ITUPublications
Recommendations

International Telecommunication Union
Standardization Sector

Recommendation

ITU-T Y.3061 (12/2023)

SERIES Y: Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities

Future networks

Autonomous networks – Architecture framework

ITU-T Y-SERIES RECOMMENDATIONS

Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities

GLOBAL INFORMATION INFRASTRUCTURE	Y.100-Y.999
General	Y.100-Y.199
Services, applications and middleware	Y.200-Y.299
Network aspects	Y.300-Y.399
Interfaces and protocols	Y.400-Y.499
Numbering, addressing and naming	Y.500-Y.599
Operation, administration and maintenance	Y.600-Y.699
Security	Y.700-Y.799
Performances	Y.800-Y.899
INTERNET PROTOCOL ASPECTS	Y.1000-Y.1999
General	Y.1000-Y.1099
Services and applications	Y.1100-Y.1199
Architecture, access, network capabilities and resource management	Y.1200-Y.1299
Transport	Y.1300-Y.1399
Interworking	Y.1400-Y.1499
Quality of service and network performance	Y.1500-Y.1599
Signalling	Y.1600-Y.1699
Operation, administration and maintenance	Y.1700-Y.1799
Charging	Y.1800-Y.1899
IPTV over NGN	Y.1900-Y.1999
NEXT GENERATION NETWORKS	Y.2000-Y.2999
Frameworks and functional architecture models	Y.2000-Y.2099
Quality of Service and performance	Y.2100-Y.2199
Service aspects: Service capabilities and service architecture	Y.2200-Y.2249
Service aspects: Interoperability of services and networks in NGN	Y.2250-Y.2299
Enhancements to NGN	Y.2300-Y.2399
Network management	Y.2400-Y.2499
Computing power networks	Y.2500-Y.2599
Packet-based Networks	Y.2600-Y.2699
Security	Y.2700-Y.2799
Generalized mobility	Y.2800-Y.2899
Carrier grade open environment	Y.2900-Y.2999
FUTURE NETWORKS	**Y.3000-Y.3499**
CLOUD COMPUTING	Y.3500-Y.3599
BIG DATA	Y.3600-Y.3799
QUANTUM KEY DISTRIBUTION NETWORKS	Y.3800-Y.3999
INTERNET OF THINGS AND SMART CITIES AND COMMUNITIES	Y.4000-Y.4999
General	Y.4000-Y.4049
Definitions and terminologies	Y.4050-Y.4099
Requirements and use cases	Y.4100-Y.4249
Infrastructure, connectivity and networks	Y.4250-Y.4399
Frameworks, architectures and protocols	Y.4400-Y.4549
Services, applications, computation and data processing	Y.4550-Y.4699
Management, control and performance	Y.4700-Y.4799
Identification and security	Y.4800-Y.4899
Evaluation and assessment	Y.4900-Y.4999

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T Y.3061

Autonomous networks – Architecture framework

Summary

Recommendation ITU-T Y.3061 provides requirements, architecture, components and related sequence diagrams that together comprise an architecture framework for autonomous networks.

This Recommendation includes:

– requirements for the architecture;

– description of the architecture and its components;

– sequence diagrams explaining the interactions between architecture components.

History *

Edition	Recommendation	Approval	Study Group	Unique ID
1.0	ITU-T Y.3061	2023-12-14	13	11.1002/1000/15735

Keywords

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

* To access the Recommendation, type the URL https://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents/software copyrights, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the appropriate ITU-T databases available via the ITU-T website at http://www.itu.int/ITU-T/ipr/.

Table of Contents

Recommendation ITU-T Y.3061

Autonomous networks – Architecture framework

1 Scope

This Recommendation provides requirements, architecture components and related sequence diagrams that together comprise an architecture framework for autonomous networks (ANs).

This Recommendation includes:

– requirements for the architecture;

– description of the architecture and its components;

– sequence diagrams explaining the interactions between the architecture components.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T Y.2701] Recommendation ITU-T Y.2701 (2007), *Security requirements for NGN release 1*.

[ITU-T Y.3101] Recommendation ITU-T Y.3101 (2018), *Requirements of the IMT-2020 network*.

[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.3172] Recommendation ITU-T Y.3172 (2019), *Architectural framework for machine learning 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*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

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

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

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

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

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

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

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

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

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

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

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

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

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

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.2.9 experimentation controller: A controller that generates potential scenarios of experimentation based on controller specifications and additional information as provided by the knowledge base, executes the scenarios in the autonomous network sandbox, collates and validates the results of the experimentation.

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

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

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

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

3.2.13 workflow: Sequence of activities to describe or realize a given task executed by a system.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

AF	Application Function
AI	Artificial Intelligence
AN	Autonomous Network
API	Application Programming Interface
AR	Augmented Reality
CI/CD	Continuous Integration and Continuous Delivery
CL	Closed loop
CN	Core Network
CP	Control Plane
DNN	Deep Neural Network
DNS	Domain Name Service
E2E	End to End
IMT-2020	International Mobile Telecommunications-2020
IoT	Internet of Things

KB	Knowledge Base
KPI	Key Performance Indicator
MANO	Management and Orchestration
MEC	Multi-access Edge Computing
ML	Machine Learning
NF	Network Function
OC	Operation Controller
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RCA	Root Cause Analysis
SDK	Software Development Kit
TOSCA	Topology and Orchestration Specification for Cloud Applications
UE	User Equipment
UPF	User Plane Function
VR	Virtual Reality
YAML	Yet Another Meta Language
ZSM	Zero touch Service Management

5 Conventions

In this Recommendation:

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

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

The phrase "can optionally" indicates an optional requirement that is permissible, without implying any sense of being recommended. This term is not intended to imply that the vendor's implementation must provide the option, and the feature can be optionally enabled by the network operator/service provider. Rather, it means the vendor may optionally provide the feature and still claim conformance with this Recommendation.

6 Introduction

ANs are those that possess the capabilities to monitor, operate, recover, heal, protect, optimize and reconfigure themselves; these are commonly known as the self-* properties, where the asterisk can be "healing", "operating", "optimization", etc. [b-Kephart].

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

One common problem in the application of ML to these use cases is the problem of model drift. Model drift is a phenomenon whereby either the goal of the ML model changes overtime (conceptual drift) or when the available data no longer enables the model to form the same relationships (data

drift). This problem can be seen most obviously in financial markets, where market predictions must be frequently revisited to address the reality that the operating environment (the market) has changed compared to when the model was made. Several tools and frameworks have been proposed to help address these issues [b-Bega], [b-Whitley], [b-Maggi], [b-Sutton], [b-AutoML], [b-Real].

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

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

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

The conclusion of the preceding is that no matter the domain of operation, technology, algorithm, intelligence or data set used, an AN requires 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-T Y.3177].

The key purpose and goal of the architecture described in this Recommendation is to support the continuous evolutionary-driven creation, validation, and application of a set of controllers to a network and its services such that the network and its services may become autonomous. A controller is a workflow, closed loop (CL) or open loop of a system under control. It is composed of modules, integrated in a specific sequence, and using interfaces exposed by the modules, to solve a specific problem or satisfy a given requirement. Modules composing the controller may be workflows, open loops, or CLs and can be developed independently of the system under control before being integrated into the system under control.

The continuous evolutionary-driven creation, validation and application of controllers is used in the use cases to realize ANs [b-ITU-T Y-Suppl. 71] and the key concepts (see the following) required to enable them.

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

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

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

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

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

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

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

In addition, the requirements for the architecture in clause 7 also consider the following concepts in ANs: knowledge and orchestration.

The analysis of data and information from the network, resulting in an understanding of what the data and information mean, is referred to as knowledge. Knowledge is used in ANs to support continuous exploratory evolution, real-time online experimental validation and dynamic adaptation.

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

7 Requirements for the architecture

This clause describes requirements for AN architecture.

7.1 Requirements for exploratory evolution

The following are requirements with respect to exploratory evolution in ANs.

Requirement	Description
AN-arch-evo-req-001	AN architecture is required to have the ability to generate and update controller designs.
AN-arch-evo-req-002	AN architecture is required to have the ability to generate potential scenarios of exploratory evolution, taking the controller designs as input. NOTE – Specific mechanisms or algorithms used for evolution lie outside the scope of this Recommendation.
AN-arch-evo-req-003	AN architecture is required to have the ability to execute the potential scenarios of exploratory evolution, taking the controller designs as input and to collate the output in the form of evolvable controller designs. NOTE – Several rounds of evolution may be applied on the same set of controller designs.
AN-arch-evo-req-004	AN architecture is required to support the generation of potential configurations for the integration of controllers to specific underlay networks. NOTE 1 – Configurations may include reference points, application programming interface (API) formats and data models. This generation of configurations may take the controller designs as input along with the description or metadata related to the underlay networks. NOTE 2 – While the specific configurations for the integration of controllers for specific use cases lie outside the scope of this Recommendation, the specification of acceptable formats for representing such configurations is for further study. NOTE 3 – Examples of underlay networks are edge networks, core networks (CNs), management plane and continuous integration and continuous delivery (CI/CD) pipelines.
AN-arch-evo-req-005	AN architecture is required to enable the management of points of metadata exchange in the AN workflow, with a peer entity.

Requirement	Description
	NOTE 1 – Examples of metadata regarding the AN workflow include current capabilities, status and context of the AN components, including knowledge base (KB), controllers, orchestrators and simulators. Managing may include identifying actors and points in the AN workflow, capturing metadata regarding the workflow that can then be exchanged. NOTE 2 – Examples of points of metadata exchange in the AN workflow are the different stages of experimentation and dynamic adaptation. NOTE 3 – The format used for the metadata exchange lies outside the scope of this Recommendation. NOTE 4 – Examples of AN workflow include the generation of scenarios for potential experimentation and potential evolution. NOTE 5 – Peers may include humans and machines.
AN-arch-evo-req-006	AN architecture is required to have the ability to integrate the impacts of metadata exchange with peer entities involved in AN workflows. NOTE – Examples of impacts of the metadata exchange are updates of KB and selection of API versions to use for adaptation.
AN-arch-evo-req-007	AN architecture is required to support the optimization of controllers. NOTE – Examples of optimization of controllers are optimization of adaptation mechanisms like data collection, data quality and frequency. Other examples are optimization of CL implementations like root cause analysis (RCA) mechanisms and recommendations on better algorithms for achieving the same objective. Other examples are formation or evolution of new controllers to address new or unforeseen problems in the underlay network.
AN-arch-evo-req-008	AN architecture is required to support the ability to discover characteristics of controllers that are relevant for enabling evolution. NOTE – Examples of characteristics of controllers that are relevant for evolution are: capabilities exposed by and requirements to be satisfied for the controllers.
AN-arch-evo-req-009	AN architecture is required to support the capability to recommend modules that can satisfy the characteristics of controllers.
AN-arch-evo-req-010	AN architecture is required to enable the integration of controllers from different domains to achieve complex use cases. NOTE – For example, an AN may integrate controllers in different domains of the network, like radio access network (RAN) and CN domains.
AN-arch-evo-req-011	AN architecture is required to enable the utilization of declarative specifications of use cases while deciding the design, deployment and management of controllers.
AN-arch-evo-req-012	AN architecture is required to allow for the utilization of declarative specifications of use cases to capture both use case requirements from applications and deployment requirements from underlay networks.
AN-arch-evo-req-013	AN architecture is required to support the capability to choose a compatible set of interfaces to integrate with underlay network services. NOTE – Examples of interfaces that may be used include those to monitor the services and controllers in the underlay networks.
AN-arch-evo-req-014	AN architecture is required to support the creation or recommendation of candidate designs for potential network services and interfaces in the underlay networks, which may possibly satisfy new use cases. NOTE – Design creation may also be triggered in response to an observed fault in underlay networks.
AN-arch-evo-req-015	AN architecture is required to consider use case specific requirements (including operator preferences) while determining operator preferences for design, deployment and management of controllers, including connectivity options between various domains.

