Technical Report on Proof of Concept

Summary

This document provides a report on the Proof of Concept activities under ITU FG AN WG3. This report provides the technical summary of the activities done under PoC and it covers the following:

- Requirements for the Proof of Concept
- Description of the Proof of Concept and results

Keywords

Architecture framework, autonomous networks, Proof of Concept, use cases

Table of contents

1 Scope 4
2 References 4
3 Definitions 4
3.1 Terms defined elsewhere 4
3.2 Terms defined in this document 4
4 Abbreviations and acronyms 4
5 Conventions 5
6 Introduction 5
7 Requirements for the PoC 6
8 2021 PoC Description 8
8.1 The PoC Design and Implementation 9
8.2 Design of closed-loops using a declarative specification 9
8.3 "Imperative actions" in the "underlay" based on the intent 12
8.4 Simulated underlay for closed-loop-based resource allocation 16
8.5 Algorithms investigation for the resource allocation in the "underlay" 21
8.6 O-RAN Control-Loop Instantiation 29
8.7 Integration of the POC 33
8.8 Observations from the PoC 35
8.9 2021 Conclusions and future research 35
9 2022 PoC Description 36
9.1 The PoC Design and Implementation 38
9.2 Import and export of knowledge in an autonomous network 38
9.3 Evolution and blockchain: An Autonomous Network Architecture PoC 40
9.4 Autonomous agents (with varied competence) in networks 51
9.5 Deriving new use cases based on link prediction algorithm. 52
9.6 A Low Latency Closed Loop for ROBOTIC grasping 62
9.7 Model inference using the MEC Test Bed 63
9.8 KPI anomaly analysis for 5G networks and beyond 67

9.9 Autonomous Log Troubleshooting (with varied competence) in networks 68

10 Conclusion and future research 69

Bibliography.. 72

Technical Report on Proof of Concept activities

1 Scope

This document provides a report on the Proof of Concept activities under ITU FG AN WG3. This report provides the technical summary of the activities done under PoC and it covers the following:

- Requirements for the Proof of Concept
- Description of the Proof of Concept and results

2 References

[FGAN-O-023] ITU-T Focus Group on Autonomous Networks Technical Specification "Architecture framework for Autonomous Networks"

[ITU-T Y.Supp 71] ITU-T Supplement 71 to ITU-T Y-3000 series Recommendations, "Use cases for Autonomous Networks" https://www.itu.int/rec/T-REC-Y.Sup71-202207-P/en

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

[ITU-T Y.3176] ITU-T RecommendationY.3176, "Machine learning marketplace integration in future networks including IMT-2020"

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

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

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

3 Definitions

None

3.1 Terms defined elsewhere

None

3.2 Terms defined in this document

None

4 Abbreviations and acronyms

This document uses the following abbreviations and acronyms:

EC – Evolution Controller

OC – Operational Controller

CC – Curation Controller

CL – Closed Loop

SC – Selection Controller

RL – Reinforcement Learning

5 Conventions

In this Technical Report, in alignment with the conventions of [Supplement 55 to ITU-T Y- series Recommendations] possible requirements which are derived from a given use case, are classified as follows:

The keywords "it is critical" indicate a possible requirement which would be necessary to be fulfilled (e.g., by an implementation) and enabled to provide the benefits of the use case.
The keywords "it is expected" indicate a possible requirement which would be important but not absolutely necessary to be fulfilled (e.g., by an implementation). Thus, this possible requirement would not need to be enabled to provide complete benefits of the use case.

The keywords "it is of added value" indicate a possible requirement which would be optional to be fulfilled (e.g., by an implementation), without implying any sense of importance regarding its fulfilment. Thus, this possible requirement would not need to be enabled to provide complete benefits of the use case.

6 Introduction

As explained in [ITU-T Y.Supp 71], the main concepts behind autonomous networks which are elaborated here are exploratory evolution, real-time responsive experimentation and dynamic adaptation. Use cases along these concepts in networks were studied in [ITU-T Y.Supp 71], especially using, a basic building block called “controller”. Controllers are used in the use cases to further elaborate autonomous networks and the key concepts required to enable them. Controller is defined in [FGAN-O-023] as 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. An architecture framework was also defined in [FGAN-O-023], with the key purpose and goal of the architecture as 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. An autonomous network is a network which can generate, adapt, and integrate controllers at run-time using network-specific information and can realize exploratory evolution, real-time responsive online experimentation and dynamic adaptation.

ITU FG-AN organized a Build-a-thon challenge in 2021 to demonstrate and validate important use cases for autonomous networks, creating PoC implementations and tools in the process relating to emergency management. Interactions between a higher closed-loop in the OSS and a lower closed-

loop in the RAN to intelligently share RAN resources between the public and emergency responder slice were used as the background scenario for this PoC. The outcomes of the challenge were submitted by the various teams that participated in creating the PoC as contributions and was the main learnings were submitted as ITU J-FET paper [TBD]. The main outputs of the Build-a-thon challenge in 2021 include: (1) the implementation of a higher closed-loop “controllers” in a declarative fashion (intent), (2) the design and implementation of a lower closed-loop with Cloud Radio Access Network (C-RAN) to trigger “imperative actions” in the “underlay” based on the intent, (3) implementation of a simulation environment for data pipeline between various components; formulation of methods/algorithms for “influencing” lower layer loops using specific logic/models, and (4) the integration of the closed-loops and systems into an Open Radio Access Network (O-RAN)-based software platform, ready to be tested in the 5G Berlin testbed. The 2021 PoC study focused on intent parsing, traffic monitoring, resource computing, and allocation autonomously. The closed-loops were implemented with several micro-services deployed as docker containers with specific functions such as monitoring, computing, ML selection, and resource allocation. 2021 was a collaborative study where we developed and implemented a hierarchical closed-loop that autonomously handles an emergency use case. Clause 8 below describes this PoC.

As a follow up, Build-a-thon challenge in 2022 focussed on creation of a crowdsourced, baseline representation for AN closed loops (controllers), reviewing and analysing them, and publishing them in an open repository. The aim of the exercise was to produce reference implementations of parser, “AN orchestrator”, “openCN” [ITU-T Y.Supp 71], Evolution controller [FGAN-O-023] and to trigger technical discussions on the standard format for representing closed loops (controllers) with FG AN members and other stakeholders. This would pave the way for further downstream extensions on top of the baseline. The main activities included (1) the implementation of a reference TOSCA orchestrator to demonstrate the parsing and validation of the format in a closed loop (2) development of an evolution/exploration mechanisms to create new closed loops based on existing closed loops or controllers. Clause 9 below describes this PoC.

As a future direction, build-a-thon Challenge 2023 is planned too, to further build upon the use cases designed as part of the 2021 and 2022 Build-a-thons, study the autonomy engine defined in [FGAN-O-023], especially regarding the possibility to plugin different evolution mechanisms as a service with clear but limited interfaces and interoperability with different knowledge bases with clear but limited interfaces.

7 Requirements for the PoC

This clause describes the requirements for the PoC.

Requirement	Description
Gen-Build-a-thon-PoC-001	It is critical that PoC development activity, builds upon a key concept in FG AN, especially aims to prove the concept practically with code, test setup and demo setup.
Gen-Build-a-thon-PoC-002	It is critical that PoC development activity create well-documented artefacts and opensource code.
Gen-Build-a-thon-PoC-003	It is critical that the maturity of the PoC is evaluated using test scenarios in the accompanying documentation.
Gen-Build-a-thon-PoC-004	It is critical that the mapping with discussions in FG AN use cases, and relationship with a focussed closed loop example(s), is documented in the PoC.

Gen-Build-a-thon-PoC-005	It is critical that PoC (proof of concept) demonstrates the feasibility (or lack of it) of specific architecture approaches.
2021-Build-a-thon-PoC-006	It is critical that Demonstration is focussed on a unique scenario.
2021-Build-a-thon-PoC-007	It is expected that AI/ML based closed loops be used to intelligently deploy and manage slice for emergency responders in an emergency scenario.
2021-Build-a-thon-PoC-008	It is expected that A higher closed loop in the OSS be used for detecting which area is affected by the emergency and deploy a slice for emergency responders to that area.
2021-Build-a-thon-PoC-009	It is expected that The lower loop can use the policy derived by the higher loop to intelligently share RAN resources between the public and emergency responder slice.
2021-Build-a-thon-PoC-010	It is expected that the lower loop intelligently manage ML pipelines across the edge and emergency responder devices by using split AI/ML models or offloading of inference tasks from the devices to the edge.
2022- Build-a-thon-PoC-011	It is critical that The reference closed loops are provided as examples, parts of controllers (may not be complete, real life use cases) but representative enough for participants in the PoC to understand how to build a closed loop.
2022- Build-a-thon-PoC-012	It is expected that participants submit other use case implementations in the same format as the reference examples.
2022- Build-a-thon-PoC-013	It is critical that FG AN will collect the submitted closed loop files, and parse them using a pre-published reference code.
2022- Build-a-thon-PoC-014	It is expected that participants submit the Evolution controllers, and given the entries in openCN, evaluate the Evolution controllers by changing utilities, testing for levels of autonomy/adaptability.
2023-Build-a-thon-PoC-015	It is expected that participants are able to submit use cases with minimal syntactical limitations e.g. using natural language.
2023-Build-a-thon-PoC-016	It is expected that Knowledge Base [FGAN-O-023] is collated and updated using the use cases collected using minimal syntactical limitations.
2023-Build-a-thon-PoC-017	It is expected that evolution mechanisms are plugged in (along with the Knowledge Base) with well-specified interfaces.

8 2021 PoC Description

The main scope of 2021 PoC study was to study the interactions between a higher closed-loop in the OSS and a lower closed-loop in the RAN to intelligently share RAN resources between the public and emergency responder slice were used as the background scenario for this PoC. In this context, the following main activities were achieved:

- We designed and implemented closed-loops using a declarative specification. In the design, the Mobile Network Operators (MNOs) instruct the OSS to detect certain emergencies and provide connectivity to emergency responders according to predefined SLA. The operator input is provided as an intent using Topology and Orchestration Specification for Cloud Applications (TOSCA). The resulting YAML file is parsed, and the resulting components are instantiated in a virtualized environment.
- A network testbed with a C-RAN architecture composed of Remote Radio Units (RRUs), Baseband Unit (BBU) pool, and the core network was designed and implemented. In the architecture, a Software-Defined Network (SDN) and RAN controllers work as information sources and agents that dynamically change the mobile and the computer network. An AI agent performs different actions (e.g. resource allocation) in the testbed according to the application, using the information provided by SDN and RAN controllers to train and execute the test stage neural networks. This study shows that validating and applying closed-loop decisions for prioritizing resource allocation for network slices can significantly increase emergency response efficiency.
- We implemented a simulation environment to generate data for model training and testing purposes and to serve as a simulation underlay for testing. Two simulation cases were considered: a standalone case and a New Radio (NR) dual connectivity case. The results show that prioritized resource allocation can be simulated in different network topologies. The simulations enable us to study various configurations and analyze them to optimize the allocations. Representation of various configurations using text files helps us to create simulation topologies easily. Thus, the SRC (source) node (generating data corresponding to resource usage) and SINK node (applying various configurations in the form of NED files) are possible in the simulation environment.
- The formulation and implementation of various algorithms for an O-RAN-based controller architecture to verify the resource allocation schemes over various domains is actualized. Two algorithms were investigated to analyze the Physical Resource Block (PRB) utilization in RAN. Results were presented considering the need for resources of each slice can vary over time under dynamic networking conditions. The results show the importance of closed-loop implementations in NS, especially for intelligent management of RAN resources during emergency scenarios.
- Lastly, the PoC describes the integration of the algorithms and closed-loops above into an O-RAN-based software platform, ready to be tested in the 5G Berlin testbed [b-FGAN-I-093]. We present the integration of the algorithms in an O-RAN near Real-Time RAN Intelligence Controller (RIC) with the Acumos model repository.

Section 8.1 describes all the contents related to PoC implementation, Section 8.2 describes the design of closed-loops using a declarative specification, Section 8.3 describes a network testbed with a C-RAN architecture composed of RRUs, baseband unit BBU pool, and core network, Section 8.4 presents the creation of a simulation environment to generate data for model training and testing purposes and serve as a simulation underlay for testing, Section 8.5 describes the various algorithms which can be integrated with an O-RAN-based controller architecture to verify the resource allocation schemes, Section 8.6 describes the implementation of the above algorithms in an O-RAN near Real-Time RAN Intelligence Controller (RIC) and its integration with the Acumos model repository, Section 8.7 describes the integration of these algorithms and closed-

loops into O-RAN-based software platform, ready to be tested in the 5G Berlin testbed, Section 8.8 presents the observations from the implementation of the 2021 PoC.

8.1 The PoC Design and Implementation

This study proposes the use of analytics in Service Management and Orchestrator (SMO) [b-oran-arch] in combination with predictive resource allocations to specific edge locations based on detected emergencies to implement the PoC. A high-level strategy/policy to reallocate resources among the slices in the non-real-time RAN Intelligence Controller (non-RT RIC), forms the first level of the closed-loop. The decision in the higher level closed-loop in the non-RT RIC to reallocate resources may depend, among other things, on the type of emergency, e.g., a natural disaster such as an earthquake, law and order situation, traffic accidents, etc. A RAN-level may complement this higher-level closed-loop that uses other inputs from emergency responders to arbitrate resources among RAN nodes. Such lower-level closed-loops may be hosted nearer to the edge, e.g., near-RT RIC. The policy input from the higher loop may indicate, among other parameters, the different sources of data for the lower loop, such as system and service data. A RAN level closed-loop might also decide to offload inference tasks from ER devices to either the edge or use a split AI/ML model to run inference tasks on edge and ER devices. This decision might be taken based on the available network and computing resources. Some layers of the AI/ML model may be hosted in the wearable devices of the emergency responders, which will help in locating persons under distress using various inputs such as Global Positioning System (GPS) coordinates

Workflows for the closed-loops at the different levels are independent of each other. The only interaction between closed-loops is via high-level intents over the inter-loop interface. The closed-loops can create new closed-loops in other network domains without human intervention. Each loop can evolve independently, although loops are deployed hierarchically. It can use different models and ML pipelines as required. Each loop may move up or down the autonomy levels as defined in ITU standard, Recommendation ITU-T Y.3173. The closed-loops can split and provision AI/ML models to other closed-loops in an automated fashion. In addition, we provide a low orchestration delay, better privacy, and flexibility for verticals (e.g., industrial campus networks) by making closed-loops in the edge domain autonomous. Higher loops can use historical knowledge to optimize and generalize lower loops using high-level intent, resulting in increased efficiency of lower loops while preserving their autonomy (e.g., the higher loop might know certain ML models that are good for cyclone emergency management based on previous cyclones). Fig. 1 presents the workflow sequence for the simulation/testbed.

8.2 Design of closed-loops using a declarative specification

The high complexity of management of future networks, which includes the ability to provide new innovative services using complex network configurations, has led to requirements for autonomous behaviour. To enable low latency response by emergency responders, the use of autonomous networking concepts, including the following factors, were found important: (1) application of intent-based mechanisms to coordinate closed-loops and (2) translation of intents into decisions and actions. These mechanisms allow seamless design, deployment, and management of emergency resources in the networks using operator-friendly intents.

Several studies have been conducted regarding close-loops. For example, Gomes, et al. [b-Gomes] presented a method for formulating and managing closed-loops using requirements communicated

through intents. They propose new management functions for intent delegation, escalation, and reporting while focusing on how intent management can be integrated into the ZSM framework. Luzar, et al. [b-Luzar] compared four TOSCA-compliant orchestrators; Opera, Yorc, Indigo, and Cloudify. This comparison was made regarding ease of usage, open-source availability, licenses, and operating systems supported by the orchestrator. Ram O.V, et al. [b-Ram] carried out a gap analysis of existing frameworks in autonomous networks. Fig. 2 shows a high-level flow chart of the intent for closed-loops activity, starting with the design of the controllers. An intent is written according to the design of the controllers. This high-level intent is parsed, and appropriate closed-loops are set up to meet the objective of the intent.

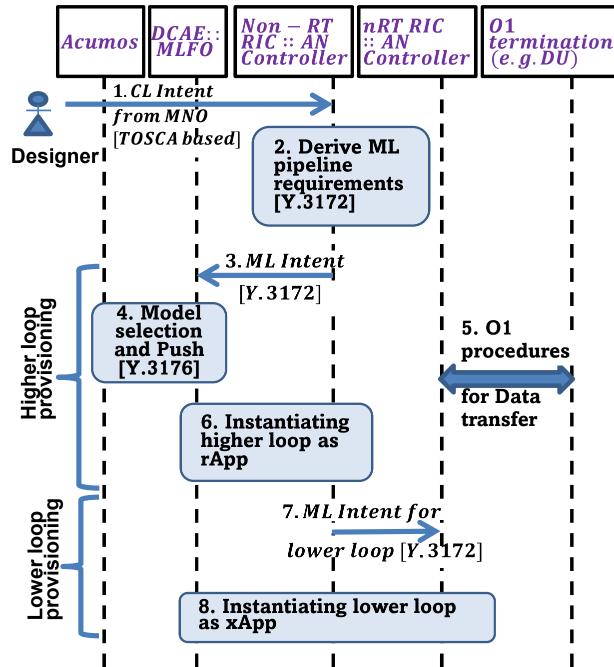


Fig. 1 – Overview of the intent-based design and implementation of hierarchical closed-loops, including simulation and testbed domain.

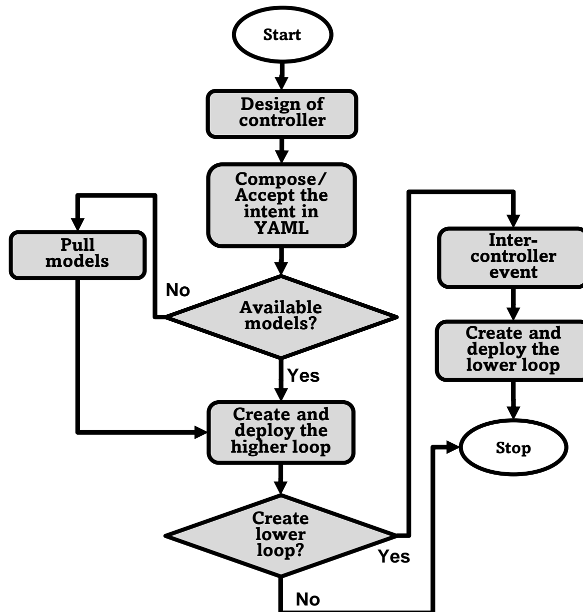


Fig. 2 – The high-level flow chart of the study toward intent for closed-loops.

```yaml
model_node:
 derived_from: tosca.nodes.SoftwareComponent
 attributes:
  download_model:
   description: URL to download ML model
   type: string
   default: "http://localhost:8080/model1"
  catalogid:
   description: catalog ID
   type: string
   default: "0"
  revisionid:
   description: revision ID
   type: string
   default: "0"
  solutionid:
   description: solution ID
   type: string
   default: "0"
 interfaces:
  Standard:
   operations:
    create:
     implementation: playbooks/create_model_node.yaml
     inputs:
      link_for_download_2:
       type: string
       default: {get_attribute: [SELF, download_model]}
```

Fig. 3 – Example - Declarative intent in YAML format.

Fig. 3 shows an excerpt from the service model showing the definition of the model node. The intent specifies the model node with attributes, including the URL for pulling the ML model from a repository. Additional attributes like catalog ID, revision ID, and solution ID may be used to identify the model. For the implementation, Opensource orchestrator xopera [b-oasis] was considered, and simple controller requirements were derived.

Fig. 4 shows the setup considered in this activity demo. The intent is written in TOSCA YAML v1.3. The intent is to create a three-node closed-loop comprising of a source, model, and sink nodes (corresponding to data collection, analysis, and application). This intent is parsed by the xopera orchestrator [b-oasis] for the deployment of the closed-loop. The model metadata and repository are defined based on the standard, Recommendation ITU-T Y.3176.

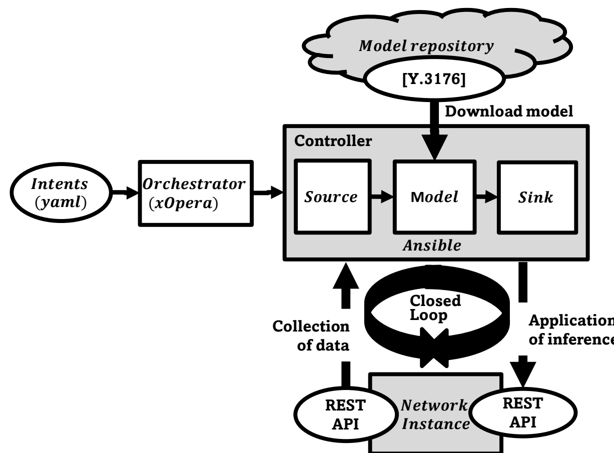


Fig. 4 – Setup design for creation and parsing of example intent.

Fig. 5 shows the outputs specified for the three nodes (source, model, and sink). The outputs specified are the attributes of the nodes, which consist of the Application Programming Interface (APIs). Fig. 6 shows the parsing of the service model for the deployment of the three nodes. The APIs in the intents are parsed into three JSON (JavaScript Object Notation) files. Three docker containers are created for implementation, which uses the APIs for data collection, analysis, and adaptation. Dummy data based on the 3Vs (Velocity, Variety, and Volume) and dummy h5 model

are downloaded from corresponding repositories according to the specified links. This study shows that a closed-loop can be represented and designed using a standard template demonstrated here using a three-node closed-loop (i.e. SRC node, ML node, and SINK node).

```
outputs:
 output_src_URI:
  description: Rest API to fetch data
  value: { get_attribute: [ source_node, rest_api_attribute ] }
 output_attribute_csv_file:
  description: Sample source data
  value: { get_attribute: [ source_node, my_csv_file_attribute ] }
 output_solution_id:
  description: Solution ID
  value: { get_attribute: [ ml_node, solutionid ] }

 output_catalog_id:
  description: Catalog ID
  value: { get_attribute: [ ml_node, catalogid ] }

 output_revision_id:
  description: Revision ID
  value: { get_attribute: [ ml_node, revisionid ] }

 output_sink_uri:
  description: Sink REST API to set values
  value: { get_attribute: [ sink_node, rest_api_attribute ] }
```

Fig. 5 – Creation and parsing of intent in YAML.

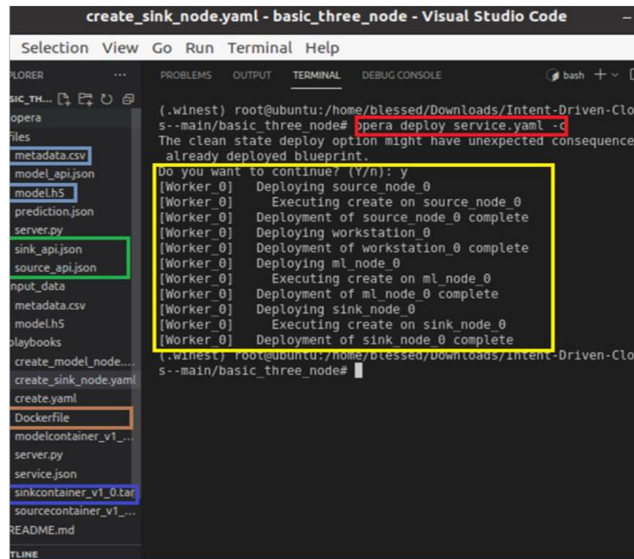


Fig. 6 – Deployment of the nodes.

