ML models.

Requirement	Description
AN-arch-req-068	It is required that AN discovers the topology and connectivity/split options and capabilities in the underlay architecture and considers such options while integrating controllers in underlays NOTE 74 – For example, 3GPP networks may have various architecture split options.
AN-arch-req-069	It is required that AN integrates intelligent controllers at various levels of the underlay. NOTE 75 – Examples of intelligent controllers are those integrating AI/ML models.
AN-arch-req-070	It is required that AN enables integration of controllers to network management and application management at various levels of the underlay. NOTE 76 – Examples of functionality of such controllers are placement of functions, choice of architecture splits.
AN-arch-req-071	It is required that AN enables designing, developing and deploying applications at various levels of the underlay. NOTE 77 – For example, in coordination with the edge network orchestrator, AN may provide a design for vertical applications to be deployed at a specific edge location. NOTE 78 – Design and development of applications may be achieved in coordination with CI/CD pipelines.
AN-arch-req-072	It is required that AN monitors the feedback from controllers deployed at various levels of the network to optimize the design and development and deployment of controllers. NOTE 79 – For example, in coordination with the edge network orchestrator, AN may optimize the existing design for vertical applications to be deployed at a specific edge location.
AN-arch-req-073	It is required that AN creates strategies for experimentation for testing and validation of controllers in sandbox environment.
AN-arch-req-074	It is required that AN deploys, tests and validates controllers in sandbox environment.
AN-arch-req-075	It is required that AN analyses the results from experiments in sandbox environment and uses those results to update the knowledge base, optimize deployed controllers in underlays as well as optimize the experimentation strategies in sandbox. NOTE 80 – Example of optimization include selection of new controllers or modules. Example of optimization of experimentation strategies are selection of new test scenarios.
AN-arch-req-076	It is required that AN captures the domain specificities differently from design of a potential controller to be deployed in that domain. NOTE 81 – Examples of domain are administrative domains in the underlays like edge network, RAN, CN. NOTE 82 – Examples of domain specificities are latency criteria, location information, data privacy requirements, etc. NOTE 83 – Example of capturing domain specificities are TOSCA service definitions. Design of controllers may also be represented using such declarative definitions.
AN-arch-req-077	It is required that AN captures service specificities. NOTE 84 – Examples of service specificities include service level requirements for QoS.

Requirement	Description
AN-arch-req-078	It is required that AN enables discovery of service level tradeoffs. NOTE 85 – Examples of service level tradeoffs are greater accuracy of inference provided larger resource for training of AI/ML models.
AN-arch-req-079	It is required that AN triggers re-design of network services based on monitoring of network services in underlays.
AN-arch-req-080	It is required that AN triggers evolution of network services based on monitoring of network services lifecycle in underlays
AN-arch-req-081	It is required that AN enables run-time discovery of new use cases for optimization in underlays.
AN-arch-req-082	It is required that AN supports usage of various types of controllers for problem discovery in various levels of the underlay. NOTE 86 – 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-req-083	It is required that AN supports usage of various types of controllers for problem isolation in various levels of the underlay. NOTE 87 – For example, collaborative communication between controllers may be used to isolate the problem.
AN-arch-req-084	It is required that AN supports design and/or selection of controllers based on the problem isolated in the underlay. NOTE 88 – For example, 3rd party controllers from repositories may be selected to address the problem.
AN-arch-req-085	It is required that AN supports deployment of new controllers with new capabilities to address the problems detected in the underlay.
AN-arch-req-086	It is required that AN supports adaptive design of controllers using hardware adaptation techniques such as detection of hardware capabilities and adaptive design. NOTE 89 – For example optimization of AI/ML models to FPGA architectures. Adaptive design may involve considerations of design tradeoffs such as energy efficiency, accuracy, etc.
AN-arch-req-087	It is required that AN supports feedback and optimization of hardware adaption process for controllers. NOTE 90 – For example, the hardware adaptation process for controllers may involve (1) translation of high level description to a intermediate representation amenable to optimization, (2) optimization considering the design tradeoffs (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-req-088	It is required that AN uses inputs from external environment and user specific models to design as well as apply controller outputs to underlays. NOTE 91 – Example of inputs from external environments is mobility prediction models for users with assistive needs or groups of users. User preferences or capabilities are examples of user specific models.
AN-arch-req-089	It is required that AN uses user preferences or user specific models while designing as well as applying controller outputs to underlays. NOTE 92 – Standard representations of user profiles or preferences or user models with assistive needs are examples of user preferences.

