Highlights of architecture concepts and components

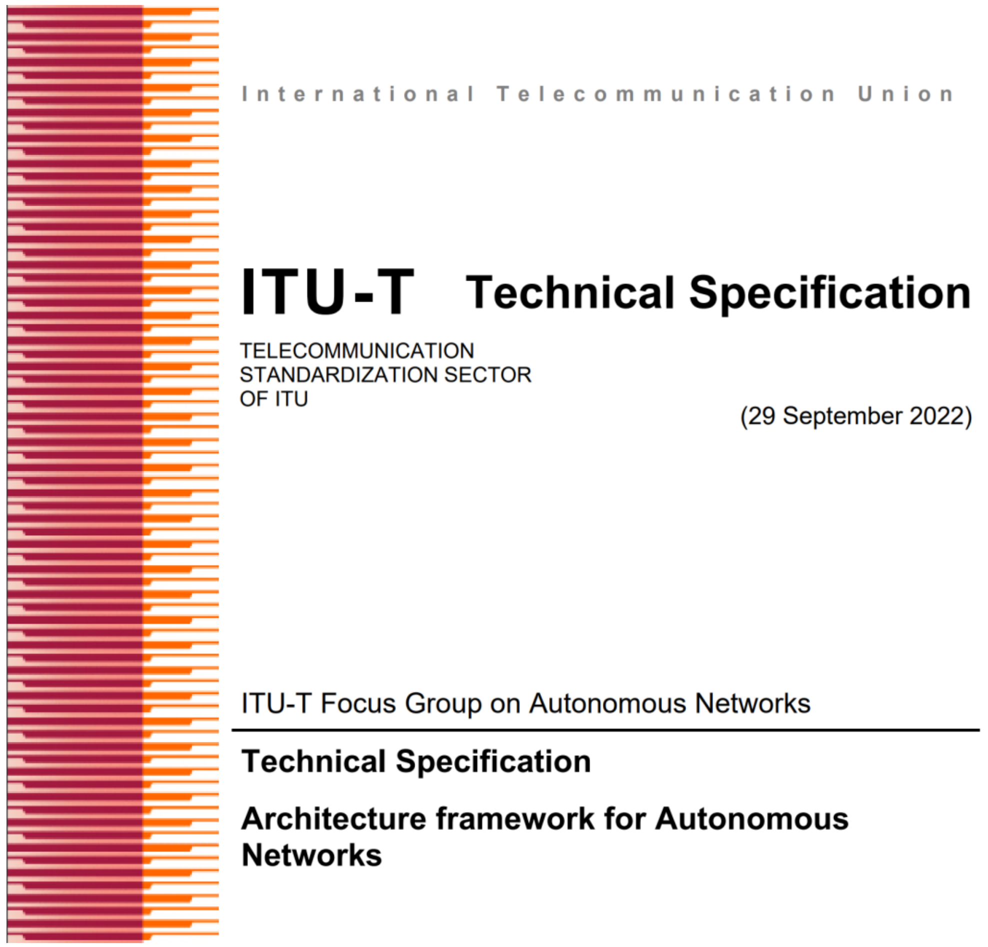
ITU-T Focus Group on Autonomous Networks

Paul Harvey

www.paul-harvey.org

FG-AN

Where to find



International Telecommunication Union

ITU-T Technical Specification

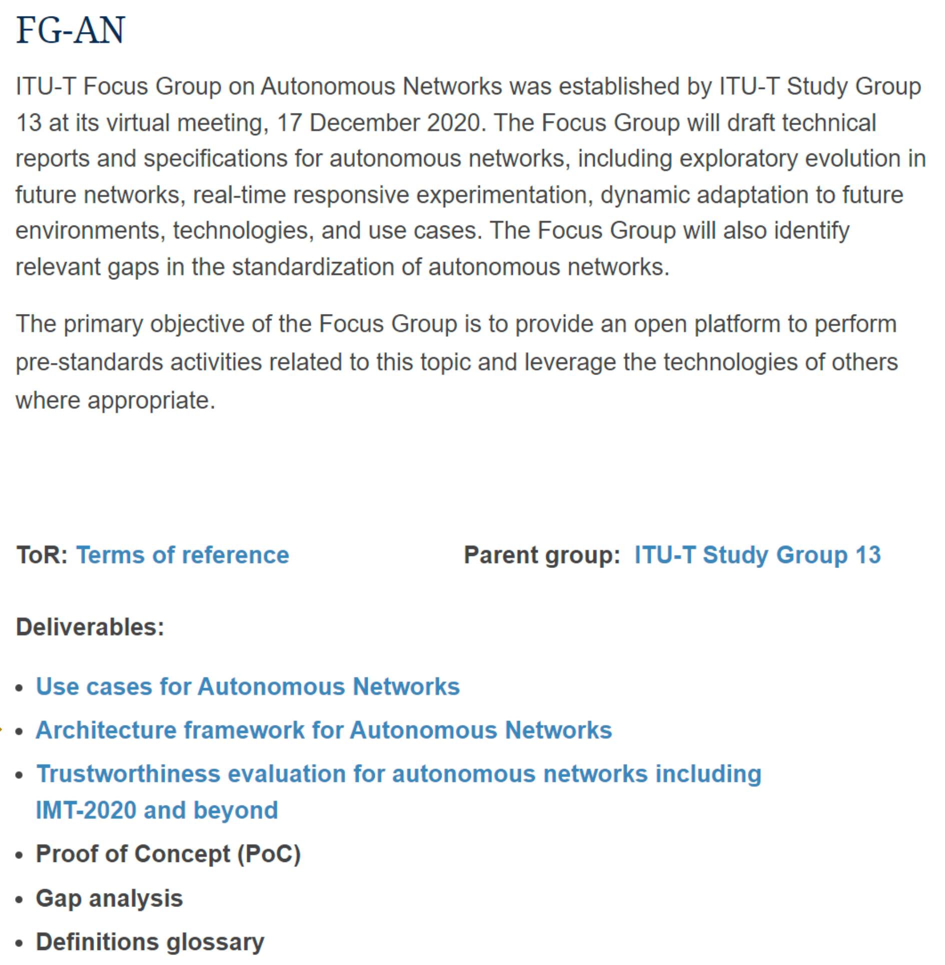
TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU

(29 September 2022)

ITU-T Focus Group on Autonomous Networks

Technical Specification

Architecture framework for Autonomous Networks



FG-AN

ITU-T Focus Group on Autonomous Networks was established by ITU-T Study Group 13 at its virtual meeting, 17 December 2020. The Focus Group will draft technical reports and specifications for autonomous networks, including exploratory evolution in future networks, real-time responsive experimentation, dynamic adaptation to future environments, technologies, and use cases. The Focus Group will also identify relevant gaps in the standardization of autonomous networks.

The primary objective of the Focus Group is to provide an open platform to perform pre-standards activities related to this topic and leverage the technologies of others where appropriate.

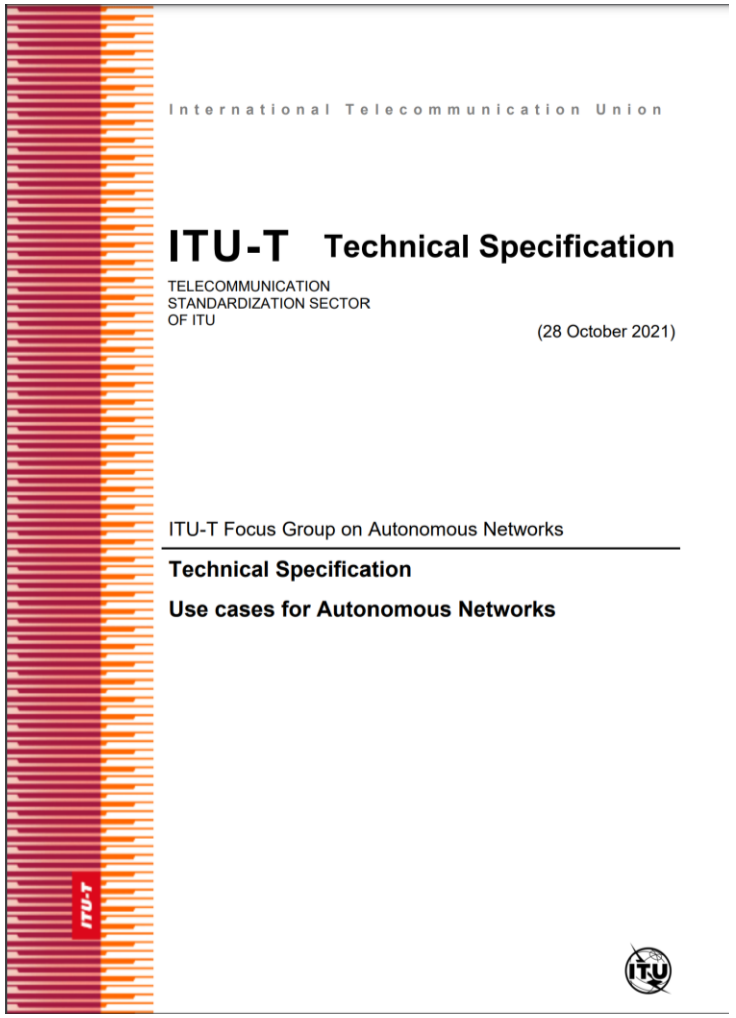
ToR: **Terms of reference** **Parent group:** **ITU-T Study Group 13**