8.3 "Imperative actions" in the "underlay" based on the intent

Validating and applying closed-loop decisions in the network is one of the major challenges in the intelligent allocation of resources for an emergency. The capability to build flexible and realistic AI-based scenarios with different network topologies for 5G and quickly deploy and assess them is important in emergency scenarios. This section describes a network testbed with a C-RAN architecture composed of RRUs, a BBU pool, and a core network. A network testbed called "Connected AI" is described in [b-Nahum]. The SDN and RAN controllers work as information sources about the network. Furthermore, they work as agents to dynamically change the mobile and the computer network. An AI agent performs different actions in the testbed according to the application using the information provided by SDN and RAN controllers to train and execute in the test stage. The ML workloads are orchestrated along the cluster to provide the AI agent processes.

Results from this study show that the validation and application of closed-loop decisions for prioritizing resource allocation for network slices can significantly increase the efficiency of

emergency response. This was demonstrated using priority assigned to an Unmanned Aerial Vehicle (UAV) drone based on a three-node closed-loop, i.e., source (SRC) node, ML node (AI Agent) and sink (SINK) node defined into ITU ML proposed architecture [ITU-T Y.3172].

8.3.1 Connected AI (CAI) network testbed

The CAI testbed deploys a 5G mobile network with a virtualized and orchestrated structure using containers while focusing on integrating AI applications [b-Nahum]. It uses open-source technologies to deploy and orchestrate the Virtual Network Functions (VNFs) to flexibly create various mobile network scenarios with distinct fronthaul and backhaul topologies. Distinctive features of the testbed are its low cost and the support for using AI to optimize the network performance.

Fig. 7 shows the testbed structure with a C-RAN architecture composed of RRUs, the BBU pool, and the CN. The transport network is emulated by software using Mininet [b-Aliyu], enabling the deployment of different network topologies without real network components (such as switches and routers).

The network contains two main controllers: the RYU SDN controller [b-Ryu], which is responsible for controlling the transport network emulated by Mininet, and the Open-Air Interface (OAI) FlexRAN controller [b-Flexran], which is responsible for controlling the base stations deployed in the testbed. Both controllers are connected to the AI agent, which receives network information from controllers and applies commands to change the network operations. No Management and Orchestration (MANO) component was implemented since the main objective of the testbed is to explore focused scenarios which do not include full end-to-end slice support to maintain simplicity and low costs.

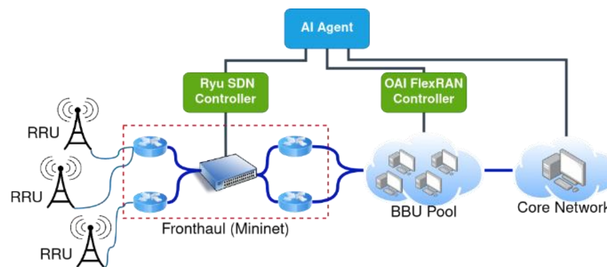


Fig. 7 – Design of the proposed testbed network [41]

To facilitate the deployment of each VNF into containers in different environments and give more flexibility to move these functions to different computers in a cluster, all the testbed components were implemented into container [b-docker] images. The RAN functions and controller were implemented using the OpenAirInterface software [b-oai], while the core network functions were implemented using the Free5GC software [b-free5gc]. These VNFs, implemented into docker containers, are orchestrated using Kubernetes software [b-kubernetes], enabling the management of the containers as well as the cluster and facilitating the deployment of different mobile network architectures. Fig. 8 shows the VNFs distributed along with the cluster and using a Software-Defined Radio (SDR) to generate Radio Frequency (RF) signals to connect the UE to the mobile network generated by the testbed. The VNFs’ location can be defined by scripts as instructed at the testbed repository publicly available [b-Nahum].

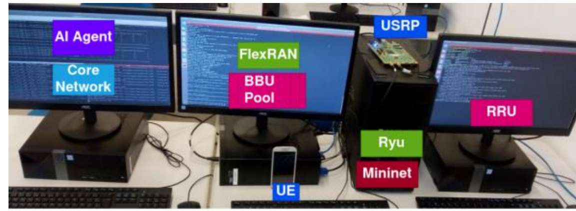


Fig. 8 – Testbed working at LASSE – UFPA lab using a C-RAN architecture.

Both the fronthaul (connecting RRU and BBU) and the backhaul (connecting the BBU and core network) are implemented over Ethernet links. Therefore, the transport network complexity usually depends on the network infrastructure available, such as switches, routers, and other network equipment. We implemented the transport network with the Mininet software to decrease costs and increase the flexibility to deploy different transport network scenarios without infrastructure changes. It emulates different topologies with routers and switches with SDN support to make the emulated network management. Then, the transport network topology can be defined in Mininet scripts and different network topologies can be tested without extra network equipment. Our scenario deployment scripts are responsible for forwarding the traffic from fronthaul/backhaul through the emulated network topology. The Mininet also allows to define some network behaviour such as packet loss rate, the bandwidth available in each emulated network link, and latency among each node and other characteristics that give a remarkable amount of flexibility to test algorithms over different network topologies and conditions. Fig. 9 shows a scenario deployed using the testbed where the backhaul link is emulated using a Mininet. A simple network composed of two routers and one switch is emulated, adding a latency of 100 ms between the BBU pool and core network.

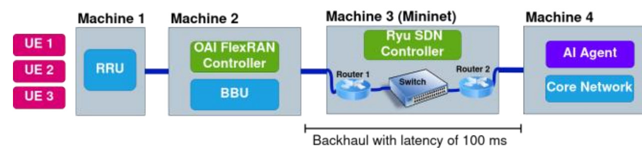


Fig. 9 – RAN slicing scenario with 3 Ues connected to a C-RAN structure with the backhaul virtualized using Mininet.

Each component from the Mininet network devices (routers and switches) was connected to an RYU SDN controller that receives information about the transport network, such as the throughput transmitted in each link, dropped packets, and latency. In addition, the SDN controller can apply commands to change transport network operations, such as changing routes and applying congestion control algorithms, thus enabling the usage of external apps to provide transport network management through communication with the SDN controller to promote changes in the network while it is operating. Fig. 10 shows information obtained from the SDN controller about switch 3 in a topology emulated with Mininet [b-free5gc48]. It gives real-time information about the switch operation, such as the number of received and transmitted packets and the port being used.

```
{
    "3": [
        {
            "tx_dropped": 0,
            "rx_packets": 126973,
            "rx_crc_err": 0,
            "tx_bytes": 2045095,
            "rx_dropped": 0,
            "port_no": 1,
            "rx_over_err": 0,
            "rx_frame_err": 0,
            "rx_bytes": 6247861542,
            "tx_errors": 0,
            "duration_nsec": 97000000,
            "collisions": 0,
            "duration_sec": 662,
            "rx_errors": 0,
            "tx_packets": 30943
        }
    ]
}
```


Fig. 10 – Information obtained from SDN controller API about the switches running in the Mininet emulated network [41].

The FlexRAN controller works as an abstraction of the RAN resources and provides an API that enables the service orchestrator entity to dynamically manage the RAN resources to provide information about the mobile network [b-Afolabi]. The FlexRAN protocol [b-flexran] defines and implements a software-defined RAN architecture integrated with the OAI platform, which incorporates an API to separate control and data planes for the mobile RAN. This architecture has a master controller represented by the FlexRAN controller in Fig. 9 and a FlexRAN agent corresponding to the OAI eNB instances. Fig. 9 also represents the FlexRAN agent in the OAI BBU instances in a C-RAN scenario. The agents can act as local controllers with a limited network view and handle the functions delegated by the master or coordinated by the master controller.

The FlexRAN agent API separates the control and data plane, allowing the control data to be managed by the FlexRAN controller and the eNB data plane on the opposite side. Fig. 11 shows the information received from a base station using the FlexRAN API, providing information such as the functional split being used, the number of user equipment (UEs) connected, buffer occupancy, and scheduling information. The FlexRAN APIs enable the development of applications related to the control and management of the RAN resources [b-foukas], e.g., schedulers, interference, and mobility manager. Moreover, applications related to improvements in the use of RAN resources make more sophisticated decisions [b-foukas], such as RAN slicing and adaptative video streaming based on channel quality.


```
{
    "bs_id": 10001,
    "agent_info": [
        {
            "agent_id": 1,
            "ip_port": "192.168.46.176:39024",
            "bs_id": 10001,
            "capabilities": [
                "LOPHY",
                "HIPHY",
                "LOMAC",
                "HIMAC",
                "RLC",
                "PDCP",
                "SDAP",
                "RRC",
                "S1AP"],
            "splits": [
                "nFAPI"]
        }
    ],
```


Fig. 11 – Information obtained from FlexRAN API about the base station running in the testbed [41].

The AI agent is implemented based on the ITU-T Y.3172 architecture that defined a logical interoperable architecture for future networks, which incorporates an ML overlay that operates on top of any specified underlay network technology [b-afolabi]. This architecture facilitates deploying ML applications in different network scenarios and is adopted in the connected AI (CAI) testbed. Specifically, ITU-T Y.3172 defined high-level architectural components to integrate ML into the network and a process pipeline [ITU-T Y.3172]. Fig. 12 shows these components, the pipeline, and their respective mapping into the CAI testbed components. This testbed orchestrates the ML workloads of the AI agent using the Kubeflow tool [51]. Kubeflow works integrated with Kubernetes to orchestrate the ML functions along with the cluster machines. Kubeflow enables the use of pipelines to define the steps of ML processing. Due to the high resource available in the cloud in real scenarios, CAI deploys the AI agent at the cloud location (with the core network) for simplicity.

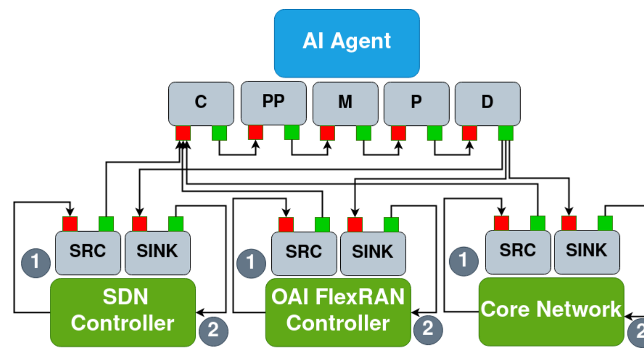


Fig. 12 – ITU-T FG-ML5G ML architecture integrated into the testbed structure [41]

8.3.2 Results and discussions

Some results exploring UAVs in critical missions using the testbed are presented in [b-lins]. This study presents how the AI agents and the network can be adapted to assist mobile network users in search, diagnostic and rescue (SDAR) missions. Fig. 13 shows the results for a scenario with an AI agent controlling the number of radio resources using the RAN slice to prioritize drones in SDAR missions about other UEs connected to the network. In the scenarios without slices, the base station tries to provide an equal amount of radio resources among the UEs without differentiating the applications. When the RAN slice is used and the AI agent set a slice to the drones in the SDAR mission, the AI agent updates the number of radio resources allocated to the drone's slice to guarantee at least 10 Mbps of throughput, the other slice with UEs receives only the remaining radio resources since it has less priority. It shows that a closed-loop can be implemented to control the testbed mobile network using AI methods despite the simplicity of the experiment the AI agent used.

8.4 Simulated underlay for closed-loop-based resource allocation

To complement the testbed described in section 2.2, this section describes the creation of a simulation environment [b-simu5g] to generate data for model training and testing purposes and also to serve as a simulation underlay for studying the impact of the

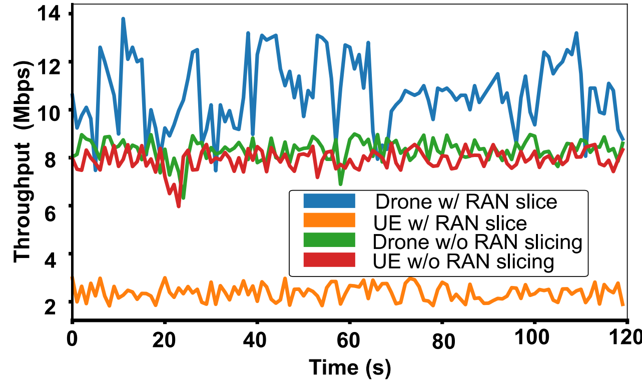


Fig. 13 – Results without RAN slicing and a scenario with RAN slicing using an AI agent.

closed-loop on resource allocation scenarios in Medium Access Control (MAC) layer. Simu5g [b-simu5g] is used to generate output data shown in the results Section 3.2 (e.g., Average served blocks in Downlink/Uplink) based on input parameters given in Table 1 (e.g., Frequency Correction Burst (Fb) Period, target block error probability(BLER), etc.) while simulating the various scenarios. This facilitates studying the machine learning algorithms' impact on resource block allocation by predicting the resource requirement at the UE. Simu5G is based on the OMNET++ simulation framework and incorporates the simulation modules from the INET library [b-simu5g]. It simulates both the data plane of 5G RAN and the core network.

Two types of simulation scenarios were considered using Simu5G- standalone case and NR dual connectivity case. In the standalone scenario, gNB is connected to the data network through the core network, while in the NR dual connectivity case, gNB is connected to the eNB through an X2 interface. In addition, the eNB provides access to core and data networks. Sections 3.1 and 3.2 discuss the simulation and the results. The simulator configurations used in this study are simu5g v1.2.0, INET v4.3.2 or above and OMNET++ v6.

8.4.1 Simulation Scenarios

This study defines two simulation scenarios: "single cell with secondary gNB" and "multi-cell with secondary gNB". These two scenarios (called "networks" in the simulator) are defined in the NED (Network Description) file in OMNET++ which the structure of a simulated network can be described. NED enables the user to declare simple modules and connect and assemble them to form compound modules. Compound modules e.g., a "single cell with secondary gNB" and a "multi-cell with secondary gNB" network, can be used as simulation modules.

Parameters can achieve their value from either the NED file or the configuration, i.e., .ini file. Every configuration file has a "General" section that has general parameters like simulation time limit ("sim-time-limit" is the physical time that is set for simulating the network). The "network" keyword is used to flag the network that needs to be simulated.

Fig. 14 shows the two networks which are defined in the NED file: "single cell with secondary gNB" and the "multi-cell with secondary gNB". The main modules that are used in this network are compound modules: carrier aggregation (carrierAggregation), packet gateway (pgw), LTE base station (masterEnb), NR base station (secondaryGnb), and UE. The carrier aggregation module is responsible for assigning multiple frequency blocks. The eNB, which directly connects to CN is called master eNB and the gNB, which is connected to the core via eNB using the X2 interface, is called secondary gNB. The number of UEs is defined using the numUe parameter of the UE

module. In the “multi-cell with secondary gNB” case an extra set of eNB which are connected via X2, and an extra set of gNB which is in turn connected to the respective eNBs are shown in Fig. 15.

Fig. 14 – NED file for SingleCell_withSecondaryGnb.

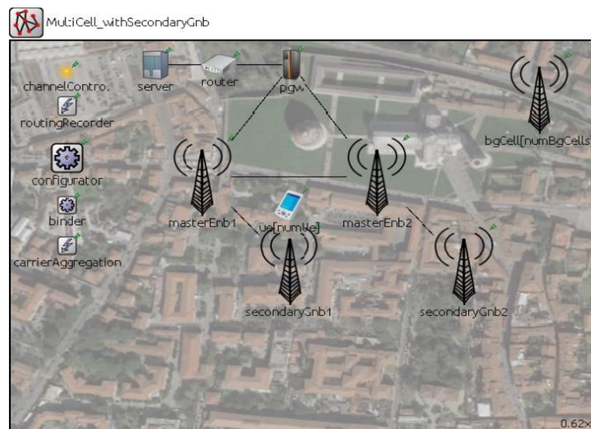


Fig. 15 – NED file for MultiCell_withSecondaryGnb.

Parameter	Value
eNodeB Transmission Power	40dB
Fb Period	10ms
Target BLER	0.01
BLER Shift	5
#Component Carriers	2
Carrier Frequency of CC1	2GHz
Carrier Frequency of CC2	6GHZ
#UE’s	10
UE mobility type	“RandomWaypointMobility”
UE speed	Between 5mps to 15mps
Dual Connectivity	True
# resource blocks for CC1	6

Parameter	Value
# resource blocks for CC2	6
#UE apps	2
Amount of UDP application on the server (server.numApps)	#UE’s * #UE apps =20

Table 1 – Parameters described in ini file for network simulation.

The configuration file (also known as an “ini” file) contains network parameters and their corresponding values for each carrier component as shown in Table 1. The number of UEs (numUe) specified in the UE module is set to 10 for this simulation. UE mobility type and UE speed are defined for each UE. Dual connectivity is enabled and each network is configured with uplink and downlink.

Carrier components are part of the carrier aggregation module and have carrier frequency and numerological index. The frequency of each carrier component is defined in Table 1 above. The number of resource blocks is also defined for each carrier component.

8.4.2 Results of the simulations

This section analyses the output of avgservingblock (average serving blocks are the resource blocks that are utilized at the time of simulation). The result files storing the simulated network's vector values and scalar values are analyzed after simulating the required network configuration. For example, avgservingblocks is a vector quantity because it varies with the simulation time. Four outputs are analyzed for each network, corresponding to two configurations: uplink and downlink.

The four output results that are obtained include: average served blocks downlink for single-cell with secondary gNB (Fig. 16), average served blocks downlink for multi-cell with secondary gNB (Fig. 17), average served blocks uplink for single-cell with secondary gNB (Fig. 18), and average served blocks uplink for multi-cell with secondary gNB (Fig. 19). The simulation time is variable, and we use a value of 50 s, with resource allocation data being collected every millisecond for each of the four output results discussed above. This gives us enough data points to study the average served blocks for each output. The total number of resource blocks allocated in the results cannot exceed those that are set in the .ini file (specified across different CC, carrier components). The blue and orange coloured line chart represents the avg served blocks for master eNB and secondary gNB, respectively in singleCell_withSecondaryGnb, data flow is downlink in Fig. 16. In contrast, the blue, orange, green, and red coloured line chart represents the avg served blocks for master eNB1, secondary gNB1, master eNB2, and secondary gNB2, respectively, in MultiCell_withSecondaryGnb, and the data flow is downlink in Fig. 17. The blue and orange coloured line chart represents the avg served blocks for master eNB and secondary gNB, respectively, in singleCell_withSecondaryGnb, where the data flow is uplink in Fig. 18. The blue, orange, green and red coloured line chart represents the avg served blocks for master eNB1, secondary gNB1, master eNB2 and secondary gNB2, respectively, in MultiCell_withSecondaryGnb, where the data flow is uplink in Fig. 19.

This study shows that prioritized resource allocation can be simulated in different network topologies. The simulations enable us to study various configurations and analyze them to optimize

the allocations. Representation of various configurations using text files defined in [b-liu] enables us to easily create simulation topologies. Therefore, the SRC node (generating data corresponding to resource usage) and SINK node (applying various configurations in the form of NED files) are possible in the simulation environment. Integrated analysis of generated data using AI/ML is for future study.

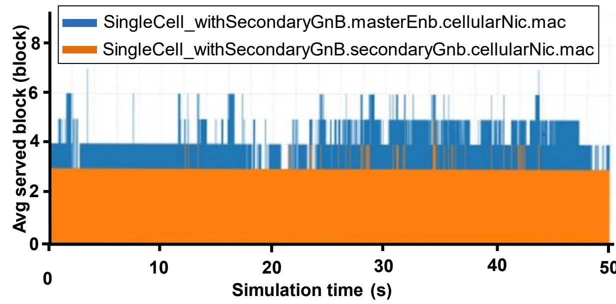


Fig. 16 – Avg served blocks, DL, SingleCell_withSecondaryGnb.

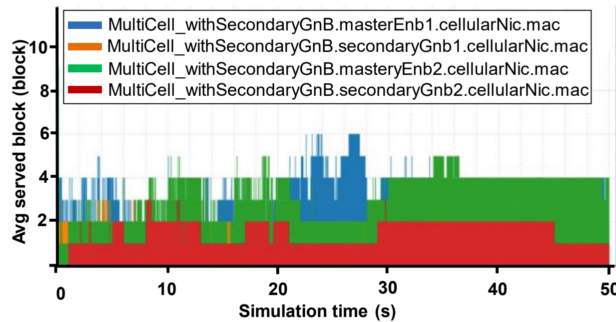


Fig. 17 – Avg served blocks, DL, multiCell_withSecondaryGnb.

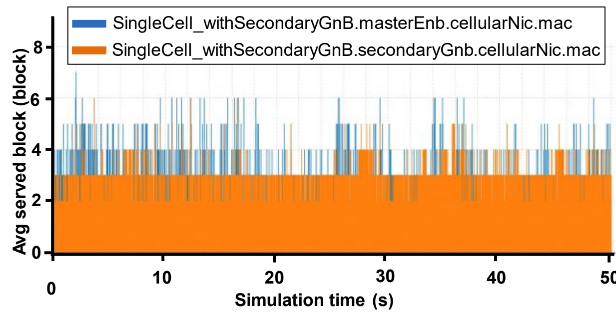


Fig. 18 – Avg served blocks, UL, singleCell_withSecondaryGnb.

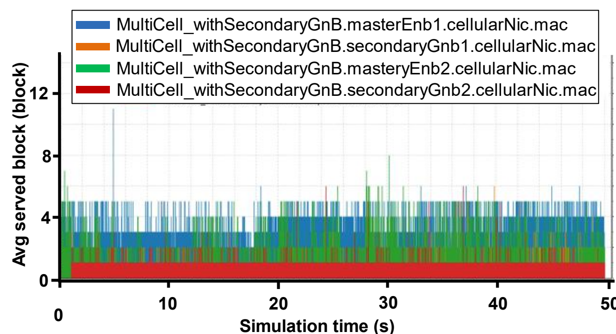


Fig. 19 – Avg served blocks. UL, MultiCell_withSecondaryGnb.

8.5 Algorithms investigation for the resource allocation in the "underlay"

This section describes the various algorithms that can be plugged into an O-RAN-based software architecture to verify the resource allocation schemes. The non-real-time RIC closed-loop intent is applied to the near real-time RIC lower loop. The lower-loop monitors RAN resources and makes decisions to achieve the intent. Fig. 20 shows the illustration of the overall process of the system.

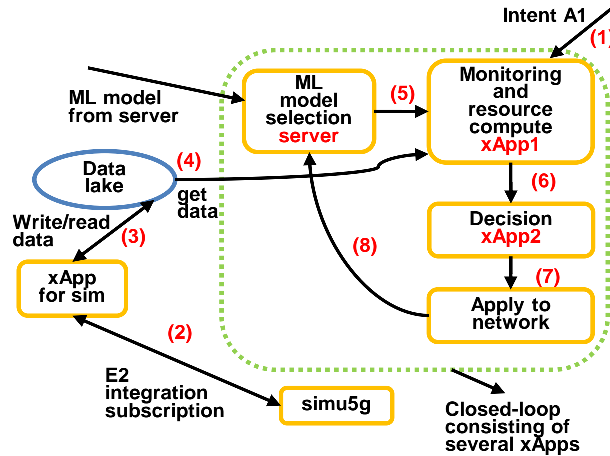


Fig. 20 – Block of closed-loop implementation for an emergency slice.

Section 2.1 describes a closed-loop representation and design using a standard template and demonstrates it using a three-node closed-loop (i.e. SRC node, ML node and SINK node). Here, we further enhance this using a model selection service and a complete ML node implementation using two RIC xApps and demonstrate their deployment using docker containers.

-*ML model selection (server):* Different ML models for inference can be available with different complexity and performance. We dynamically select different ML models from a server based on the declarative specification of the ML model, as described in Section 5. The models are implemented as a docker container and selection may be done either periodically or based on an external request. These ML models can be either specific to a particular problem or a general purpose one. The idea is that some ML algorithms might be too costly but can have a good prediction accuracy. On the other hand, there might be cheap ML algorithms with low-quality inference. Depending on the requirements, the best ML model can be selected.

-*Monitoring and resource compute (xApp 1):* Advanced ML algorithms are applied for monitoring RAN resources (i.e., PRB, physical resource block utilization). This paper uses Gaussian Process Regression (GPR) as a non-parametric prediction technique. xApp1 reads data from a data lake either periodically or when needed. Then, it predicts how much resources will be available in the near future using this data. This information is used for other xApps to make resource allocation decisions.