Requirement	Description
AN-arch-req-090	It is required that AN experiments to generate changes to user specific models which may help ease of experience for users in hitherto unforeseen circumstances. NOTE 93 – AN experiments may be done in a sandbox using simulators.
AN-arch-req-091	It is required that AN enables transfer of user specific models to other domains in the underlay. NOTE 94 – This may help in update of models in other domains, update of simulators, etc.
AN-arch-req-092	It is required that AN provides evolution of controllers as a service to underlays. NOTE 95 – Underlays 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 underlays.
AN-arch-req-093	It is required that AN discovers and utilizes the services provided by other evolution-as-a-service for controllers. NOTE 96 – Underlays may have 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-req-094	It is required that AN provides experimentation of controllers as a service to underlays. NOTE 97 – Underlays may have different types of controllers deployed including from 3rd parties. AN discovers the characteristics of different types of controllers and provides experimentation as a service and provides results as output for various types of experimentation scenarios.
AN-arch-req-095	It is required that AN enables import and export of configurations for simulators. NOTE 98 – Examples of configurations for simulators are simulated network topologies, simulated number of devices, simulated traffic settings, closed loop interfaces.
AN-arch-req-096	It is required that AN enables asynchronous trigger of experimentation. NOTE 99 – Examples of asynchronous triggers 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.
AN-arch-req-097	It is required that AN enables validation of results of experimentation. NOTE 100 – Examples of validation include sanity checks, functional and non-functional tests.
AN-arch-req-098	It is required that AN enables feedback to the design of controllers based on the results of experimentation. NOTE 101 – Examples of feedback are experimentation logs, test scenarios along with detailed results.
AN-arch-req-099	It is required that AN enables design of experimentation scenarios based on use case descriptions. NOTE 102 – Example of design representation is TOSCA definitions, derived from use case representations.

Requirement	Description
AN-arch-req-100	It is required that AN discovers and utilizes the services provided by other experimentation-as-a-service for controllers. NOTE 103 – Underlays 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-req-101	It is required that AN selects reference points in the underlay where controllers could be deployed. NOTE 104 – 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 notional cost of training of models or communication overheads.
AN-arch-req-102	It is required that AN selects the experimentation scenarios, data generation and simulators, based on the selected reference points in the underlay where the controllers could be deployed.
AN-arch-req-103	It is required that AN selects the controllers for integration in the underlay based on the results of the experimentations which are in turn based on the reference points in the underlay where the controllers could be deployed.
AN-arch-req-104	It is required that AN integrates the underlay controllers, selected based of the experimentation results, at the reference points in the underlay where the controllers could be deployed. NOTE 105 – This may involve control and data flow modifications according to the integration methods for controllers in the underlay.
AN-arch-req-105	It is required that AN integrate data collection mechanisms. NOTE 106 – 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-req-106	It is required that AN creates a virtual model of real environment for experimentation. NOTE 107 – Virtual model may use visualization and perception mechanisms like AR/VR, simulation engines and data generation mechanisms.
AN-arch-req-107	It is required that AN uses the virtual model along with the real-world inputs to analyse and optimize the underlay and to provide feedback to humans. NOTE 108 – Analysis may use AI/ML models. Optimization may involve configurations in the underlay. Feedback to humans may be generated in AR/VR formats.
AN-arch-req-108	It is optional that 3rd party modules or applications can be integrated in the collection, analysis or feedback steps in the AN. NOTE 109 – For example, a software development kit (SDK) may be exposed to 3rd party developers who may develop new applications to analyse the AR-collected data.

Requirement	Description
AN-arch-req-109	It is required that AN enable discovery of capabilities at the various levels of underlays. NOTE 110 – Examples of levels in underlays are RAN and core network. capabilities of the underlays may differ based on the resource availabilities e.g., compute, memory, already deployed controllers etc.
AN-arch-req-110	It is optional that AN enables creation of application definitions which can then be decomposed to controllers which can be deployed at various levels of the underlay, based on the capabilities at those levels of the underlay. NOTE 111 – Application definitions may be in the form of intents.
AN-arch-req-111	It is optional that AN enables optimal placement of controllers based on the application requirements and capabilities at various levels of the underlay.
AN-arch-req-112	It is required that AN enables continuous monitoring of capabilities at various levels of the underlays and trigger update of controllers based on any changes to the capabilities. NOTE 112 – Examples of changes to capabilities of underlays are addition, deletion of network functions, updates to software versions, etc.
AN-arch-req-113	It is required that AN uses interoperable format for storing controllers. NOTE 113 – Various components in AN may read/write from the stored controllers. E.g., 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-req-114	It is optional that AN supports experimentation and derivation of inter-controller coordination strategies. NOTE 114 – Examples of inter-controller interaction 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-req-115	It is optional that AN supports monitoring and optimization and creation of new inter-controller coordination strategies, along with design of new controllers. NOTE 115 – Optimization may use AI/ML mechanisms.
AN-arch-req-116	It is required that AN enables creation, storage, customization and export of controllers which can be deployed in various types of network underlays.
AN-arch-req-117	It is required that AN enables customization of controllers which can be deployed in various types of network underlays, based on the characteristics of underlays.
AN-arch-req-118	It is optional that AN enables discovery of the characteristics of underlays at runtime.
AN-arch-req-119	It is optional that AN enables 3rd party toolsets for development and visualization of the controllers. NOTE 116 – Graphical user interfaces to edit workflows are examples of 3rd party toolsets.