Requirement	Description
	NOTE – For example, in rural areas, end-users may have usage patterns with characteristics dependent on applications (e.g., low mobility, high bandwidth). The choice of last mile connectivity options may be influenced by such preferences.
AN-arch-evo-req-016	AN architecture is required to support capabilities enabling the evolution of inter-domain connectivity among controllers deployed in various domains of the underlay network.
AN-arch-evo-req-017	AN architecture is required to support adaptation to the evolution of applications at run-time. NOTE – An example of adapting to the evolution of applications at run-time is onboarding new applications or changes in existing applications deployed by service providers in the underlay network.
AN-arch-evo-req-018	AN architecture is required to support adaptation to changes in external systems that the AN interfaces with. NOTE – Examples of external systems are various management and orchestration (MANO) systems, policies and corresponding management systems, workflow management systems and user management systems. External systems deployed by the operator may be provided by multiple vendors.
AN-arch-evo-req-019	AN architecture is required to support the ability to recommend changes in capabilities to monitor, configure and analyse parameters from underlay networks.
AN-arch-evo-req-020	AN architecture is required to support learning of metric derivations from collected parameters and measurements, where such learning may change. NOTE – Derivation of metrics may use artificial intelligence (AI)/ML techniques. Derivation mechanisms may change over time or events. In such cases, mechanisms such as re-training may be applied to update derivation models.
AN-arch-evo-req-021	AN architecture is required to support the derivation of requirements for service life cycle management in underlay networks based on analysis of requirements from different domains in the network. NOTE 1 – For example, CLs for optimization in a RAN may be configured based on analysis of requirements from the CN. NOTE 2 – Service life cycle management includes resource allocation, scaling and optimization.
AN-arch-evo-req-022	AN architecture is required to support optimization of intent based on monitoring of the performance of derived controllers deployed in various domains of the underlay network and their life cycles. NOTE – For example, derivation of intent in the CN may be optimized based on the feedback obtained from monitoring the (derived) closed loops deployed in the RAN.
AN-arch-evo-req-023	AN architecture is required to support application development capabilities to automate the design and instantiation of underlay network services. NOTE – For example, platforms such as Kubernetes [b-kubernetes], multi-access edge computing (MEC) [b-ETSI GS MEC 012] or O-RAN [b-ORAN] may expose software development kits (SDKs) or APIs for automation of design and deployment of network services.
AN-arch-evo-req-024	AN architecture is required to support application development capabilities to automate the design or instantiation of controllers in various levels of the underlay network. NOTE – For example, platforms such as Open Networking Foundation [b-ONF], O-RAN [b-ORAN] and Open Network Automation Platform [b-ONAP] allow design and deployment of xApps [b-ORAN] and AI/ML models

Requirement	Description
	respectively. In some cases, third party repositories may be accessed to select and deploy xApps or AI/ML models.
AN-arch-evo-req-025	AN architecture is required to enable design, development and deployment applications by the AN at various levels of the underlay network. NOTE 1 – For example, in coordination with the edge network orchestrating function, AN may provide a design for vertical applications to be deployed at a specific edge location. NOTE 2 – The design and development of applications may be achieved in coordination with CI/CD pipelines.
AN-arch-evo-req-026	AN architecture is required to support capabilities to monitor feedback from controllers deployed at various domains of the underlay network in order to optimize the design, development and deployment of controllers in general. NOTE – For example, in coordination with the edge network orchestrating function, AN may optimize the existing design for vertical applications to be deployed at a specific edge location.
AN-arch-evo-req-027	AN architecture is required to support the ability to trigger re-design of network services based on monitoring of network services in underlay networks.
AN-arch-evo-req-028	AN architecture is required to provide means for the AN to trigger an evolution of network services based on the monitoring of their lifecycle in underlay networks.
AN-arch-evo-req-029	AN architecture is required to support usage of various types of controller for problem discovery in various domains of the underlay network. NOTE – For example, data collection agents may be designed separately from the analysis. Data collection agents may be deployed in the edge network, whereas analysis may be performed in the core cloud.
AN-arch-evo-req-030	AN architecture is required to allow the AN providing evolution of controllers as a service to underlay networks. NOTE – Underlay networks may have different types of controller deployed including those from third parties. The AN discovers the characteristics of different types of controller and provides evolution as a service that results in evolved controller candidates for deployment in underlay networks.
AN-arch-evo-req-031	AN architecture is required to support capabilities for the AN to discover and utilize different evolution services for controllers. NOTE – Underlay networks may need evolution of controllers deployed including those from third parties. AN discovers the characteristics of different types of evolution service 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 or controller repositories as input and accepting evolved controller candidates as outputs.
AN-arch-evo-req-032	AN architecture is required to support the ability for the AN to customize controllers that can be deployed in various types of underlay network, based on their characteristics.
AN-arch-evo-req-033	AN architecture is required to support decomposition of controller designs into parts, which can be mapped to various parts of the underlay network based on capabilities and requirements. NOTE 1 – An example of decomposition is splitting of user plane programs into modules that can be hosted in various user plane functions in the underlay network. NOTE 2 – An example of considerations on decomposition of controller designs is resource requirements of the controllers and capabilities of the underlay network.

Requirement	Description
AN-arch-evo-req-034	AN architecture is recommended to support monitoring and optimization of the decomposition, design, placement or deployment of controllers in various parts of the underlay network.
AN-arch-evo-req-035	AN architecture is required to support composition of controllers deployed in various parts of the underlay network to form complex controllers.
AN-arch-evo-req-036	AN architecture is recommended to enable the integration and plugin of algorithms into controllers. NOTE – Algorithms may be provided by a third party, ML or AI models
AN-arch-evo-req-037	AN architecture is recommended to enable AN discovery of service level trade-offs. NOTE – An example of a service level trade-off is greater accuracy of inference versus larger resource for training of AI/ML models.
AN-arch-evo-req-038	AN architecture is recommended to support adaptive design of controllers using hardware adaptation techniques. NOTE 1 – Examples of hardware adaptation techniques include detection of hardware capabilities and adaptive design. NOTE 2 – Examples of hardware adaptation techniques include optimization of AI/ML models to field programmable gate array architectures. Adaptive design may involve considerations of design trade-offs such as energy efficiency and accuracy.
AN-arch-evo-req-039	AN architecture is recommended to provide capabilities for the AN to support feedback and optimization of hardware adaption process for controllers. NOTE – For example, the hardware adaptation process for controllers may involve: 1) translation of high-level description to an intermediate representation amenable to optimization; 2) optimization considering design trade-offs; and 3) hardware implementation and integration. Translation and optimization steps 1) and 2) may themselves be tuned based on monitoring and feedback from the controllers integrated in the hardware.
AN-arch-evo-req-040	AN architecture can optionally support the capability to recommend new capabilities and requirements for the network functions (NFs) deployed in underlay networks.
AN-arch-evo-req-041	AN architecture can optionally support the ability for the AN to monitor, optimize and create new intercontroller coordination strategies, along with the design of new controllers. NOTE – Optimization may use AI/ML mechanisms.

7.2 Requirements for online experimentation

The following are requirements with respect to online experimentation in ANs.

Requirement	Description
AN-arch-exp-req-001	AN architecture is required to support the validation and processing of controller descriptions, so that exploratory evolution can be applied to these controller designs. NOTE 1 – Exploratory evolution may result, among others, in interconnection of descriptions together to form complex controller designs or in a list of controllers. NOTE 2 – Exploratory evolution may be a triggered or periodic process
AN-arch-exp-req-002	AN architecture is required to support the ability to generate potential scenarios for experimentation, taking controller designs as input. NOTE 1 – Specific configurations and limits of experiments may be specified in the metadata and constraints related to controller designs.

Requirement	Description
	NOTE 2 – Specific mechanisms for arriving at scenarios for experimentation may use AI/ML analytics or other forms of analytics and lie outside the scope of this Recommendation.
AN-arch-exp-req-003	AN architecture is required to have the ability to execute experimentation and to collate and validate its results, considering the metadata and constraints, as well as corresponding controller designs. NOTE 1 – Experimentations may have several phases, e.g., simulation driven, testbed driven or canary test driven. The phases of an experimentation may be configurable and automated, for example, as per a workflow. NOTE 2 – The specific success and failure criteria for the experiments lie outside the scope of this Recommendation. Acceptable formats for representing metadata and constraints related to potential success and failure as related to a use case are for further study.
AN-arch-exp-req-004	AN architecture is required to support the ability to verify, before the actual integration of controllers into the underlay networks, that the proposed evolution of the controllers is compatible with the underlay networks.
AN-arch-exp-req-005	AN architecture is required to support the automation and abstraction of experimentation of underlay network services.
AN-arch-exp-req-006	AN architecture is required to provide capabilities for the creation of strategies regarding the deployment of experimentation scenarios, testing and validation of controllers in a sandbox environment.
AN-arch-exp-req-007	AN architecture is required to support capabilities allowing the AN to deploy, test and validate controllers in a sandbox environment.
AN-arch-exp-req-008	AN architecture is required to provide capabilities allowing the AN to analyse the results from experiments in a sandbox environment and to use those results to update the KB, optimize deployed controllers in underlay networks as well as optimize the experimentation strategies in sandbox. NOTE – An example of controller optimization is the selection of new controllers or modules. An example of experimentation strategy optimization is the selection of new test scenarios.
AN-arch-exp-req-009	AN architecture is required to support the ability of the AN to experiment with the generation of changes to user specific models that may help ease of experience for users in hitherto unforeseen circumstances. NOTE – AN experimentation may be done in a sandbox using simulators.
AN-arch-exp-req-010	AN architecture is required to support capabilities for the AN to provide experimentation of controllers as a service to underlay networks. NOTE – Underlay networks may have different types of controller deployed including those from third parties. An AN discovers the characteristics of different types of controller, providing different experimentation services and results as output for various types of experimentation scenario.
AN-arch-exp-req-011	AN architecture is required to support means for the AN to import and export configurations for simulators. NOTE – Examples of configurations for simulators are simulated network topologies, simulated number of devices, simulated traffic settings and CL interfaces.
AN-arch-exp-req-012	AN architecture is required to enable the AN to asynchronously trigger experimentations. NOTE – Examples of asynchronous triggering are evolution of new set of controllers that need to be validated, updated network configurations by operator, provisioning or update of NFs in the underlay network.
AN-arch-exp-req-013	AN architecture is required to enable the validation of experimentation results by the AN.

Requirement	Description
	NOTE – Examples of validation include sanity checks, functional and non-functional tests.
AN-arch-exp-req-014	AN architecture is required to support the ability of the AN to provide feedback on the design of controllers based on the results of experimentation. NOTE – Examples of feedback are experimentation logs, test scenarios along with detailed results.
AN-arch-exp-req-015	AN architecture is required to support the ability of the AN to design experimentation scenarios based on use case descriptions. NOTE – An example of design representation is a topology and orchestration specification for cloud applications (TOSCA) definition, derived from use case representations.
AN-arch-exp-req-016	AN architecture is required to provide the ability for the AN to discover and utilize other experimentation services for controllers. NOTE – Underlay networks may have experimentation of controllers deployed including that from third parties. AN discovers the characteristics of different types of experimentation service 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 or controller repositories as input and accepting experimentation results as outputs.
AN-arch-exp-req-017	AN architecture is required to support the AN ability to select experimentation scenarios, data generation and simulators, based on the selected reference points in the underlay network where the controllers can be deployed.
AN-arch-exp-req-018	AN architecture is recommended to enable the AN to create a virtual model of real environment for experimentation. NOTE – Creating a virtual model may use visualization and perception mechanisms like augmented reality/virtual reality (AR/VR), simulation engines and data generation mechanisms.
AN-arch-exp-req-019	AN architecture is recommended to support experimentation and derivation of intercontroller coordination strategies. NOTE – Examples of intercontroller interactions are resolution of conflicting goals for various use cases, like power consumption versus coverage optimization. An example of a relevant strategy is a game theory-based cooperative and non-cooperative mechanism.
AN-arch-exp-req-020	AN architecture is recommended to enable the integration of various forms of testing components including simulators and data generators, including those provided by third parties.
AN-arch-exp-req-021	AN architecture can optionally support experimentation in a domain-specific sandbox and optimization of domain-specific intents. NOTE – An ML sandbox [ITU-T Y.3172] is an example of a domain-specific sandbox.

7.3 Requirements for dynamic adaptation

The following are requirements with respect to dynamic adaptation in ANs.

Requirement	Description
AN-arch-adp-req-001	AN architecture is required to support the ability to select candidates from a set of controllers ready for integration and to execute their integration to specific underlay networks, taking as input the generated configurations for integration. NOTE 1 – The specific criteria for selecting controllers for integration lie outside the scope of this Recommendation. Formats for representing such criteria are for further study.