Deliverables:

- **Use cases for Autonomous Networks**
- **Architecture framework for Autonomous Networks**
- **Trustworthiness evaluation for autonomous networks including IMT-2020 and beyond**
- **Proof of Concept (PoC)**
- **Gap analysis**
- **Definitions glossary**

https://www.itu.int/en/ITU-T/focusgroups/an/Pages/default.aspx

Process


International Telecommunication Union

ITU-T Technical Specification

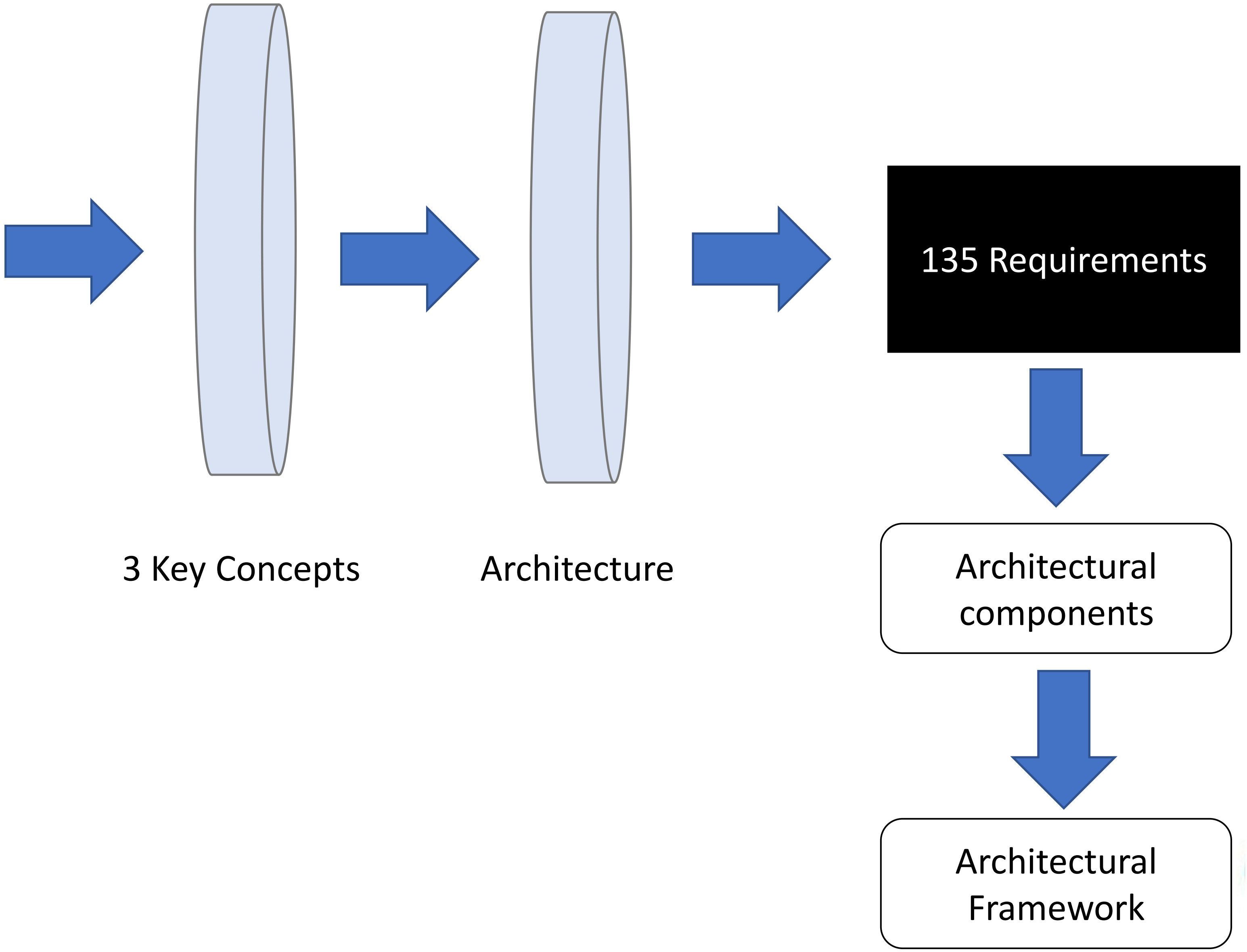
TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU

(28 October 2021)

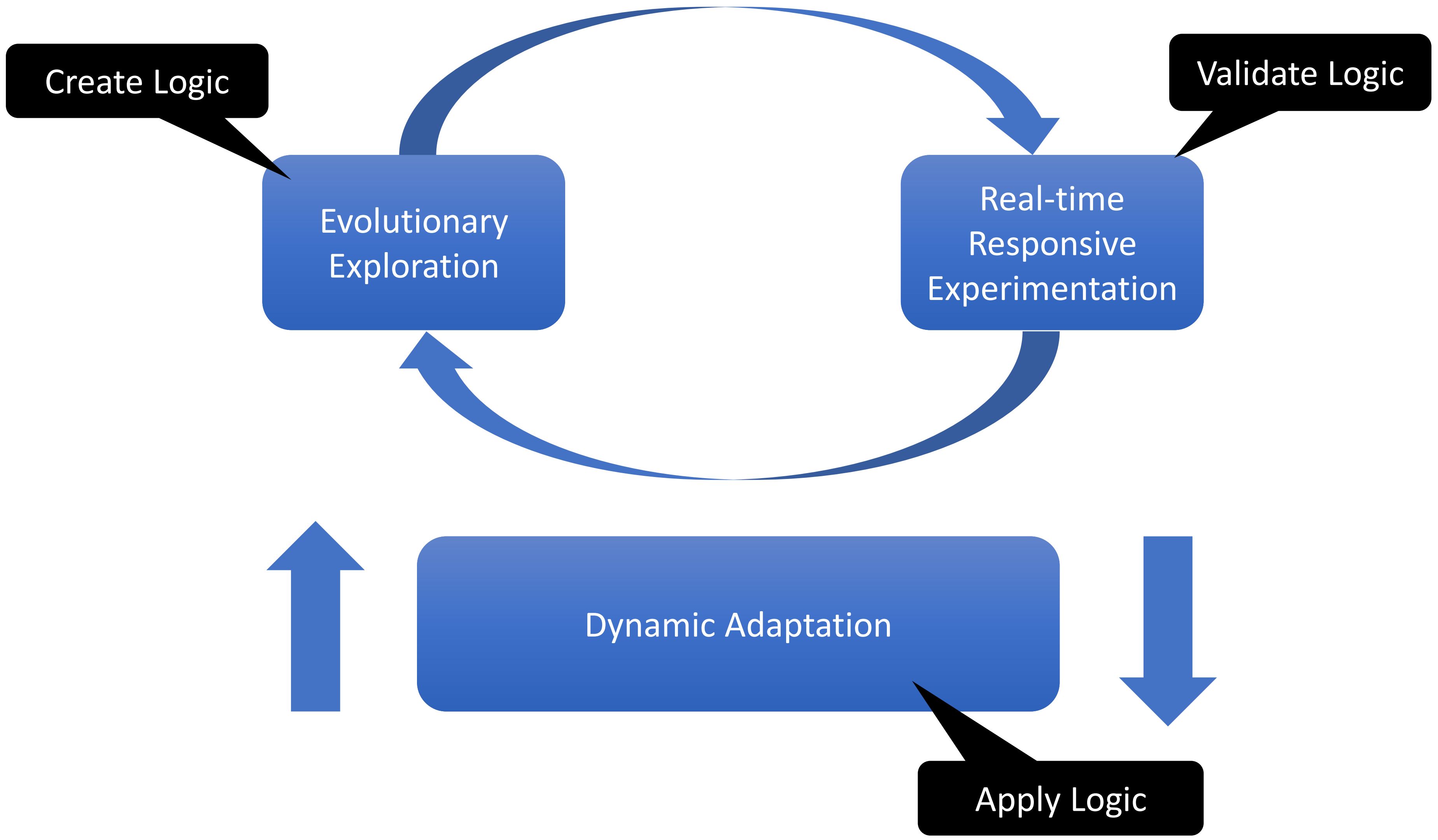
ITU-T Focus Group on Autonomous Networks