Requirement	Description
AN-arch-req-120	It is critical that AN supports decomposition of controller designs into parts, which can be mapped to various parts of the underlay based on the capabilities and requirements. NOTE 117 – example of decomposition is splitting of user plane programs into modules which can be hosted in various user plane functions in the network. NOTE 118 – Example of considerations while splitting of controller designs is resource requirements of the controllers and capabilities of the underlay.
AN-arch-req-121	It is critical that AN supports placement or deployment of controllers in various parts of the underlay. NOTE 119 – Deployment may consider resource availability and other capabilities of the underlay along with the requirements of the controller.
AN-arch-req-122	It is optional that AN supports monitoring and optimization of decomposition, design, placement or deployment of controllers in various parts of the underlay.
AN-arch-req-123	It is critical that AN supports composition of controllers deployed in various parts of the underlay to form complex controllers.
AN-arch-req-124	It is optional that AN supports input of use case design whereas design, and deployment of controllers in underlays may be done by third parties.
AN-arch-req-125	It is critical that AN supports description of use case in a declarative format. NOTE 120 – AN use case may be described at high level.
AN-arch-req-126	It is critical that AN supports derivation of domain specific intents from the use case description. NOTE 121 – ML intent [ITU-T Y.3172] is an example of domain specific intent.
AN-arch-req-127	It is optional that AN supports experimentation in domain specific Sandbox and optimization of domain specific intents. NOTE 122 – ML Sandbox [ITU-T Y.3172] is an example of domain specific sandbox.
AN-arch-req-128	It is critical that AN supports tightly coupled and loosely coupled integration with underlays, and capability and preference of the underlay (to perform a tightly coupled or loosely coupled AN integration) may be discovered at the time of integration with the underlay. NOTE 123 – In tightly coupled integration, underlay may utilize the components provided by AN provider. Whereas in loosely coupled integration, underlay may deploy and use its own providers for AN.
AN-arch-req-129	It is critical that AN enables controllers to use domain specific mechanisms to manage resource sharing, service integrations, across administrative domains in the underlays. NOTE 124 – E.g., dynamic service agreements, distributed ledger technologies.
AN-arch-req-130	It is critical that AN interfaces with resource orchestration mechanisms in the underlay. NOTE 125 – 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 126 – interface may be used to trigger actions or monitoring.

Requirement	Description
AN-arch-req-131	It is optional that AN supports discovery of need for new controllers based on monitoring of the underlay. NOTE 127 – Need for new controllers may be discovered based on monitoring of various parameters or KPIs.
AN-arch-req-132	It is critical that, based on the discovery of the need for new controllers, AN supports selection of new controllers from controller repositories, as candidates to be deployed in the underlay. NOTE 128 – Controller repositories may be external or internal to the AN provider.
AN-arch-req-133	It is critical that, based on the candidates selected from repositories, AN supports evaluation of new controllers, as candidates to be deployed in the underlay. NOTE 129 – Evaluation may be done based on use case specific metrics.
AN-arch-req-134	It is critical that, based on the candidates evaluated from repositories, AN supports deployment of new controllers, at runtime, in the underlay.
AN-arch-req-135	It is optional that AN supports peer interaction between controllers without the intervention of a central coordinating function. NOTE 130 – Example of peer interaction is exchange of metadata for resource allocation and load balancing.

8 Architecture Description

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

8.1 Description of Controller

As noted in the description of use cases for autonomous networks [ITU-T Y.Supp 71], 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.

Exploratory evolution and experimentation are examples of functionalities in the AN which act upon controllers. Exploratory Evolution hosts evolution controllers which provides 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 1 – Examples of system under control are managed entities, workflows and/or processes in an IMT-2020 network.

NOTE 2 – 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.

In this architecture, we introduce the term controller to refer specifically to a workflow, open loops, or closed loops that are composed of modules.

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.

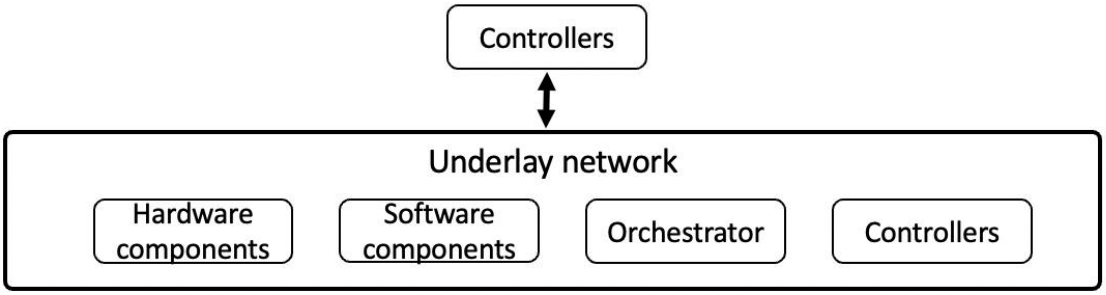

Figure 1 – Controllers and Underlay Interaction

Figure 1 shows the different forms interaction of controllers with the underlay network. The interactions shown here are:

- Controller interacting with a hardware components [b-LogicNets].
- Controller interacting with software components.
- Controller interacting with an orchestrator or other software control mechanism [b-FRINX].
- Controller interacting with another controller.

NOTE 3 – Building upon this simple representation, hierarchies of controllers may be formed.

8.2 Description of the Architecture Components

This clause describes the components of the architecture for autonomous networks to achieve the requirements mentioned in clause 7.

It also describes the components of the architecture for autonomous networks to realize the key concepts of exploratory evolution, experimentation and dynamic adaptation.

8.2.1 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 controllers 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 modules 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.

8.2.2 Knowledge Base

Knowledge in AN is a collection of resources that helps in solving a specific type of problem.

NOTE 1 – Examples of resources are description of a problem along with the description of corresponding potential solutions to that type of problem. The descriptions may be in the form of standard metadata. Resources may include possible causes of the problem, corresponding solutions and their advantages, disadvantages and optimization approaches etc. Problems may have sub-problems e.g., QoE problems may have sub-problems including coverage problems and/or interference problems.

A knowledge base component manages knowledge derived from and used in autonomous networks. It is updated and accessed by various components in the autonomous network.