Requirement	Description
	NOTE 2 – The specific mechanisms used for the integration of controllers to underlay networks lie outside the scope of this Recommendation. Examples of such mechanisms are service-based architectures [b-ETSI TS 129 500] and continuous integration mechanisms [b-ITU-T Y.3525].
AN-arch-adp-req-002	AN architecture is required to allow AN decisions about new opportunities for the deployment of controllers in underlay networks.
AN-arch-adp-req-003	AN architecture is required to allow AN decisions about the configuration of controllers that are to be deployed in underlay networks.
AN-arch-adp-req-004	AN architecture is required to enable the reporting and monitoring of AN components and procedures by humans or other automation mechanisms.
AN-arch-adp-req-005	AN architecture is required to support the discovery of deployed controllers in underlay networks, including those deployed by third party providers.
AN-arch-adp-req-006	AN architecture is required to support the discovery and consumption of services provided by service management frameworks. NOTE – Examples of service management frameworks are ONAP [b-ONAP] and Open Source Management and Orchestration [b-OSM].
AN-arch-adp-req-007	AN architecture is required to support the discovery of interfaces with underlay networks used for integration. NOTE – Interfaces with underlay networks may use specific APIs, e.g., for data collection or configuration of NFs. Discovery of interfaces may involve API metadata including parameters, versions and range of parameters.
AN-arch-adp-req-008	AN architecture is required to enable, on a per use case basis, the discovery of the underlay networks' specific parameters candidate for optimization, data points for collection in the underlay network and the relevant key performance indicators (KPIs) for tracking. NOTE – For example, in edge deployments, underlay networks may use MEC APIs [b-ETSI GS MEC 012].
AN-arch-adp-req-009	AN architecture is required to support the ability to customize the integration of controllers into underlay networks, considering the integration options exposed by the underlay networks. NOTE – Examples of integration options are interfaces, parameters and configurations exposed by underlay networks. Examples of underlay networks are those for industry vertical applications.
AN-arch-adp-req-010	AN architecture is required to support the automation and abstraction of the evolution of underlay network services. NOTE 1 – Examples of evolution of underlay network services include updates to support new features, migration to new service platforms and technologies. NOTE 2 – A service provider may monitor the evolution but does not manually execute the evolution of underlay network services.
AN-arch-adp-req-011	AN architecture is required to support the discovery of service management frameworks used by underlay networks. NOTE – This may help in managing and automating the lifecycle of underlay network services by the AN.
AN-arch-adp-req-012	AN architecture is required to support the E2E integration of controllers, taking into account the evolution of connectivity options in the underlay networks. NOTE – For example, access network may be using various different types of connectivity technology that may evolve over time.
AN-arch-adp-req-013	AN architecture is required to support the monitoring of dynamic changes in capabilities of monitoring, configuration and analysis of parameters from underlay networks. NOTE – Given the independent evolution of controllers, of the underlay networks and of the applications deployed, mechanisms such as those for

Requirement	Description
	discovery, publishing and subscription may be used to provide flexibility in monitoring parameters.
AN-arch-adp-req-014	AN architecture is required to support the utilization of dynamic changes in capabilities of monitoring, configuring and analysis of parameters from underlay networks.
AN-arch-adp-req-015	AN architecture is required to support capabilities enabling the deployment of controllers that utilize both simulated and real networks as those for underlay.
AN-arch-adp-req-016	AN architecture is required to support capabilities for the discovery of topology and architecture connectivity or split options, as well as capabilities in the underlay networks, and to consider such options while integrating controllers in underlay networks. NOTE – For example, 3GPP networks may have various architecture split options [b-3GPP TR 38.801].
AN-arch-adp-req-017	AN architecture is required to have capabilities to integrate intelligent controllers at various levels of the underlay network. NOTE – Examples of intelligent controllers are controllers integrating AI/ML models.
AN-arch-adp-req-018	AN architecture is required to enable the integration of controllers to manage networks and applications at various levels of the underlay network. NOTE – Examples of functionalities of such controllers are placement of functions and choice of architecture splits.
AN-arch-adp-req-019	AN architecture is required to enable AN run-time discovery of new use cases for optimization in underlay networks.
AN-arch-adp-req-020	AN architecture is required to support usage of various types of controller for problem isolation in various domains of the underlay network. NOTE – Collaborative communication between controllers may be used to isolate the problem.
AN-arch-adp-req-021	AN architecture is required to support AN design or selection of controllers based on problems isolated in the underlay network. NOTE – For example, third party controllers from repositories may be selected to address the problem.
AN-arch-adp-req-022	AN architecture is required to support AN deployment of new controllers with new capabilities to address problems detected in the underlay network.
AN-arch-adp-req-023	AN architecture is required to support AN ability to select reference points in the underlay network where controllers could be deployed. NOTE – Examples of considerations for AN while selecting the reference points are trade-offs in terms of benefits (i.e., spectral efficiency, latency, etc) as against the computational overheads of training of models or communication overheads.
AN-arch-adp-req-024	AN architecture is required to support AN ability to select controllers for integration into the underlay network based on the results of experimentation, which are in turn based on the reference points in the underlay network where the controllers could be deployed.
AN-arch-adp-req-025	AN architecture is required to support AN ability to integrate controllers, selected based on experimentation results, at reference points in the underlay network where the controllers could be deployed. NOTE – This may involve control and data flow modifications according to the integration methods for controllers in the underlay network.
AN-arch-adp-req-026	AN architecture is required to enable AN continuous monitoring of capabilities at various levels of underlay networks and trigger updates of controllers based on any changes to the capabilities.

Requirement	Description
	NOTE – Examples of changes to capabilities of underlay networks are addition or deletion of NFs, updates to software versions.
AN-arch-adp-req-027	AN architecture is required to support the placement or deployment of controllers in various parts of the underlay network. NOTE – Deployment may consider resource availability and other capabilities of the underlay network along with the requirements of the controller.
AN-arch-adp-req-028	AN architecture is required to support integration that is coupled tightly and loosely with underlay networks. NOTE 1 – The capability and preference of the underlay network to perform a tightly or loosely coupled AN integration may be discovered at the time of integration with the underlay network. NOTE 2 – In tightly coupled integration, the underlay network may utilize the components provided by an AN. Whereas in loosely coupled integration, the underlay network may deploy and use the underlay network provider components for an AN.
AN-arch-adp-req-029	AN architecture is required to enable controllers to use domain-specific mechanisms to manage resource sharing and service integrations, across domains in the underlay networks. NOTE – Examples of domain-specific mechanisms include dynamic service agreements and distributed ledger technologies.
AN-arch-adp-req-030	AN architecture is required to support the discovery by the AN of the need for new controllers based on monitoring of the underlay network. NOTE – The need for new controllers may be discovered based on monitoring of various parameters or KPIs.
AN-arch-adp-req-031	AN architecture is required to support, based on the discovery of the need for new controllers, the selection of new controllers from controller repositories, as candidates to be deployed in the underlay network. NOTE – Controller repositories may be external or internal to the AN provider.
AN-arch-adp-req-032	AN architecture is required to support the evaluation of new candidate controllers to be deployed in the underlay network from those selected from repository candidates. NOTE – Evaluation may be done based on use case specific metrics.
AN-arch-adp-req-033	AN architecture is required to support the run-time deployment of new controllers in the underlay network based on the candidates evaluated from repositories.
AN-arch-adp-req-034	AN architecture is recommended to enable the monitoring of controllers that are already deployed in underlay networks.
AN-arch-adp-req-035	AN architecture can optionally provide the ability to influence the services provided by service management frameworks. NOTE – Examples of AN influence upon services provided by service management frameworks are passing policies and intents to service management frameworks. Service management frameworks may use them to design or deploy new or modified services.
AN-arch-adp-req-036	AN architecture can optionally support capabilities enabling the correlation of declarative specifications of network services with those of controllers and the use of that correlation to integrate controllers in the same or different domains of the network. NOTE – Examples of correlation of declarative specifications of controllers with those of network services include mapping of interfaces, capabilities and requirements. Other examples are identifying opportunities for deriving specifications, e.g., using substitution mechanisms in TOSCA [b-TOSCA].

Requirement	Description
AN-arch-adp-req-037	AN architecture can optionally enable optimal placement of controllers by the AN based on application requirements and capabilities at various domains of the underlay network.
AN-arch-adp-req-038	AN architecture can optionally support peer-to-peer interaction between controllers without the intervention of a centralized coordinating function. NOTE – Example of peer interaction is exchange of metadata for resource allocation and load balancing.

7.4 Requirements for knowledge

The following are requirements with respect to knowledge in ANs.

Requirement	Description
AN-arch-knw-req-001	AN architecture is required to provide capabilities for the management of knowledge related to ANs. NOTE – Managing knowledge includes its storage, querying, export, import and optimization.
AN-arch-knw-req-002	AN architecture is required to enable the update of knowledge based on the various processes involved in the AN.
AN-arch-knw-req-003	AN architecture is required to support the utilization of components like stored controllers and knowledge to deploy and manage controllers in underlay networks. NOTE – Examples of components include stored controllers and knowledge.
AN-arch-knw-req-004	AN architecture is required to enable the storage and management of supporting artefacts for the lifecycle management of controllers. NOTE 1 – Examples of supporting artefacts are knowledge, AI/ML or other types of models, workflow representations and policies which need to be applied while managing the lifecycle of controllers. NOTE 2 – Examples of management of supporting artefacts are storage in a KB, creation, modification, deletion, and storage of AI/ML models in an ML model repository [b-ITU-T Y.3176] and of policies, query and discovery of various artefacts.
AN-arch-knw-req-005	AN architecture is required to support the production of human and machine-readable reports of periodic or aperiodic nature.
AN-arch-knw-req-006	AN architecture is required to support capabilities for import and export of controller specifications at various stages of their management. NOTE – Examples of various stages of management of controller specifications are before and after exploratory evolution.
AN-arch-knw-req-007	AN architecture is required to support the integration of derivation mechanisms provided by a third party for metrics from collected parameters and measurements. NOTE – An example of derived metrics is quality of experience (QoE) while an example of collected measurements is a quality of service (QoS) parameter.
AN-arch-knw-req-008	AN architecture is required to support the capture of both service KPI requirements, as well as deployment preferences and considerations in the intent.
AN-arch-knw-req-009	AN architecture is required to provide capabilities for the AN to capture domain specificities. NOTE 1 – Examples of domain specificities are latency criteria, location information and data privacy requirements. NOTE 2 – Examples of capturing domain specificities are TOSCA service definitions. The design of controllers may also be represented using TOSCA declarative definitions.

Requirement	Description
AN-arch-knw-req-010	AN architecture is required to provide the ability for the AN to capture service specificities. NOTE – Examples of service specificities include service level requirements for QoS.
AN-arch-knw-req-011	AN architecture is required to support the AN capability to use inputs from an external environment and user specific models to design as well as apply controller outputs to underlay networks. NOTE – An example of input from external environments is a mobility prediction model for users with assistive needs or groups of users.
AN-arch-knw-req-012	AN architecture is required to support the AN capability to use user preferences while designing and applying controller outputs to underlay networks. NOTE – Standard representations of user profiles or preferences, or user models with assistive needs are examples of user preferences.
AN-arch-knw-req-013	AN architecture is required to provide means for the AN to integrate data collection mechanisms. NOTE – Data collection mechanisms may include AR/VR glasses or other types of sensor. Data collection mechanisms may be provided by third parties.
AN-arch-knw-req-014	AN architecture is required to enable discovery by the AN of capabilities available in the various domains of underlay networks. NOTE – Capabilities of the underlay networks may differ based on their resource availability, e.g., compute, memory, already deployed controllers.
AN-arch-knw-req-015	AN architecture is required to support means for the AN to use an interoperable format for storing controllers. NOTE – Various components in AN may read and write from the stored controllers, e.g., an evolution controller may read existing controllers (even from third parties) and utilize them to compose new ones, which are in turn written in the storage.
AN-arch-knw-req-016	AN architecture is required to support description of use cases in a declarative format.
AN-arch-knw-req-017	AN architecture is required to support derivation of domain-specific intents from the use case description. NOTE – ML intent [ITU-T Y.3172] is an example of domain-specific intent.
AN-arch-knw-req-018	AN architecture is recommended to enable the analysis and correlation of domain-specific, unstructured data in natural languages from underlay networks. NOTE 1 – Examples of domain-specific unstructured data in natural languages are logs from NFs. NOTE 2 – Advances in analysis of natural language text may be exploited from third party models and repositories.
AN-arch-knw-req-019	AN architecture is recommended to support capabilities to derive knowledge from the analysis and correlation of domain-specific unstructured data in natural languages from underlay networks.
AN-arch-knw-req-020	AN architecture is recommended to enable the transfer by the AN of user specific models between different domains in the underlay network. NOTE – This may help in updating of models in other domains, updating of simulators.
AN-arch-knw-req-021	AN architecture is recommended to provide means for the AN to use virtual models along with real-world input to analyse and optimize the underlay network and to provide feedback to operators.