Technical Specification

Use cases for Autonomous Networks



Three Key Concepts



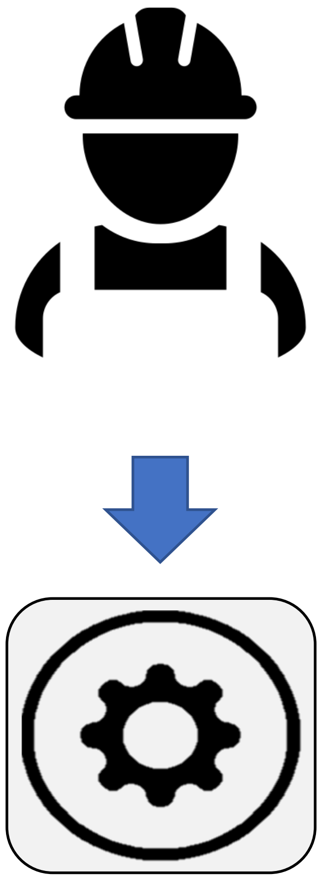
AN-arch-req-001	It is required that AN parses, validates and translates an abstracted use case description, with high level objectives of a controller (use case) into a controller description.

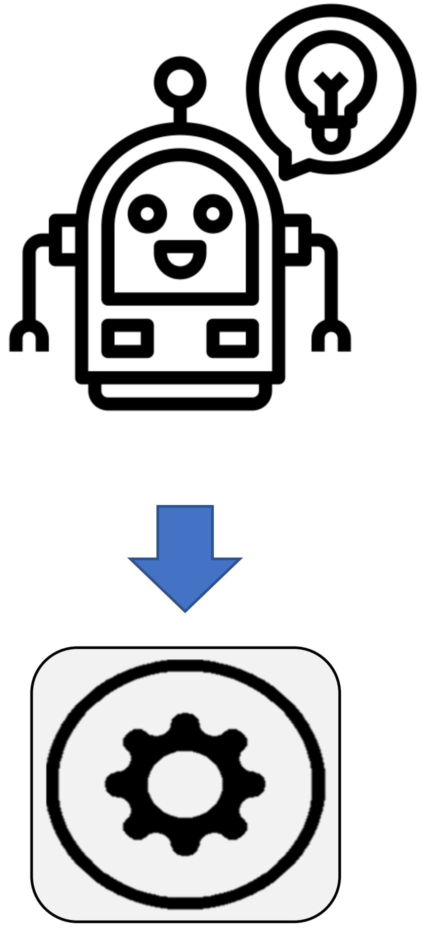
NOTE 2 – the abstracted use case description may be hand-crafted as unstructured text or derived from "software modules specifications"
NOTE 3 – Controller description may use structured languages formats e.g. TOSCA and may have a structure which facilitates downstream exploration, experimentation, and adaptation.
NOTE 4– controller description may use/enable properties derived from various domains in the network e.g. properties to describe physical layer, network layer and application layer use cases.

NOTE 5– examples of use cases are
(a) root cause analysis and diagnosis of network elements based on real time analysis of data .
(b) Intelligent energy saving solution based on automatic data acquisition, AI-based energy consumption modelling and inference, facilities parameters control policies decision, facilities adjustment actions implementation, energy saving result evaluation and control policies continuous optimization.
(c) optimal adjustment of antenna parameters with AI capabilities of multi-dimensional analysis and prediction.
(d) management of industry vertical applications and related services in the network.

Requirement	Description
AN-arch-req-002	it is required that AN validates and processes the controller description, so that exploratory evolution can be applied on the controller descriptions.

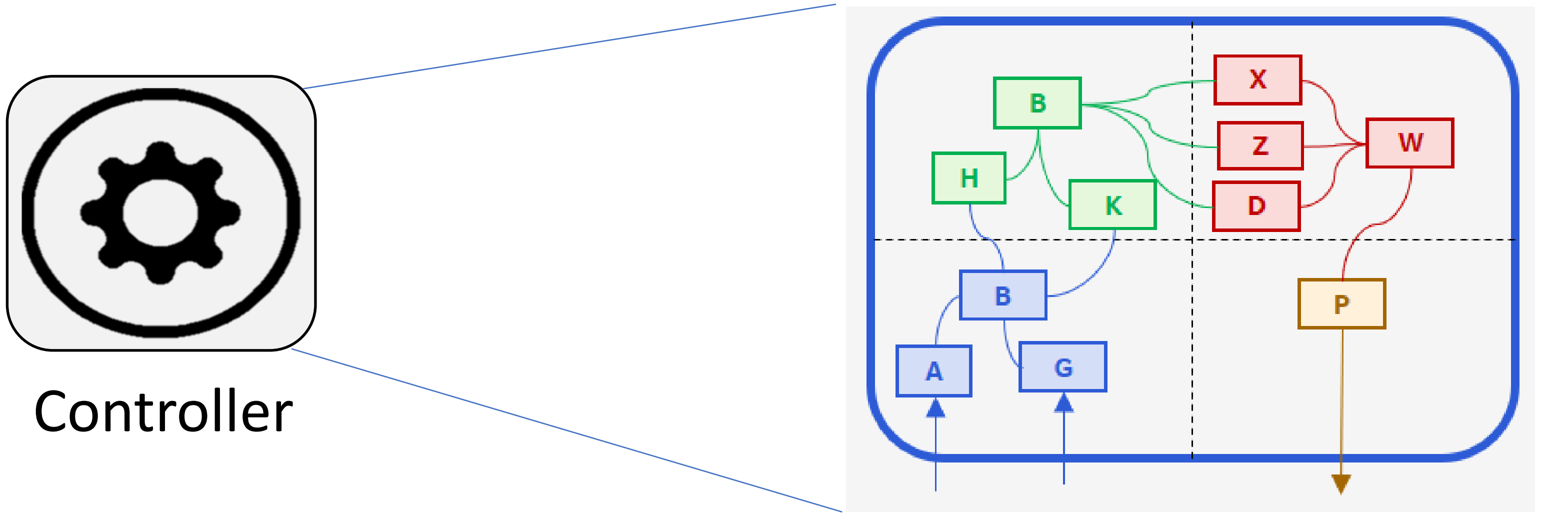
NOTE 6 – exploratory evolution may include the following: interconnecting of descriptions together to form complex controller descriptions.
NOTE 7 – exploratory evolution may result in a list of evolvable controllers
NOTE 8 – exploratory evolution may be a continuous process.

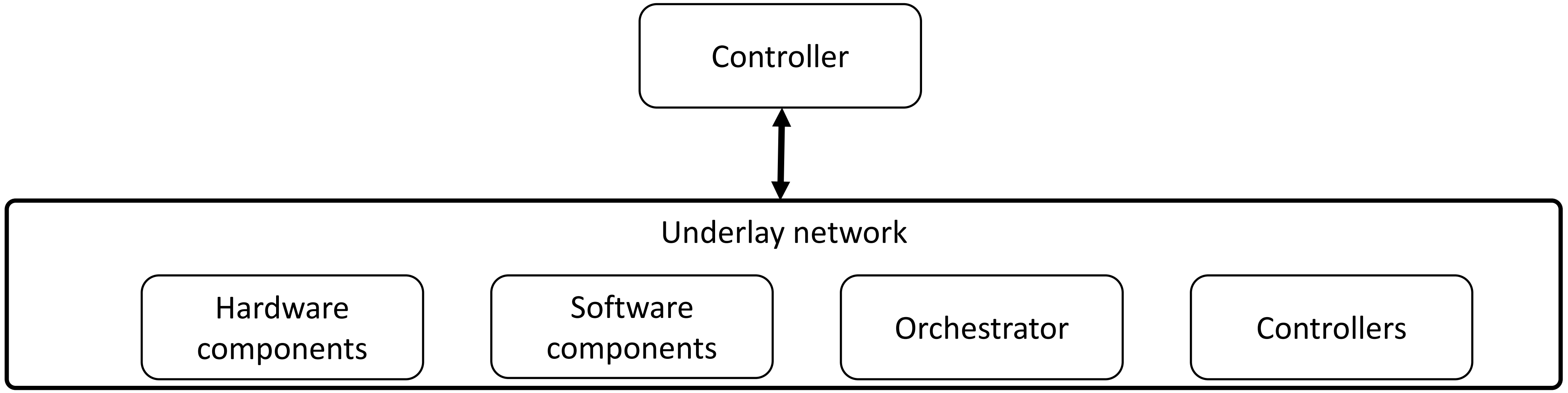

Controller


Controller

Definition 3.2.4: Controller

A controller is a workflow, open loop or closed loop [ITU-T Y.3115] composed of modules, integrated in a specific sequence, using interfaces exposed by the modules, which can be developed independently of the system under control before integration into the system under control, to solve a specific problem or satisfy a given requirement