NOTE 2 – 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.

NOTE 3 – 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 4 – 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.

8.2.3 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 that provided by 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) 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.

8.2.4 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.

8.2.5 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.

8.2.6 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 necessitates validation of controllers, e.g., software update of controllers.

8.2.7 External functionalities

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

Examples may include:

- AN Controller repositories [b-dagsthul]: This is a repository which contains controllers and modules. AN components (e.g., evolution controller) may access this repository 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: Domain orchestrators (may be implemented by third parties) may provide specific functions associated with specific technologies such [b-ETSI GS ZSM 009-1, b-FRINX, b-ONAP].

- Development pipelines (CI/CD pipelines): provides continuous development environment for components, modules and controllers.
- Model repositories [ITU-T Y.3176]: the model repository would store the specifications of models used in the AN.

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

- 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.

8.3 Description of Subsystems

8.3.1 Exploratory Evolution Subsystem

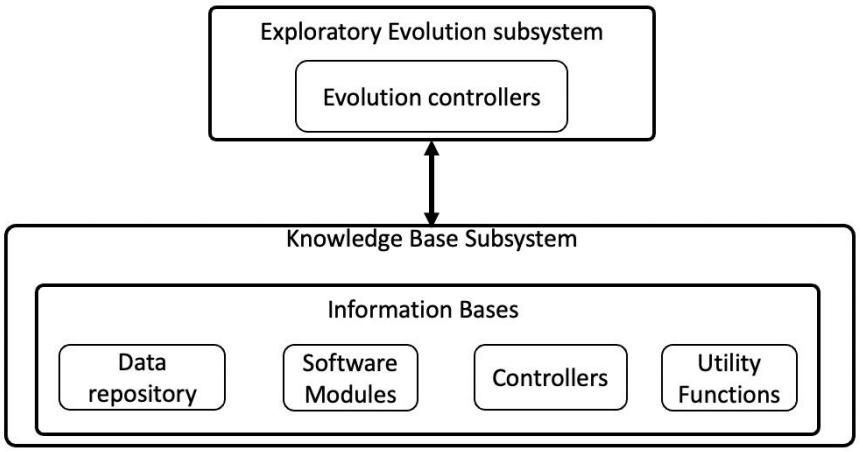

Figure 2 – Exploratory Evolution Overview

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 to use to operate autonomous networks (i.e., the controller). 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 posse 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 representation 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.

Collection of controllers may be formed with each of them tasked with the same purpose but with various different compositions. These are called Controller hierarchies.

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 hierarchy designs from controller specifications.
4) The automatic modification of controller hierarchy designs based on existing controller hierarchy specifications and 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].

8.3.2 Experimentation Subsystem

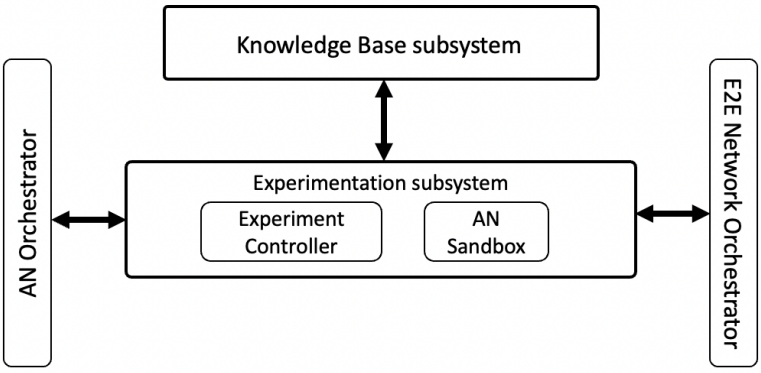

Figure 3 – Experimentation Subsystem

Controllers must be validated before being integrated to the underlay to ensure that it is free of errors and meets both the functional and non-functional requirements.

In the architecture, the experimental evaluation subsystem is concerned with the validation of controllers, as shown in Figure 3. Specifically, an experiment controller will design, orchestrate, and execute an experimental scenarios.

Validation is a spectrum of activities that may encompass one or more activities, 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 experiment controller also requires additional input from about the underlay network and its configuration. As shown in Figure 3, this information should be stored and made available from the knowledge base. Representative examples of such data are discussed in clause 8.3.4.

8.3.3 Dynamic Adaptation Subsystem

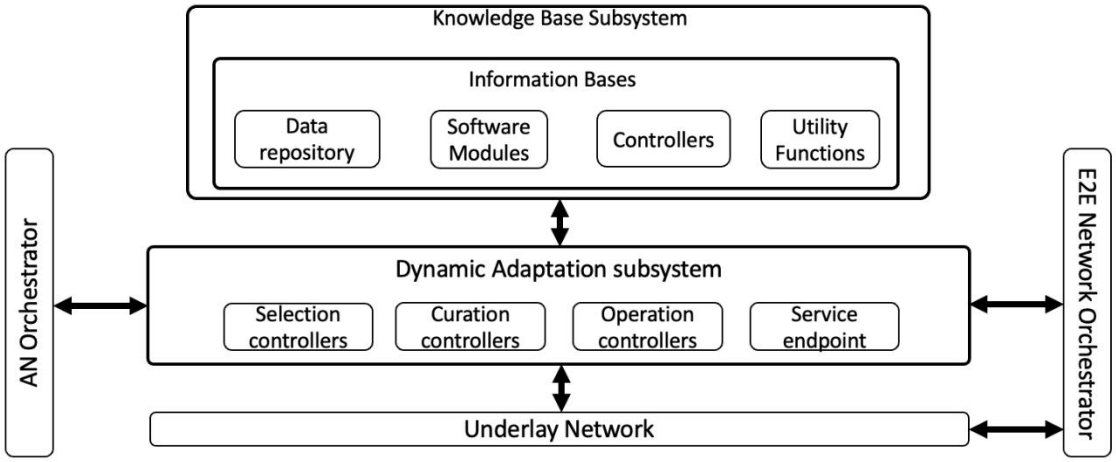

Figure 4 – Dynamic Adaptation Subsystem

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", 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 8.3.2).