Requirement	Description
	NOTE – Analysis may use AI/ML models. Optimization may involve configurations in the underlay network. Feedback to operators may be generated in AR/VR formats.
AN-arch-knw-req-022	AN architecture can optionally support the ability to integrate in the AN third party modules or applications for collection, analysis, or feedback. NOTE – For example, an SDK may be exposed to third party developers who may develop new applications to analyse AR collected data.
AN-arch-knw-req-023	AN architecture can optionally enable the creation of use case descriptions that can then be decomposed to controllers, which can be deployed at various domains of the underlay network, based on the capabilities at those levels of the underlay network. NOTE – Use case descriptions may be in the form of intents.
AN-arch-knw-req-024	AN architecture can optionally support the ability of the AN to discover the characteristics of underlay networks at run-time.

7.5 Requirements for autonomous network orchestration

The following are requirements with respect to orchestration in ANs.

Requirement	Description
AN-arch-ano-req-001	AN architecture is required to support the ability to parse, validate and translate abstracted use case descriptions, with high-level objectives of a controller into controller designs. NOTE 1 – The abstracted use case descriptions may be hand crafted as unstructured text or derived from controller specifications. NOTE 2 – Controller designs may be provided using structured languages formats (e.g., TOSCA [b-TOSCA]) and may be structured in a way that facilitates downstream exploration, experimentation, and adaptation. NOTE 3 – Controller designs may use and enable properties derived from various domains in the network, e.g., properties allowing description of use cases of physical layer, network layer and application layer. NOTE 4 – Examples of use cases are: a) RCA and diagnosis of network elements based on real-time analysis of data – see FG-AN-usecase-006 in [b-ITU-T Y-Suppl. 71]. b) Intelligent energy-saving solution based on automatic data acquisition, AI-based energy consumption modelling and inference, facilities parameters control policies decision, facilities adjustment actions implementation, energy-saving result evaluation and control policies continuous optimization – see FG-AN-usecase-007 in [b-ITU-T Y-Suppl. 71]. c) Optimal adjustment of antenna parameters with AI-enabled multi-dimensional analysis and prediction – see FG-AN-usecase-008 in [b-ITU-T Y-Suppl. 71]. d) Management of third party vertical applications and related services in the network – see FG-AN-usecase-010 in [b-ITU-T Y-Suppl. 71].
AN-arch-ano-req-002	AN architecture is required to have the ability to manage the lifecycle of controllers. NOTE 1 – Examples of management tasks of a controller's lifecycle include creating a configuration for the controller (based on the capabilities of the underlay network), creating an instance of the controller in the underlay network, monitoring the execution of the controller in the underlay network and subsequently optimizing controllers or related parameters. NOTE 2 – Examples of subsequent optimization are recommendations on new AI/ML analysis techniques, data collection techniques, evolution of controllers to move up the intelligence level [b-ITU-T Y.3173].

Requirement	Description
AN-arch-ano-req-003	AN architecture is required to enable the management of lifecycle of controllers based on their output. NOTE – Examples are management of lifecycle of controllers in underlay network (such as RAN), by controllers in underlay network (e.g., CN), creation and optimal positioning of controllers in the RAN by controllers in a higher domain, as well as evolution, experimentation and deployment of controllers.
AN-arch-ano-req-004	AN architecture is required to enable the consumption of services exposed by service management frameworks used by underlay networks. NOTE 1 – Services exposed by service management frameworks include configuration, lifecycle management and customization. NOTE 2 – Examples of service management frameworks include ETSI zero touch service management (ZSM) framework [b-ETSI GS ZSM 009-1].
AN-arch-ano-req-005	AN architecture is required to support capabilities enabling AN adaptation to the evolution of underlay network services, which may take place independently of AN evolution. NOTE – The modification of underlay network services may be managed by an entity different to that managing AN evolution.
AN-arch-ano-req-006	AN architecture is required to utilize the available connectivity options provided by the underlay networks to deploy and integrate controllers in different levels of underlay networks.
AN-arch-ano-req-007	AN architecture is required to support the monitoring of changes to underlay network connectivity among controllers deployed in various domains.
AN-arch-ano-req-008	AN architecture is required to support the exposure of a single point to monitor and manage AN functionalities deployed by the operator.
AN-arch-ano-req-009	AN architecture is required to support the discovery of changes to NFs in underlay networks and to include the changed functions in the AN while providing AN functionalities like evolution. NOTE – This enables plug and play of new NFs in underlay networks.
AN-arch-ano-req-010	AN architecture is required to support the provision of inputs to external systems regarding potential scenarios and requirements. NOTE – Examples of inputs are reports generated from AN on new use case scenarios, such as those for experimentation.
AN-arch-ano-req-011	AN architecture is required to support the ability of the AN to create, store, customize and export controllers that can be deployed in various types of underlay network.
AN-arch-ano-req-012	AN architecture is required to support interfaces between the AN and resource orchestration functions in the underlay network. NOTE 1 – Examples of resource orchestration mechanisms are not only network function virtualization MANO but also domain-specific resource managers like multi-user schedulers in RAN. NOTE 2 – The AN interfaces may be used to trigger actions or monitoring.
AN-arch-ano-req-013	AN architecture can optionally support the ability of the AN to use third party toolsets for the development and visualization of controllers. NOTE – Graphical user interfaces to edit workflows are examples of third party toolsets.
AN-arch-ano-req-014	AN architecture can optionally support input of use case design within the AN, whereas design and deployment of controllers in underlay networks may be done by third parties.

8 Architecture framework description

8.1 High-level architecture framework

This clause describes the high-level architecture framework for ANs. The goal of this architecture is to support continuous evolution-driven creation, validation and application of a set of controllers to a network and its services to render them autonomous.

As shown in Figure 1, the high-level architecture framework for ANs consists of the autonomy engine (clause 8.3.1), the dynamic adaptation subsystem (clause 8.3.2), the KB system (clause 8.3.3), the AN orchestrator (clause 8.3.4), the underlay network (clause 8.3.5), and the E2E network orchestrator (clause 8.3.6).

Clause 8.3 describes them in detail; they are briefly introduced here. The autonomy engine embodies the key concepts of exploratory evolution and online experimentation and is responsible for the creation and validation of controllers. The dynamic adaptation subsystem embodies the key concept of dynamic adaptation and is responsible for equipping the underlay network with autonomy via controllers. The KB system manages the lifecycle and optimization of knowledge. The AN orchestrator is responsible for managing the workflows and processes in the AN, as well as the steps in the lifecycle of controllers. The underlay network is a telecommunication network with its NFs. The E2E network orchestrator is responsible for managing and orchestrating control entities within the AN, including the underlay.

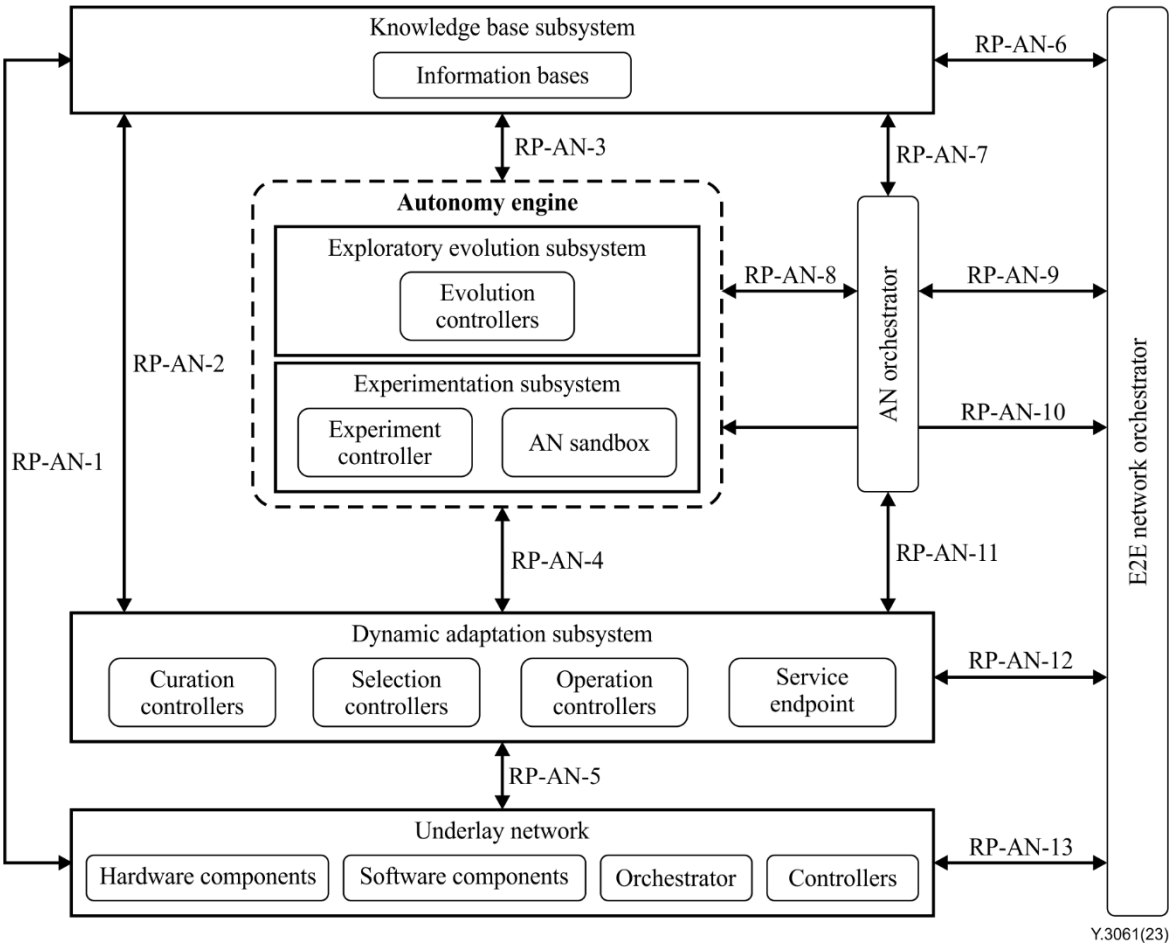

Figure 1 – High-level architecture framework for autonomous networks

The reference points, as shown in Figure 1, are as follows.

RP-AN-1, RP-AN-2, RP-AN-3 and RP-AN-6: Reference points between the KB subsystem and underlay network, dynamic adaptation subsystem, autonomy engine, E2E network orchestrator and AN orchestrator respectively. As discussed in clause 7.4, these reference points enable access to the KB from other subsystems in the architecture framework.

RP-AN-4: Reference point between the autonomy engine and dynamic adaptation subsystem. This reference point is used to provide evolutionary exploration and experimentation functionalities to the dynamic adaptation subsystem.

RP-AN-5: Reference point between the dynamic adaptation subsystem and underlay network. This reference point is used to provide selection and integration of controllers to an underlay, as the underlay undergoes changes at run-time.

RP-AN-7, RP-AN-8 and RP-AN-11: Reference points between the AN orchestrator and KB, autonomy engine and dynamic adaptation subsystem, respectively. These reference points enable the AN orchestrator to manage the workflows and processes in the AN, and the lifecycle of controllers.

RP-AN-9, RP-AN-10, RP-AN-12: Reference points between the E2E network orchestrator and AN orchestrator, autonomy engine and dynamic adaptation subsystem, respectively. These reference points are used by the E2E network orchestrator to manage and orchestrate control network entities in the AN framework. These reference points may use existing procedures as specified in [b-ITU-T Y.3100].

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

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

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

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

NOTE 2 – The details of the interaction of these external functionalities with the architecture framework through the architecture framework reference points lie outside the scope of this Recommendation.

8.2 Description of controller

In this architecture, the term controller is introduced. As introduced in clause 6, a controller is a workflow, open loop or CL [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 within it to solve a specific problem or satisfy a given requirement.