The interactions shown here are:

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

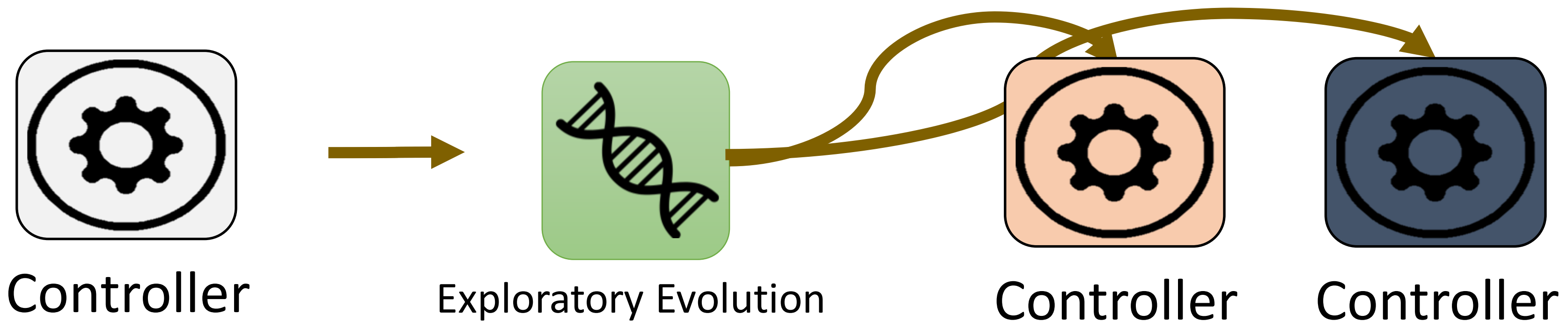
NOTE 3- Building upon this simple representation, hierarchies of controllers may be formed.

Exploratory Evolution

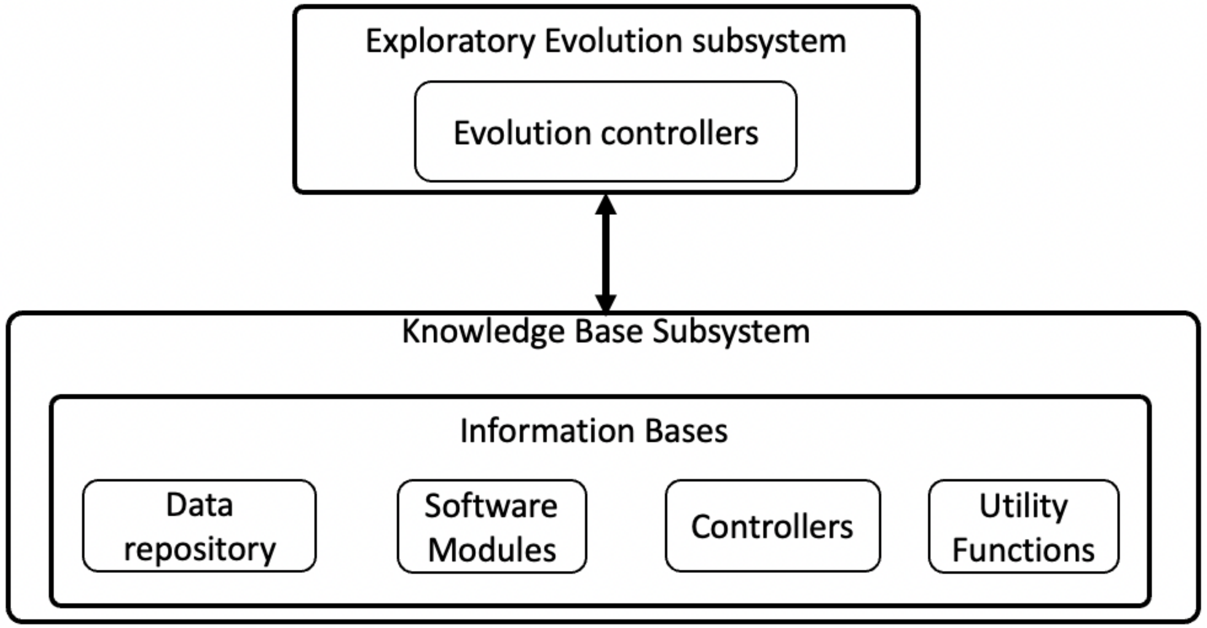
Exploratory evolution is the process that **creates and modifies a controller** in accordance with the system under control and the real-time changes therein.

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

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



Exploratory Evolution



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

- Examples of processes to drive the modification of a controller are:
    - biologically inspired artificial evolution(e.g evolutionary computing or genetic programming [b-large-evolution, b-evolution])
    - Bayesian optimisation [b-bayesian-radio]
    - game theoretic approaches [b-game-theory].

Examples of Controller Evolution

1) A “RAN channel scheduling controller” is an example of a controller used to allocate radio resources to users in a multi-user environment. Exploratory evolution is applied to a RAN channel scheduling controller in response to the change of radio channel feedback from the UE. This may include selecting the most appropriate algorithm from a set of alternatives.

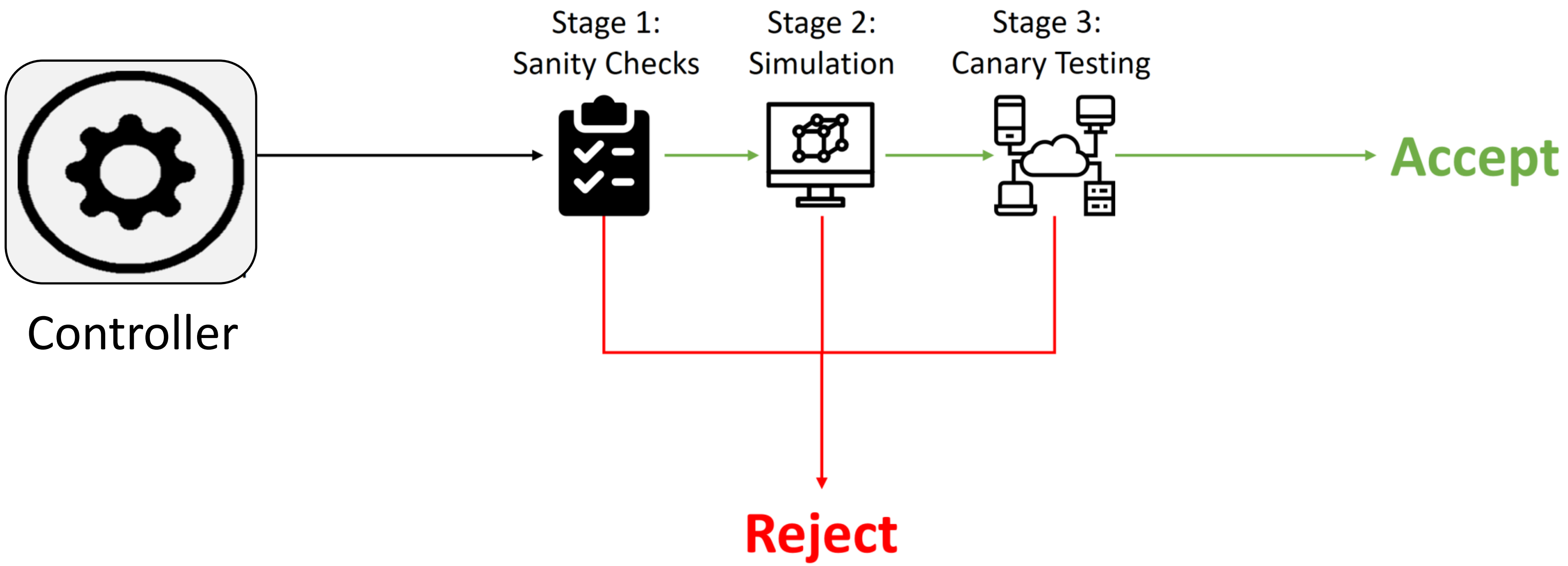
2) An “anomaly detection controller” is an example of a controller used to detect abnormal states in the operation of a network service, such as security attacks or peaks in resource usage for network function. In this context, the new approaches of data fusion algorithms [b-data-fusion] may be applied. Exploratory evolution is applied to “anomaly detection controller” by optionally using and configuring newly provided data fusion algorithms as the input of an “anomaly detection controller”,

3) A “time-to-live controller” is an example of a controller used to configure the time duration for which a certain content is cached in a CDN server. In a time-to-live controller in a caching system at the edge, optimisation of the timeout parameter(s) is an example of application of exploratory evolution.