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 later in Figure 5, the evolvable 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.

NOTE 3 – For example, a selection controller may be composed of modules which send the trend of fluctuation of use populations, network traffic, or resource demands. As discussed in clause 8.1 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 would correspond to the *design-time* and *run-time* concepts, respectively, as expressed in [ITU-T Y.3177].

8.3.4 Knowledge Base subsystem

Autonomous networks requires 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.

8.4 Autonomy Engine

Autonomy engine refers to the grouping of the evolutionary exploration subsystem and the real-time online experimentation subsystem. Together, these architectural components enable the more general trial and error process where new candidate controllers generated in the former and validated by the latter.

8.5 Description of Architecture

This clause describes the high-level architecture framework for Autonomous Networks, as shown in Figure 5. As noted previously, the goal of this architecture is to support the continuous evolutionary-driven creation, validation, and application of a sea of CL controllers to a network and its services such that the network and its services may become autonomous.

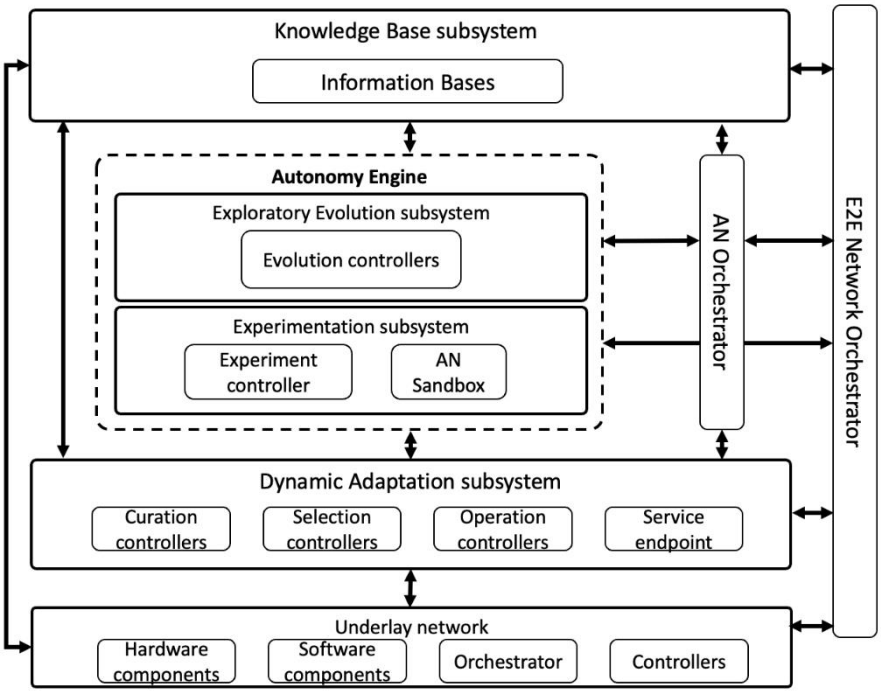

Figure 5 – High-Level Framework for Autonomous Network

8.5.1 Controller Roles

As shown in Figure 5, there are four different roles performed by Controllers. These different controller roles are:

NOTE – Even though the roles are distinct, from a compositional perspective, the exploratory evolution (as explained in clause 8.3.1), real-time experimentation (as explained in clause 8.3.2) and dynamic adaptation (as explained in clause 8.3.3) can be applied to these controllers. See clause 8.5.2 for details.

- **Operation Controller**: 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 these actions (e.g., scale in/out or migration) to the managed entity. 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.
- **Selection Controller**: A controller responsible for the selection of a services' operational controller from the curated controller lists. Selection controller utilizes metrics including the data related to the status and performance of the different components of the architecture and managed entities as described in clause 8.3.4 recorded in the knowledge base to select the operation controller to be deployed in the underlay.
- **Curation Controller**: A controller responsible for the selection and maintenance of the controllers within the curated controller lists from the evolvable controllers.
- **Evolution Controller**: A controller responsible for evolution of controllers. Evolution Controller may use as input, a set of controllers called evolvable controllers, which are subject to the exploratory evolution process.

These controller roles are used in many workflows in the architecture. For example, a end-to-end workflow which starts with formulation of controller specifications to controller deployment in the underlay is given below.