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

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

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

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

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

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

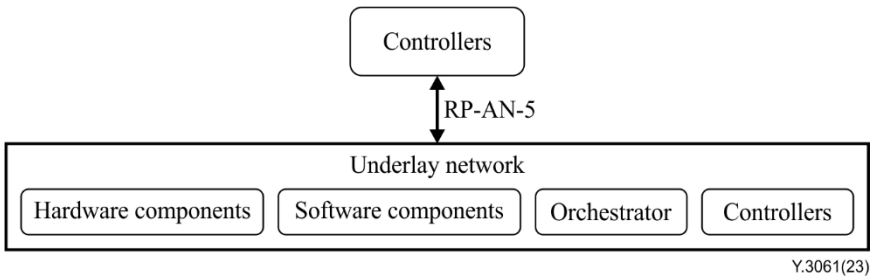

Figure 2 – Controllers and underlay network interaction

The controller interactions are with:

- hardware components [b-Umuroglu];
- software components;
- an orchestrator or other software control mechanisms;

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

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

8.3 Description of the sub-systems and their components

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

8.3.1 Autonomy engine

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

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

8.3.1.1 Exploratory evolution subsystem

Exploratory evolution enables exploration and evolution to adapt controllers in response to changes in the underlay network. Knowledge stored in the KB subsystem is used in ANs for supporting continuous exploratory evolution. As explained in clause 8.1, reference point RP-AN-3 allows the exploratory evolution subsystem to interface with the KB subsystem and RP-AN-4 allows exploratory evolution subsystem to interface towards the dynamic adaptation subsystem. Figure 3 shows an overview of the exploratory evolution subsystem and its relation to the KB subsystem as new controllers are generated or existing controllers are updated (evolved) as part of exploratory evolution.

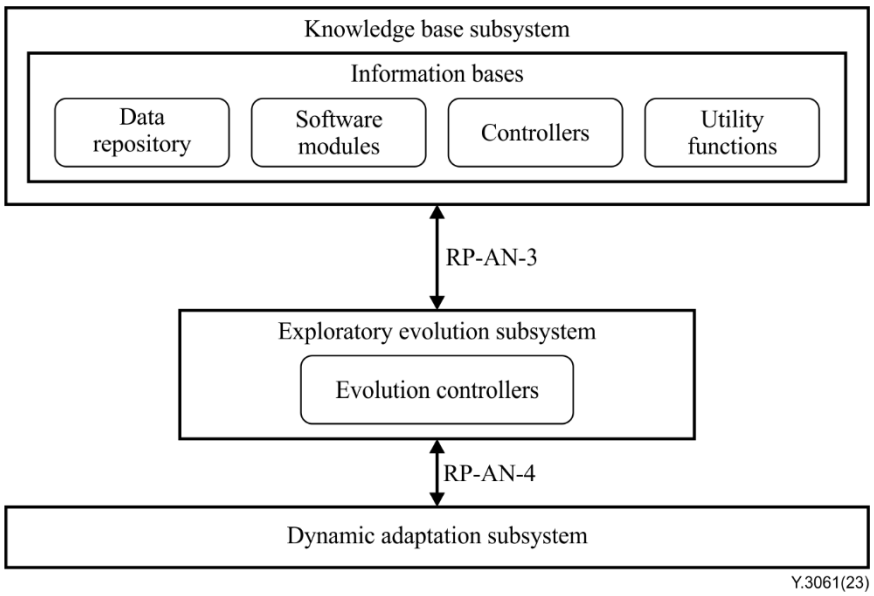

Figure 3 – Exploratory evolution overview

As stated in clause 0, any approach towards an AN 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 0, this architecture requires the ability to alter the logic that is used to operate ANs (i.e., controllers). Without such functionality, it is not possible to achieve adaptation that is sufficiently flexible across the spectrum of use cases, operational environments, technological innovations and potential human errors.

NOTE 1 – It is important to remember that controllers may themselves possess the ability to adapt their outputs based on learning or experience – so called *cognitive* controllers [b-Mwanje]. 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 (i.e., a new transport protocol) or to handle error. In all cases, human intervention is required.

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

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

Hence, the evolutionary exploration subsystem is responsible for:

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

Exploratory evolution will enable automated design or modification of controllers and their hierarchies to explore 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 AI techniques, such as evolutionary computing [b-Whitley].

8.3.1.1.1 Evolution controller

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

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

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

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

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

The process of exploratory evolution is agnostic about 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 KB, applying such evolutions to various types of controller. 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 cases, provided by AN orchestrator, is used as input by the evolution controller. Controller designs 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-Anderson], [b-Whitley];
2) Bayesian optimization [b-Maggi];
3) game theoretic approaches [b-Ahmad].

Specific algorithms, including those provided by third party solution providers, used for exploratory evolution lie outside the scope of this Recommendation.

NOTE 4 – Examples of application of exploratory evolution in various application contexts follow.

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 user equipment (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 NF. In this context, the new approaches of data fusion algorithms [b-Bleiholder] may be applied. Exploratory evolution is applied to an anomaly detection controller by optionally using and configuring newly provided data fusion algorithms as input to it.
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 content distribution network server. In a time-to-live controller in a caching system at the edge, optimization 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 an NF. 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 – Optimization 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 optimization scenarios lie outside the scope of this Recommendation.

8.3.1.2 Experimentation subsystem

The experimentation subsystem is concerned with the validation of controllersError! Reference source not found.. It designs, orchestrates and executes experimental scenarios. These are supported by a KB subsystem, AN orchestrator and E2E network orchestrator.

Figure 4 shows an overview of the experimentation subsystem and its relationship with the KB subsystem, AN orchestrator and E2E network orchestrator through various reference points described in clause 8.1.

Reference point RP-AN-3 allows the experimentation subsystem to interface with the KB subsystem as experimentations are designed and updated as part of experimentation process. Reference point RP-AN-4 allows the experimentation subsystem to interface with dynamic adaptation subsystem. Reference points RP-AN-8 and RP-AN-10 allow the experimentation subsystem to orchestrate the experimentations.

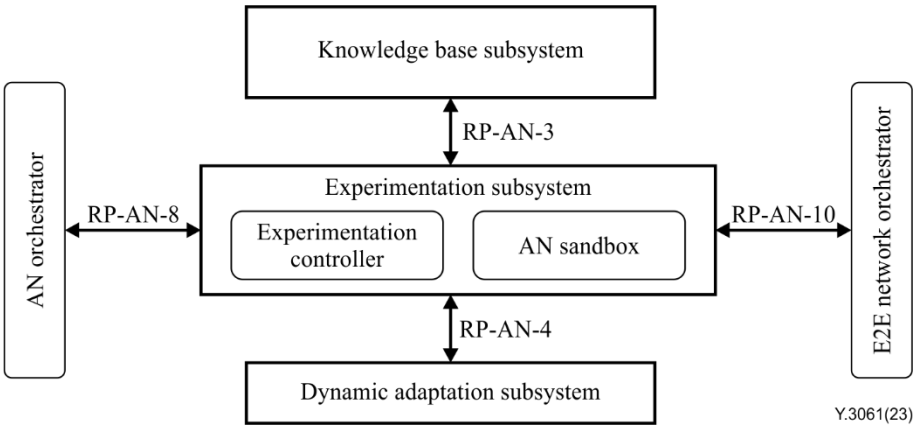

Figure 4 – Experimentation subsystem

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

Validation is a spectrum of activities that may encompass one or more tasks, including static testing, simulation, testbed deployment and canary testing. In addition, validation can also be used to assess non-functional properties, such as trust, providing confidence in the "handover of work, duties, or decisions" to the architecture.

To compliment these validation activities, the experimentation controller also requires additional input from the underlay network and its configuration. As shown in Figure 4**Error! Reference source not found.**, this information should be stored and made available from the KB. Representative examples of such data are discussed in clause 8.3.3.

8.3.1.2.1 Experimentation controller

Experimentation is the process that validates controllers using inputs from a combination of underlay network, simulators 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 that generates potential scenarios for experimentation based on controller designs and representations of the use cases. The experimentation controller uses

additional information, as provided by the KB and AN orchestrator, in the process of generating scenarios for experimentation.

NOTE 1 – Methods for generating scenarios for experimentation are assisted by additional information including knowledge captured in the KB or ML. The experimentation controller may exploit the structured representation (e.g., TOSCA yet another meta language (YAML) [b-TOSCA]) of the controllers to derive scenarios for experimentation. Experimentation scenarios can also be provided by third parties for use by the experimentation controller.

In addition to generating scenarios for experimentation, the experimentation controller executes the scenarios in the AN sandbox, collates and validates the results of the experimentation. Reports may be generated by the 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, the experimentation controller can use different types of component such as simulators, data generators hosted in the AN sandbox, including those provided by third parties. Experimentation may be triggered by various AN workflows that necessitate validation of controllers, i.e., their software updates. The process of experimentation may be configurable, e.g., it may be triggered periodically, asynchronously.

NOTE 4 – Examples of experimentation in various application contexts follow.

- The use of static sanity checking, such as formal methods [ITU-T Y.3320] or model checking to ensure that provided MANO solutions are well formed against pre-set 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-Almasan] 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 preceding to achieve broader coverage of validation, from the offline validation to online validations during the operation of the underlay.

8.3.1.2.2 AN sandbox

An 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 – The AN sandbox generates reports regarding the experimental validation of controllers. These reports are collated by the experimentation controller and the KB 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, the AN sandbox simulates scenarios that are rarely or never seen in underlays. For example, a burst of traffic which rarely occurs in real network.

An AN sandbox hosts different types of components such as simulators, data generators, including those provided by third parties. Experimentation controller may trigger experiments of various AN workflows that necessitate validation of controllers, i.e., software update of controllers.

8.3.2 Dynamic adaptation subsystem

Dynamic adaptation equips the underlay network with autonomy and the ability to handle new and hitherto unseen changes in network scenarios. Knowledge stored in the KB subsystem is used in ANs to support continuous dynamic adaptation. The AN orchestrator and E2E network orchestrator support the orchestration needed for dynamic adaptation towards an underlay network.

As explained in clause 8.1, reference point RP-AN-2 allows the dynamic adaptation subsystem to interface with the KB subsystem. Reference points RP-AN-11 and RP-AN-12 allow the dynamic adaptation subsystem to interface with the AN orchestrator and E2E network orchestrator, respectively. Reference point RP-AN-4 allows the dynamic adaptation subsystem to interface with experimentation subsystem and exploratory evolution subsystem. RP-AN-5 allows the dynamic adaptation subsystem to interface with underlay network. RP-AN-13 allows the E2E network orchestrator to interface with underlay network.

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

NOTE 1 – Here, fitness for purpose is evaluated based on the fitness function or utility score obtained from the experimentation subsystem (clause 8.3.1.2).

This set of controllers is drawn from the controllers that were validated by the experimentation 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 and QoE are expected to be represented within a controller's utility function.

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

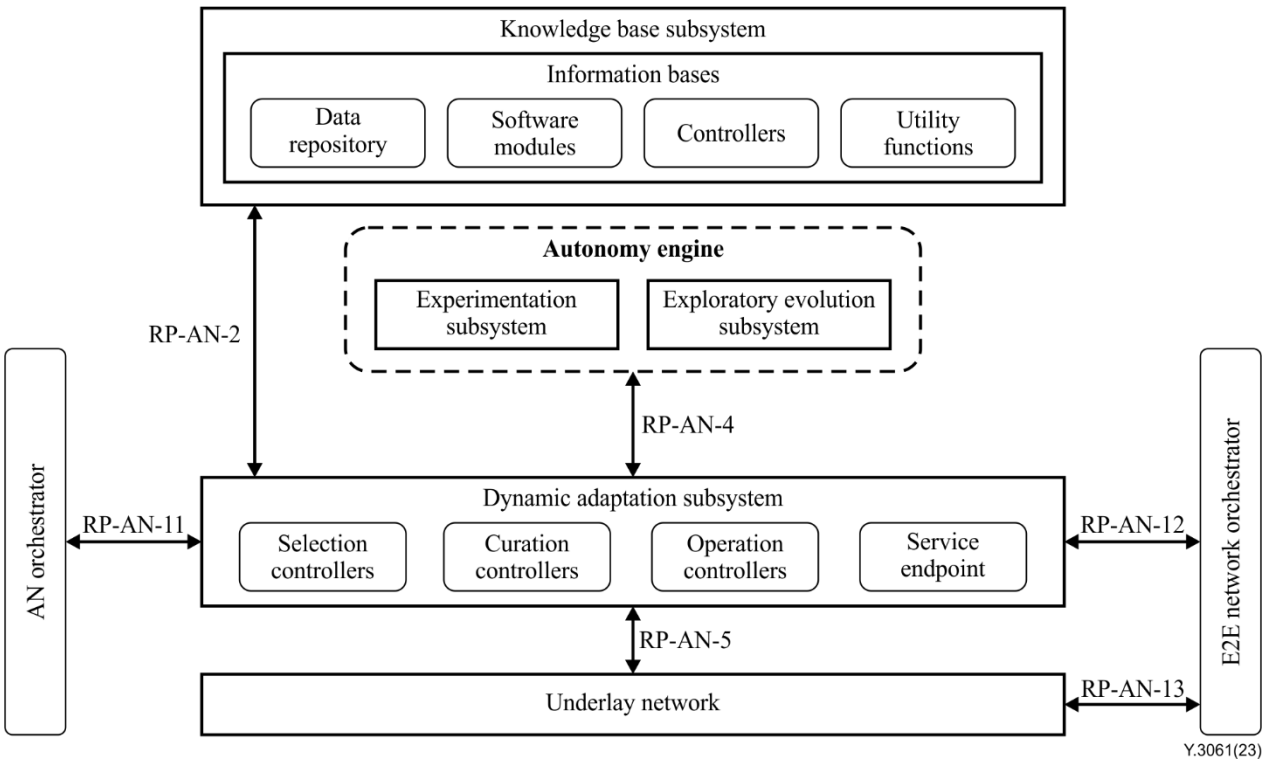

Figure 5 – Dynamic adaptation subsystem

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

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

Selection and integration of controllers to a managed entity require a stable set of functioning controllers that can respond correctly in subsecond timescales, depending on the use case in question.