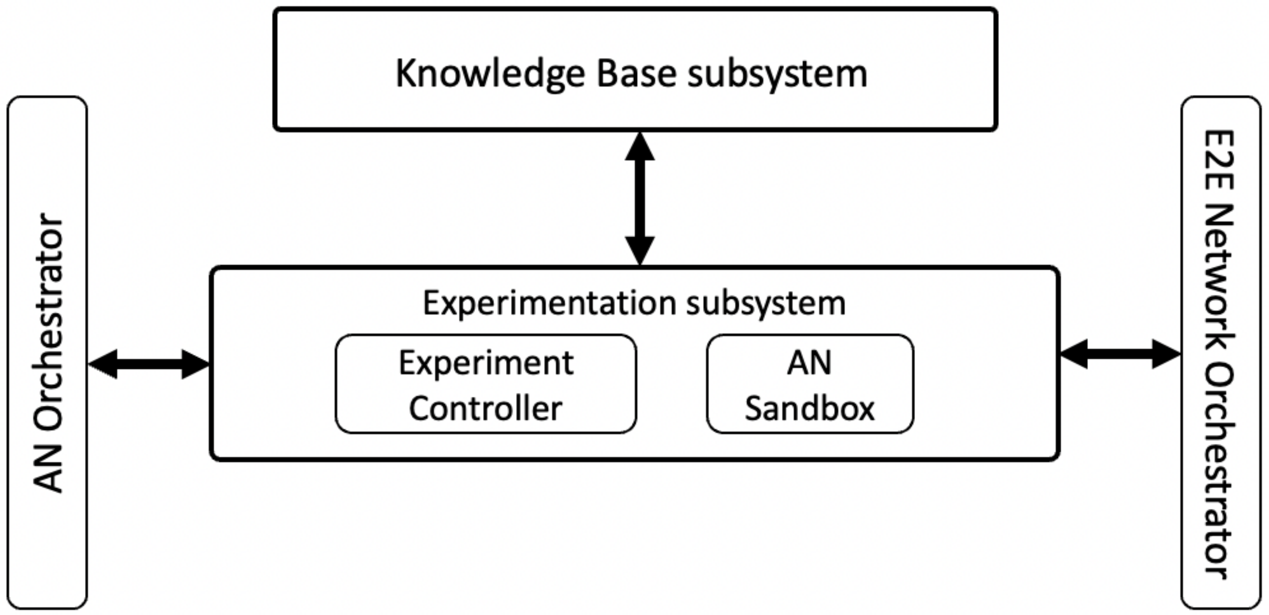
4) A “scaling controller” is an example of a controller used to increase or decrease the resource allocation for a network function. In this context, exploratory evolution may be applied by controlling the configuration of the scaling method of deployed controllers in a specific network domain.

Realtime Online Experimentation

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



Realtime Online Experimentation



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

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

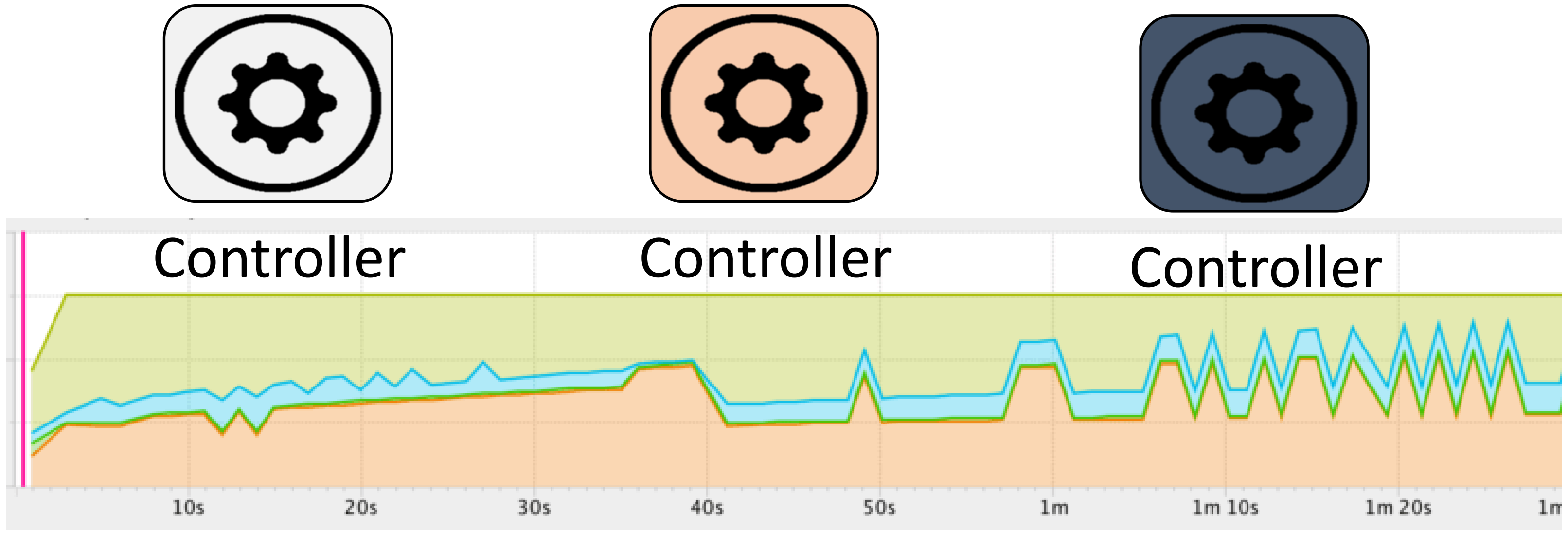
Examples of Controller Experimentation

Examples of experimentation in various application contexts are given below:

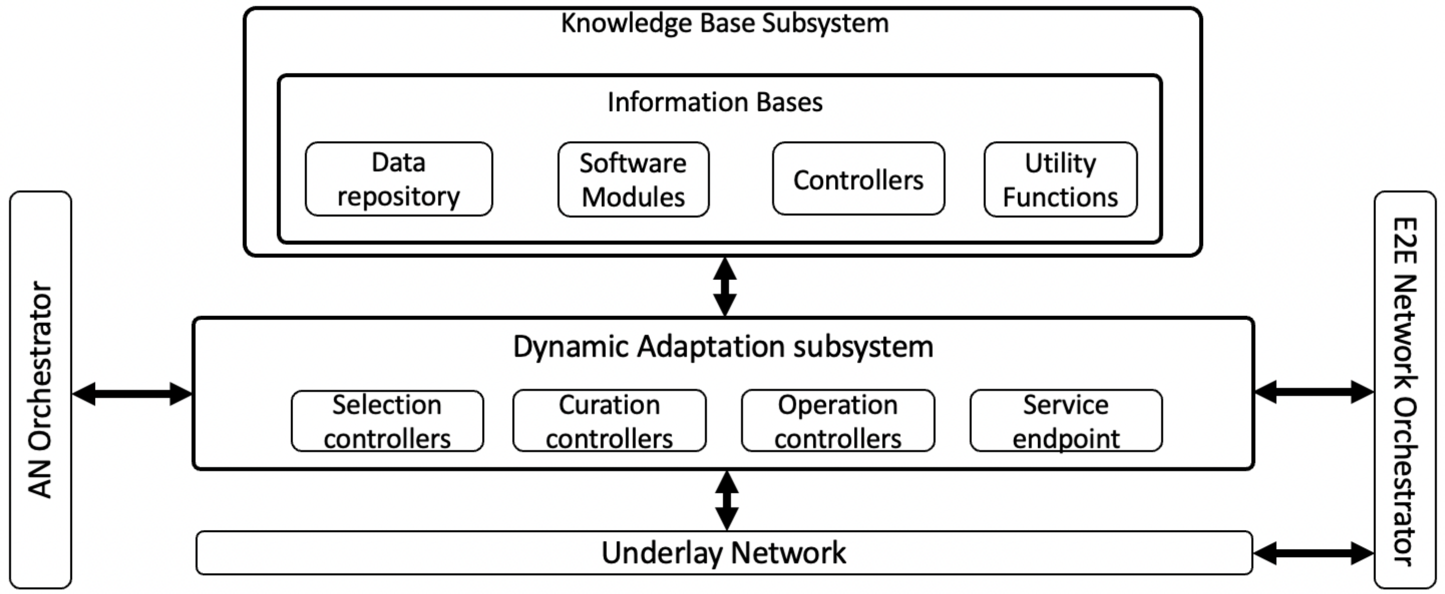
- The use of static “sanity checking” such as formal methods [ITU-T Y.3320] or model checking to ensure that provided management and orchestration solutions are well-formed against pre-defined rules
- The use of simulators or digital twins in offline validation of controllers. These simulators or digital twins can support the same interfaces as underlays.
- The use of digital twins [b-Digital-twin] in online validation of controllers before deployment
    - NOTE 5 - online validation involves use of timescales comparable to real underlays e.g. validation of controllers (xApps) [b-ORAN] using digital twins.
- Combinations of the above to achieve broader coverage of validation, from the offline validation to online validations during the operation of the underlay.