-*Decision (xApp 2):* After receiving the forecasted RAN resource in the near future, xApp 2 makes a resource allocation decision for the current and emergence slices depending on their SLA requirements.

This section designs and studies the closed-loop analysis and decision parts. Communication between xApps is provided through RIC message router (RMR)messaging used within the O-RAN software community. The workflow of the implementation shown in Fig. 20 is as follows:

(1) Get intent from a higher loop. It indicates if there is an emergency case and monitoring xapp is triggered.

(2) Subscribe to SRC to get the simulator/testbed data.

(3) Write data to the data lake to be used later for ML training.

(4) Data is sent to the ML node (implemented in xApp1) for model training and inference .

(5) Different ML models can be selected from the server here and sent to xApp1 for inference.

(6) xApp1: Resource monitoring such as PRB utilization. Here, the ML model is used, which can be fetched from our local repository. It also analyses whether there is an overutilization/underutilization.

(7) Result obtained on (6) is sent to xApp2, which will make the final decision. It decides whether there is a need to allocate more resources on RAN for an emergency slice. Then it applies the decision to the real network (allocate more PRB for the emergency slice, E2 CONTROL).

8.5.1 The system implementation

A low-level closed-loop needs to be instantiated that monitors and computes RAN resources and makes a resource allocation decision for emergency cases based on the high-level intent.

The xApp1 monitors RAN resources and makes forecasts for the future PRB usage of the network. It also computes the available resources in the RAN domain. The forecasting and resource information is sent to xApp2, which is our decision xApp, through the RMR. RMR is developed by the O-RAN Software Community (SC), and we also utilize this messaging protocol in our implementation. xApp2 receives the necessary information from xApp1 and solves the problem P2 (or P1), which are given next, to find out the necessary PRB resources needed for ES and make the resource allocation. The output of xApp2 will be sent to the real network to be applied through the E2 interface of O-RAN when the real integration starts.

We implement a docker container that acts as a web server where we keep different ML models to be used for monitoring or any other activities to test the model selection. model_handler.py implements an ML selection/pulling task in which we select ML dynamically depending on the performance of the current ML, thus enabling us to use another ML that may have a better performance than the current model. Fig. 21 shows the implementation details of this activity.

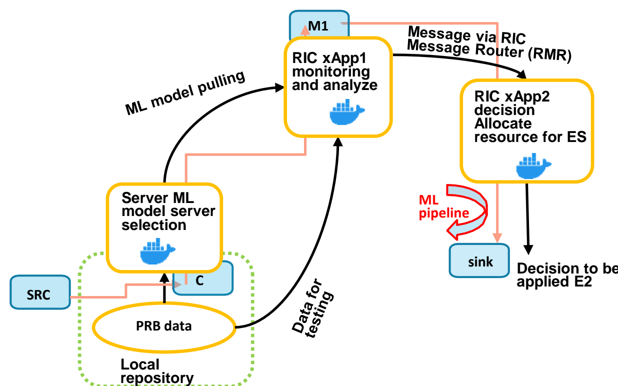


Fig. 21 – Workflow of the implementation.

8.5.2 Results and analysis

8.5.2.1. Time-series forecasting of traffic for monitoring using Gaussian Process Regression

This section studies NS implementation in the network to have dedicated network resources over various domains. For example, the operator can allocate dedicated frequency resources (PRBs) to each slice at the RAN domain. Furthermore, different slices may have different SLA requirements on latency, bandwidth, reliability, etc. Even though each slice's need for resources can vary over time under dynamic networking conditions, the operator needs to ensure that the underlying infrastructure SLAs for each slice is guaranteed. For example, an operator can deploy a separate slice for video streaming. It needs to allocate additional resources to meet the SLA requirements on the slice during peak hours of the day. In case of an emergency, a new slice, Emergency Slice, ES, must be deployed by operators to handle the traffic in the emergency area, and the necessary amount of resources must also be allocated to the ES. In this study, NS with ES is a dynamic resource allocation problem in the RAN domain. When an emergency occurs, the ES is deployed and the resources needed for the ES are maintained autonomously.

To achieve autonomous resource management, traffic prediction of each slice is critical to gather information on the minimum amount of resources needed for the SLA requirements. It is complex to capture the dynamics through linear models due to the highly dynamic and non-linear patterns exhibited by wireless traffic. Artificial Neural Networks (ANNs), also known as deep neural networks or recurrent neural networks are commonly applied for traffic prediction. However, NN has well-known training challenges, and it is complicated to interpret the outcome of the NN prediction. Comparatively, Gaussian Process Regression (GPR) has continuously gained attention due to its interpretability and prediction accuracy. In addition, GPR can also provide information on the uncertainty of prediction, which is important when making resource allocation. In this study, PRB usage measurement is used to reflect the traffic characteristics and a time-series forecasting problem is formulated in which PRB utilization is predicted using GPR.

We study the use of GPR for the prediction of traffic. The PRB usage characterizes traffic which enables us to predict PRB utilization in the RAN domain. Real-world data from [b-raida] in an urban area shows PRB utilization measured over a Long-Term Evolution (LTE) network for a user and collected and reported at every 500 ms. Fig. 22 shows 1000 samples of PRB utilization data.

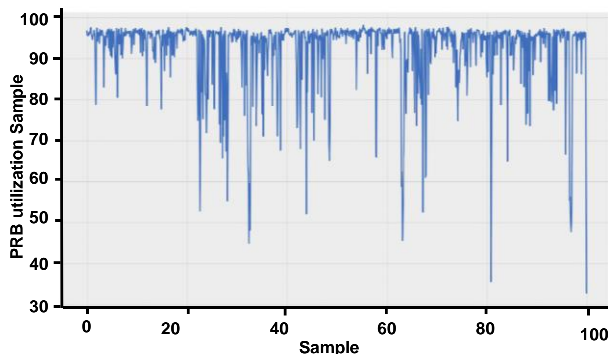


Fig. 22 – PRB utilisation over 1000 sample points.

Understanding the characteristics of the data is important in selecting the best kernel for GPR. Periodic and varying data characteristics observed in Fig. 23 and Fig. 24 enables us to determine a good kernel for PRB prediction with GPR, representing both types of characteristics.

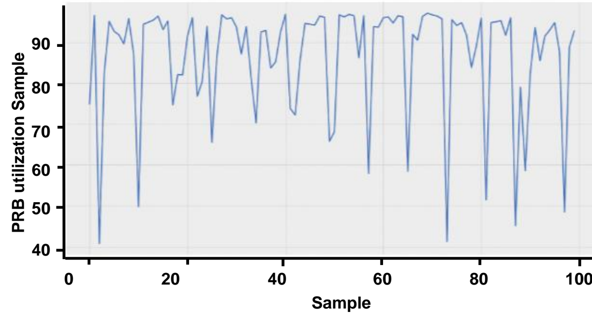


Fig. 23 – Periodic-type characteristics over a period of 100 time-step.

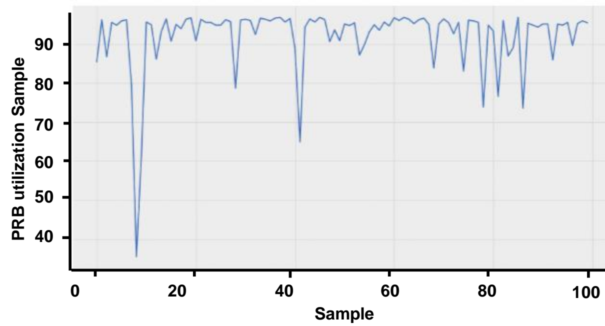


Fig. 24 – Constant-type characteristics over a period of 100 time- time.

Forecasting with GPR

Fig. 25 shows PRB forecasting with GPR. The GPR model is trained with the last 100 samples and the chosen kernel as described above. We note that more data may need to be used for training depending on the application. Then, the trained GPR is used to make predictions for the future 50 samples. It can be concluded that the prediction with GPR is good enough to make efficient resource allocation proactively, as shown in Fig. 25. Note that GPR also provides information on the uncertainty of these predictions as we point to the upper bound when making predictions for the next 50 points. These upper bounds on the predictions can be utilized when making resource allocation to ensure that the correct amount of PRBs is allocated while satisfying the SLA requirements.

The PRB data is stored in a local repository. It is also possible to use different ML models for inference. An example implementation for O-RAN integration is implemented as a separate xApp and prediction_xapp.py creates a docker container for the inference implementation as a micro-service to be used for O-RAN.

We evaluate the performance of GPR prediction. We train with 1000 data points and evaluate the performance in terms of Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). The MAE and RMSE for future 4000 points are 0.077 and 0.147 for entity reference of 0.046 and 0.081, respectively.

8.5.2.2. Resource allocation at RAN for an emergency slice

After the predicted traffic is obtained through GPR, the next step is to determine how many resources the ES should allocate. Therefore, we need to consider the SLAs of other slices in the network. The SLAs of other slices may degrade if we allocate more resources than the ES needs. The emergency case cannot be handled if we allocate fewer resources for the ES than it needs.

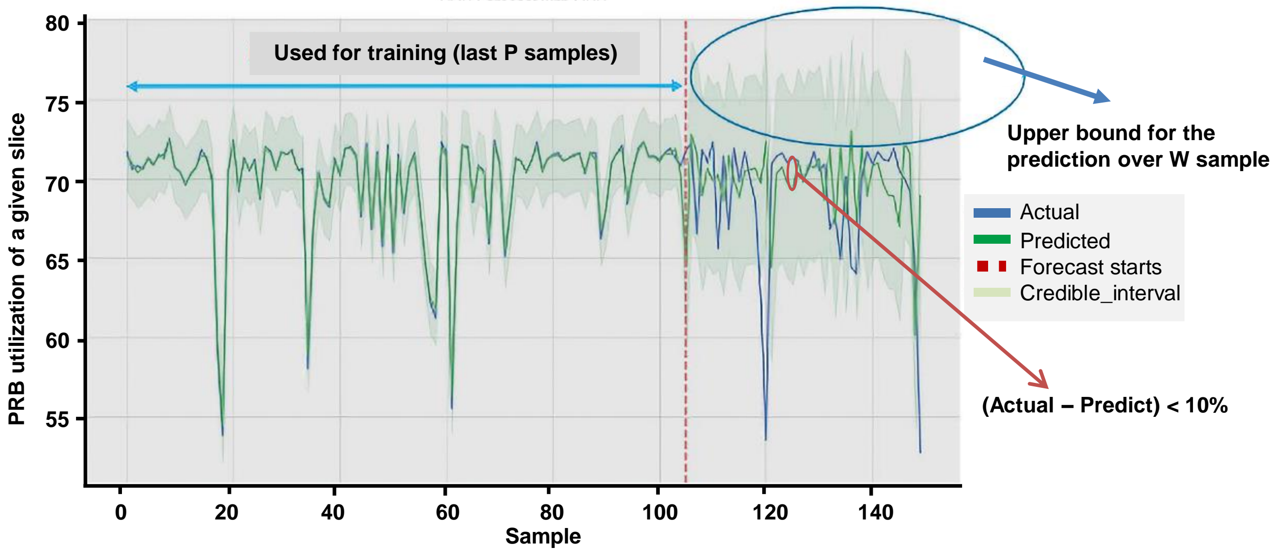


Fig. 25 – Time-series forecasting of PRB utilization with GPR.

We assume that RAN resources are given in terms of either frequency resources or PRBs. Different slices allocated to different amounts of PRBs can be determined and fixed by the operator. However, it is not always efficient because not every slice is active all the time and uses its PRBs with 100% utilization although this strategy is good enough to have a dedicated network. This means that there can be some leftover PRBs that are not used by the corresponding slices [b-foukas], [b-okic].

We consider two cases:

- Case-1: The ES does not have any dedicated PRB allocated, but it can only use the unused PRBs from other slices. The advantage of this strategy is it guarantees the SLAs of other slices, but ES can have significant degradation because it can only use the leftover PRBs, and after some time the leftover PRBs may not be large enough to support the emergency case.
- Case-2: we dynamically borrow PRBs from other slices to support the emergency case to minimize the degradation of the SLAs of other slices. In this strategy, priority is given to the ES and we guarantee that the emergency case is solved successfully while also minimizing the negative impact of borrowing PRBs on the SLAs of other slices.

We develop two algorithms to implement these two strategies:

Leftover GPR-based PRB allocation to ES algorithm (ALG1)

ALG 1 implements the first strategy in which only the leftover PRBs from other slices are allocated to the ES. The details of ALG1 are given in ALGORITHM 1.

To illustrate the operations of ALG1, let us consider two slices and the allocated PRBS to these slices: T1 = 40 PRBs and T2 = 60 PRBs and in total, the system has T = 100 PRBs. Let us also assume that the PRB utilization of these slices is 80% and 90%, respectively. That means the first slice uses only 40*0.8=32 PRBs and the second slice uses only 60*0.9=54 PRBs. Hence, 40-32= 8 PRBs from the first slice and 60-54=6 PRBs from the second slice (in total 14 PRBs) can be allocated to the ES with ALG1 for this example.

Priority GPR-based PRB allocation to ES algorithm (ALG2)

ALG2 implements the second strategy in which we borrow PRBs from the other slice while minimizing

the negative impact on both of them. We assume the ES needs an E amount of PRBs. First, we allocate the available leftover PRBs to the ES. If it is not enough, we borrow PRBs from other slices by minimizing their performance degradation. The details of ALG2 are given in ALGORITHM 2.

PRBs are borrowed from other slices to meet the requirement of ES and also minimize the resource shortage of other slices. Thus, the P1 optimization problem is formulated in ALG 2

Since it involves a non-linear operation with a max b-tosca] operator, the problem is difficult to solve. However, we use an auxiliary trick and convert this problem to an easily solvable integer program. This problem is transformed into a solvable integer problem using the auxiliary variable u_n as shown P2 in ALG 2.

The importance of P2 is to decide how many PRBs are to be taken from each of the other slices and allocated to the ES. These two algorithms are implemented, and decision_xapp.py creates a docker file to run this implementation as a microservice to be ready for use in the O-RAN platform.

***ALGORITHM* 1**: *Leftover GPR-based PRB allocation to ES algorithm (ALG1)*

Input : T = total available PRBs of the system (i.e., For LTE, 100 PRBs).

W = Prediction window (i.e., next prediction time, 500 ms).

P = Past training window (the last 100 samples).

o = Compensation.

N= number of slices in the network.

T_n= Amount of PRBs allocated to slice n.

D_n= PRB utilization time-series data for each slice.

Output : Allocate PRBs to ES: P_{ES}

1 **For** *each other slice n*

2 ***Step 1:*** Train GPR with the latest P training data.

3 ***Step 2:*** Forecast PRB utilization over the next W samples with GPR : U_n.

4 ***Step 3:*** Calculate maximum possible PRB utilization using upper bound: $C_n = U_n + o_n$.

5 ***Step 4:*** Calculate the forecasted PRB usage of all other slices over next W samples : $B_n = T_n C_n$

6

7

8

9

10

11

12 **End**

11 ***Step 5:*** Calculate available PRBs for Emergency Slice:

12 $P_{ES} = T - \sum_{n=1}^{N} T_n C_n$

13 ***Step 6:*** Allocate PRBs to ES: P_{ES}

$\boldsymbol{ALGORITHM\ 2}$: *Priority GPR-based PRB allocation to ES algorithm (ALG 2)*

Input :

T =Total available PRBs of the system.

T_n= Amount of PRBs allocated to slice n.

W = Prediction window.

P = Past training window.

o = Compensation.

N= number of slices in the network.

D_n= PRB utilization time-series data for each slice

E = amount of PRBs needed for an emergency slice

Output : $x_n(t)$ = amount of PRBs needed for slice n at time t

$y_n(t)$ = amount of PRB taken from slice n at time t

1 **P1:**

2 $min \sum_{t=1}^{W} \sum_{n}^{N} max \{0, x_n(t) - (T_n - y_n(t))\}$

3 s.t. $0 \leq y_n(t) \leq T_n \; \forall n$ $\sum_{n}^{N} \; y_n(t) \geq E$

4

5 $-\; x_n(t)$: PRB usage for slice n at time t

6 $-\; T_n$: Total PRBs given to slice n

7 $-\; y_n(t)$: Number of PRBs taken from slice n at time t to be used for emergency slice

8

9

10

11

12 **P2:**

13

$$\min \sum_{t-1}^{W} \sum_{n}^{N} u_n(t)$$

$$\tilde{x}_n(t) + o_n(t) - (T_n - y_n(t)) \leq u_n(t)$$

14

$$0 \leq y_n(t) \leq T_n \ \forall n$$

15

$$\sum_{n}^{N} y_n(t) \geq E$$

16

$$u_n(t) \geq 0$$

where $x_n(t) = \tilde{x}_n(t) + o_n(t)$

17

$x_n(t)$: Actual PRB usage at time t in future. This cannot be known in advance.

18

$\tilde{x}_n(t)$: Estimated PRB usage with GPR

19

$o_n(t)$: Estimation error. Upper bound provided by GPR can be used.

20

21

In a simulation scenario, we assumed that we have two slices with different PRB requirements:

- T1: number of PRBs assigned to Slice 1 by the operator (e.g., T1=40)
- T2: number of PRBs assigned to Slice 2 by the operator(e.g., T2=60)
- T: total number of PRBs in the system (e.g., T= 100 PRBs)
- Time series PRB utilization for each slice, in percentage, with a granularity of 100ms and 200ms

The performance of ALG2 was studied under the scenario that there are two other slices and T1=40 and T2 = 60 PRBs allocated to them. By applying ALG2, we borrow PRBs to satisfy the requirement of the ES when those slices do not need the resources. We assume the ES needs 20 PRBs. Fig. 26 shows the number of PRBs taken from other slices over 45 time instants. Depending on the predicted PRB usage of other slices, ALG2 takes 11 or 12 PRBs and 8 or 9 PRBs from the first and second slices, respectively, and 20 PRBs in total are ready to be used by the ES simultaneously.

This section studies NS implementation in the network to have dynamic resource allocation over various domains. To analyze the PRB utilization in RAN, two algorithms were studied. Results are presented considering the need for resources of each slice which can vary over time under dynamic

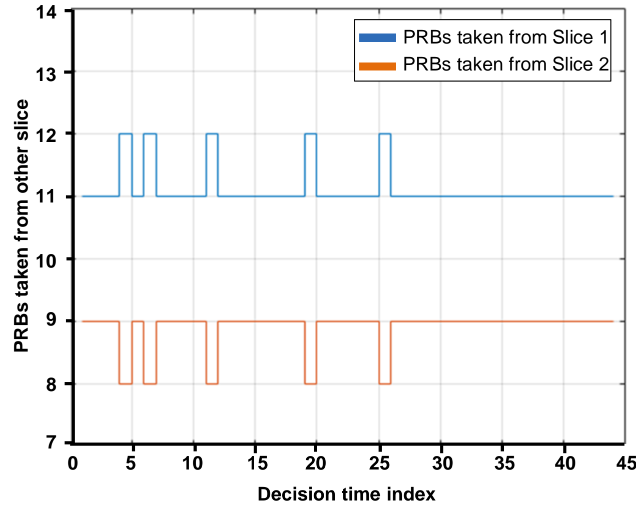


Fig. 26 – PRB allocation for ES.

networking conditions. The results show the importance of closed-loop implementations in NS, especially for intelligent management of RAN resources during emergency scenarios.

8.6 O-RAN Control-Loop Instantiation

This section describes the implementation of the algorithms described in Section 4 in an O-RAN near Real-Time RAN Intelligence Controller (RIC) [b-nrt-ric] and its integration with the Acumos [b-acumos] model repository. The model description is included in the declarative specification of closed-loop as discussed in Section 2.1. In this study, a pretrained model is fetched from Acumos based on the given description and deployed as xApp [b-oran-sdk] in the O-RAN platform (See Fig. 27 for details).

8.6.1 A solution workflow

The following workflow is used for the implementation:

- RAN (E2-SIM [b-oran-sc] is used) is registered and associated with O-RAN near RT RIC.
- RIC receives policy updates from A1 for triggering closed-loop PRB allocation.
- An ML model is fetched based on the A1 policy details.
- PRB utilization is predicted based on the analysis of test data used instead of actual data from E2.
- The PRB to be allocated is computed and an E2 control message is sent based on the inference. PRBs are always reserved for the emergency slice and additional resources can be reallocated based on situational considerations.
- The allocation decision is continuously monitored, evaluated, and improved upon.

The workflow steps are further explained in Fig. 27 and discussed as follows:

- Points 1 and 2 show that E2 SIM is up and that association with RIC is set up.
- Point 3 shows the nRT RIC receives the A1 policy update to trigger closed-loop monitoring.
- Point 4 shows the A1-mediator sends A1 Policy REQ to the "*prbpred*" xApp.
- Points 5a and 5b show the model is fetched from the model store as per policy guidelines and "*prbpred*" instructs DataMon/Alloc xApp to start monitoring the data.
- Points 6,7, and 8 show the messaging done for subscribing to E2 for data.

E2 Indication is for future reference; currently, data is not monitored through RIC Indication. In future, data needs to be monitored and sent to predict xApp.

- Point 9 shows data reception from the E2 node. The received metrics are stored in metrics DB as in Point 10.
- Upon timer expiry as in Point 11, a request for prediction is sent to “prbpred” xApp as in Point 12.
- “*prbpred*” uses the ML model to predict future utilization. Retraining may be done based on the new data model. The predicted values may be sent to DataMon/Alloc xApp as in Point 13 and Point 14.
- DataMon/Alloc xApp computes the PRB to be
  allocated and sends the E2 control message towards E2 as in Point 15.

8.6.2 Resulting implementation

This section presents the implementation of the algorithms in Section 4. The algorithms can be instantiated in the O-RAN-RIC platform and prediction based on the xApp onboarding/deployment process and RIC platform components can be achieved.