1) Controller specifications, module specifications, experimental specifications are loaded in the Knowledge base.
2) Controller design for default evolutional, curation, and selection controllers are loaded in the Knowledge base.
3) Evolution Controller is instantiated based on default controller designs in the KB.
4) Evolution Controller generates new evolvable controller designs based on modules and controller specifications or pre-existing designs in the KB and stores them in the KB.
5) Controller designs from the KB are picked up by Experiment controller and prepared for validation.
6) Experiment controller designs and builds experiments needed to validate controllers based on experimental specifications from the KB, Controllers are validated against the experiments.
7) The results from the analysis & report of the experiment are stored in KB.
8) Curation Controller curates suitable controllers from the Evolvable Controllers to be used in the management of the managed entity for specific context/intent.
9) Selection Controller analyses and decides most suitable controllers from the curated list to manage the managed entity in the current context.
10) AN orchestrator will instantiate or replace existing Operation Controller with the most suitable controller to perform actions on intended managed entity.

8.5.2 Self-reflective use of the AN architecture framework

The AN architecture framework shown in Figure 5 is used for creating/adapting controllers, validating controllers, 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.

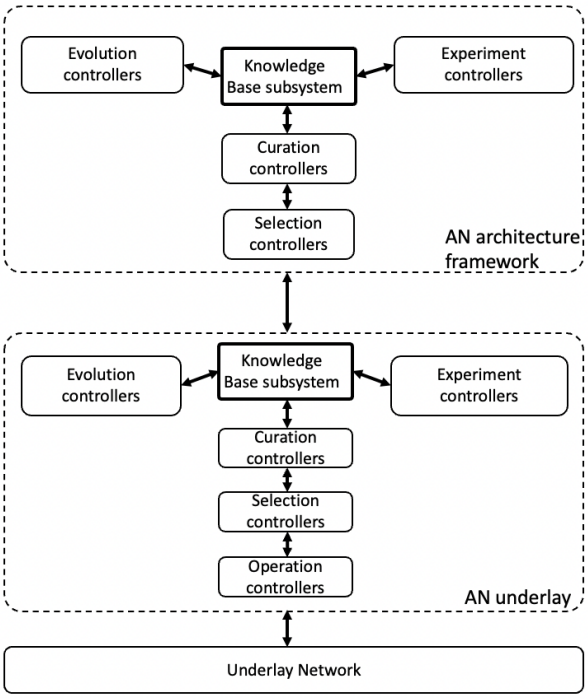

Figure 6 – Self-Reflective use of the AN architecture framework

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 Figure 6. Thus, the architecture itself becomes a collection of managed entities.

NOTE 1 – Even though Figure 6 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 – Figure 6 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.

9 Sequence diagrams

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

9.1.1 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.

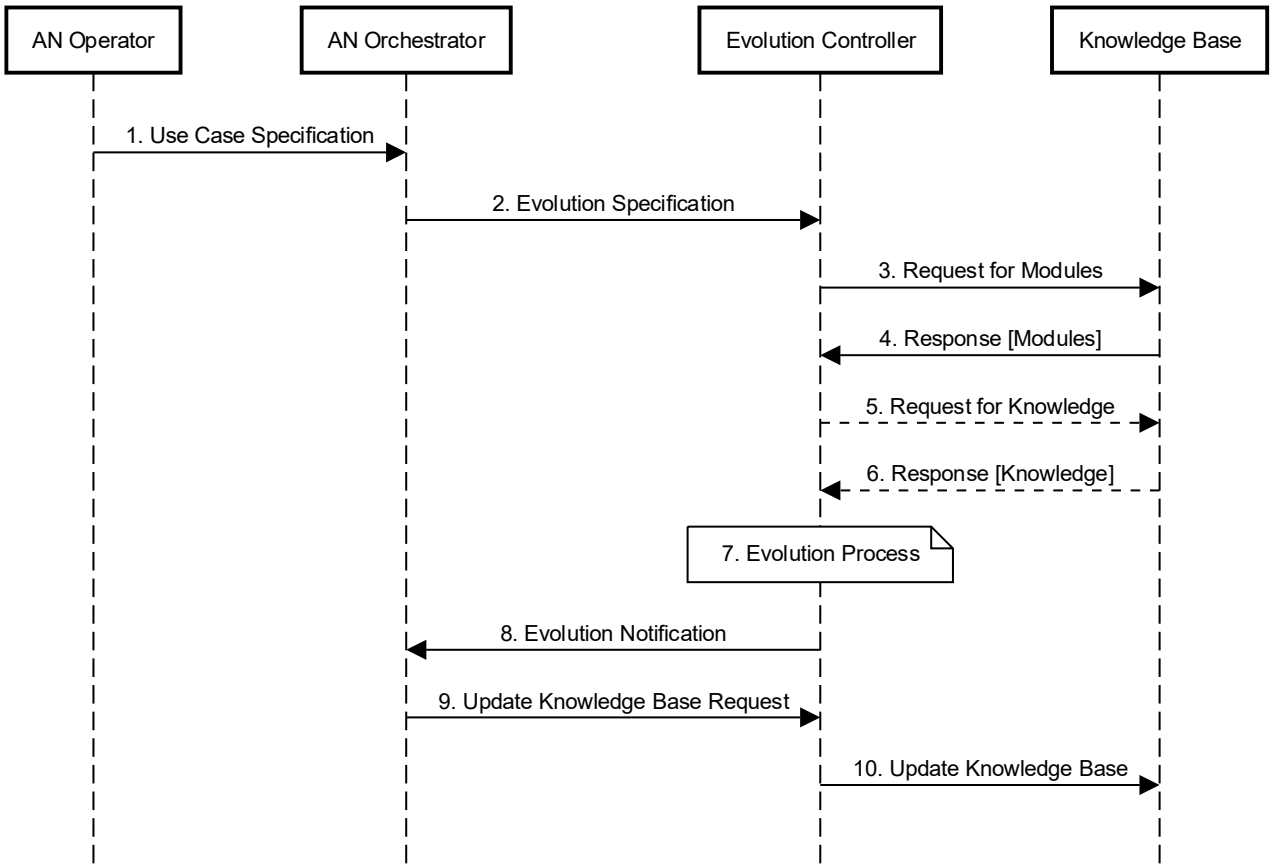

Figure 7 – Creation of Controllers

The steps involved in the scenario described in Figure 7 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 8.2.