Dynamic Adaptation

- 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.



Dynamic Adaptation



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

NOTE - Adaptation controller has two parts:

- **Curation controller** (responsible for selection and maintenance of the controllers within the curated controller lists from the evolvable controllers) and
- **Selection Controller** (responsible for the selection of a services' operational controller from the curated controller lists).

Examples of Dynamic Adaptation

Examples of adaptation in various application contexts are given below:

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

Architecture Components

Evolution Controller

Creates and modifies a controller in accordance with the system under control and the real-time changes therein.

Knowledge Base

Manages knowledge derived from and used in autonomous networks. It is updated and accessed by various components in the autonomous network.

AN Orchestrator

managing workflows and processes in the AN and steps in the lifecycle of controllers

Experimentation Controller

Validates controllers using inputs from a combination of underlay network, simulators and/or testbeds.

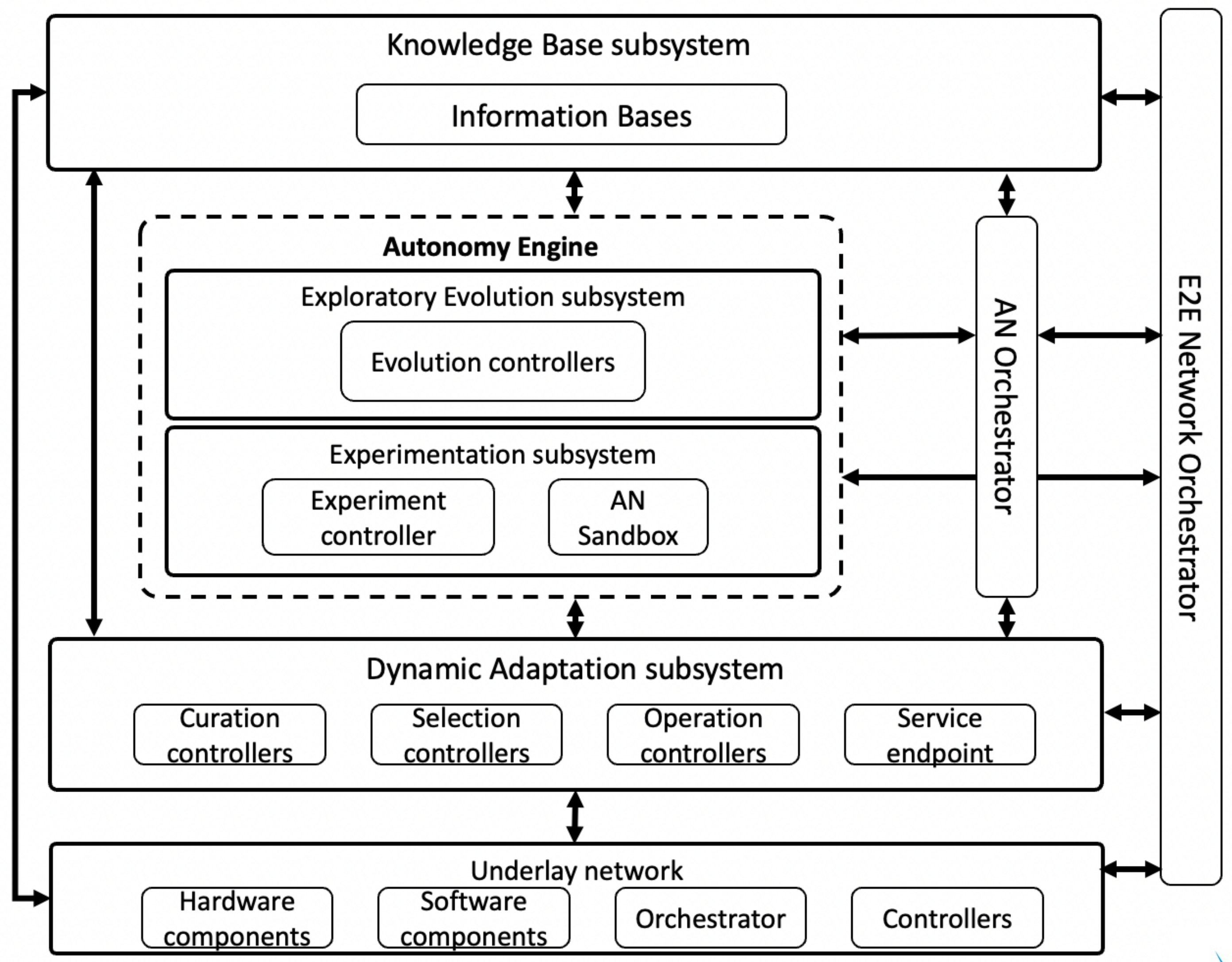
An Sandbox

environment in which controllers can be deployed, experimentally validated with the help of (domain specific) models of underlays