The xApps are developed based on the xApp Framework for Python. Separate xApp-descriptor files were defined detailing the configuration, RX & TX messages supported:

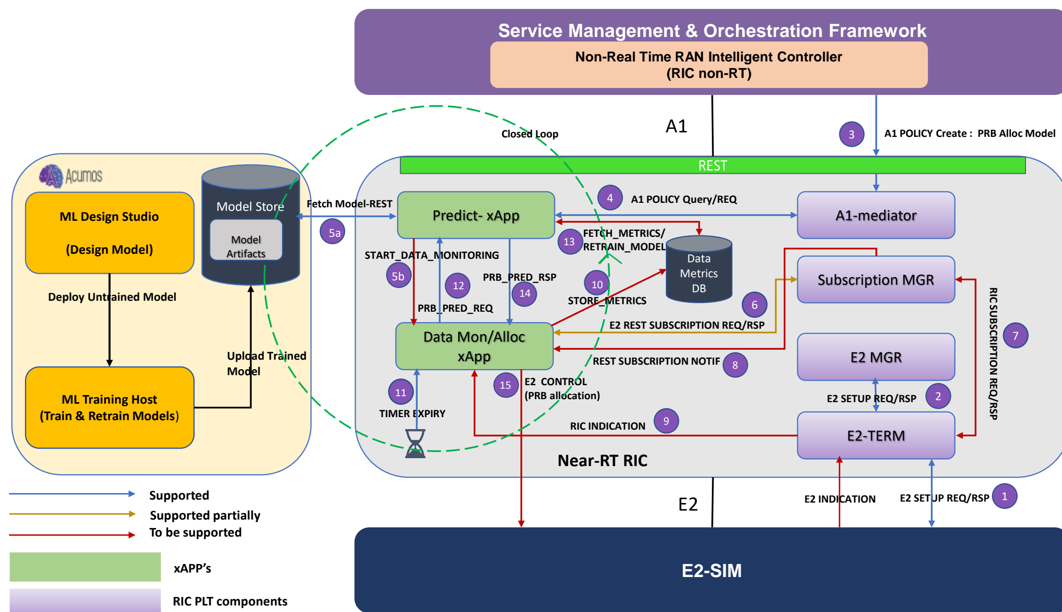


Fig. 27 – Modified xApp process model.

```
        "xapp_name": "alloc",
        "version": "0.0.2",
        "containers": [
            {
                "name": "alloc",
                "image": {
                    "registry": "nexus3.o-ran-
sc.org:10002",
                    "name": "o-ran-sc/ric-app-alloc",
                    "tag": "0.0.2"
                }
            }
        ],
        "messaging": {
            "ports": [
                {
                "name": "http",
                "container": "alloc",
                "port": 10005,
                "description": "http service"
                },
                {
                    "name": "rmr-data",
                    "container": "alloc",
                    "port": 4560,
                    "txMessages":
["PRB_PRED_REQ","RIC_HEALTH_CHECK
_RESP"],
                    "rxMessages":
["PRB_PRED_RESP",
"SUBSCRIPTION_REQ","RIC_HEALTH_CHE
CK_REQ"],
                    "policies": [],
                    "description": "rmr receive data port
for alloc"
                },
                {
                    "name": "rmr-route",
                    "container": "alloc",
                    "port": 4561,
                    "description": "rmr route port for
alloc"
                }
            ]
        },
        "rmr": {
            "protPort": "tcp:4560",
            "maxSize": 2072,
            "numWorkers": 1,
            "rxMessages": ["PRB_PRED_RESP"],
            "txMessages": ["PRB_PRED_REQ"],
            "policies": []
        },
        "controls": {
            "fileStrorage": false
        },
        "db" : {
            "waitForSdl": false
        }
}
```


Fig. 28 – xApp-descriptor file for prbpred xApp.

```
{
    "xapp_name": "prbpredxapp",
    "version": "0.0.2",
    "containers": [
        {
            "name": "prbpredxapp",
            "image": {
                "registry": "nexus3.o-ran-
sc.org:10002",
                "name": "o-ran-sc/ric-app-prbpred",
                "tag": "0.0.2"
            }
        }
    ],
    "messaging": {
        "ports": [
            {
            "name": "http",
            "container": "prbpredxapp",
            "port": 10003,
            "description": "http service"
            },
            {
                "name": "rmr-data",
                "container": "prbpredxapp",
                "port": 4560,
                "rxMessages": [
                    "A1_POLICY_REQ",
                    "PRB_PRED_REQ"
                ],
                "txMessages":
[ "A1_POLICY_RESP",
"PRB_PRED_RSP","A1_POLICY_QUERY" ],
                "policies": [20008],
                "description": "rmr receive data port
"
            },
            {
                "name": "rmr-route",
                "container": "prbpredxapp",
                "port": 4561,
                "description": "rmr route port "
            }
        ]
    },
    "rmr": {
        "protPort": "tcp:4560",
        "maxSize": 2072,
        "numWorkers": 1,
        "txMessages": [
            "PRB_PRED_RSP",
            "A1_POLICY_RESP",
            "A1_POLICY_QUERY"
        ],
        "rxMessages": [
            "PRB_PRED_REQ",
            "A1_POLICY_REQ"
        ],
        "policies": [20008]
    }
}
```


Fig. 29 – xApp-descriptor file for allocator xApp.

- *prbpred* xApp: Initially, this xApp registers for PRB_PRED_REQ (PRB Prediction Request) and A1_POLICY_REQ (A1 Policy Request), and queries A1-mediator to get the policy details. A specific policy was created which gives model information and model version information to be used. This xApp is responsible for receiving *A1_POLICY_REQ* and saving the policy details. Fetch the model from the *modelStore* and save it. Predict the future PRB utilization and respond to *alloc* xApp for further processing based on the timer trigger. Upon reception of *PRB_PRED_REQ*, the xApp predicts PRB utilization for each slice and sends a response to *Alloc* xApp based on the model fetched. Fig. 28 shows the xApp-descriptor file for the *prbpred* xApp.
- Allocator xApp: Initially, this xApp registers with the subscription manager for E2 information and starts a timer to trigger PRB_PRED_REQ periodically. PRB is allocated for an emergency slice based on predicted future PRB utilization. A simple algorithm for PRB allocation in Section 4 is used here. In addition, some PRBs are shown as reserved for emergency/high priority events.

The assumption taken is the total number of PRBs in the system is 100. Slice #1 and Slice#2 were configured with 35 PRBs each. 30 PRBs were reserved for emergency/high priority events.

The actual Value of PRB utilized is computed based on the predicted PRB utilization received for each slice.

Utilised_PRB_slice1=PRB_ALLOC_SLICE1*(slice1_utilisation/100)

Utilised_PRB_slice2=PRB_ALLOC_SLICE2*(slice2_utilisation/100)

total_prb_avail = Total_PRB – (Utilised_PRB_slice1 + Utilised_PRB_slice2)

The reserved PRBs are also made available because this is an emergency event. Alloc xApp sends the E2 control message to allocate the available PRBs from the calculation. Fig. 29 shows the xApp-descriptor file for allocator xApp.

Successful communication between the xApp and other RIC platform components was achieved as part of this. A model store was developed to mimic Acumos and have access to the pretrained model. E2 SIM setup was registered with the E2 component in the RIC platform.

In the Dawn release, the creation of the A1 policy instance doesn’t trigger the A1 policy to send a message towards the xApp [b-oran-sc]. The workflow was modified to send a timer-based event from alloc XApp to trigger PRB prediction. When the policy instance is created (CREATE/UPDATE messages are sent to xApp by A1 mediator), the prbpred xApp can store the model information and perform prediction based on the trigger.

8.7 Integration of the POC

This section describes the integration of the above implementation of closed-loops into P-RAN-based software platform ready to be tested in the 5G Berlin testbed [b-FGAN-I-197]. The operator inputs the declarative intent to the Service Management Orchestrator (SMO)/Non-RT RIC, which describes the use case to detect emergencies and maintain the required SLA as described in Section 2.1. Similar to the mechanism described in 2.1, SMO/Non-RT RIC then creates a higher loop that monitors various parameters like network activities, input from emergency responders (ER), social media trends, etc. to detect and locate the emergency (e.g., fire in a building). This can be realized using either a hosted model in Acumos or Open Network Automation Platform data collection analytics engine (ONAP DCAE) or O-RAN rApp, as discussed in sections 2.2 and 4. Once the emergency is detected, the higher loop sends an intent over the A1 interface to the Near-RT RIC, instructing it to handle the increased load for the corresponding RAN node. Real-time ML/AI inference might be needed by some of the ERs' devices; for firefighters a helmet-mounted camera may use image recognition to detect humans in a burning building. However, the devices might not have enough computing and might need to offload the task to the network edge or use split AI/ML models for inference. The Near-RT RIC receives the intent and creates a closed-loop which can monitor the network and compute resources of the edge and the ER device and maintains the SLA/QoS (quality of service) of the inference task as discussed in Section 3 above. This loop can be realized using xApp. Fig. 30 shows the simulator-based sequence for the integration of the activities.

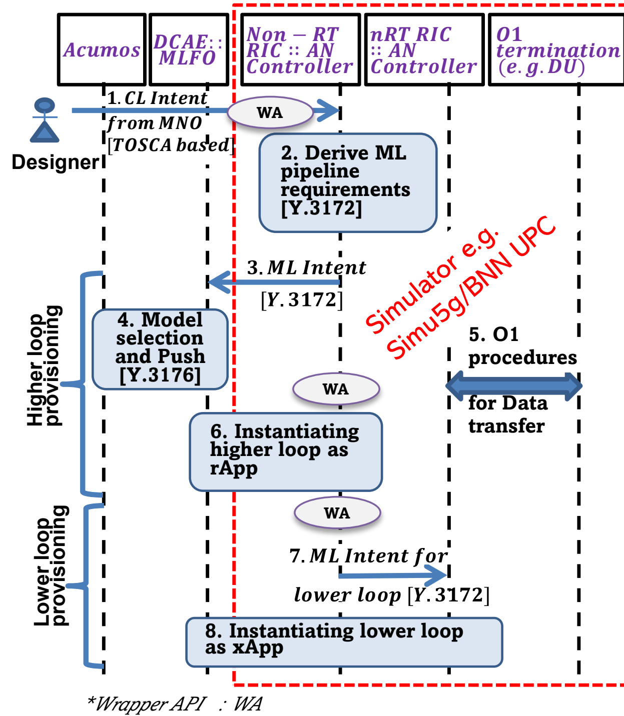


Fig. 30 – Simulator-based sequence for the integration of the activities.

Fig. 31 shows the extensions to the sections above to integrate the implementation of the algorithms described in Section 4 in an O-RAN- RIC and its integration with the Acumos [b-acumos] model repository. The first addition was the A1 poller which pulls the A1 mediator at regular intervals and converts the A1 policy to a TOSCA template described in Section 2.1. It uses HTTP-based interfaces to communicate with the A1 mediator and the orchestrator. The dms_cli tool provided by the O-RAN-SC was used to enable the orchestrator to orchestrate the xapps as specified in the A1 policy. Playbooks (workflows) described in Section 2.1 were updated to integrate relevant command line (dms_cli) commands. These commands are used to onboard and install corresponding xApps.

The overall flow of the final integrated solution (see Fig. 31) is as follows:

1. The human user or a higher loop applies an A1 policy to the near-RT RIC. This policy is received by the A1 mediator.

2. The A1 poller gets the policy, translates it into the TOSCA template and sends it to the orchestrator.

3. The orchestrator manages the RIC xapps according to the TOSCA template using dms_cli.

4. The newly orchestrated xapps pull the necessary models from the model repository server.

5. *Pred* xapp makes a time-series prediction for future traffic in the network and how much resources (PRB) will be available for an emergency slice in the near future.

6. *Alloc* xapp sends an RMR request to *pred* to get the prediction and allocates PRBs to the emergency slice based on that.

7. *Alloc* xapp then sends a message over the E2 interface to the RAN. Slice allocation messages are verified from the console.

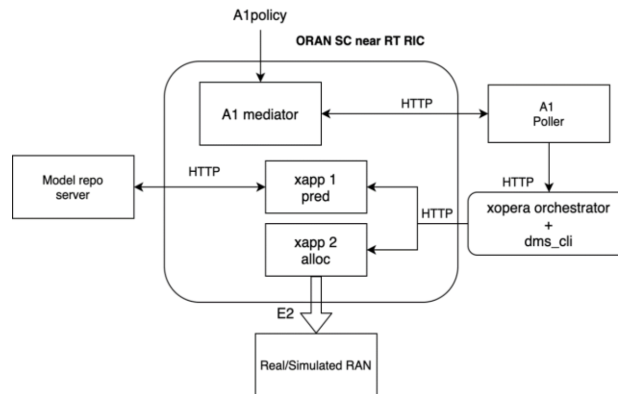


Fig. 31 – Overall flow of the final integrated PoC .

8.8 Observations from the PoC

Abstraction of nodes allows the service template to select concrete nodes that best match the requirements of the abstract nodes during deployment. The concrete nodes can be provided in a repository known to the orchestrator. Abstract requirements can be achieved in TOSCA YAML using the *node_filter* feature. However, this study found that abstraction features like *node_filter* and *substitution* are not supported by certain implementations of orchestrators. A feature-based comparison of orchestrators concerning TOSCA compliance may be made as part of future studies.

Besides the RAN slicing experiments exploring a closed-loop using the FlexRAN controller, other AI applications can interact with the SDN controller and the VNF placement functions to attend to different network requirements. An End-to-End (E2E) network slice cannot be completely implemented in the testbed because a MANO implementation was not used to avoid computational costs and network complexity in this first phase and focus on AI integration. Future studies may include integrating the software developed in our testbed into ONAP software, a popular MANO implementation, to provide E2E NS with a centralized closed-loop. The verification and validation of resource allocation during simulation in line with the traffic pattern (e.g. full buffer) when simulating the scenarios, e.g., dual connectivity, is an essential future step as we broaden the simulation into more scenarios.

Creating a closed-loop with several modules brings communication and computation problems. Overall integration, including A1/O1/E1 interface integrations, is critical and which parts of this integration can be realized autonomously can be explored. The real-time system performance will have to be tested to ensure compliance with closed-loop specifications. Integration issues with platforms highlight the importance of close coordination with underlays, as mentioned in sections 2.2 and 5.

8.9 2021 Conclusions and future research

This is a collaborative study where we developed and implemented a hierarchical closed-loop that autonomously handles an emergency case. The study focused on intent parsing, traffic monitoring, resource computing, and allocation autonomously. The closed-loops were implemented with several micro-services deployed as docker containers with specific functions such as monitoring, computing, ML selection, and resource allocation. Future activities will focus on enhancing the attributes of the nodes in the template, e.g., data parameters in the SRC [e.g., 3xVs: velocity, variety and volume], Model metadata (as defined in ITU-T Y.3176), and SINK parameters [e.g., underlay specific APIs]. After integration, data pulling, model pulling, and adaptation can be demonstrated

based on such enhanced attributes. The machine learning agent presented in the Connected AI study will be implemented for the built environment with ONAP and Acumos integration for future activities. Enhancing the simulator to include inputs from an intent and integration with the SRC, ML and SINK nodes to form the closed-loop in the simulation domain is also an important future step. Apart from advanced algorithms studied, e.g., multivariant time-series models with monitored data and arriving at intelligent inference, resource reservation for emergencies and resource reallocation from lower priority services should be explored. Easy onboarding of xApps and the triggering of policy towards lower closed-loops and supporting visualizations can increase usability. In addition, extending the solution to self-learning closed-loops with continuous collection, analytics, decision and actuation and model performance detection needs further study. With the self-learning close-loops, the network could trigger a switchover to another better performing model, analyze and trigger a different set of data/measurements for data analysis and perform synchronization and management across the edge and emergency responder devices.

9 2022 PoC Description

The main contributions of this study are summarised as follows:

- Analysis of specific use cases such as:

AN-usecase-001 [ITU-T Y.Supp 71], “Import and export of knowledge in an autonomous network”, produce a design as per the reference design in the Build-a-thon repository. We also provide the corresponding code based on the reference code in the Build-a-thon 2022 repository.

Use case 15 from [ITU-T Y.Supp 71] and analysed the design. Solution will scan all incoming log traces continuously, detect system issues and will report issues as Incidents with probable root-cause automatically. Achieves it through historical log data learning and prediction is from new live log sentences. While producing the report solution will take feedback from user and re-train the Log Anomaly detection to do self-correction. We provide extensions to the reference code provided in [Build-a-thon 2022] and build our own graph based on the reference code.

We analyze use case 23, Autonomous agents (with varied competence) in networks. Data by sensors such as logs, packet measurements, Deep packeted inspection, etc are managed by autonomous agent. The agent specifically create collects the data, analyze and perform action with minimal human intervention.

We analyse Network resource allocation for emergency management based on closed loop. Solution will scan all incoming KPIs and slice config continuously, including any changes in slice config for new slice deployment and RL agent will optimise the slice config, accordingly, using calculated rewards based on SLAs.

We analyse FG-AN-usecase-41(3GPP TR 28.809 V17.0.0), “KPI anomaly analysis for 5G networks and beyond” and produce a design as per the reference design in the Build-a-thon repository. Solution will scan list of Network Functions(NF) from EMS, identifies NF for KPI retrieval, forecasts the KPIs from these NF for future time period, identifies anomalies from the forecasted KPIs, identifies Rouge NF causing KPI degradation, blacklists Rouge NF. Achieves it through historical KPI data from NF.

We analyse FG-AN-usecase-38, 1.1 “AN enabled end-to-end supply chain”, produce a design as per the reference design in the Build-a-thon repository. We also provide the corresponding code based

on the reference code in the Build-a-thon 2022 repository. Enabled end-to-end of Unified Standards & Guidelines Manufacturing & Operators.

We analyse "Slip Detection (and Force Estimation)" and "Object Detection" in a robotic arm, and produce a design as per the reference design in the Build-a-thon repository. We also provide the corresponding code based on the reference code in the Build-a-thon 2022 repository. After analyzing the use cases, we trained two ML models, one for "Slip Detection (and Force Estimation)" and another one for "Object Detection" we have tested and validated the models for the use cases

- Implementation of a PoC for the distributed autonomous evolution of controllers based on the FGAN architecture [FGAN-O-023]. We specifically address important issues, such as the security, auditability, explainability and scalability of the process. Additionally, the PoC also includes the subsequent automatic deployment of any controller obtained from the evolution.
- Deriving new use cases for the autonomous networks can help us understand the potential of using autonomous networks, in this trial we got a document that contains old use cases between some network components, and from these old ones we apply the link prediction algorithm to investigate the future opportunities of making use of autonomous network, we have reached 86% of accuracy of predicting if the link exists or not, and as a start we predicted more than 6.5k possible relations that can be investigated further

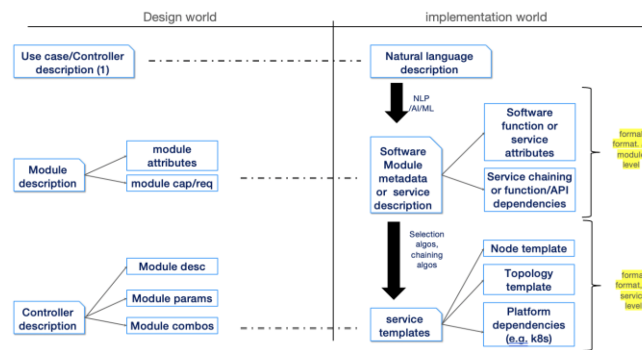


Fig. 32 – Design vs. implementation.

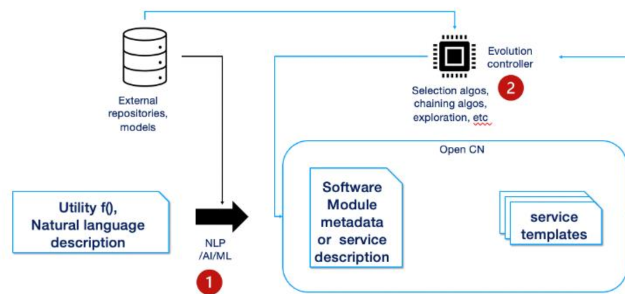


Fig. 33 – Part 1 and part 2 of Build-a-thon.

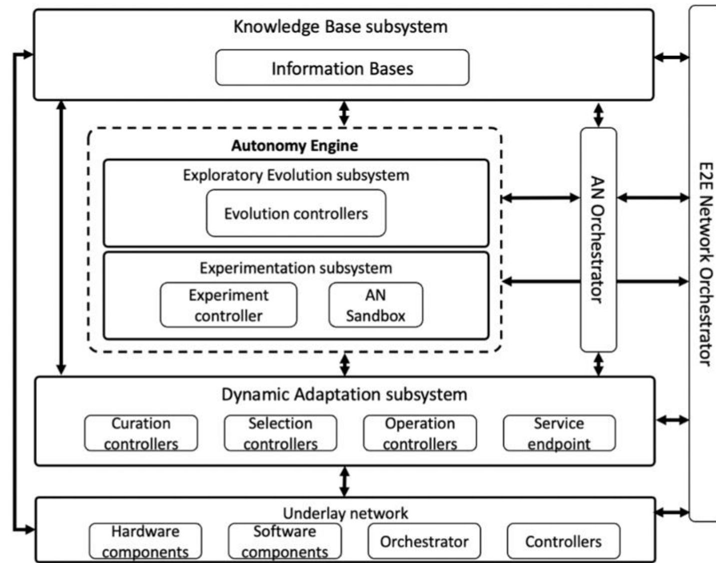


Fig. 34 –High-Level Framework for Evolutionary-Driven Autonomous Network [FGAN-O-023]

9.1 The PoC Design and Implementation

The document [FGAN-O-023] describes a High-Level Architecture Framework for AN, including evolutionary-driven networks. This architecture is shown in Figure 34. A module is "a building block consisting of executable code and a module specification from which controllers are assembled". A module's specification includes its input and output interfaces, and a metadata description of its functionality. Examples of possible modules include aggregation functions, DNS configuration interfaces, an entire deep neural network (DNN) model, a single layer of a DNN model, etc. On the other hand, a controller can be considered as a software closed-loop composed of modules. A controller instance is "an executable representation of a controller including modules, their configurations, and parameter values". Exploratory evolution tries to find a suitable controller for a required use case specification using evolutionary algorithms.

This report describes an end-to-end structural PoC based on the FGAN architecture and on the secure and traceable evolution of controllers. The evolutionary algorithms that would drive the process and the reactive evolution of controllers to operational conditions are beyond the scope of this work.

9.2 Import and export of knowledge in an autonomous network

Knowledge is essential to fulfil the key concepts of autonomous networks (evolution, experimentation and adaptation) while minimizing human intervention. Knowledge in autonomous network ranges from autonomous system environmental data representation, actions/consequences, key configuration options to potential parameter indices. This section presents the use case PoC of knowledge in the AN actors.

9.2.1 The PoC design

As shown in the Figure 34, the system consist of orchestrator, KB manager, auto controller, open CN, ML pipeline, and human operator.

The *orchestrator* is the component responsible for managing workflows and processes in the AN and steps in the lifecycle of controllers. To manage the workflows and processes in AN, AN

orchestrator coordinates with various other functions in the AN as well as outside the AN. Being part of the management plane, AN orchestrator provides interface to human operators in the form of reports regarding the functioning of the AN and human interfaces for configuring the AN, where applicable.

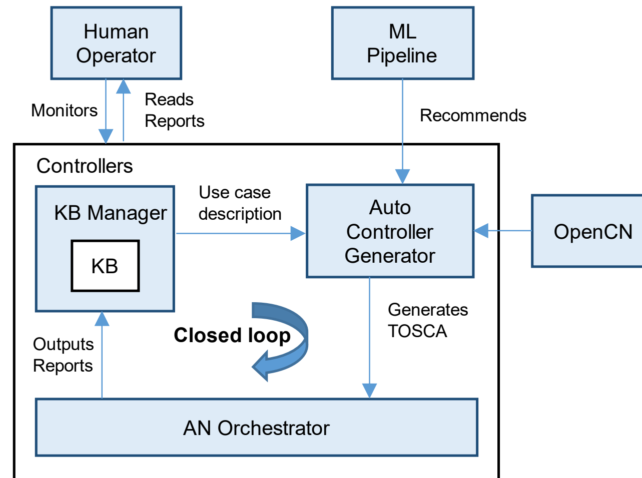


Fig. 35 – High-level flow chart for creation and parsing of example intent.

The *KB manager* is a subsystem which manages storage, querying, export, import and optimization and update knowledge, including that derived from different sources including structured or unstructured data from various components or other subsystems. KB manager is a node which optimizes and manages data available on the closed loop but requires a node which can host its resources. Since AN orchestrator has the capability of hosting node resources it is designed to be a host to the KB manager. Knowledge in AN is a collection of resources that helps in solving a specific type of problem. A knowledge base component manages knowledge derived from and used in autonomous

networks. It is updated and accessed by various components in the autonomous network. The knowledge includes metadata which is derived from the capabilities, status of AN components. This knowledge is stored and exchanged as part of interactions of AN components with knowledge base. Knowledge can be derived from different sources including structured or unstructured data

The *auto controller generator* represent generic software component that can be managed and run by a TOSCA Compute Node Type. It generate controller specifications using the existing repository in OpenCN, the knowledge base and an analytics function aided by AI/ML. Meanwhile the Open CN stores the controllers for the AN.

The *ML pipeline* node is able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyze and draw inference from patterns in data. It hosts analytics and recommends controllers in the AN. On the other hand, the *Human operator* is just like a user friendly interface for human base instructions

9.2.2 The PoC Implementation

The TOSCA metamodel uses the concept of service templates that describe cloud workloads as a topology template, which is a graph of node templates modelling the components a workload is made up of and of relationship templates modelling the relations between those components. TOSCA service template are instantiated at runtime using a TOSCA orchestrator (xOpera) and the

order of component instantiation is based on the relationship between components. TOSCA is one of best and most often used automated testing tools. It is widely employed in large-scale applications to achieve successful outcomes.