9.1.2 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. Experiment controller derives scenarios for experimentation based on the experiment specifications. Based on these scenarios, Experiment 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 Experiment controller reuses an existing experiment specifications (stored in the KB) and designs the experiments to validate the controller included in the experiment specification provided by the AN orchestrator.

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

Pre-requisites:

0. Experiment specifications and evolvable controllers are populated in the KB. This may be done based on Creation of controllers in Figure 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 manager requests the current list of experiments from the KB.
3) The knowledge base replies with the requested data, if any.
4) Experimentation manager requests the current list of controllers from the KB.
5) The knowledge base replies with the requested data, if any.
6) Experimentation manager requests additional knowledge from the KB, needed to support the experiment design.

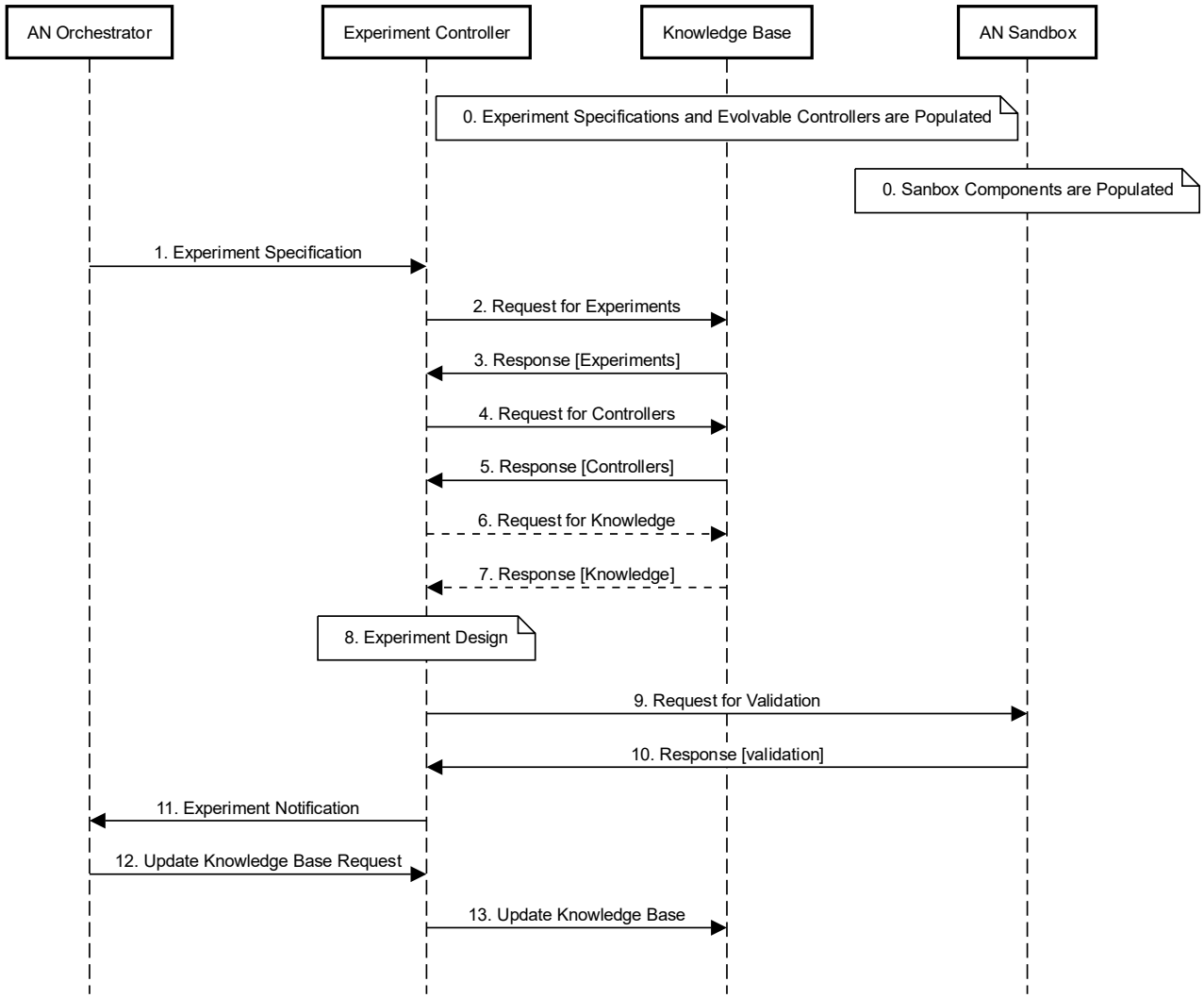

Figure 8 – Validation of Controllers

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.