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

8.3.2.1 Adaptation controller

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

NOTE 1 – Examples of underlays are edge networks, CNs, management plane and CI/CD pipelines. 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 third 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 (CP) via DNS updates or SDN configurations, see FG-AN-usecase-040 in [b-ITU-T Y-Suppl. 71];

- scaling via the number, replication and distribution of bare metal, virtual machine or container resources attached to network services, see FG-AN-usecase-018 in [b-ITU-T Y-Suppl. 71];
- scaling via the priority or relative bandwidth allocation of radio spectrum to different user or use case categories in RAN, see FG-AN-usecase-032 in [b-ITU-T Y-Suppl. 71].

NOTE 2 – Specific configurations for integration of controllers for specific use cases may lie outside the scope of this Recommendation.

An adaptation controller is the component in an AN responsible for selecting candidate controllers from a set of generated controller configurations that are ready for integration and executes the integration to the underlay. An adaptation controller monitors deployed controllers and the underlay, deciding opportunities for new controller integrations to the underlay. In monitoring a deployed controller, an adaptation controller discovers underlay specific parameters (including those provided by third parties) for optimization, data points of collection and KPIs for tracking, and may update such knowledge to the KB.

An adaptation controller has two parts: a curation controller (responsible for selection and maintenance of the controllers within the curated controller lists from the evolvable controllers) and a selection controller (responsible for the selection of an operational controller of services from the curated controller lists).

Generation of configurations for adaptation may take the controller design 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 an adaptation controller that captures information from the process of adaptation. These reports may be shared with humans or used for analysis by algorithmic methods. An adaptation process may be optimized as a result of analysis of the reports.

NOTE 3 – Examples of adaptation in various application contexts follow.

- The need to use different traffic shaping algorithms for various geographical contexts, such as urban vs rural.
- Business priorities may change over time, e.g., prioritization of performance KPIs over energy efficiency or prioritization of internal applications over third party applications. These changes in business priorities may necessitate the use of different controllers for scheduling virtual machines or containers.
- There could be a need to deploy new technology in order to improve or optimize operation, including adding new capabilities that previously did not exist, i.e., 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, i.e., 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 determined against a controller's performance;
- replacement, based on a PID controller [b-Bennet], of one deployed controller with another;
- the use of an AI/ML model in the prediction of future operation and response to pre-emptively exchange a deployed controller [b-ITU-T Y-Suppl. 71];
- combinations of the preceding in concert with knowledge stored within the KB.

8.3.2.2 Operation controller

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

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

8.3.2.3 Service endpoints

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

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

8.3.3 Knowledge base subsystem

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

Data, information and knowledge are required for the controllers to operate the managed entity in its goals of supporting the continuous exploratory evolution, real-time online experimental validation and dynamic adaptation.

A KB is a subsystem that manages storage, querying, export and import, as well as optimization and update knowledge.

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

Managing knowledge includes its storage, querying, export, import and optimization. AN workflows, including exchange of knowledge between AN components, may in turn result in update of the KB.

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

Examples of knowledge stored in a KB follow.

1) Relevant descriptions of modules and controller metadata taxonomies and ontologies.
2) Configurations: 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 real-time experimentation 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-TOSCA]
3) Metrics: metrics are the data related to the status and performance of different components of the architecture, controllers, operating environment, underlay network and managed

entities, e.g., resource usage such as central processing unit usage, workload such as packet rate, performance metrics such as QoS.

8.3.4 AN orchestrator

The 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, the AN orchestrator coordinates with various other functions within and outside the AN.

NOTE 1 – Examples of workflows and processes in the AN are interactions with the E2E network orchestrator, KB and AN component repositories. Examples of controller instances are:

- a set of Java objects to be executed on the Java virtual machine;
- 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-Kumar];
- an ML pipeline [ITU-T Y.3172].

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

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

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

8.3.5 Underlay network

An underlay network is a telecommunication network with its NFs. An underlay network may provide interfaces that facilitate application of controllers.

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

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

8.3.6 E2E network orchestrator

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

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

9 Sequence diagrams

This clause gives the sequence diagrams showing the interactions between architecture framework components and subsystems. Specific reference points over which the messages are exchanged are not shown here, but all the reference points are shown in Figure 1.

9.1 Exploratory evolution of controllers

Exploratory evolution of controllers involves creation and modification of controllers in accordance with the underlay network and the real-time changes therein. An example scenario where controllers are created is shown in Figure 6. In this example, the AN operator provides a new use case specification from which new controller specifications are derived. There are additional example

scenarios in which the evolution controller reuses an existing controller specification and applies the exploratory evolution process.

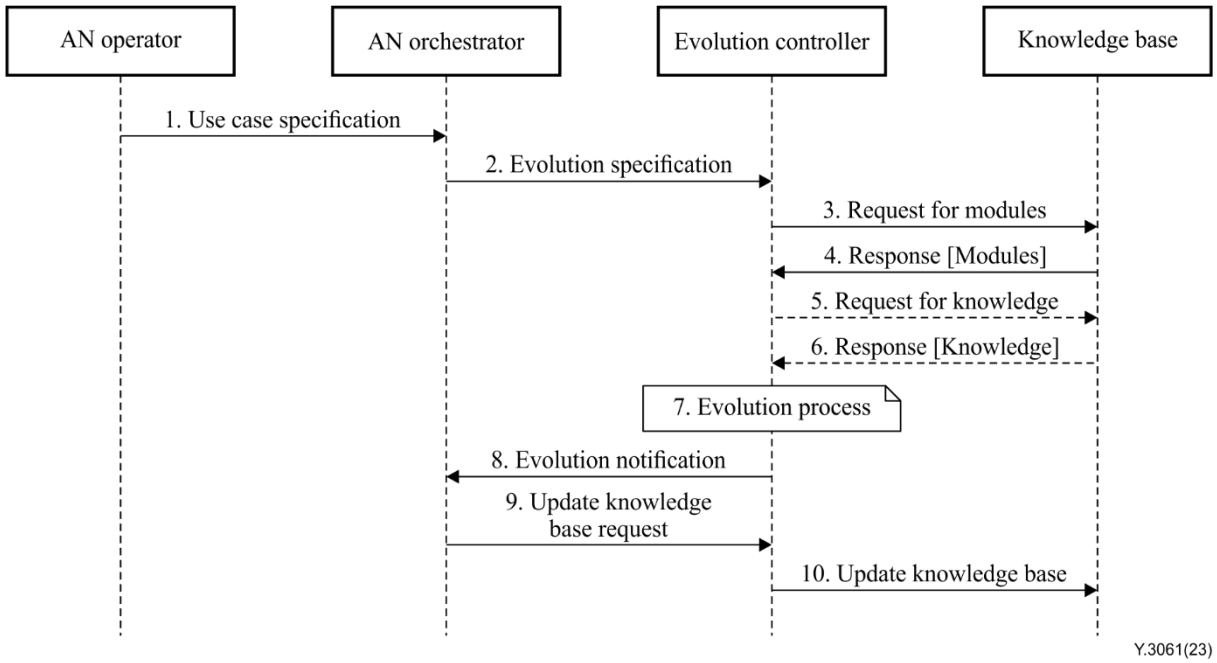

Figure 6 – Creation of controllers

The steps involved in the scenario described in Figure 6 are as follows.

1) The AN operator provides a use case specification to the AN orchestrator. The use case specification includes the actors, their relationships and utility functions corresponding to the use case.
2) The AN orchestrator derives an evolution specification from that of the use case. The evolution specification has a controller specification with the metadata corresponding to the necessary functionality of the controller and a utility function to be achieved (after the exploratory evolution process).
3) The evolution controller queries the KB for modules corresponding to the controller specification.
4) The KB 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 KB relevant to the use case specification or the exploratory evolution process.
6) Corresponding to the optional step 5, the KB Base responds with the knowledge requested.
7) The evolution controller applies the exploratory evolution process to create new controller(s). This includes composition of controllers from modules or other CLs as described in clause 8.2.
8) The evolution controller updates the KB. This includes storing the generated controllers in the KB.

NOTE – Discussion of the logic used to drive the exploratory evolution process lies outside the scope of this Recommendation. Examples of such processes can be found in clause 8.3.1.1.

9.2 Experimentation for controllers

Experimentation for controllers involves their validation using inputs from a combination of underlay network, simulators or testbeds. An example scenario where evolvable controllers are validated is

given in Figure 7. In this example, a new experiment specification, which has controller specifications to be validated, is provided by the AN orchestrator. The experimentation controller derives scenarios for experimentation based on the experiment specification. Based on these scenarios, the experimentation controller interacts with the KB to gather additional supporting specifications (experiments or controllers) and relevant knowledge to design an experiment to validate the controllers included in the experiment specification.

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

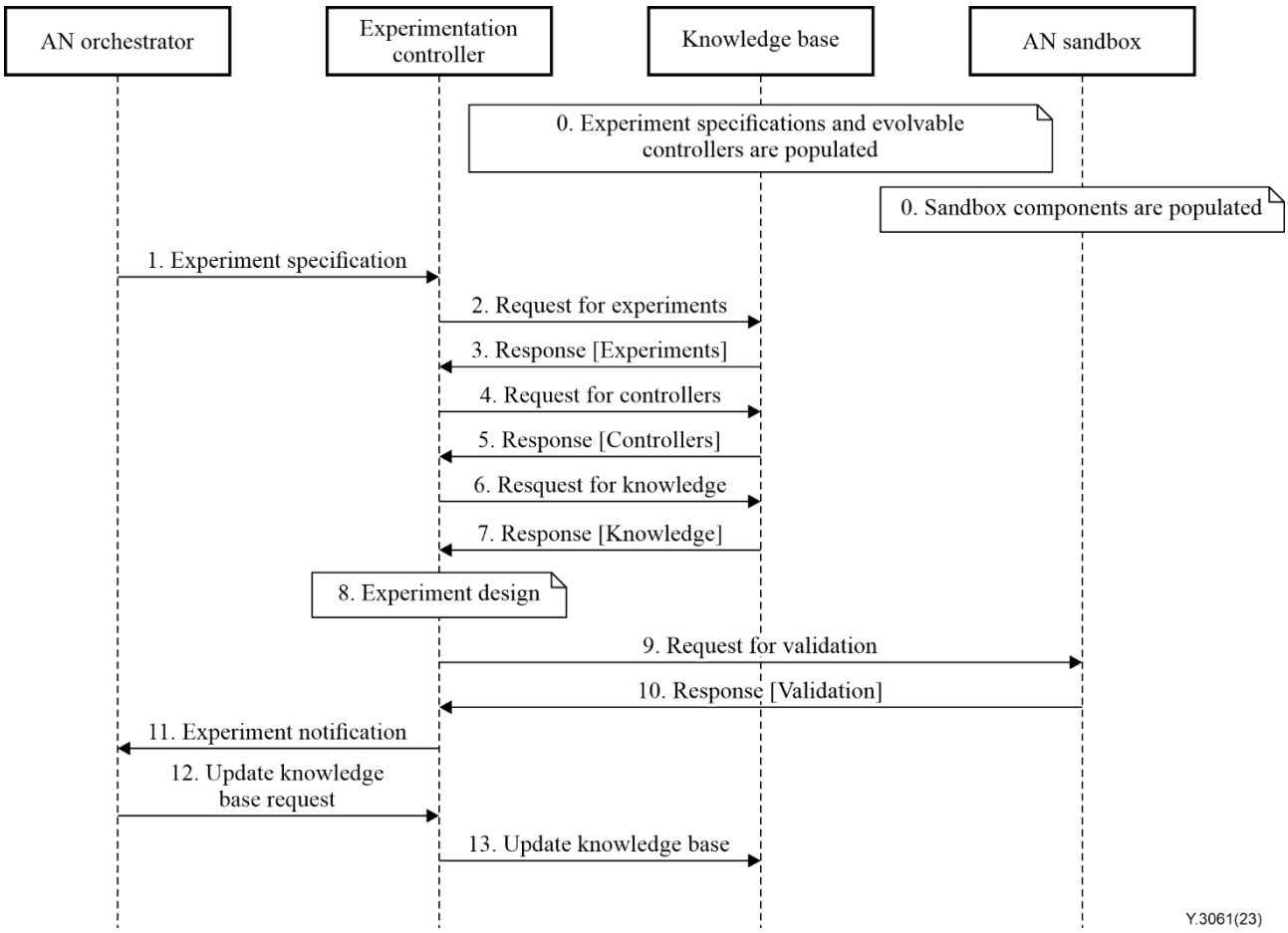

Figure 7 – Validation of controllers

Pre-requisites follow:

Experiment specifications and evolvable controllers are populated in the KB. This may be done based on creation of controllers in Figure 6 or based on offline provisioning by AN operator. In addition, the AN sandbox is populated with components that are ready for instantiation and execution to validate the controllers.