In the implementation, the YAML file is generated and stored in a graph database is using neo4j. After the basic template YAML template f the use case is written, Ruamel (a python YAML editing library) was used to populate the nodes of the use case with the actors and relationships from our use case in TOSCA by using information obtained from querying the graph database. Finally, the generated YAML was then validated by parsing it using xOpera. Figure 36. shows the flow chart of the service template generation from the graphDB.

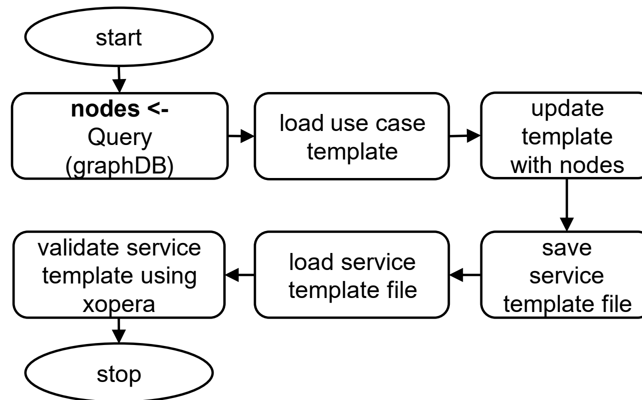


Figure 36: Flow chart of the service template generation from the graphDB.

9.3 Evolution and blockchain: An Autonomous Network Architecture PoC

This section presents a Proof of Concept (PoC) of the decentralized controller evolution architecture for Autonomous Networks (AN) using a distributed Marketplace based on the FGAN architecture (FGAN-I-198 [5]). Such a marketplace serves as a decentralized mechanism for the secure and traceable evolution of controllers, which is achieved by a private Ethereum blockchain and a distributed and decentralized filesystem, the Interplanetary Filesystem (IPFS).

We specifically address important issues, such as the security, auditability, explainability and scalability of the process. Additionally, the PoC also includes the subsequent automatic deployment of any controller obtained from the evolution. This last part contains the TOSCA parsing of a Representational State Transfer (REST) service for the chosen controller.

9.3.1 MODULES, CONTROLLERS AND EXPERIMENTATION

Figure 2 shows the high-level Neo4J graph representation for the PoC closed-loop, which uses all of the concepts presented in Figure 34, except for the Selection and Operation Controller. Note that for the Real Time Online Experimentation the use of a Digital Twin (DT) is considered. A DT, which concept is described in FGAN-I-058 [6], is “a mathematical representation of a physical and/or logical object”. For example, a DT could be a Graph Neural Network (GNN) that reproduces the behaviour of a network, producing outputs such as latency or packet loss rate, given inputs that describe its state and policy. This PoC uses the concept of a DT in the architecture, but does not focus on its implementation, which is a problem out of the scope of this project.

The blue nodes represented in Figure 2 are labelled as actors in the graph, while the green ones are denoted as elements. Actors are controllers that can evolve, although the evolution of actors is

out of the scope of this PoC. Element include supporting artefacts, such as software modules, evolvable controllers, repositories, etc.. The Evolution Controller (Evol_Ctr) stores the list of available modules and uses them to compose an Evolvable Controller (Evola_Ctr) that is sent to the Marketplace (Mark_Plc). Although the Marketplace is represented as a single node, it is a distributed network of several nodes that are part of the rest of the actors. That means that the Marketplace is distributed across all the instantiations of the Evolution Controller, Experimentation Manager, Digital Twin and Curation Controller, ensuring the security and traceability of the evolution process. In this PoC, both modules and controllers are implemented as Python classes. The source code for both definitions is available in [7], in the *Mod_Ctr.py* file, which is reproduced in all the nodes. The fact that all nodes work with the same definition is part of a list of important assumptions that are now explained.

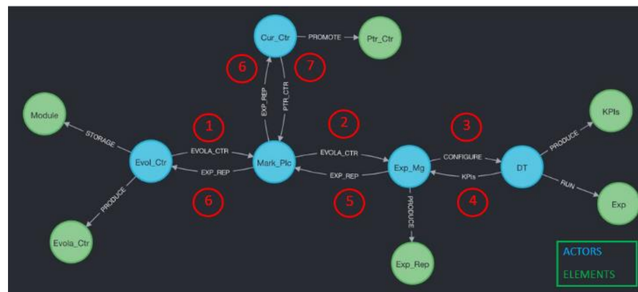


Fig. 37 – Neo4J graph representation of the high-level architecture of the PoC

9.3.2 IMPORTANT ASSUMPTIONS

- The list of available modules, as well as the definition of the classes *Module* and *Controller*, are contained in the *Mod_Ctr.py* file, having each node (blue circles in Figure 37) an identical copy of it. In the case of introducing new modules, something that is not considered in this PoC, a protocol for distributing the information and ensuring consistency is required. However, the Marketplace provides the perfect tool for implementing this feature.
- Since the PoC is focused on laying the groundwork for the evolution architecture, the modules and controller considered are rather simple, reproducing the behaviour of simple mathematical operations. However, we assume that this approach is sufficient for proving our architecture and that it can be improved in the future.
- As for the evolution, the algorithm uses random combination of modules, avoiding repetition of the combinations that have already being tried for which their result is testing is known. We assume that our use of random module combinations is structurally equivalent in this context to those controllers produced by evolution. Of course, they are not expected to be equivalent in their operation.

All these assumptions simplified the PoC implementation, but do not limit its effectiveness when laying the groundwork for the evolution of controllers in a distributed system. New features can be added with a few or no changes in the base work presented here.

The class *Module* have four attributes: module id, function, parameter and representation.

The list of possible functions is limited to the mathematical operations add, subtract, pow, and multiple. These functions are also defined as Python functions in the file *Mod_Ctr.py*.

Figure 38 shows the Python code for the multiple function, *f_mul*. Figure 39 shows an instantiation of a *Module* object with *f_mul* as function with the float 3 as a parameter. The id is a

string explanation of the function, while the representation is the symbol that denotes the mathematical operator. The *Module* class has the method *execute* represented in Figure 40 that, given an input, returns the output of applying the function with the designed parameter. Once a *Module* object is instantiated is possible to modify its parameter with the method *set_method.*

```
def f_mul (x, param):
        return x * param
```

Figure 38: Python code for the definition of the function *f_mul.*

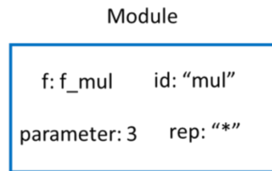


Figure 39: Representation of a *Module* object.

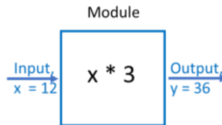


Figure 40: Representation of the method *execute* of a *Module* object.

9.3.3 CONTROLLER DEFINITION

The class *Controller* has three attributes:

- The controller id, which is unique
- A list of the modules that compose the controller
- A list of the parameters for each of its modules

Figure 41 represents the instantiation of a *Controller* object. In that case the two modules implement the functions substract and multiply tuned with the parameters 2 and 10 respectively. After the object is instantiated, it is possible to call the method *execute* as represented in Figure 42. The *executed* method for each *Module* objtect will be called in order, being the output of each phase the input for the next one. Additionally, the *Controller* has a method *get_dict* that returns a Python dictionary that describes its attributes.

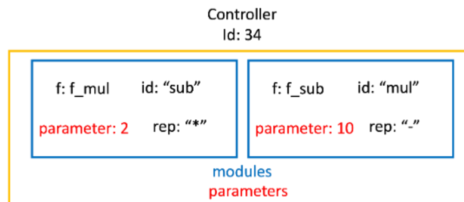


Figure 41: Representation of a *Controller* object.

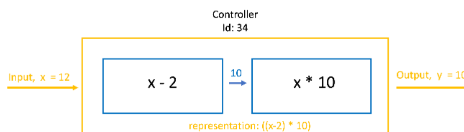


Figure 42: Representation of the method *execute* of the *Controller* class.

A Python dictionary can be easily converted into a JavaScript Object Notation (JSON) file which is the selected format for transmitting the controller information between nodes. The transmitter node sends the JSON description of a controller and since the receiver also has the list of all available modules, it is able to recreate a copy of the controller by calling the function *json_to_ctr (dictionary)*. We have chosen JSON since it is a lightweight data-interchange format that is readable by humans. Figure 43 shows the JSON representation of the *Controller* object introduced in Figure 41.

9.3.4 EXPERIMENTATION AND EVOLUTION

All blue nodes in Figure 37 communicate by exchanging JSON files via the Hypertext Transfer Protocol (HTTP). On the Marketplace side, we have used the language JavaScript for handling the HTTP communication, while in the rest of the blue nodes we have chosen the combination of the Python framework Flask [8] and Gunicorn [9], a Python HTTP server for Web Server Gateway Interface (WSGI) applications.

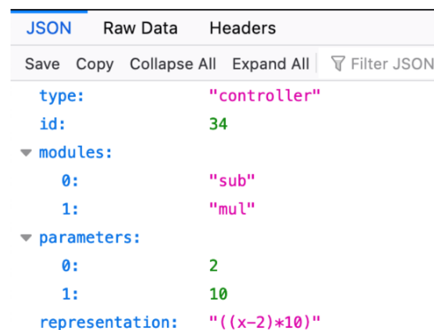


Figure 43: JSON representation of a *Controller* object extracted from the Marketplace.

Following the architecture presented in Figure 37, the Evolution Controller (Evol_Ctr) sends the JSON representation of a new Evolvable Controller (Evola_Ctr) as the one shown in Figure 10. After storing the JSON file, the Marketplace forwards it to the Experimentation Manager (Exp_Mg). This node configures the Digital Twin (DT) by sending both the JSON representation of the controller and a list of experiments to be run. This PoC presents a simple case where only two experiments implemented as Python functions are available in the DT:

- ***Average*:** Return the average of the controller output after 100000 iterations, being the input drawn from a uniform distribution between 1 and 10.
- ***Value*:** Given a fixed input, return the absolute value of the difference between the controller output and a desired value. By default, the input is 5 and the desired output is 35.

The DT insatiate a Controller object using the JSON representation and, after running the experiments, sends back to the Experimentation Manager a JSON containing the results or Key Performance Indicators (KPIs, Figure 37). The Experimentation Manager composes an Experiment Report (Exp_Rep) that is a JSON containing the id of the controller under testing and its results in each experiment. The Experiment Report is sent to the Marketplace and then forwarded to the Curation Controller (Cur_Ctr). An example of an Experiment Report (running the experiments with the default settings) is shown in Figure 44. The Experiment Report is also forward to the Evolution Controller, since the results are useful feedback if an evolutionary algorithm is implemented.

JSON Raw Data Headers
Save Copy Collapse All Expand All

results:
    average: 35.081
    value: 5
type: "exp_rep"
id: 34

Figure 44: JSON representation of the Experiment Report of the controller described in Figure 10, extracted from the Marketplace.

Based on the Experiment Report, the Curation Controller (Cur_Ctr) will decide if the Evolvable Controller referenced by its id is promoted to Protected Controller (Ptr_Ctr) for one or more experiments. Treating the protected category independently for each experiment is based on the idea that a controller suitable for a specific use case may perform poorly if the case changes. For example, Figure 45 shows the notification of the Curation Controller to the Marketplace informing that the controller with id 34 (Figure 43) is a Protected Controller for the experiment/use case *value*. Note that the same controller has not the category of protected for the experiment *average*. This decision depends on the criteria followed by the Curation Controller, that for this example is:

- ***Average*:** Protected Controller if the result is greater than 500.
- ***Value:*** Protected Controller if the result is smaller than 20.

Therefore, these values serve as configurable thresholds to decide if an evolvable controller may be promoted to a protected controller.

JSON Raw Data Headers
Save Copy Collapse All Expand All

type: "ptr_ctr"
id: 34
exp: "value"

Figure 45: JSON file registration of Protected Controller, extracted from the Marketplace.

9.3.5 FUNCTIONAL IMPLEMENTATION

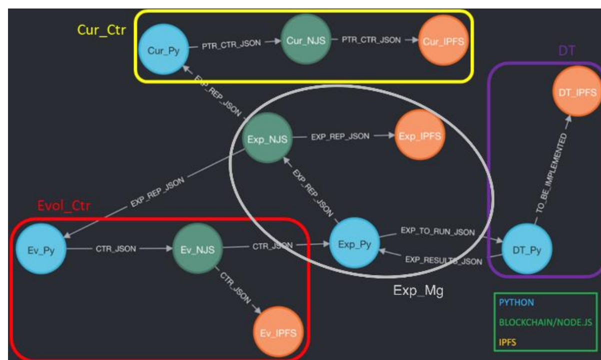


Figure 46: Functional level architecture of the PoC.

Figure 46 shows the functional level architecture of the PoC, where each node represents a specific technology implemented. As can be seen from the picture, each one of the actors shown in Figure 37 is composed by a subset of services that represent a specific function within the PoC. Different technologies are used depending on the type of functionality implemented by each service. A description of each type of service and the technology it uses is provided below:

- **Experimentation and Evolution**: responsible for the evolution and experimentation logic of each actor. As described in previous sections of this report, different actors have different functionalities, so implementation varies from actor to actor (the Cur_Ctr promotes controllers based on experimentation reports, the Evol_Ctr creates new controllers and sends them to the Exp_Mg etc.).
  - Programming language/Frameworks: Python.
  - Naming convention: *actor name abbreviation + _Py* (e.g., *Cur_Py* for Curation Controller)
- **Marketplace:** services that make up the functionalities of the Marketplace.
  - Ethereum service: responsible for receiving and executing transactions through smart contracts in Ethereum. These transactions include uploading a file to IPFS, retrieving an uploaded file by its CID, or by its id.
    - Programming languages/Frameworks: Node.js, Hardhat, Solidity.
    - Naming convention: *actor name abbreviation + _NJS* (e.g., *Cur_NJS* for Curation Controller)
  - IPFS service: node connecting each actor to a private IPFS distributed filesystem network.
    - Naming convention: *actor name abbreviation + _IPFS* (e.g., *Cur_IPFS* for Curation Controller)

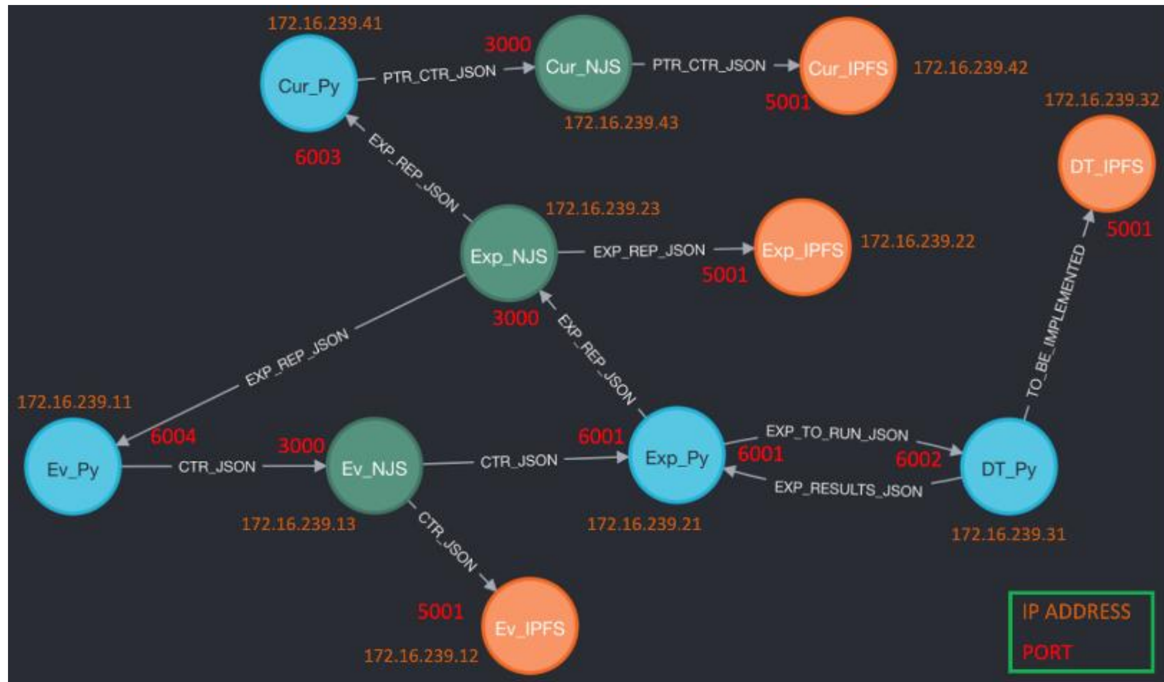


Figure 47: Network architecture of the PoC.

All independent services have been implemented through Docker containers, orchestrated and configured via a Docker Compose *(docker-compose.yml)* file. Each container is interconnected creating a network that is shown in Figure 47. From it, it can be seen that each container/service of the system has an IP address and a specific set of ports exposed for communication with other nodes within the network. The distribution of these ports and addresses is detailed in Table 2.

Node	Container name	IP address	Ports	Function
Evol_Ctr	evol-ctr-py	172.16.239.11	6004	Builds controllers combining modules Sends controllers for experimentation
	evol-ctr-ipfs	172.16.239.12	4001 8090:8080 5001	Libp2p swarm connection HTTP gateway and read-only API IPFS API
	evol-ctr-mkp	172.16.239.13	8545 3000	JSON RPC connection to Ethereum Marketplace HTTP API
Exp_Mg	exp-mg-py	172.16.239.21	6001	Receives controllers Passes controllers and experiment list and parameters to the Digital Twin
	exp-mg-ipfs	172.16.239.22	4001 8091:8080 5001	Libp2p swarm connection HTTP gateway and read-only API IPFS API
	exp-mg-mkp	172.16.239.23	8545 3000	JSON RPC connection to Ethereum Marketplace HTTP API
DT	dt-py	172.16.239.31	6002	Runs experiments
	dt-ipfs	172.16.239.32	4001 8092:8080 5001	Libp2p swarm connection HTTP gateway and read-only API IPFS API
Cur_Ctr	cur-ctr-py	172.16.239.41	6003	Decides if a controller is protected
	cur-ctr-ipfs	172.16.239.42	4001 8093:8080 5001	Libp2p swarm connection HTTP gateway and read-only API IPFS API
	cur-ctr-mkp	172.16.239.43	8545 3000	JSON RPC connection to Ethereum Marketplace HTTP API

Table 2: Containers and Ports

9.3.6 THE MARKETPLACE

As described in Section 9.3.2 of this report, rather than a single node of the network that makes up the PoC, the designed Marketplace is in itself a distributed network of several interconnected nodes. As shown in the previous sub-section, these nodes are composed of Docker containers incorporating two different services (Ethereum and IPFS) that make up the Marketplace and are included into every actor that is part of the PoC and shown in Figure 37.

This means that the Marketplace is distributed across all the instantiations of the Evolution Controller, Experimentation Manager, Digital Twin and Curation Controller, connecting every actor to the traceable ledger of evolution artifacts created throughout the process and stored in IPFS and accessible through the private Ethereum blockchain. Figure 48 presents an example of how the Marketplace works when uploading an artifact. The process is described below:

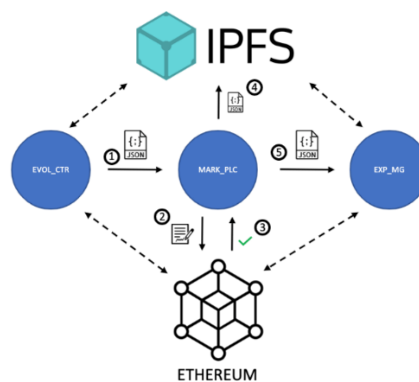


Figure 48: Example of uploading process in the Marketplace.

1. The Evolution Controller sends an Evolvable Controller in the form of a JSON file to the Marketplace. This JSON file has the structure described in Section 9.3.3, containing information that identifies the controller according to its definition, and is sent via a POST HTTP request to port 3000 of the *evol-ctr-mkp* container, which handles the HTTP API of the Marketplace (see Table 2).

2. The Marketplace receive the JSON artifact via port 3000 and registers the file into Ethereum via the *Marketplace.sol* SC (see sub-section 9.3.3 for further details). The operation is run by the *marketplace.mjs* script, that is contained in every *_NJS* container and handles the operation of the Marketplace, interconnecting Ethereum with IPFS, and handling transactions on behalf of each node.
3. Ethereum validates the transaction, confirming in turn the registration of the JSON artifact into the chain.
4. Once the transaction is confirmed, the *marketplace.mjs* script uploads the JSON file to IPFS via the IPFS API (port 5001).
5. After uploading to IPFS, the JSON artifact is sent to the next corresponding actor/actors (in this case, the Experimentation Manager).

9.3.7 THE MARKETPLACE SMART CONTRACT

Just as has been stated in earlier sections of this report, the Marketplace SC is responsible of registering artifacts generated throughout the evolution and experimentation process into the blockchain, ensuring traceability, security, accessibility and scalability. While the private IPFS network stores artifacts and provides scalability while ensuring availability, the Ethereum

blockchain keeps track of all transactions throughout the process and constitutes a historic record of file versions, upload times and other metadata.

In order to achieve this successfully, file registration is managed by *Marketplace.sol,* a SC that has been laid out to capture meaningful information in order to optimally ease identification of files and browsing the register. To this end, a structure named *File* has been created in the *Marketplace.sol* SC, whose elements are listed below:

- File number *(uint256)*: number of the file uploaded to the Marketplace chronologically.
- CID *(string)*: the file's Content ID, computed before being uploaded to IPFS.
- Name *(string)*: name of the file.
- Object Type (*string*): type of the object uploaded: controller, experimentation report etc.
- Uploader *(address payable)*: Ethereum address of the node that is registering the file.

A mapping is then assigned to the File structure and the total number of files uploaded to the Marketplace, which is used to keep track of files in the system. This number is in turn saved in the chain as the File number attribute.

It is worth noting that, while the *id* of a controller as described in section 9.3.3 represents a specific instance of a controller, the CID is a unique cryptographic hash that represents a file and does not change as long as the contents of the JSON stay the same, i.e., when the evolution is run multiple times.

9.3.8 CONTROLLER DEPLOYMENT USING TOSCA

This PoC also includes the automatic deployment of a controller as a REST service. The code for this part can be found in the folder *ctr_deployment* [7]. For this purpose, a YAML file is written according to the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) [10], as it is recommended in the Build-a-Thon proposal [11]. The TOSCA file is then parsed and executed by using the orchestrator xOpera [12]. Given the CID of a controller as input, the following task are performed by the orchestrator (Figure 49):

- Fetch the JSON representation of the selected controller from the Marketplace.
- Create an instance of the class Controller based on the JSON representation.
- Deploy a REST service in a Docker container.

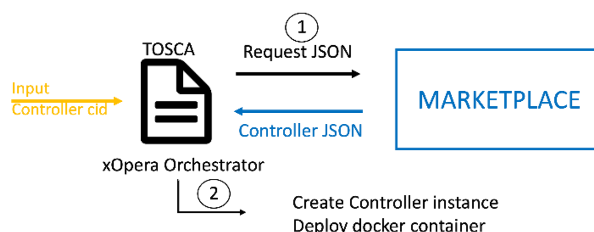


Figure 49: Representation of controler deployment.

For the development of the REST service, we have used the Python tools Flask and Gunicorn (Section 9.3.4). Once the service is deployed it will reply to HTTP GET requests (Figure 50). The HTTP response is a JSON that contains the information about the controller and the output based on the request's input (Figure 51).

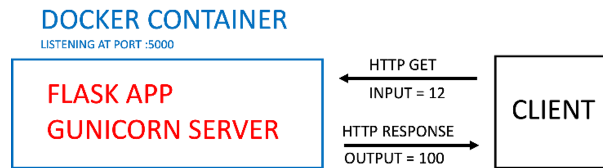


Figure 50: Controller REST service representation.