1) The knowledge base replies with the requested data, if any.
2) The experiment controller will design potential experimentation scenarios.
3) For each experimentation scenarios, the experimentation manager will request the AN sandbox to perform the validation.
4) The AN sandbox will report the results to the experimentation manager.
5) The experimentation manager will perform any necessary analysis of the results, and notifies the AN orchestrator.
6) AN orchestrator triggers the experimentation manager to update the knowledge base.
7) 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 8.2.

9.1.3 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.

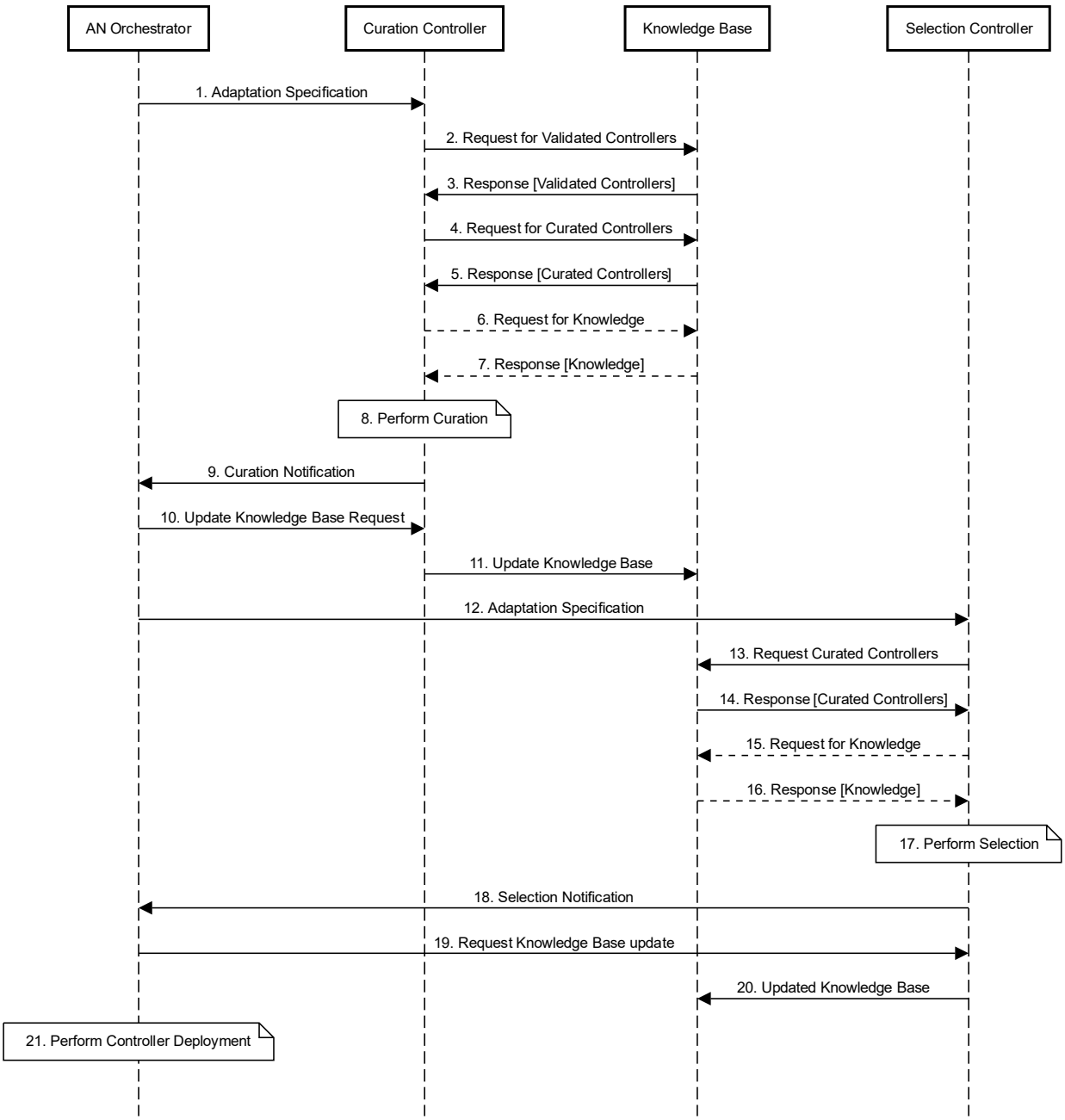

Figure 9 – Dynamic Adaptation of Controllers

The steps involved in the scenario described in Figure 9 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 information 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 8.2.

9) The Curation Controller notifies the AN Orchestrator that it has completed the curation process.
10) The AN Orchestrator requests 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 curation process is beyond the scope of this document. Examples of such processes can be found in clause 8.2.

18) The Selection Controller notifies the AN Orchestrator that it has completed the selection process.
19) The AN Orchestrator requests 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.

Bibliography

[b-ITU-T Y.3173] Recommendation ITU-T Y.3173 (2020), *Framework for evaluating intelligence levels of future networks including IMT-2020.*

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

[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. 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-ETSI-AN] ETSI White Paper No. 40 (2020), *Autonomous Networks, supporting tomorrow's ICT business*.

[b-ETSI GS ENI 002] ETSI GS ENI 002 V1.1.1 (2018), *Experiential Networked Intelligence (ENI); ENI requirements*.

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

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

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

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

[b-FRINX] 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-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] https://irtf.org/nmrg

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

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

[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-TMFORUM-AN-WP] *Autonomous Networks: Empowering Digital Transformation For The Telecoms Industry*, whitepaper

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