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

1) the AN orchestrator provides experiment specification that has the controller specification for the controller to be validated;
2) the experimentation controller requests the current list of experiments from the KB;
3) the KB replies with the requested data, if any;
4) the experimentation controller requests the current list of controllers from the KB;
5) the KB replies with the requested data, if any;

6) the experimentation controller requests additional knowledge from the KB, needed to support the experiment design;

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

7) the KB replies with the requested data, if any;

8) the experimentation controller designs potential experimentation scenarios;

9) for each experimentation scenario, the experimentation controller requests the AN sandbox to perform the validation;

10) the AN sandbox reports the results to the experimentation controller;

11) the experimentation controller performs any necessary analysis of the results and notifies the AN orchestrator;

12) the AN orchestrator triggers the experimentation controller to update the KB;

13) the experimentation controller updates the KB – this includes storing the experiment results for the validated controllers in the KB.

NOTE 3 – Discussion of the logic used to drive the experiment design lies outside the scope of this Recommendation. Examples of such processes can be found in clause 8.3.1.2.

9.3 Dynamic adaptation of controllers

Dynamic adaptation is the process of continuous integration of controllers to an underlay, as the underlay undergoes changes at run-time. An example scenario where validated controllers are curated, selected and deployed to the underlay is shown in Figure 8.

In this example, the AN orchestrator provides the curation controller with an adaptation specification, which contains controller specifications, to drive the curation process. The curation controller queries the KB for validated controllers and relevant knowledge. Then the AN orchestrator provides the selection controller with an adaptation specification, which contains controller specifications, to drive the selection process.

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

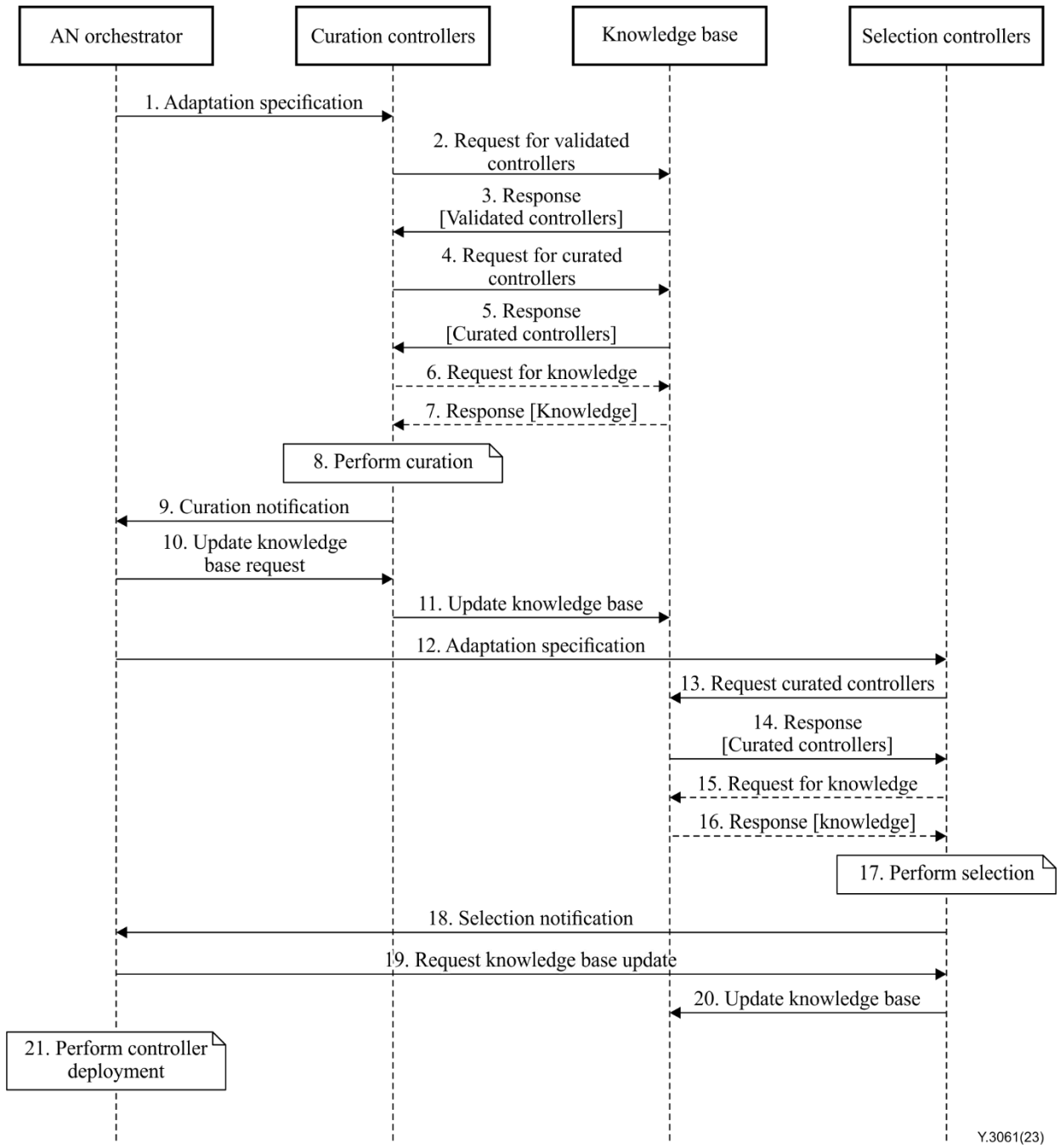

Figure 8 – Dynamic adaptation of controllers

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

1) the AN orchestrator provides the curation controller with an adaptation specification, which contains 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 KB;
3) the KB replies with the requested data, if any;
4) the curation controller requests the list of curated controllers for the use case from the KB;
5) the KB replies with the requested data, if any;
6) the curation controller requests additional knowledge from the KB 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 KB replies with the requested data, if any;

8) the curation controller performs the curation process that 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 lies outside the scope of this Recommendation. Examples of such processes can be found in clause 8.3.2.1.

9) the curation controller notifies the AN orchestrator that it has completed the curation process;

10) the AN orchestrator requests the curation controller to update the KB with the curated list of controllers;

11) the curation controller updates the KB 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 KB;

14) the KB replies with the list of requested data, if any;

15) the curation controller requests additional knowledge from the KB needed to support the curation process;

NOTE 4 – 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.

16) the KB replies with the requested data, if any;

17) the selection controller performs the selection process which decides the curated controller that should be deployed to the underlay – the deployed controllers are known as operational controllers for the specified use case, if any;

NOTE 5 – Discussion of the logic used to drive the selection process lies outside the scope of this Recommendation. Examples of such processes can be found in clause 8.3.2.1.

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 KB with the controller to be deployed as the operational controller;

20) the selection controller updates the KB;

21) the AN orchestrator performs the necessary lifecycle actions to deploy the operational controller for the use case.

10 Security considerations

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

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

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

Appendix I

An example realization of the architecture framework for autonomous networks with technology specific underlays

(This appendix does not form an integral part of this Recommendation.)

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

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

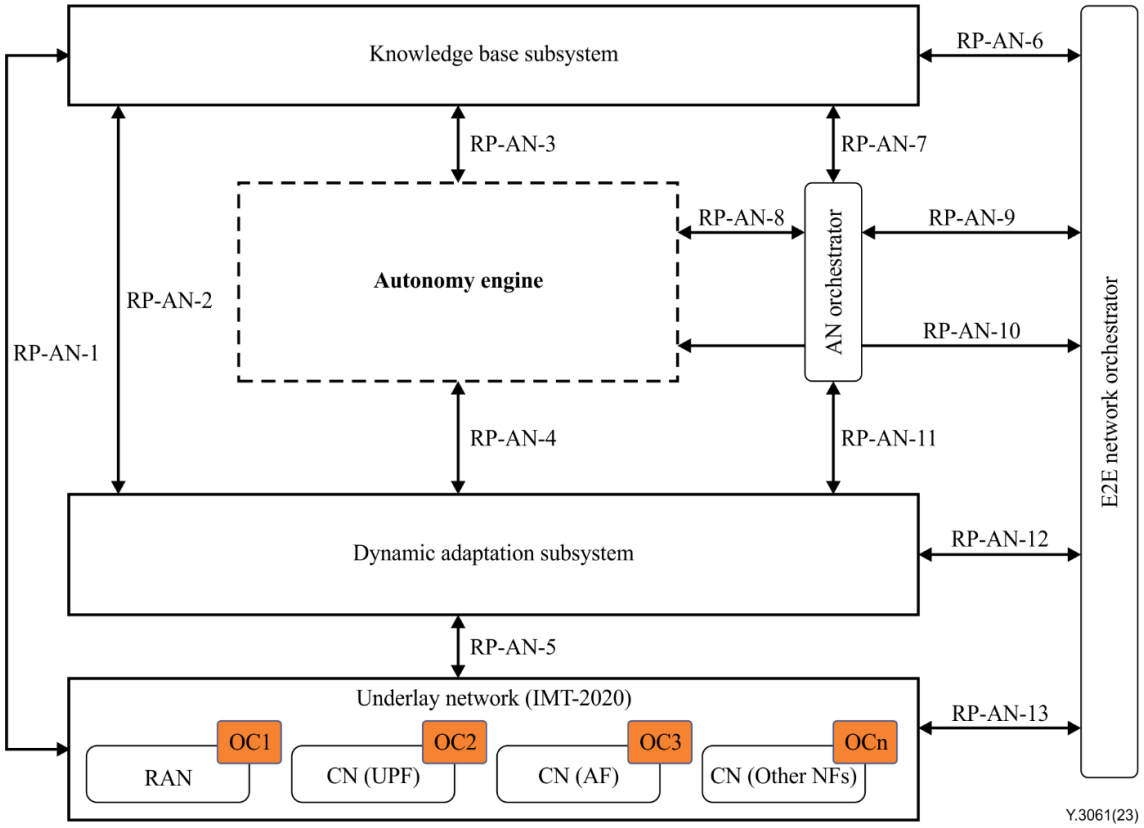

Figure I.1 – Example of a realization of the AN architecture framework with an IMT-2020 network underlay

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

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

I.1 Examples of deployment locations of controllers

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

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

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

The dynamic adaptation subsystem interfaces with the AN orchestrator to deploy the OCs to the underlay via the E2E orchestrator. This can be realized via reuse of reference points described in [b-ITU-T Y.3111].

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

NOTE 2 – While the deployment of an operational controller on UE [b-ITU-T Y.3104] may be technically possible, the associated operational challenges, such as data privacy or security concerns, lie outside the scope of this Recommendation. Examples of such an operation controller may include one tasked with monitoring the state of a service in UE as input to E2E decision-making (e.g., QoE measurements for media-streaming services).

I.2 Example realization of exploratory evolution

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

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

I.3 Example realization of online experimentation

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

As shown in steps 6 to 9 of Figure 7 (clause 9.1.2), based on data stored in the KB regarding the status and configuration of the access network in the underlay (steps 6 and 7 of Figure 8), the experimentation controller (clause 8.3.1.2) generates possible experimental scenarios for OC1 (step 8 of Figure 7).

I.4 Example realization of dynamic adaptation

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

As shown in steps 15 to 21 of Figure 8 (clause 9.1.3), based on data stored in the KB regarding the status and configuration of the CN in the underlay, the status and configuration of currently deployed OC2, and information regarding other relevant curated controllers (steps 15 and 16 of Figure 8), the adaptation controller (clause 8.3.2.1) decides which curated controller is to be deployed in the underlay (step 17 of Figure 8) as the new revision of OC2 (step 21 of Figure 8).