JSON Raw Data Headers

Save Copy Collapse All Expand All


```
controller:
  id:                 34
  modules:
    0:                "sub"
    1:                "mul"
  parameters:
    0:                2
    1:                10
  representation:     "((x-2)*10)"
  type:               "controller"
input:                12
output:               100
```


Figure 51: JSON response by the controller REST service.

The controller implemented is rather simple, and the REST service is not actually a close-loop (we already implemented a close-loop for this contribution in the evolution process). However, the purpose of this part is to prove that is feasible to deploy a controller with just its CID, getting its representation from the Marketplace. The CID of the controller with id 34 is the default input in the file *inputs.yaml* inside the *ctr_deployment* folder [7], although it can be replaced by any other controller's CID. As Section 9.3.3 explains, this CID will remain the same as long as the JSON representation of the controller is the same, no matter how many times the evolution is run. This feature ensures a "check-point" to which the service can always return.

9.3.9 PoC RESULTS

This section presents the results of the PoC which code is included in [7]. Along with this report, the BT submission includes a demo video of the PoC execution. The configuration parameters for the presented PoC are:

- Each controller is composed of 2 modules.
- The parameter of each module can only adopt two possible values: 2 and 10
- Evolution is done by randomly combining pairs of modules, avoiding repetition of the same modules in the same order with the same parameters (that would yield an identical controller).
- The PoC stops after trying out all possible combinations
- As for the Curation Controller criteria, the default parameters are used. As explained in Section 9.3.4 that means:
  - ***Average*:** Protected Controller if the result is greater than 500.
  - ***Value*:** Protected Controller if the result is smaller than 20.

- All docker containers run on a local machine and in a private network as explained in Section 9.3.2.

The purpose of this configuration is to illustrate the mechanics of the proposed architecture by a clear and quick demo that can be reproduced easily in a local machine by anyone who clones the repository [7]. The close-loop chronological order was explained in Section 2.1 and it is shown in Figure 37. Figure 46 shows the representation of the actual docker container nodes. As shown in the demo video, one must look at log messages of the docker containers to analyse the process.

Figure 52 shows the log messages of the Evolution Controller (its Python part), denoted as Ev_Py in Figure 46. The output shows the JSON representation of each controller that the Ev_Py node sends to Ev_NJS. The JSON is forwarded to Ev_IPFS and its response, the CID assigned to each controller, is sent back to Ev_Py by the Ev_NJS. With the printed CID it is possible to access the controller JSON representation in the marketplace, as shown in Figure 53 for the controller with id 34. It is interesting to change the port in the address, to access a different node (Figure 54). The port 8091 corresponds to the Exp_IPFS node and the 8092 to the DT_IPFS (see Figure 47 and Table 2) This proves that the information about each controller is reproduced in all the IPFS nodes.


```
evol-ctr-py  | {'file': '{"type": "controller", "id": 32, "modules": ["sum", "pow"], "parameters": [10, 10], "representation": "((x+10)^10)"}'}
evol-ctr-py  | {'cid': 'QmcRhK3o8tW7m6rQAxrargU6a8x7X9Smyxd7rDvV7xiyUR'}
evol-ctr-py  | {'file': '{"type": "controller", "id": 33, "modules": ["sub", "mul"], "parameters": [2, 2], "representation": "((x-2)*2)"}'}
evol-ctr-py  | {'cid': 'QmY9KpCHhefRCCBQowpCqZoDacRBycyMQBLgrisS34FU8V'}
evol-ctr-py  | {'file': '{"type": "controller", "id": 34, "modules": ["sub", "mul"], "parameters": [2, 10], "representation": "((x-2)*10)"}'}
evol-ctr-py  | {'cid': 'QmP5ffWUvKWUDpoMCSvdw6rwfy3r2CoZAgDJJtiY8h84B4'}
evol-ctr-py  | {'file': '{"type": "controller", "id": 35, "modules": ["sub", "mul"], "parameters": [10, 2], "representation": "((x-10)*2)"}'}
evol-ctr-py  | {'cid': 'QmWW2qBK9TsRskivWXKryG6caFq7fxS99trB5RJRMKB5Dx'}
evol-ctr-py  | {'file': '{"type": "controller", "id": 36, "modules": ["sub", "mul"], "parameters": [10, 10], "representation": "((x-10)*10)"}'}
```


Figure 52: Log messages of the Ev_Py docker container.

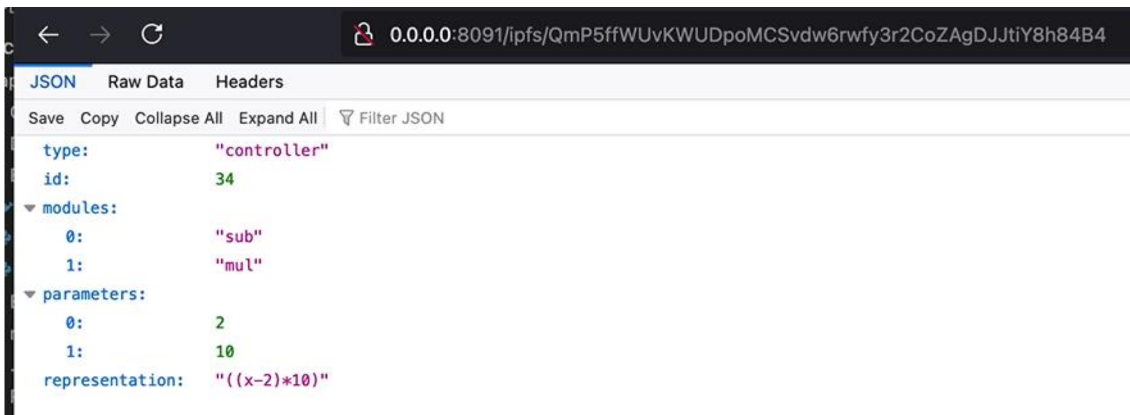


Figure 53: Access to the JSON representation of a controller in the Marketplace via its CID. The Marketplace node that is being consulted is Exp_IPFS.

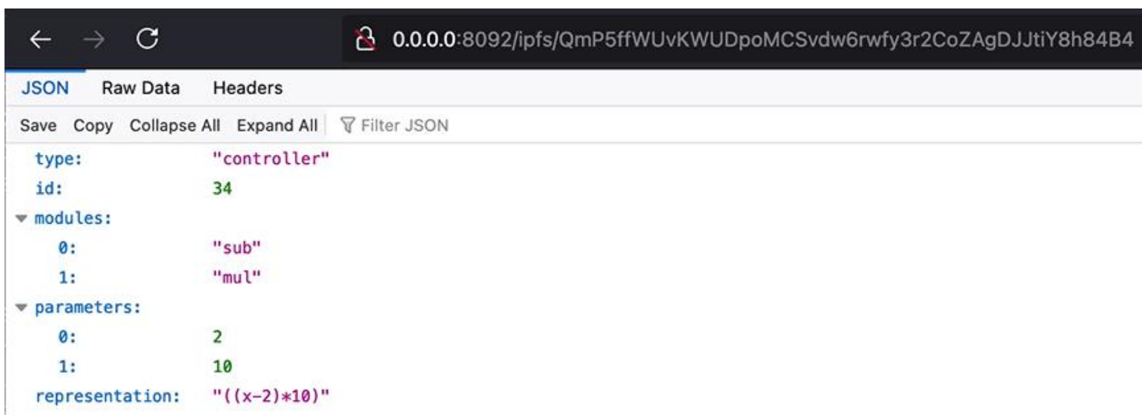


Figure 54: Access to a controller JSON representation in the Marketplace via its CID. The Marketplace node that is being consulted is DT_IPFS.

9.4 Autonomous agents (with varied competence) in networks.

This section presents the PoC on autonomous agent in network case study. The case study is focused on the level of autonomy for autonomous system with estimation and judgement of competence as key criteria.

The autonomous system requirements are presented [4]. Among the requirement, the autonomous agent should determine and adjust the level of autonomy depending on environment and type of task in contrast to capabilities of the system in combination with policies to support. The capability will enable system to dynamically adjust to situation corresponding human intervention requirement at run-time or design. The main highlight of an autonomous workflow include task specification, understanding, feasibility, planning and execution. Request for human support and subsequent learning by the autonomous system are other important considerations.

A simple workflow scheme for autonomy with varying autonomy levels was discussed. This includincludeded task specification by human and task understanding, feasibility check, task planning, task execution by system. Request for support to human and control by human can be to any of these workflow steps. Monitoring of performance levels by humans and learning by the system are added steps.

9.4.1 The Poc design and implementation

For the PoC, we consider sensors being an integral part of literally all equipment, gadgets, mobility solutions, communication devices, medical devices, defence applications, athletics, electronics, computers, etc. Sensors are among the fastest growing markets along with mobile phones and computers. So, the requirements for use case [4] involves the type of data that is collected and detected from the environment.

The data detected from the network in particular includes include text data (logs), packet measurements, deep packet inspection, latency time measurements, packet loss measurements, etc. The sensors are thus autonomous agents because they are created (instance) and managed (data collection, data analysis and actions) with minimal human intervention.

Also, as part of the requirements for use case 23, it is essential that autonomous networks (AN) enable fault isolation through collaborative analysis of multiple sensors to identify the cause of the fault. Another aspect is enabling bug fixes without human interaction, such as discovering the correct version of software or hardware to update.

The creation of new agents with new capabilities, requirement collection and analysis of existing problems (and in some cases, recurring) in the network (RAN or base station failed) warrants the autonomous management of close loops. In this this PoC , we consider the validation of the actors and the relationship between them in the design of a graph in Neo4j. other important tools use including Graph DB, Colab, Python programming, Regular meetings and Interactions

Autonomous controllers and data are considered as the main actors. The controllers consist of data stream sensors or data delivering technology while cloud-based (Environment) methods are employed to distribute communication loads through relay nodes. Table 3 presents the actors and their relationship.

Actor-1	Actor-2	Description
Controllers	Autonomous	Sensors for data stream collection (or delivery) technologies.
Data	Environment Data	Cloud-based (Environment) methods to distribute communication loads by relay nodes.

Table 3: Different actors and their relationships

9.5 Deriving new use cases based on link prediction algorithm.

This section presents PoC on deriving new use cases based on link prediction algorithm. It's important to investigate the hidden use cases for the autonomous networks to be able to extract how to use the AN concepts in future networks effectively, and an effective way to do so is to make use of the recent use cases, to predict the possible ones.

Instead of analysing the existent use cases manually we can benefit from the huge advancements in the computational intelligence A.K.A. AI or ML, but for that purpose we need a dataset that contains information about the recent use cases.

Firstly we analyse and automate the process of getting information (text and graphs) as describe in [ITU-T Y.Supp 71]. Through the data presentation, the network structure and entities are captured and a model is proposed to predict the links. The rest of the section discusses the Text and document parsing, figure parsing process, graph database representation, and the link prediction algorithms.

9.5.1 Text And Document Parsing:

In [1], We can find the use cases for the build-a-thon challenge, and in order to make predictions for the new use cases we need first to get the existing details of the use cases to be able to represent them.

In that document, we can find tables, texts and images that are used to represent the use cases, in this section we will describe how we parsed the paragraphs and tables, and in the next section we will describe the parsing of images.

By using the docx library[8], which is a programming library based on Python programming language to process word documents, we can extract all the information we need from that document.

First we need to extract information that is found in the tables, because it represents the meta data and also the description of the use case itself, but the first problem we faced was we have a lot of tables other than use cases' one like the following tables, so we need to select those tables only by some sort of conditioning.

after Parsing those tables, we found that the fields on the tables is not consistent in naming, for example we have some problems like:

- [Category, Use case category, Notes on use case category] they all refer to the category of the use cases.
- [Base contribution, Base Contribution, Base contributions] All refers to the same information.
- [Use case description, Description, Use case description\n] All refers to the same information.
- [Open issues, Open issues (as seen by the proponent)] All refer to the same information.

So we need to make sure we have a consistent naming, we edited the confusion manually, and finally we got the following table that summarises all 40 cases:

	Use case id	Use case name	Base contribution	Creation date	Use case context	Use case description	Open issues	Use case category	Reference	Notes on priority of the use case
0	FG-AN-usecase-001	Use of knowledge in autonomous network	[FGAN-I-12-R1]	21/January/2021	Discussions during ITU webinar on autonomous n...	To satisfy the key concepts of autonomous netw...	- Representation mechanisms and transfer proto...	Cat 1: describes a scenario related to core au...	[b-Clark], [b-AN2020], [b-Jimenez-Ruiz], [b-My...	NaN
1	FG-AN-usecase-002	Configuring and driving simulators from autono...	[FGAN-I-12-R1]	21/January/2021	Discussions during ITU workshop on autonomous ...	To explore and experiment with various scenari...	Are simulators encapsulated in Sandbox? Or are...	Cat 1: describes a scenario related to core au...	[b-Y.ML-IMT2020-SANDBOX]	NaN
2	FG-AN-usecase-003	Peer-in-loop (including humans)	[FGAN-I-12-R1]	21/January/2021	Discussions during ITU workshop on autonomous ...	To guide the autonomous behaviour autonomous n...	Are some peers more equal than others (e.g. hu...	Cat 1: describes a scenario related to core au...	[b-Y.ML-IMT2020-MLFO]	NaN
3	FG-AN-usecase-004	Configuring and driving automation loops from ...	[FGAN-I-12-R1]	21/January/2021	Inspired by discussions on "demand mapping" du...	There are different automation loops in variou...	Where are the automation loops hosted? Are the...	Cat 1: describes a scenario related to core au...	[ITU-T Y.3173]	NaN
4	FG-AN-usecase-005	Domain analytics services for E2E service mana...	[FGAN-I-12-R1]	21/January/2021	Based on discussions with ETSI ZSM via [ML5G-I...	Section 6.5.3.2 of [ETSI ZSM ARCH] describes t...	Open issues (refer also those discussed in [ML...	Cat 2: describes a scenario related to applica...	[b-ETSI GS ZSM 002]	NaN
5	FG-AN-usecase-006	Automation and intelligent OAM(operation, main...	[FGAN-I-008]	NaN	NaN	Background: Dynamic radio environment, network...	NaN	Category 1 - Use case for autonomous behaviour	None	NaN

Table 4: Final parsed data from the tables

Next step is to parse the requirements of each use case, the requirements in the document is represented as paragraphs after the table of each use case, and as an example, the first requirement in the 10th use case is :

For this example we can find that we can extract everything about the requirement like the use case number and the number of the requirement itself, also the priority level of the requirement can be extracted, and we have 3 levels of priority will be as follows:

- Critical: Important and required for the use cases implementation.[1]
- Expected: Possible requirement which would be important but not absolutely necessary to be fulfilled.[1]
- Added Value: possible requirement which would be optional to be fulfilled (e.g., by an implementation), without implying any sense of importance regarding its fulfilment.[1]

Finally We managed to get the final table that represents all the requirements as follows:

	id	001	002	003	004	005	006	007	008	009	010
0	UC001	It is critical that AN enable exchange of kno...	It is critical that AN enable optimization of...	It is critical that AN enable creation of rep...	It is critical that AN enable exchange of kno...	It is critical that AN use knowledge base for...	It is expected that AN enable exchange of kno...	it is of added value that AN use Auto-control...	NaN	NaN	NaN
1	UC002	It is critical that AN components arrive at p...	It is critical that autonomous networks (AN) ...	It is critical that autonomous networks (AN) ...	It is critical that autonomous networks (AN) ...	NaN	NaN	NaN	NaN	NaN	NaN
2	UC003	It is critical that autonomous networks (AN) ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	UC004	It is critical that autonomous networks (AN) ...	It is critical that autonomous networks (AN) ...	It is critical that closed loops monitor the ...	It is critical that AN components consider th...	NaN	NaN	NaN	NaN	NaN	NaN
4	UC005	It is critical that AN support discovery and ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	UC006	It is critical that autonomous networks enabl...	It is critical that AN enables data quality a...	It is critical that AN enables capturing and ...	It is expected that AN uses AI and big data t...	It is of added value that a varying set of KP...	It is of added value that AN solutions may be...	NaN	NaN	NaN	NaN
6	UC007	It is critical that autonomous networks (AN) ...	It is expected that autonomous networks (AN) ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table 5: Final requirements table

Now we have now two tables, one for requirements and the other one for the information about the use cases, now we need to combine those tables together to have all details in one place, and the final details will be as follows:

Open issues	Use case category	Reference	Notes on priority of the use case	...	006	006_importance	007	007_importance	008	008_importance
- Representation mechanisms and transfer proto...	Cat 1: describes a scenario related to core au...	[b-Clark], [b-AN2020], [b-Jimenez-Ruiz], [b-My...	NaN	...	It is expected that AN enable exchange of kno...	expected	it is of added value that AN use Auto-control...	added value	NaN	NaN
Are simulators encapsulated in Sandbox? Or are...	Cat 1: describes a scenario related to core au...	[b-Y.ML-IMT2020-SANDBOX]	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
Are some peers more equal than others (e.g. hu...	Cat 1: describes a scenario related to core au...	[b-Y.ML-IMT2020-MLFO]	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
Where are the automation loops hosted? Are the...	Cat 1: describes a scenario related to core au...	[ITU-T Y.3173]	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
Open issues (refer also those discussed in [ML...	Cat 2: describes a scenario related to applica...	[b-ETSI GS ZSM 002]	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
	Category 1 -				It is of					

Table 6: Final combined tables of the requirements

Now we have all the details about each use case in a row, next step we need to parse the figures that represent the relations between actors.

9.5.2 Figures representation:

In the document, we saw two types of figures, one for possible components (Figure 52) and the other in a form of sequence diagram (Figure 53), both representing the interactions between the actors.

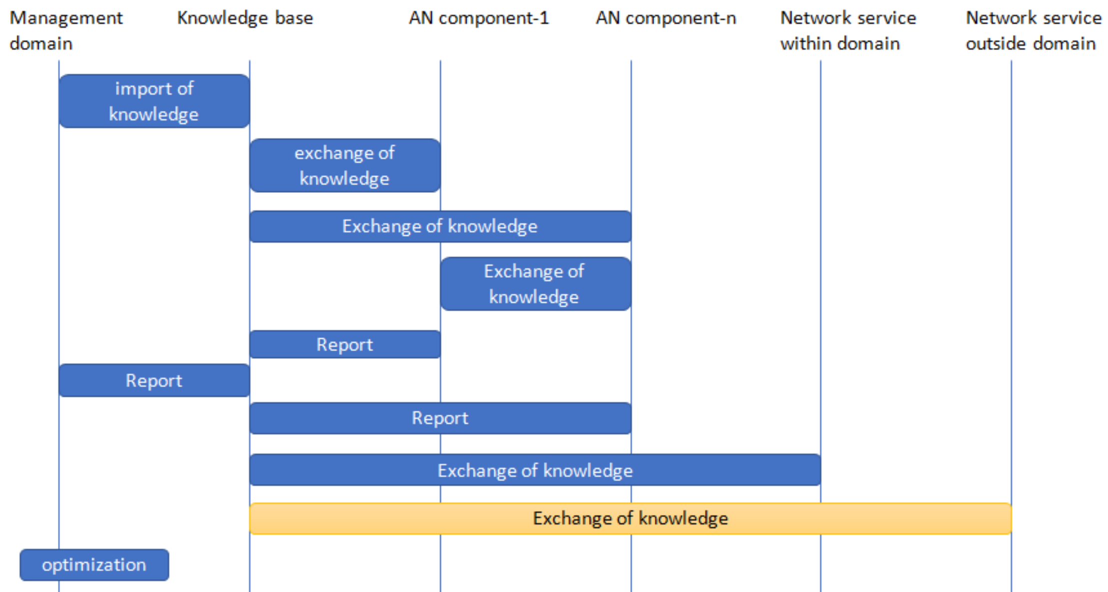


Fig. 55: Example Actor Interaction diagram

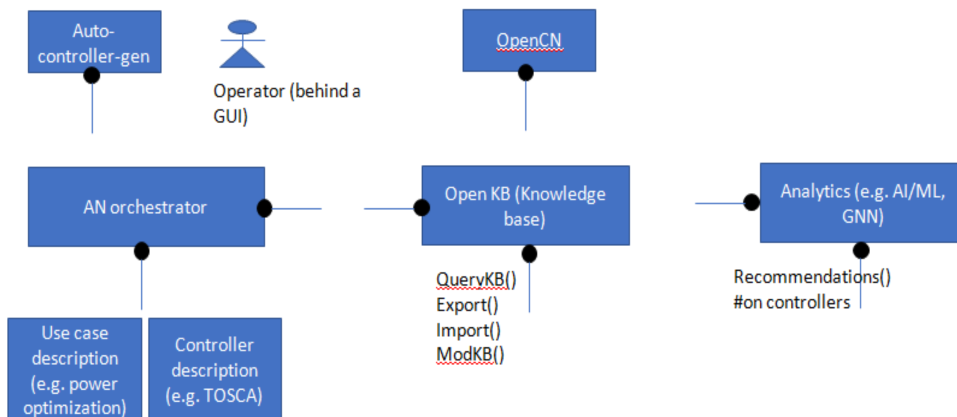


Fig. 56: Possible components diagram

The problem with images is that they are not parsable, we cannot extract the desired information from these images in an automated fashion, so to be able to extract the information, we will convert it to a machine readable format and then extract the desired information.

To do that job we tried several tools to make a convenient representation, first tool is PlanUML[6], and it's a tool to make several types of diagrams using the UML format, by using this tool we are able to create sequence diagrams, and the result our first try was as in the next figure:

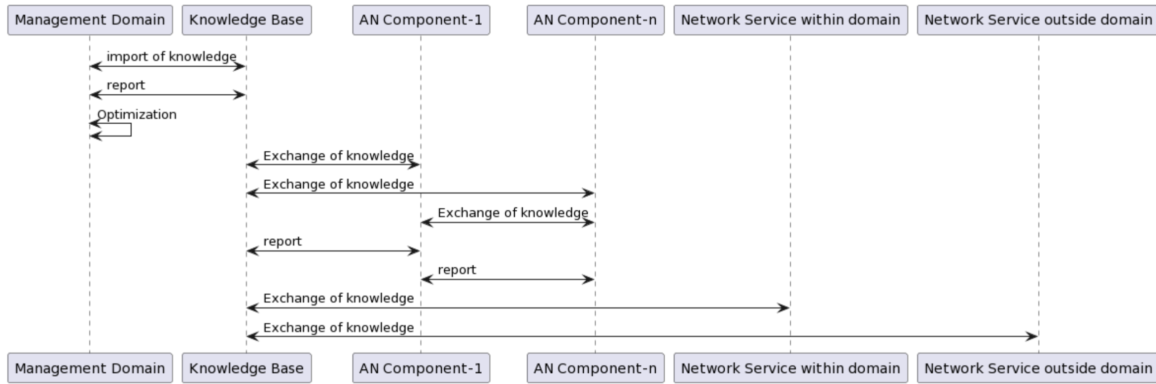


Fig. 57: First try of using PlantUML

By comparing this figure 54and figure 52 we can see that this representation is most like the desired one, but has one but crucial disadvantage, which is the optimization function of the management domain here is represented as a self message, but in the real time it's a function inside the management domain component.