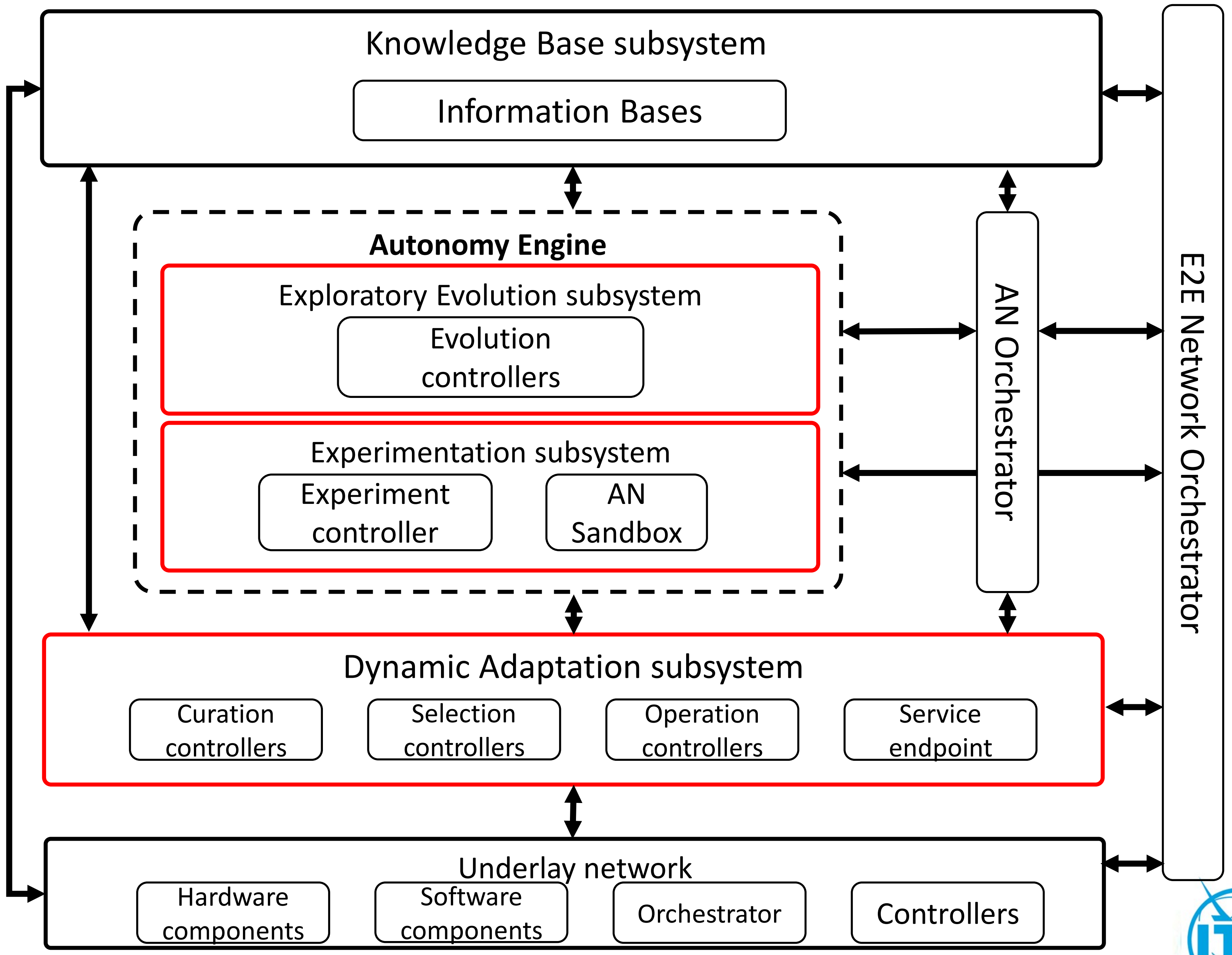
Adaptation Controller

Continuous integration of controllers to an underlay, as the underlay undergoes changes at run-time.