Appendix II

Self-reflective use of the AN architecture

(This appendix does not form an integral part of this Recommendation.)

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

The architecture is self-reflective in its operation, i.e., the 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 II.1. Thus, the architecture itself becomes a collection of managed entities.

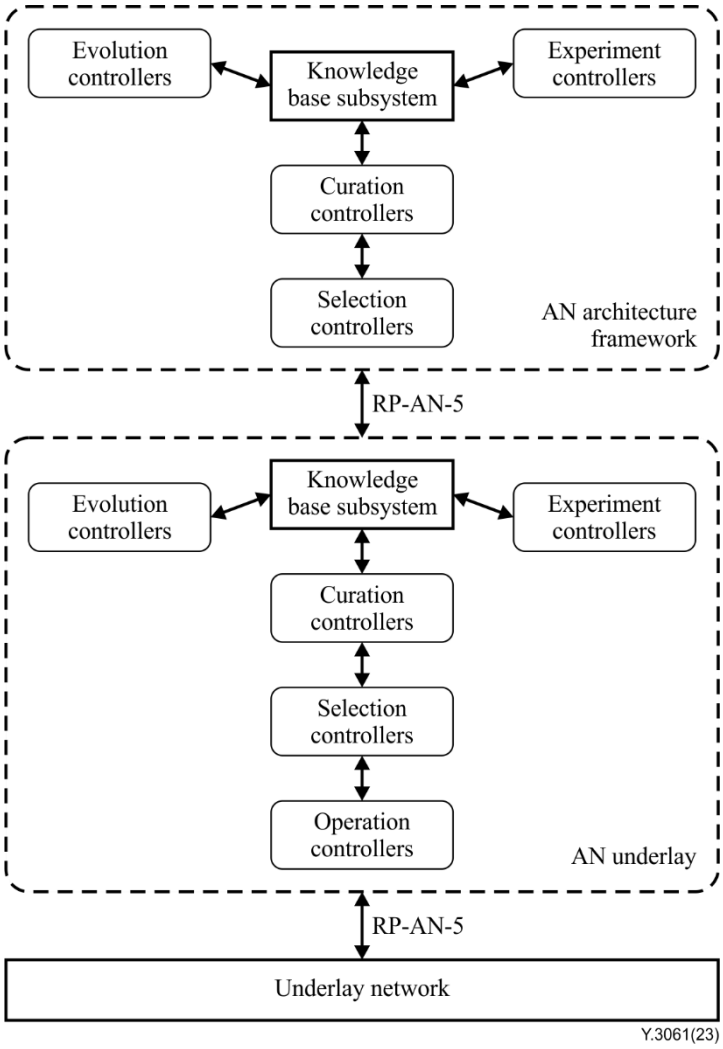

Figure II.1 – Self-reflective use of the AN architecture framework

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

NOTE 2 – Figure II.1 relates to an AN underlay. This concept relates to an instance of the AN architecture framework (or a subset of it) used, in turn, as an underlay of another instance of the AN architecture framework.

Appendix III

External functionalities

(This appendix does not form an integral part of this Recommendation.)

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

Examples may include the following.

- AN controller repositories [b-Dagsthul]: These are repositories that contain 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 KBs implemented within the AN architecture, there are external knowledge repositories used by AN architecture to access such knowledge.
- Domain orchestrators: These may be implemented by third parties and may provide specific functions associated with specific technologies such [b-ETSI GS ZSM 009-1], [b-FRINX], [b-ONAP].
- Development pipelines (CI/CD pipelines): They provide a continuous development environment for components, modules and controllers.
- Model repositories [b-ITU-T Y.3176]: They store the specifications of models used in the AN.

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

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

Bibliography

[b-ITU-T Y.3100] Recommendation ITU-T Y.3100 (2017), *Terms and definitions for IMT-2020 network.*

[b-ITU-T Y.3102] Recommendation ITU-T Y.3102 (2018), *Framework of the IMT-2020 network.*

[b-ITU-T Y.3104] Recommendation ITU-T Y.3104 (2018), *Architecture of the IMT-2020 network.*

[b-ITU-T Y.3111] Recommendation ITU-T Y.3111 (2017), *IMT-2020 network management and orchestration framework.*

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

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

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

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

[b-ITU-T Y-Suppl. 71] ITU-T Y-series Recommendations – Supplement 71 (2022), *ITU-T Y-3000 series – Use cases for autonomous networks*.

[b-3GPP TR 38.801] Technical Report 3GPP TR 38.801 V14.0.0 (2017), *3rd Generation Partnership Project; Group Radio Access Network; Study on new radio access technology: Radio access architecture and interfaces (Release 14)*

[b-ETSI GR ENI 004] Group Report ETSI GR ENI 004 V3.1.1 (2023), *Experiential networked intelligence (ENI); Terminology*.

[b-ETSI GS ENI 005] Group Specification ETSI GS ENI 005 V3.1.1 (2023), *Experiential networked intelligence (ENI); System architecture.*

[b-ETSI GS MEC 012] Group Specification ETSI GS MEC 012 V2.2.1 (2022-02), *Multi-access edge computing (MEC); radio network information API.*

[b-ETSI GS ZSM 009-1] Group Specification 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] Technical Specification ETSI TS 129 500 V17.13.0 (2024), *5G; 5G system; Technical realization of service based architecture; Stage 3 (3GPP TS 29.500 version 17.13.0 Release 17).*

[b-Ahmad] Ahmad, I., Kaleem, Z., Narmeen, R., Nguyen, L.D., Ha, D.B. (2019). Quality-of-service aware game theory-based uplink power control for 5G heterogeneous networks. *Mobile Netw. Appl.,* **24**(2), pp. 556-563. DOI: 10.1007/s11036-018-1156-2

[b-Almasan] Almasan, P., Ferriol-Galmés, M., Paillisse, J., Suàrez-Varela, J., Perino, D., López, D., et al., Network digital twin: Context, enabling technologies and opportunities. *IEEE Commun. Mag.*, **60**(11), pp. 22-27. DOI: 10.1109/MCOM.001.2200012

[b-Anderson] Anderson, D., Harvey, P., Kaneta, Y., Papadopoulos, P., Rodgers, P., Roper, M. (2022). Towards evolution-based autonomy in large-scale systems. *GECCO '22: Proc. Genetic and Evolutionary Computation Conference*, pp. 1924–1925. New York, NY: Association for Computing Machinery. DOI: 10.1145/3520304.3533975

[b-AutoML] Google Cloud (Internet). *AutoML*. Available [viewed 2024-02-20] from: https://cloud.google.com/automl

[b-Bega] Bega, D., Gramaglia, M., Fiore, M., Banchs, A., Costa-Perez, X. (2020). AZTEC: Anticipatory capacity allocation for zero-touch network slicing, *IEEE INFOCOM 2020 – IEEE Conf. Comput. Commun.*, pp. 794-803, DOI: 10.1109/INFOCOM41043.2020.9155299.

[b-Bennett] Bennett, S. (1993). Development of the PID controller. *IEEE Control Syst. Mag.*, **13**(6), pp. 58-62. DOI: 10.1109/37.248006

[b-Bleiholder] Bleiholder, J., Naumann, F. (2009). Data fusion. *ACM Comput. Surv*. **41**(1), Article 1, 41 pp. DOI: 10.1145/1456650.1456651

[b-Blessed] Blessed, G., Aliyu, I., Agajo, J., Sarmento,T.L., Nahum, C.V., Novoa, L., *et al*. (2022).Network resource allocation for emergency management based on closed-loop analysis. *ITU J. Future Evolv. Technol*. **3**(2), pp. 175-201. DOI: 10.52953/HVPI8935

[b-Boyd] Boyd, J.R., Hammond, G.T. (editor). (2017). *A discourse on winning and losing*. Maxwell AFB, AL: Air University Press, 392 pp.

[b-Dagsthul] Demestrescu, C., Seidel, R, editors (Internet). *Dagsthul artifacts series*. Wadern: Schloss Dagsthul. Available [viewed 2024-02-22] from: https://drops.dagstuhl.de/opus/institut_darts.php

[b-El Hattachi] El Hattachi, R., Erfanian, J. (editors) (2015). *NGMN 5G white paper*. Reading: Next Generation Mobile Networks Alliance. Available [viewed 2024-02-20] at: https://ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf

[b-FRINX] GitHub (2024). *FRINX machine*. San Francisco, CA: GitHub. Available [viewed 2024-02-21] at: https://github.com/FRINXio/FRINX-machine

[b-Kephart] Kephart, J.O., Chess, D.M. (2003). The vision of autonomic computing. *Computer*, **36**(1), pp. 41–50. DOI: 10.1109/MC.2003.1160055

[b-Kubernetes] Kubernetes authors (2024). *Overview*. San Francisco: Linux Foundation. Available [viewed 2024-02-21] at: https://kubernetes.io/docs/concepts/overview/

[b-Kumar] Kumar, V.V. (2020). *Acumos DCAE integration*. Sydney: Atlassian. Available [viewed 2024-02-22] at: https://wiki.onap.org/display/DW/Acumos+DCAE+Integration

[b-Maggi] Maggi, L., Valcarce, A., Hoydis, J. (2021). Bayesian optimization for radio resource management: Open loop power control. *IEEE J- Selected Areas Commun*. **39**(7), pp. 1858-1871. DOI: 10.1109/JSAC.2021.3078490.

[b-MAPE-K] IBM (2006). *An architectural blueprint for autonomic computing*, Academic Computing White Paper, fourth edition. Hawthorne, NY: IBM 36 pp.

[b-Mwanje] Mwanje, S.S., Mannweiler, C., editors (2020). *Towards cognitive autonomous networks: Network management automation for 5G and beyond*. Chichester:Wiley. 560 pp. DOI:10.1002/9781119586449

[b-ONAP] Open Network Automation Platform (2024). *About.* San Francisco: Linux Foundation. Available [viewed 2024-02-21] at: https://www.onap.org/about

[b-ONF] Technical Steering Team (2024). *ONF mobile network projects*. Palo Alto, CA: Open Networking Foundation. Available [viewed 2024-02-21] at: https://opennetworking.org/onf-mobile-projects/

[b-ORAN] Dryjański, M. (2022). *The O-RAN whitepaper 2022 – RAN intelligent controller*. Poznań: Rimedo Labs. Available [viewed 2024-02-21] from: https://rimedolabs.com/blog/the-oran-whitepaper-2022-ran-intelligent-controller/

[b-OSM] Open Source MANO (2020). *1. OSM Quickstart*. Sophia Antipolis: ETSI OSM. Available [viewed 2024-02-21] at: https://osm.etsi.org/docs/user-guide/latest/01-quickstart.html

[b-Real] Real, E., Liang, C., So, D.R., Le, Q.V. (2020). Automl-zero: Evolving machine learning algorithms from scratch. *Proc. 37thInt. Conf. Machine Learning*, pp. 8007-8019. PMLR.

[b-Rossi] Rossi, F., Cardellini, V., Presti, F. L. (2020). Hierarchical scaling of microservices in Kubernetes. *2020 IEEE Int. Conf. Auton. Comput. Self-Organizing Syst.*, pp. 28–37. DOI: 10.1109/ACSOS49614.2020.00023.

[b-Sutton] Sutton, R.S. Barto, A.G. (2018). *Reinforcement learning: An introduction*, 2nd edition. Cambridge, MA: MIT Press. 552 pp.

[b-TOSCA] OASIS (2020). *TOSCA simple profile in YAML version 1.3*. Open-source Alliance for Social Innovation & Sustainability. Available [viewed 2024-02-21] at: https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-Profile-YAML-v1.3-os.pdf.

[b-Umuroglu] Umuroglu, Y., Akhauri, Y., Fraser, N.J., Blott, M. (2020), LogicNets: Co-designed neural networks and circuits for extreme-throughput applications. *30th Int. Conf. Field-Programmable Logic and Applications (FPL)*, Gothenburg, Sweden, 2020, pp. 291-297. DOI: 10.1109/FPL50879.2020.00055

[b-Whitley] Whitley, D. (2001). An overview of evolutionary algorithms: Practical issues and common pitfalls. *Inform. Softw. Technol.*, **43**(14), pp, 817-831. DOI: 10.1016/S0950-5849(01)00188-4.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	**Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities**
Series Z	Languages and general software aspects for telecommunication systems

Published in Switzerland
Geneva, 2024