Then we tried another representation, in this time we tried another type available in PlantUML, and the result was as follows:

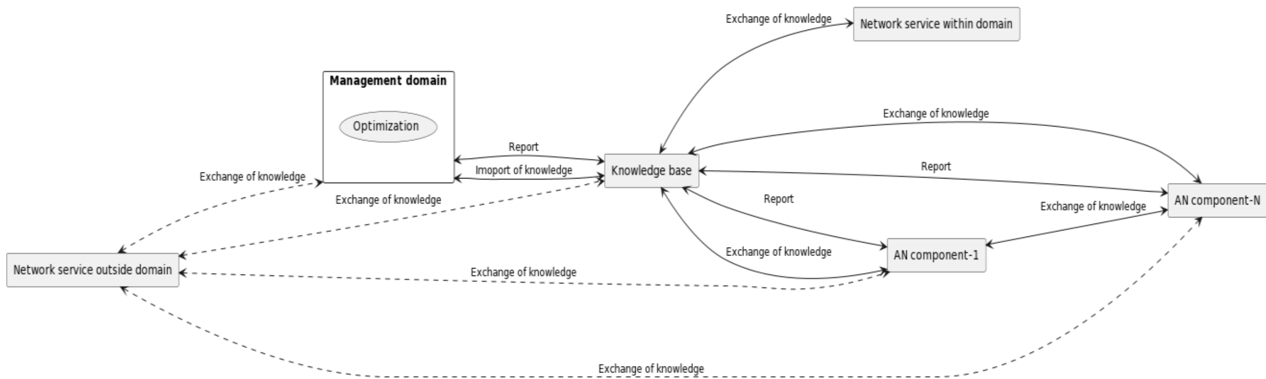


Fig. 58: Second try result of figure representation using PlantUML

This representation is sufficient if we don't want to see a visual representation, here we overcame the problem of representing the function, but we have a bad visual that is not like the one we want to be consistent.

The final try we used another tool to make the representation, this time we used Draw.io[9], which is a GUI based tool, from which we can extract the XML or any other machine readable format, and the result was as follows:

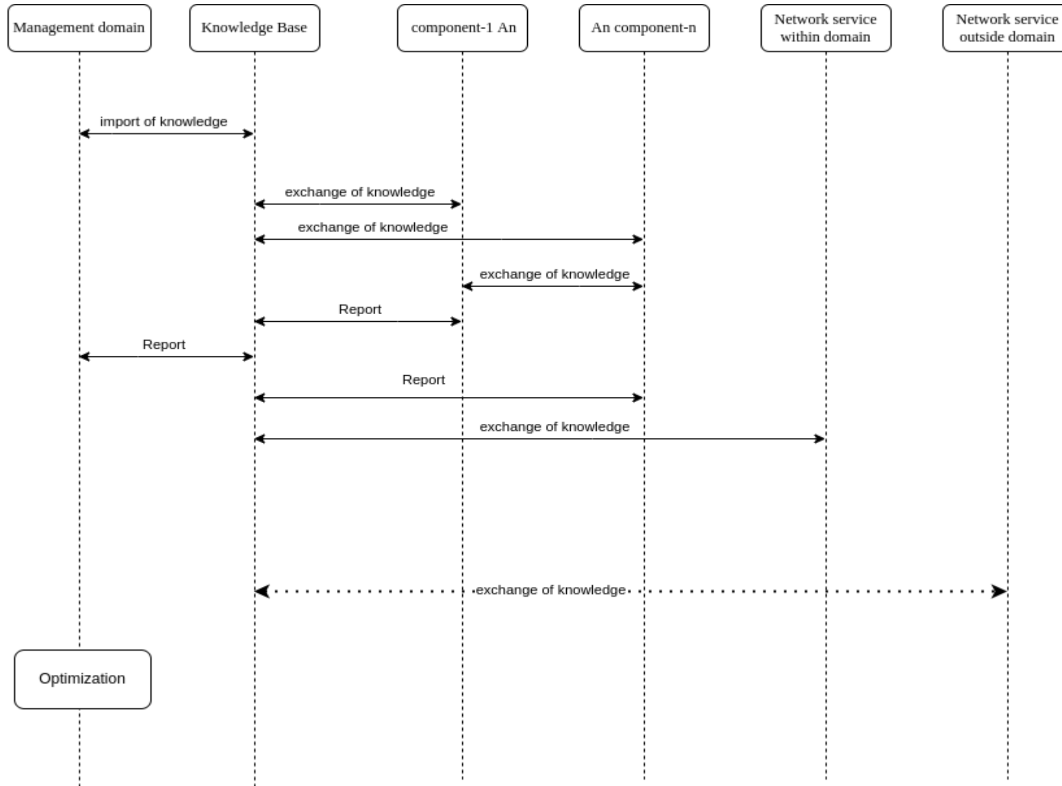


Fig. 59: Final try in figure representation

As we can see here an identical copy of the diagram in figure 52, also there are other types of shapes that can represent any other type of diagrams, also it gives us the flexibility to enter any other metadata that can help us when we parse the figures, and we used these metadata as follows:

ID: 3nuBFxr9cyL0pnOWT2aG-1
Value: Knowledge Base
class: Component
Enter Property Name
Add Property

(a) Component metadata

ID: 3nuBFxr9cyL0pnOWT2aG-5
class: function
value: optimization
parent: 3nuBFxr9cyL0pnOWT2aG-1
Enter Property Name
Add Property

(b) Functions Metadata

ID: 5fA4P5xMXdoz5tjVhF9F-12
class: critical relationship
value: Report
Enter Property Name
Add Property

(c) Relations Metadata

Fig. 60: Added Metadata

We added the following meta data:

- Class: To represent the class of the item, whether it was component, function or the type of the relation, this field is added to all items in the figure.
- Value: The name of the item or the relation to all items.
- Parent: To contain the parent component ID that contains this function, only included in the function items.

After successfully representing the figure in the XML, which is a machine readable format, we can use Python[10] and its library XML[8] to parse the figure, and we ended up with the following table:

	source	target	message	class	id
0	Knowledge base	AN component-N	exchange of knowledge	critical relationship	UC001
1	AN Component-1	AN component-N	exchange of knowledge	critical relationship	UC001
2	Knowledge base	AN Component-1	Report	critical relationship	UC001
3	Knowledge base	AN Component-1	exchange of knowledge	critical relationship	UC001
4	Management domain	Knowledge base	import of knowledge	critical relationship	UC001
...	...	...	...	...	...
125	AN_1	AN_3	negative acknowledgement for the unicast reply	critical relationship	UC040
126	AN_1	AN_2	Configuration and context setup request requir...	critical relationship	UC040
127	AN_2	AN_1	context setup response	critical relationship	UC040
128	AN_1	AN_2	AN_1 configures the network underlays for rout...	critical relationship	UC040
129	AN_1	AN_2	AN_1 pays AN_2 for the service	critical relationship	UC040

Table 7: Relations Table

And also we have a dictionary that contains the functions and its corresponding parent actors.

Now we have completed parsing the document and the following figure represents what we did till now:

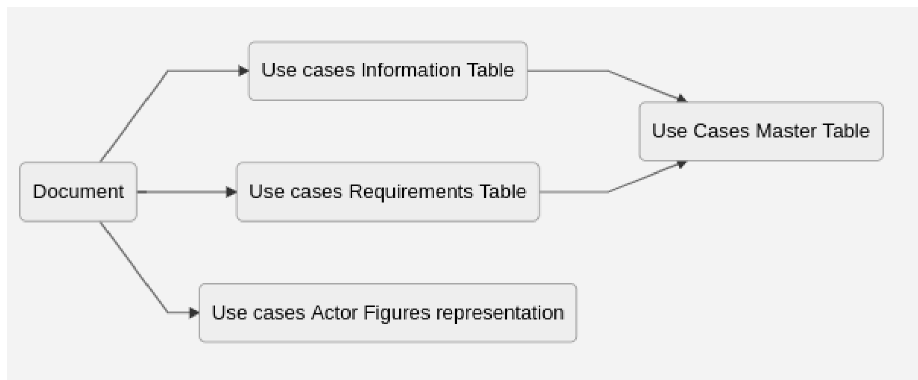


Fig. 61: Resources we have

After that, we need a way to convert this into a type of dataset to be able to run machine learning algorithms on it.

9.5.3 Graph database representation:

There are several graph databases, we chose Neo4j[5] which is one of the most popular graph databases. Also it has some advantages like native representation of graphs, a lot of other packages especially in data science, has its Querying language Cypher which help us in query graphs and Finally Has an API with Python so it can help us automating our stuff.

It's more convenient to represent the network with a graph database, in the graph database we represent the data in the form of nodes and relations, which is the most relevant format for network structure.

We made some iterations to represent, the first try based only in the relations table, which contains only the components in the relations table with 130 relations only, but this representation has some drawbacks which is special to those use cases that have figures to represent it on the document, which are not all use cases, so using only these relations will ignore important parts of the use cases.

Next try is to read each use case carefully and extract the true relations and functions inside the graphs and also the hidden relations in the requirements and descriptions of each use case, and the final product is the reference code, which is sufficient to represent the use cases, now we have +500 relations and +300 Component.

A part of our representation is in the following figure:

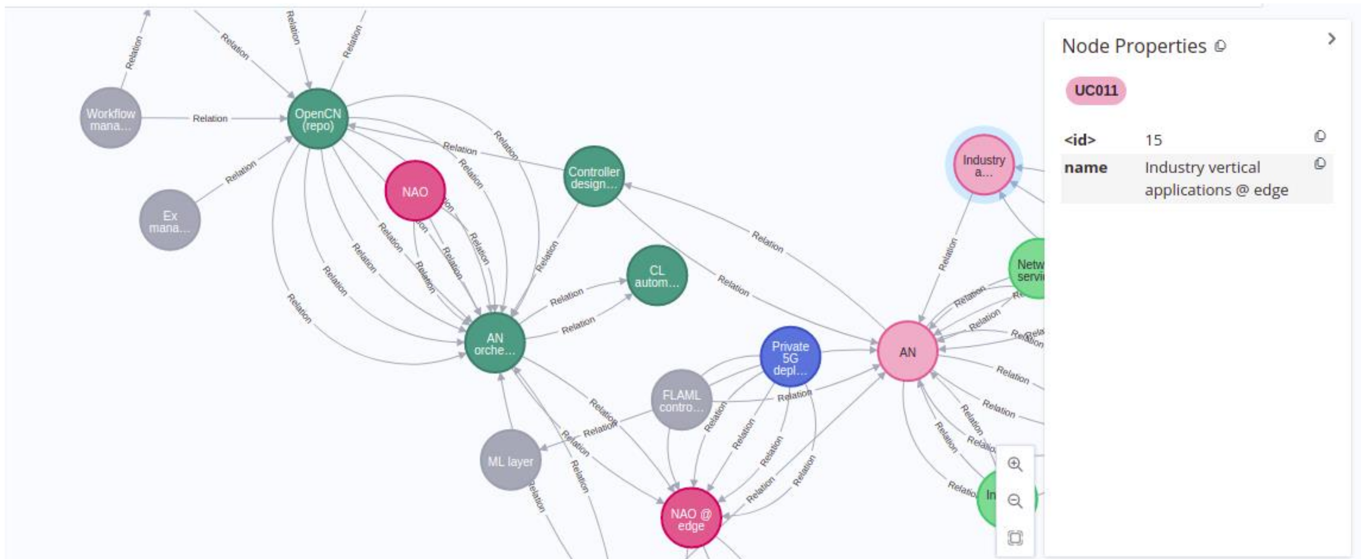


Fig. 62: Part of the graph representation

Now we have represented the data, next step is to run the link prediction algorithm on it to be able to investigate new use cases.

9.5.4 Link Prediction Algorithm:

We can consider the link prediction pipeline as a classification problem to classify if a certain relation exists between two nodes or not, and this classification can be more accurate if we calculate something that measures the how likely these two nodes to have a relation in between, and actually the current representation needs some numeric properties.

And the pipeline of the link predictions contains several steps as follows:

9.5.4.1. Make the projection:

First step is to project the graph database in the memory, this is done by using the native projections in neo4j.

9.5.4.2. Add node properties:

We need to add some properties to the nodes, these nodes are:

9.5.4.2.1. Node Embeddings:

For each node we need specific embedding like what is happening when we decode a categorical feature in the ordinary data preprocessing.

Node embedding algorithms compute low-dimensional vector representations of nodes in a graph. These vectors, also called embeddings, can be used for machine learning.

We have used the FastRP algorithm because it's the only one in production quality, and works in directed and undirected relationships.

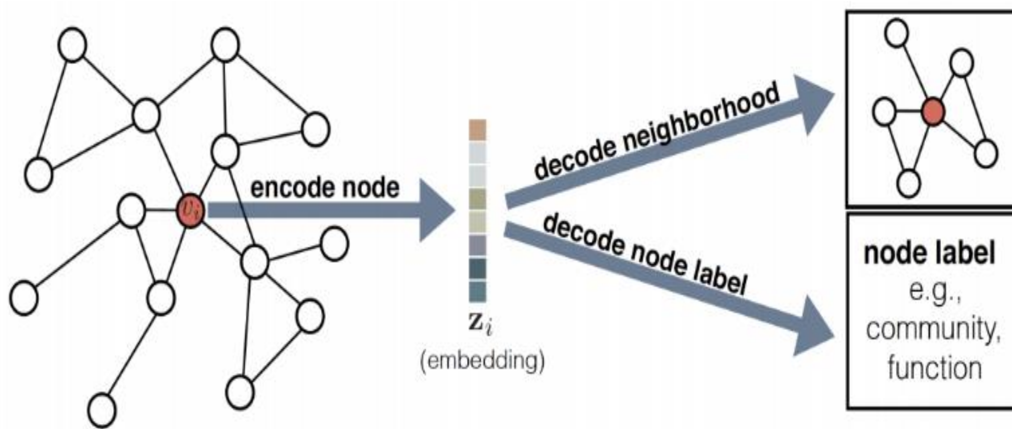


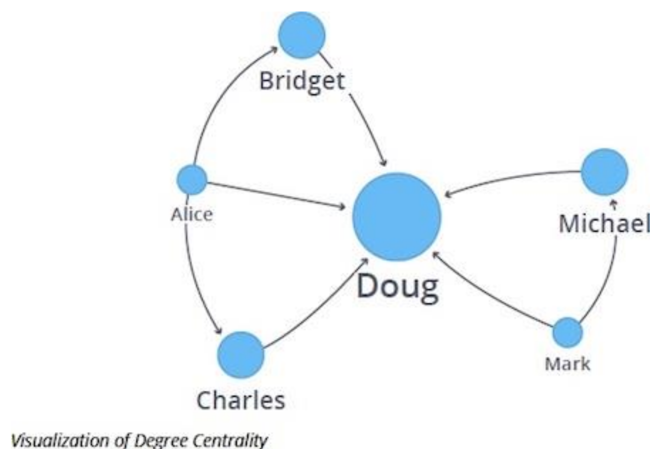
Fig. 63: Node embedding

9.5.4.2.2. Centrality:

Centrality algorithms are used to determine the importance of distinct nodes in a network.

We have used the Degree Centrality algorithm, as it works well in directed and undirected relationships.

degree centrality measures the number of incoming or outgoing (or both) relationships from a node, depending on the orientation of a relationship projection.



Visualization of Degree Centrality

Fig. 64: Example of Centrality

9.5.4.2.3. Community detection:

Community detection algorithms are used to evaluate how groups of nodes are clustered or partitioned, as well as their tendency to strengthen or break apart.

We have used the Label propagation algorithm, as it works well for directed and undirected relationships.

The Label Propagation algorithm (LPA) is a fast algorithm for finding communities in a graph. It detects these communities using network structure alone as its guide, and doesn't require a pre-defined objective function or prior information about the communities.

Now we have added all the properties that may help the algorithm more and more, next step is to calculate combined features using these properties.

9.5.4.3. Combined Features:

We are using the properties that are created on nodes, and combine them in each potential link According to one of the following functions we choose for every property:

L2	$f = [(s_1 - t_1)^2, (s_2 - t_2)^2, ..., (s_d - t_d)^2]$
HADAMARD	$f = [s_1 * t_1, s_2 * t_2, ..., s_d * t_d]$
COSINE	$f = \frac{\sum_{i=1}^{d} s_i t_i}{\sqrt{\sum_{i=1}^{d} s_i^2}\sqrt{\sum_{i=1}^{d} t_i^2}}$

Fig. 65: Similarity algorithms

9.5.4.4. Training and prediction:

Now all of the pipeline is ready, we trained algorithms in the dataset and after making hyper tuning we found that the RF model behaves best in all use cases, now we need to test the models in all scenarios:

- First scenario: we use only the 130 relation data set, which is the result of using only use cases that have figures on it.

- Second scenarios: we use the full representation of the reference code, that have +500 relations and +300 Nodes.

- Third scenario: we use the second scenario but we add the functions of the actors as properties in these nodes.

And the following was the results in each try:

Representation	Best Model	Test Accuracy (AUCPR)
Scenario 1	RF	88 %
Scenario 2	RF	73.6%
Scenario 3	RF	86 %

Table 8: Results

Now it's obvious that the third scenario is the best scenario, and when we made the prediction, we found that 6954 new links with more than 99% of probability to be a true relation.

9.5.5 Discussion

We can conclude from all of our findings that we found a way to parse the document and text, also the draw.io is the best tool that has a lot of customizations that can help us in making more representative figures.

Best algorithm for link prediction is the Random Forest model with 86% AUCPR accuracy in the full reference code, and also we need to find a more enhanced way to represent the use cases in the reference code, i.e. to convert all functions to be properties.

9.6 A Low Latency Closed Loop for ROBOTIC grasping

In this PoC use case, "Slip Detection (and Force Estimation) and Object Detection" in a robotic grasping. Primarily, universal grasp action requires object picking with compensatory grip force control to avoid gross object slippage and then the secondary task for in hand manipulation is to identify the object type across different properties primarily detecting the class. A Low Latency Closed Loop System between the haptic hands / algorithms and real robotic hand (Allegro Hand) using MEC Test Bed is developed. Figure 66 presents the low latency closed loop for the robotic graping problem. The slip detection, force estimation and object detection are discuss in the following subsections.

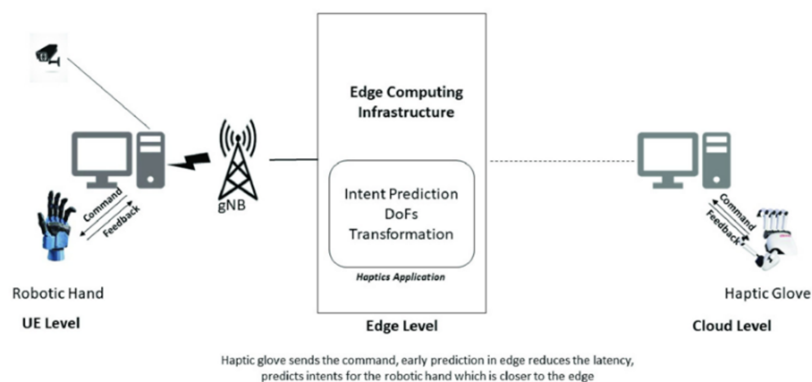


Figure 66: The low latency closed loop for the robotic graping problem

9.6.1 SLIP DETECTION, FORCE ESTIMATION AND OBJECT DETECTION IN A ROBOTIC GRASPING

Slip detection is a crucial component of robotic grasping. It is the ability of a robotic arm to sense when its grasp is slipping or losing its grip on an object. This is important for robotic arms so that they can adjust their grip and maintain a secure hold on the object. Slip detection is achieved by the use of tactile sensors or through gripper's joints force feedback, which detect changes in pressure and friction as the robotic hand interacts with the object. This data is then used to identify when the object is slipping from the grasp and the robot can adjust its grip accordingly. When the object begins to slip, the robot applies more grip force, and when it is secure, less force is applied. This helps the robot maintain its grasp of the object more effectively, allowing it to manipulate it more precisely. Slip detection in robotic grasping is a process of ensuring that the robotic arm can accurately grasp objects and keep them in place. To enable this, a low latency machine learning model control approach can be used. The machine learning model can be trained on a dataset containing objects, their physical properties, and the desired robotic grip parameters. The model can then be used to detect when the object is slipping out of the robotic grip, allowing the robot to adjust its grip parameters accordingly. This approach ensures that the robot can quickly and accurately detect the slipping objects and make the necessary changes to maintain its grip. Additionally, the low latency of the machine learning model control approach allows the robot to respond to slipping objects in a timely manner, ensuring that objects are not dropped or damaged during the robotic grasping process.

Object recognition in robotic grasping is a complex field of study that has been the focus of many research projects in the domain of robotics and computer vision. At its core, object recognition is the process of identifying a specific object within an image or video frame. This task is typically accomplished by applying specialized algorithms such as convolutional neural networks or other machine learning techniques. In recent years, object recognition has become increasingly important for applications in robotics and automation. Haptic force based object recognition is a method of robotic grasping that relies on the tactile feedback from physical contact with an object. By measuring the forces applied to the object, the robot can identify the shape and size of objects, and determine the best way to grasp and manipulate it. This technique can be used in a variety of circumstances, such as when the object is occluded from view, or when it is too small or too large to be detected by vision-based object detection. Compared to vision based object detection, haptic force based object detection offers greater accuracy and reliability, as it is not affected by lighting conditions or the presence of occluders in the environment. Additionally, haptic force based object detection is not limited to detecting flat or rigid surfaces, and can even detect the texture and properties of an object, such as its weight or material composition.

9.7 Model inference using the MEC Test Bed

This section presents the PoC use case solution design and raw data processing into logical features for learning model to predict valid outputs. For each use case a process of feature engineering and the methodology used is discussed. Figure 67 present the MEC testbed for the model inference investigation. The two cases considered are presented as follows.

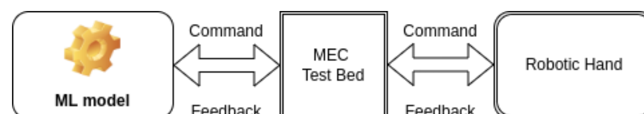


Figure 67: Model inference using MEC Test Bed .

9.7.1 Case 1 : Slip Detection

Any combination of the following individual features forms the basis of the feature set for the model to train upon. All individual features and possible combinations were fed to the learning model during experimentation.

Feature Engineering: Model Features Space-

1. Joint Force F_t
2. Joint Position,Θ_t
3. Force Derivative, ΔF_t
4. Position Derivative $\Delta\Theta_t$

The target label for slip event and crumple event are transformed into 4 possible combined classes for the model to predict the slip and crumple state at any given time step as a combined output depicted in Table 9.

Events		Resulting Events
Slip	Crumple	
No	No	0
Yes	No	1
No	Yes	2
Yes	Yes	3

Table 9: Transformed events label for multi-class prediction.

Methodology: For the defined problem statement of detecting the slip event as well as crumple event during the grasp-lift phase of object picking , the time-series readouts data from the 16 Joints of gripper is transformed into the required input format of the LSTM based model presented in Figure 68. The data is transformed by applying a sliding window to the time series data set with window size equal to the number of previous observations before the model predicts the output for next time_step. The transformed dataset is shuffled to avoid biased learning.

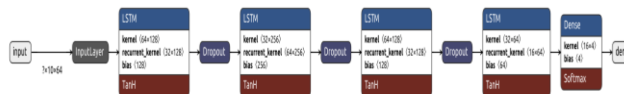


Figure 68: LSTM model for Slip detection

- Input shape : `(n_samples,n_previous_time_steps, n_features)` The build model was tried on different features combinations (`n_features` : [16 - F_t ,16 - Θ_t ,16 - ΔF_t ,16 - $\Delta\Theta_t$])
- Output_shape: `(n_samples,n_output)` The model predicts a class out of n_outputs (one-hot encoded) classes (`n_outputs` : [0,1,2,3]) corresponding to each transformed target class.

9.7.2 Case 2 : Object Recognition

For this case, the following individual features forms are used for training the model.

Feature Engineering: Model Features Space-

1. Joint Force, F_t
2. Joint Position, Θ_t
3. Mass, M_{kg}
4. Joint Force x Joint Position , F_t x Θ_t

The target labels can be segregated into unique 13 object type (class : unique shapes and sizes, properties) as well as 6 unique object categories(same shape objects).

- Classes: 13 Objects
- Class categories : 6 Objects categories.

The data-set is pre-processed further based on the unique categories to get the extracted data-set which is then fed to the classifier. Any combination of above individual features forms the basis of the feature set for the model to train upon. All individual features and possible combinations were fed to the learning model during experimentation.