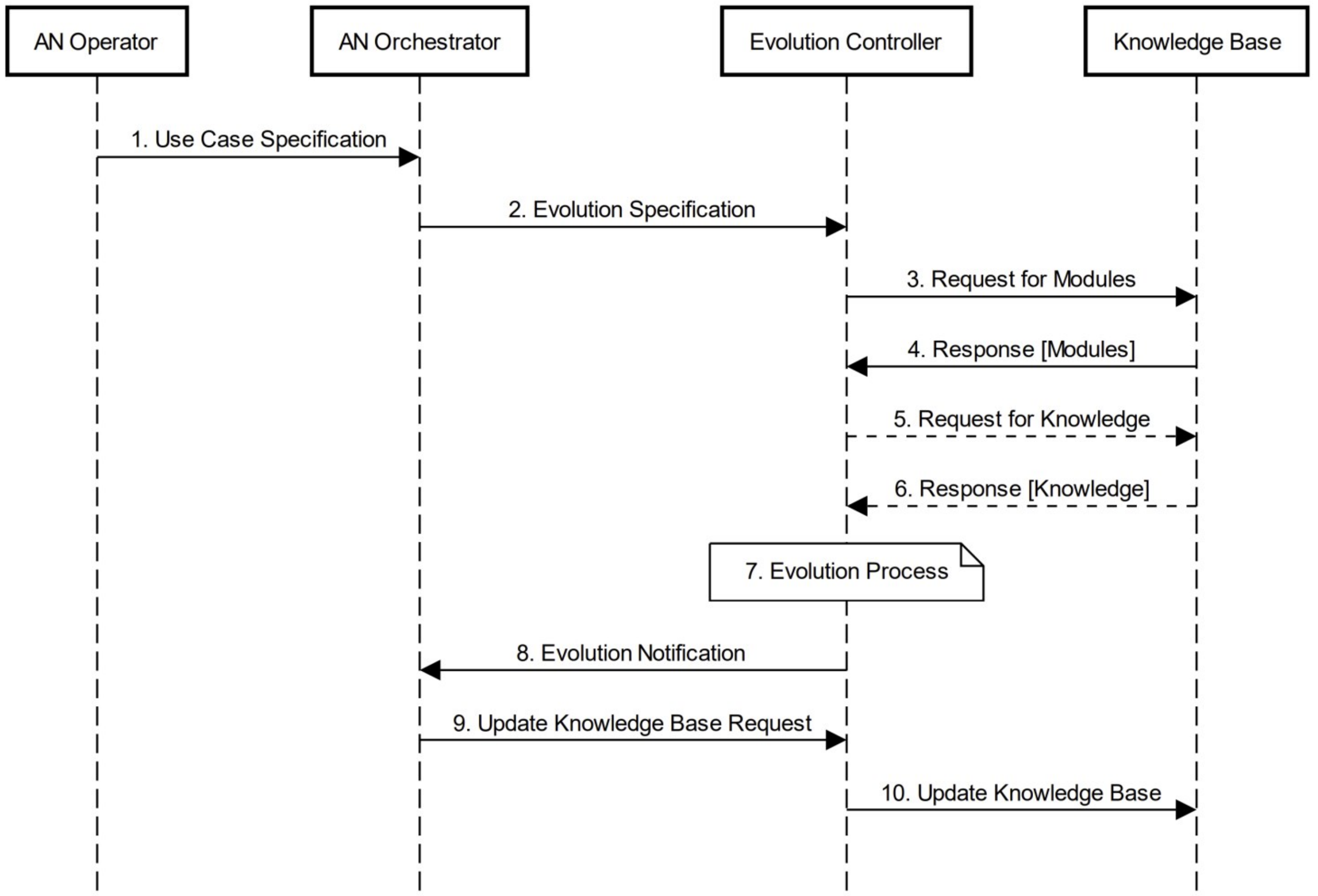
Architecture Framework Overview



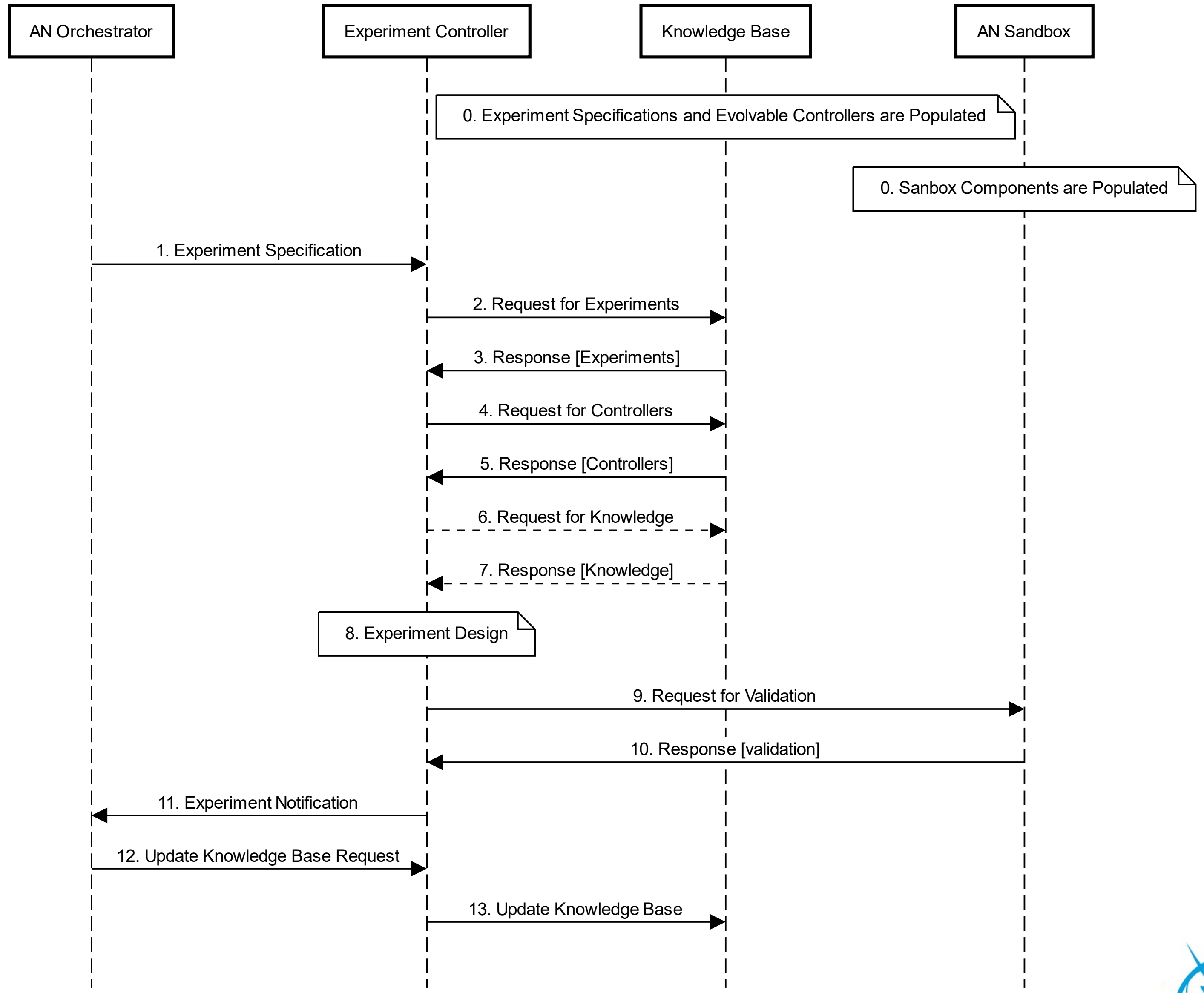
3 Key Concepts



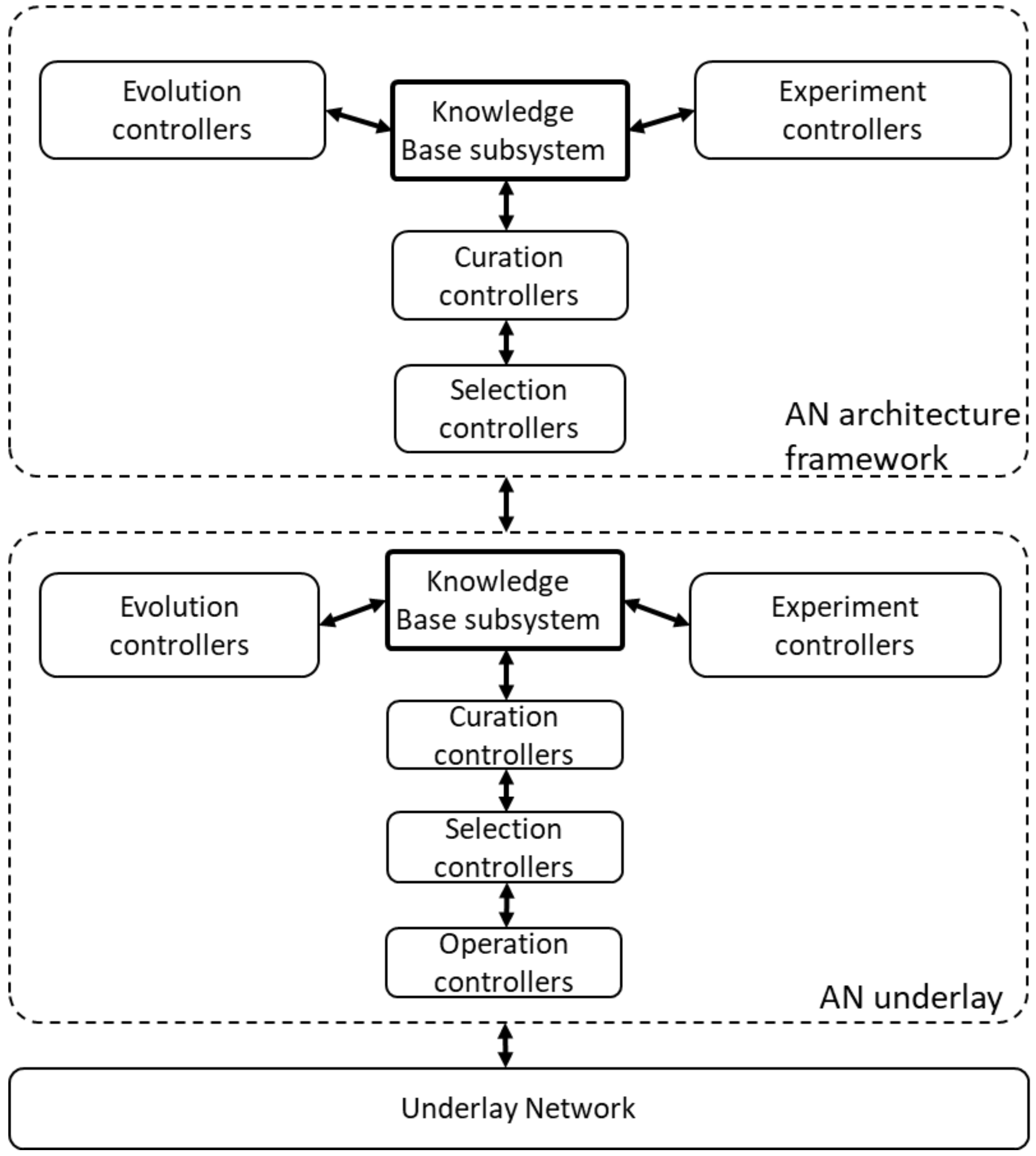
Exploratory Evolution of Controllers



Realtime Responsive Experimentation of Controllers


AN Orchestrator
Experiment Controller
Knowledge Base
AN Sandbox
0. Experiment Specifications and Evolvable Controllers are Populated
0. Sanbox Components are Populated
1. Experiment Specification
2. Request for Experiments
3. Response [Experiments]
4. Request for Controllers
5. Response [Controllers]
6. Request for Knowledge
7. Response [Knowledge]
8. Experiment Design
9. Request for Validation
10. Response [validation]
11. Experiment Notification
12. Update Knowledge Base Request
13. Update Knowledge Base

Self-Evolution

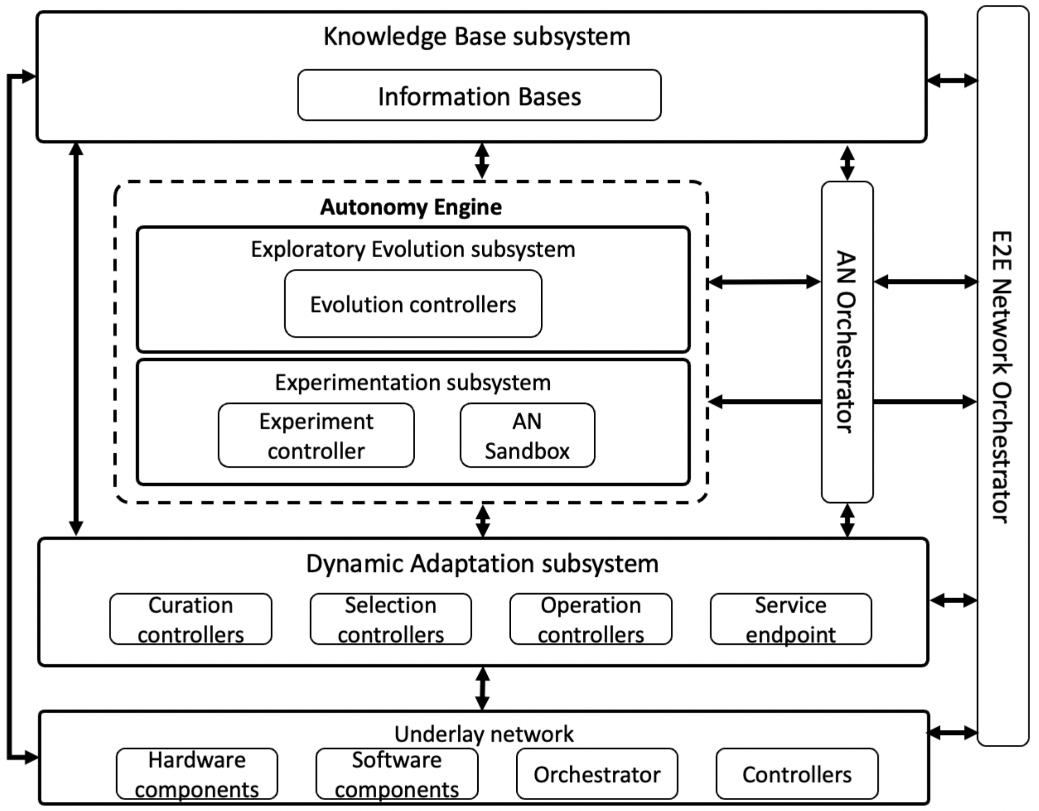


Questions – (FG)AN Architecture Highlights

- https://www.itu.int/en/ITU-T/focusgroups/an/Pages/default.aspx

#autonomousnetworks

@jhebus


Knowledge Base subsystem
Information Bases
Autonomy Engine
Exploratory Evolution subsystem
Evolution controllers
Experimentation subsystem
Experiment controller
AN Sandbox
AN Orchestrator
E2E Network Orchestrator
Dynamic Adaptation subsystem
Curation controllers
Selection controllers
Operation controllers
Service endpoint
Underlay network
Hardware components
Software components
Orchestrator
Controllers