Methodology: For the defined problem statement of class recognition the raw dataset was highly imbalanced. Class Imbalance : The raw dataset with unique object classes was processed to get the balanced dataset as to avoid the biased learning of the model due imbalanced samples of data of individual classes. The train dataset is generated by extracting each class dataset equivalent to the category which has the least sample in the raw dataset.

- Input data : `(n_samples,n_features)` The build model was tried on different features combinations (`n_features` : [16 - F_t ,16 - Θ_t ,1 - M_{kg}, 16 - F_t x Θ_t])
- Output_shape: `(n_samples,n_output)` The model predicts a class out of n_outputs (one-hot encoded) classes (`n_outputs` : 13 classes) corresponding to each transformed target class.

9.7.3 Result and Discussion

For case 1, we considered 10 previous time-steps as the length of the observed window for the model to predict the event class. The model is trained with Adam optimizer with default learning rate of 0.01 and categorical_crossentropy as loss function.

Furthermore, Random forest is used for the case 2. As the name implies, Random forest consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

Illustrative results for both use cases across combination of raw and constructed features trials for slip detection and object recognition are presented in Figure 69 and Figure 70, respectively, and summarise in Table 10. It can be observed that there were minor differences in the classifier behavior across the features trials for the object recognition case, but the overall performance was highly accurate due to well constructed features and cleaning of noise from the raw data performed

during feature engineering stage. The model trained for slip detection case also performed reasonably well but a significant increase in the accuracy can be observed for the feature set involving derivative features of raw data. Thus the derivative features like $\Delta\Theta, \Delta F$ reasonably impacts the model performance for inference of resultant slip and crumple events during the grasp-pick phase of lifting the objects.

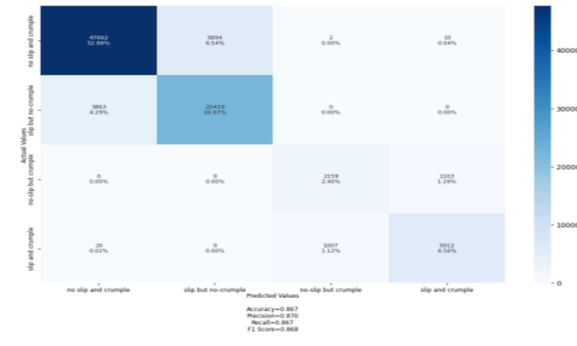


Figure 69: Confusion matrix for LSTM model for Case 1: slip detection

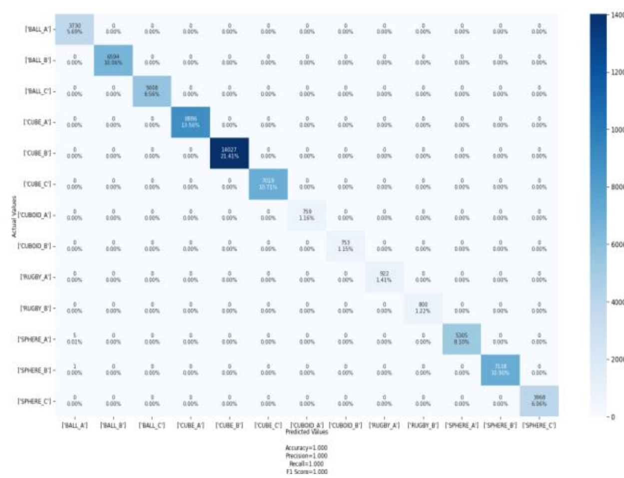


Figure 70: Confusion matrix for RF model for Case 2: object detection

<table>
<tr><th rowspan="2">Method</th><th rowspan="2">Fetures</th><th colspan="3">Slip Detection</th></tr>
<tr><th>Train</th><th>Validation</th><th>Test</th></tr>
<tr><td rowspan="2">LSTM</td><td>Θ, F</td><td>79.6</td><td>80.4</td><td>80.6</td></tr>
<tr><td>$\Delta F, \Delta\Theta, \Delta F$</td><td>86.4</td><td>84.2</td><td>85.6</td></tr>
<tr><td rowspan="4">RF</td><td rowspan="2"></td><td colspan="3">Object Detection</td></tr>
<tr><td>Train</td><td>Validation</td><td>Test</td></tr>
<tr><td>Θ, F</td><td>96.1</td><td>97.5</td><td>96.9</td></tr>
<tr><td>$\Delta F, \Delta\Theta, \Delta F$</td><td>98.1</td><td>99.5</td><td>98.9</td></tr>
</table>

Table 10 - Result table for both use cases (update table with Roger)

9.8 KPI anomaly analysis for 5G networks and beyond

This section presents PoC on examine KPI anomaly analysis for 5g networks and beyond [4] and produce a reference design. The proposed solution scans list of Network Functions(NF) from EMS, identifies NF for KPI retrieval and forecasts the KPIs from these NF for future time period. It also identifies anomalies and it source from the forecasted KPIs.

9.8.1 Design

Using historical KPI data, an ML Models is developed to be re-train at scheduled interval to learn new insights from the data and improve performance.

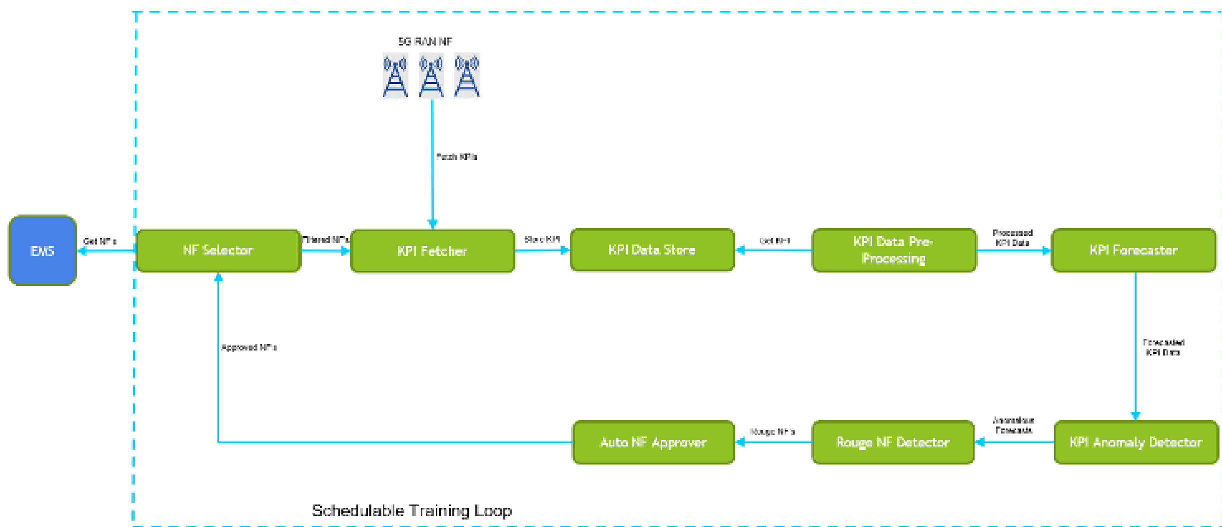


Figure 71: KPI anomaly analysis for 5G networks and beyond

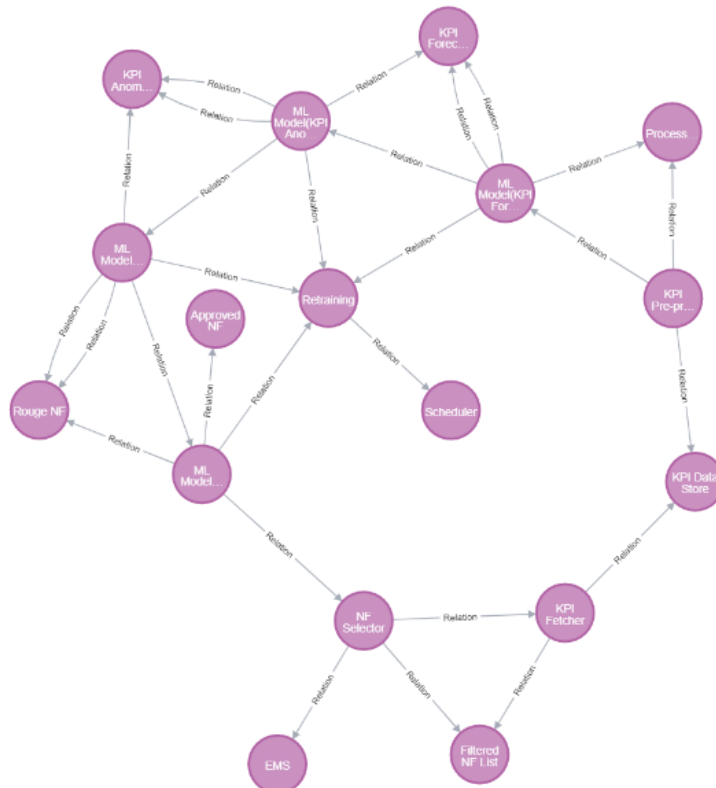


Figure 72: Relationship graph representation of KPI anomaly analysis for 5G networks and beyond

9.9 Autonomous Log Troubleshooting (with varied competence) in networks

This section presents the PoC on autonomous log troubleshooting (with varied competence) in networks. The case study focuses on fault prediction and isolation based on log analysis. The logs are general unstructured with no standard format, making it tedious to correlating logs across various vendors.

Through historical logs, machine learning can learn and predict failure. The proposed solution entails scanning all incoming log traces continuously, detect and report system issues such as incidents with probable root-cause automatically.

9.9.1 Design

The system takes feedback from user and re-train the Log anomaly detection to improve performance through self-correction. The main steps for the system include:

1. Logs collection from various open interfaces and NFs

2. Correlation and Analysis of the collected logs

3. performance optimization of log analysis mechanism

9.9.2 Summary

In this document, we discussed use case 15 from [ITU-T Y.Supp 71] and analyzed the design. We provide extensions to the reference code provided in [Build-a-thon 2022] and build our own graph based on the reference code.

.

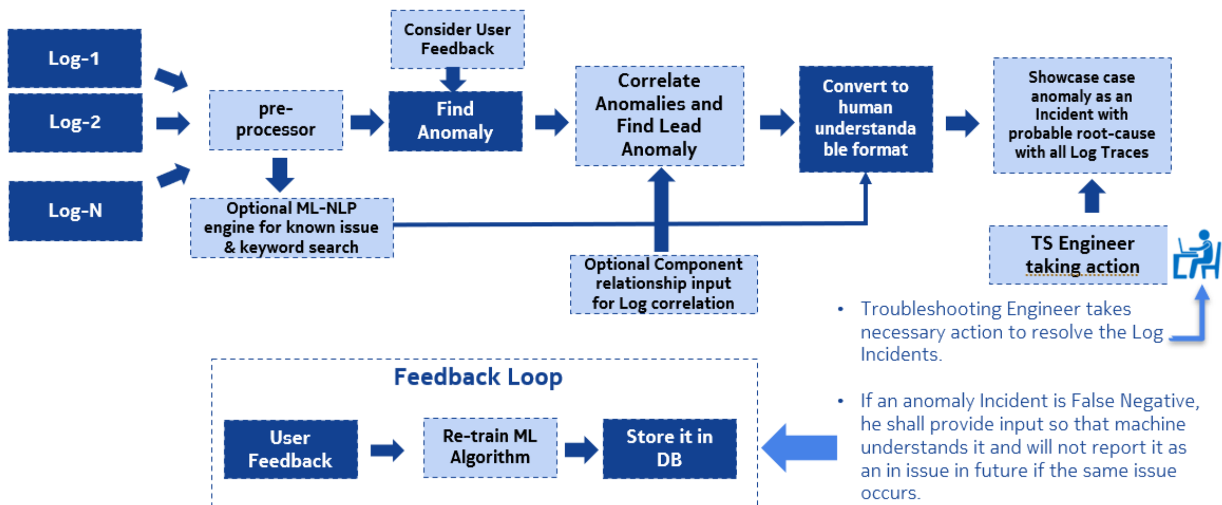


Figure 73: ML-based automated Log Troubleshooting

Fig. 74 – (7. AVANGERS) AUTONOMOUS LOGTROUBLESHOOTING (WITH VARIED COMPETENCE) IN NETWORKS.

10 Conclusion and future research

The PoC described here lays out the skeleton for an evolution framework based on the concepts described in [4], while leaving details regarding each of its components for further development. To this end, simplified implementations of modules and controllers are used for the PoC, as well as random combinations instead of a more complex evolutionary algorithm. Nevertheless, specific improvements on each of these examples can be made relying on the presented work, with little or no change in the architecture. That key feature makes this PoC valuable for the future development of the use cases for AN.

Another key point of focus of our approach is the assurance of security and traceability of the evolution process through the implementation of a decentralized Marketplace. The design and operation of the Marketplace has been described throughout this report, and we believe it represents a strong feature of our approach as well, as, thanks to the implementation of Ethereum and IPFS, it successfully provides with security, auditability, privacy and scalability through decentralization and a distributed means of storage. Furthermore, the choice to use the popular and widely adopted open-source technologies Ethereum and IPFS facilitates its future development and accessibility.

On top of security and auditability, using JSON as the data format for exchanging the controller and experiment information in the Marketplace enhance the explainability of the process, since it is a human readable format. Moreover, the deployment based on docker make the PoC portable and by using open-source tools and providing the source code we have ensured its future use and reproduction.

Regarding future lines of work, some of the ideas are:

- Implement more complex definitions of Modules and Controllers for solving a specific problem in an AN.
- Add a list of experiments that is suitable for that use case.

- Implement more complex evolutionary algorithms than random combination that can make use of the feedback that the Evol_Ctr receives about the experiment results.
- Add the possibility of updating new Modules after the evolution process has started. In order to make this possible all nodes must be notified of the new upload. Fortunately, IPFS provides the tools for making this possible without changing the architecture.
- Implement more complex Smart Contracts that would allow for more complex access control, updating and deleting artifacts, as well as an easier and more intuitive access to files.
- Add a more efficient detection and communication of updates within the system.

We can conclude from all of our findings that we found a way to parse the document and text, also the draw.io is the best tool that has a lot of customizations that can help us in making more representative figures.

Best algorithm for link prediction is the Random Forest model with 86% AUCPR accuracy in the full reference code, and also we need to find a more enhanced way to represent the use cases in the reference code, i.e. to convert all functions to be properties.

We need a way to include parsable shapes inside the document of use cases instead of images, to help automate the process further.

Also we need more representative figures or an algorithm for example and NLP to convert the descriptions of use cases into the reference code.

Another use case study was undertaken to create and develop a PoC for in-grasp manipulation with a low-latency inference system to solve grasping use cases of slip avoidance and in-hand object recognition [15]. The allegro gripper's in-hand joint force sensors detected the grasping state, and a outlined model in previous section were used to effectively process the information. As a result, the proposed method has the potential to generate successful inference outputs for low-latency in-grasp manipulation with abundant kinesthetic information (especially adding the derivative features for slip detection) by employing a LSTM which captures the temporal change of the joint information. The logical next step in our research is extending this to other rich tactile based multi-fingered in-grasp manipulation tasks because the tactile sensor can detect shear forces, which is highly subjected to change during multi-fingered in-grasp manipulation (e.g. slip and grasping from different orientations). Moreover, achieving several tasks (e.g. slip avoidance and object recognition) with one network for avoiding re-training on each task can be a next challenge.

Thus, Build-a-thon challenge in 2022 focussed on creation of a crowdsourced, baseline representation for AN closed loops (controllers), reviewing and analysing them, and publishing them in an open repository. The aim of the exercise was to produce reference implementations of parser, "AN orchestrator", "openCN" [ITU-T Y.Supp 71], Evolution controller [FGAN-O-023] and to trigger technical discussions on the standard format for representing closed loops (controllers) with FG AN members and other stakeholders. This would pave the way for further downstream extensions on top of the baseline. The main activities included (1) the implementation of a reference TOSCA orchestrator to demonstrate the parsing and validation of the format in a closed loop (2) development of an evolution/exploration mechanisms to create new closed loops based on existing closed loops or controllers. Clause 9 below describes this PoC.

Build-a-thon challenge in 2021 demonstrated and validated important use cases for autonomous networks, creating PoC implementations and tools in the process relating to emergency management. Interactions between a higher closed-loop in the OSS and a lower closed-loop in the

RAN to intelligently share RAN resources between the public and emergency responder slice were used as the background scenario for this PoC. The outcomes of the challenge were submitted by the various teams that participated in creating the PoC as contributions and was the main learnings were submitted as ITU J-FET paper [TBD]. The main outputs of the Build-a-thon challenge in 2021 include: (1) the implementation of a higher closed-loop “controllers” in a declarative fashion (intent), (2) the design and implementation of a lower closed-loop with Cloud Radio Access Network (C-RAN) to trigger “imperative actions” in the “underlay” based on the intent, (3) implementation of a simulation environment for data pipeline between various components; formulation of methods/algorithms for “influencing” lower layer loops using specific logic/models, and (4) the integration of the closed-loops and systems into an Open Radio Access Network (O-RAN)-based software platform, ready to be tested in the 5G Berlin testbed. The 2021 PoC study focused on intent parsing, traffic monitoring, resource computing, and allocation autonomously. The closed-loops were implemented with several micro-services deployed as docker containers with specific functions such as monitoring, computing, ML selection, and resource allocation. 2021 was a collaborative study where we developed and implemented a hierarchical closed-loop that autonomously handles an emergency use case. Clause 8 below describes this PoC.

As a future direction, build-a-thon Challenge 2023 is planned too, to further build upon the use cases designed as part of the 2021 and 2022 Build-a-thons, study the autonomy engine defined in [FGAN-O-023], especially regarding the possibility to plugin different evolution mechanisms as a service with clear but limited interfaces and interoperability with different knowledge bases with clear but limited interfaces.

Bibliography

[b-FGAN-I-078] Evolutionary algorithms, and how they could be applied to achieve adaptation in control system.

[b-FGAN-I-093] S. Sultana and A. Mittermaier. "Build-a-Thon (PoC) – 5G Berlin Test Network Functional Specification", https://extranet.itu.int/sites/itu-t/focusgroups/an/input/FGAN-I-093.zip (accessed 07/06, 2022).

[b-oran-arch] O-RAN. "O-RAN Architecture Overview", https://docs.o-ran-sc.org/en/latest/architecture/ architecture.html (accessed 03/09, 2022).

[b-Gomes] P. H. Gomes, M. Buhrgard, J. Harmatos, S. K. Mohalik, D. Roeland, and J. Niemöller, "Intent-driven closed loops for autonomous networks", Journal of ICT Standardization, pp. 257–290-257–290, 2021

[b-Luzar] A. Luzar, S. Stanovnik, and M. Cankar, "Examination and Comparison of TOSCA Orchestration Tools", in European Conference on Software Architecture, 2020: Springer, pp. 247-259.

[b-Ram] V. Ram O.V et al. "Proposal for a “Build-a-thon” for ITU AI/ML in 5G Challenge (second edition, 2021), aligned with FGAN WG3", ITU Focus Group on Autonomous Network (FG-AN). https://extranet.itu.int/sites/itu-t/focusgroups/an/input/FGAN-I-170-R1.docx (accessed 02/23, 2022).

[b-oasis] OASIS TOSCA Simple Profile in YAML v1.3, xopera, 2021. [Online]. Available: https://xlab-si.github.io/xopera-docs/

[b-Nahum] C. V. Nahum et al., "Testbed for 5G connected artificial intelligence on virtualized networks", IEEE Access, vol. 8, pp. 223202-223213, 2020.

[b-Aliyu] I. Aliyu, M. C. Feliciano, S. Van Engelenburg, D. O. Kim, and C. G. Lim, "A blockchain-based federated forest for SDN-enabled in-vehicle network intrusion detection system", IEEE Access, vol. 9, pp. 102593-102608, 2021.

[b-Ryu] Ryu. "Ryu API Reference", https://ryu.readthedocs.io/en/latest/api_ref.html (accessed 01/25, 2022).

[b-Flexran] Flexran. "Mosaic5G", https://mosaic5g.io/ flexran/ (accessed 01/25, 2022).

[b-docker] Empowering App Development for Developers | Docker, Docker. [Online]. Available: https://www.docker.com

[b-oai] O. OpenAirInterface. Accessed: Sep. 13. "5G Software Alliance for Democratising Wireless Innovation", http://www.openairinterface.org/ (accessed 01/25, 2022).

[b-free5gc] Free5GC. "Free5GC: Open-Source 5GC", https://www.free5gc.org/ (accessed 01/25, 2022).

[b-kubernetes] Kubernetes. "Kubernetes", (accessed 01/25, 2022).

[b-Afolabi] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions", IEEE Communications Surveys & Tutorials, vol. 20, no. 3, pp. 2429-2453, 2018.

[b-foukas] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "FlexRAN: A flexible and programmable platform for software-defined radio access networks", in Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies, 2016, pp. 427-441.

[b-lins] S. Lins et al., "Artificial Intelligence for Enhanced Mobility and 5G Connectivity in UAV-Based Critical Missions", IEEE Access, vol. 9, pp. 111792-111801, 2021.

[b-simu5g] G. Nardini, D. Sabella, G. Stea, P. Thakkar, and A. Virdis, "Simu5G–An OMNeT++ Library for End-to-End Performance Evaluation of 5G Networks", IEEE Access, vol. 8, pp. 181176-181191, 2020.

[b-liu] F. Liu, Y. Guo, Z. Cai, N. Xiao, and Z. Zhao, "Edge-enabled disaster rescue: a case study of searching for missing people", ACM Transactions on Intelligent Systems and Technology (TIST), vol. 10, no. 6, pp. 1-21, 2019.

[b-raida] V. Raida, P. Svoboda, and M. Rupp, "Real World Performance of LTE Downlink in a Static Dense Urban Scenario-An Open Dataset", in GLOBECOM 2020-2020 IEEE Global Communications Conference, 2020: IEEE, pp. 1-6.

[b-foukas] X. Foukas, M. K. Marina, and K. Kontovasilis, "Orion: RAN slicing for a flexible and cost-effective multi-service mobile network architecture", in Proceedings of the 23rd annual international conference on mobile computing and networking, 2017, pp. 127-140.

[b-okic] A. Okic, L. Zanzi, V. Sciancalepore, A. Redondi, and X. Costa-Pérez, "π-ROAD: A learn-as-you-go framework for on-demand emergency slices in V2X scenarios", in IEEE INFOCOM 2021-IEEE Conference on Computer Communications, 2021: IEEE, pp. 1-10.

[b-tosca] TOSCA Simple Profile in YAML Version 1.3, C. L. OASIS Committee Specification 01..Edited by Matt Rutkowski, Claude Noshpitz, 2019. [Online]. Available: https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/TOSCA-SimpleProfile-YAML-v1.3.html

[b-nrt-ric] K. Kristiansen. "Near Realtime RIC ", https://wiki.o-ran-sc.org/display/GS/ Near+Realtime+RIC+Installation (accessed 03/09, 2022).

[b-acumos] S. Zhao, M. Talasila, G. Jacobson, C. Borcea, S. A. Aftab, and J. F. Murray, "Packaging and sharing machine learning models via the acumos ai open platform", in 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 2018: IEEE, pp. 841-846.

[b-oran-sdk] Z. Huang. "App Writing Guide", https://wiki.o-ran-sc.org/display/ORANSDK/ App+Writing+Guide (accessed 03/09, 2022).

[b-oran-sc] O-RAN. "O-RAN Software Community",https://github.com/o-ran-sc (accessed 03/09,2022).

[b-FGAN-I-197] S. Sultana and A. Mittermaier. "Updates on the5G test network, Plans on intent based-networkslicing", ITU-T FGAN. https://extranet.itu.int/sites/itu-t/focusgroups/an/input/FGAN-I-197.zip (accessed 03/12,2022).

____