ICT-56-2020 "Next Generation Internet of Things"

Grant Agreement number: 957218

IntelIIoT

Deliverable D5.4 Integrated IntellIoT framework & use case implementations (final version)

Deliverable release date	30/08/2023
Authors	1. SIEMENS: Andreas Zirkler, Andreas Ziller, Christian Bachmann 2. EURECOM: Jérôme Härri 3. AAU: Beatriz Soret, Andreas Casparsen 4. UOULU: Sumudu Samarakoon, Mohamed Abdelaziz 5. TTC: Martijn Rooker 6. TSI: Andreas Brokalakis, Babis Savvakos, Thomas Kyriakakis, Fotini Karetsi, Charalampos Ioannis Mitropoulos, Ioannis Kopanakis 7. PHILIPS: Anca Bucur, Bas Flaton 8. SANL: Antonios Paragioudakis, Sotirios Michail, Despoina Ntolka 9. HSG: Simon Mayer 10. HOLO: Clayton Gordy, Nour Fendri 11. AVL: Wolfgang Hollerweger, Holger Burkhardt
Editor	Wolfgang Hollerweger (AVL)
Reviewer	Simon Mayer (HSG), Clayton Gordy (HOLO)
Approved by	PTC Members: (Vivek Kulkarni, Konstantinos Fysarakis, Sumudu Samarakoon, Beatriz Soret, Arne Bröring, Maren Lesche) PCC Members: (Vivek Kulkarni, Jérôme Härri, Beatriz Soret, Mehdi Bennis, Martijn Rooker, Sotiris Ioannidis, Anca Bucur, Georgios Spanoudakis, Simon Mayer, Holger Burkhardt, Maren Lesche, Georgios Kochiadakis)
Status of the Document	In Work
Version	1.0
Dissemination level	Public

Table of Contents

FIGURES

ACRONYMS AND DEFINITIONS

Acronym	**Definition**
5G NR	5th Generation New Radio
AES	Advanced Encryption Standard https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
AI	Artificial Intelligence
AMQP	Advanced Message Queuing Protocol https://www.amqp.org/
API	Application Programming Interface
AR	Augmented Reality
CAN	Controller Area Network
CBC	Cipher Block Chaining
CIDR	Classless Inter-Domain Routing
CN	Core Network
COVID-19	Corona Virus Disease of 2019
Dapps	Decentralized apps
DLT	Distributed Ledger Technology
ECG	Electrocardiogram
EMS	Edge Management System
ETH	*Eth*er crypto currency
FML	Fast Machine Learning
GPS	Global Positioning System
HMAC	(Keyed) Hash-based Message Authentication
HMU	Head Mounted Unit
HyperMAS	Hypermedia Multi-agent System
IAKM	Infrastructure Assisted Knowledge Management
ID	Identification
IDS	Intrusion Detection System
IMSI	International Mobile Subscriber Identity
IO	Input/Output
IoT	Internet of Things
ISAR	Interactive Streaming for Augmented Reality
IT/OT	Information Technology/Operational Technology
JSON	JavaScript Object Notation

JSON-LD	JSON for Linking data
JSON-RPC	JSON-based Remote Procedure Call
JWT	JSON Web Token
Keycloak	Keycloak Open-Source Identity and Access Management
LLDP	Link-Layer Discovery Protocol
MEC	Mobile Edge Computing
MTD	Moving Target Defenses
Node-RED	Low-code programming for event-driven applications https://nodered.org/ built on Node.js
NodeJS	The node.js open-source cross-platform JavaScript runtime environment
OAuth	Open Authorization (https://oauth.net/) is a way to get access to protected data from an application.
OpenID Connect	OpenID Connect (https://openid.net/connect) is a simple identity layer on top of the OAuth 2.0 protocol
OpenAPI	OpenAPI (https://www.openapis.org) is a specification language for HTTP APIs.
QoS	Quality of Service
RabbitMQ	RabbitMQ (https://www.rabbitmq.com) is an open-source message broker
RAN	Radio Access Network
ROS	Robot Operating System, https://www.ros.org
RNTI	Radio Network Temporary Identifier
SAP	Security Assurance Platform
Solidity	Solidity (https://soliditylang.org) is a statically-typed programming language designed for developing smart contracts that run on Ethereum
TD	W3C WoT Thing Description
TLS	Transport Layer Security
TSN	Time Sensitive Networking
TUN	Network TUNnel, a kernel virtual network device
UC	IntellIoT Use Case (1: agriculture, 2: healthcare, 3: manufacturing)
UDP	User Datagram Protocol
UE	User Equipment
UML	Unified Modelling Language
UR5	Universal Robots UR5 Collaborative Robot Arm (https://www.universal-robots.com/products/ur5-robot)
VLAN	Virtual Local Area Network
VPN	Virtual Private Network

VR	Virtual Reality
W3C	World Wide Web Consortium
WoT	Web of Things
WP	Work Package of IntellIoT project

1 INTRODUCTION

This deliverable D5.4 summarizes the work that has been done in the second cycle of Task 5.1, related to the integration of components and their interfaces in the 3 UCs of IntellIoT. The predecessor of this document is the first version D5.1. The aim of Task 5.1 is to implement the IoT applications, their underlying services, and the planned demonstrators of the use cases, while remaining in the developer environment and utilizing mostly simulated data. The step towards the demonstration environment is done in Task 5.2. Deliverable D5.4 therefore provides the final version of the integrated IntellIoT framework of technology components (from WP3 and WP4) as well as the final implementation of use case applications and services. The components of the IntellIoT framework are based on the 3 pillars: (a) human-defined autonomy, (b) distributed, self-aware IoT applications and (c) an efficient, reliable, and trustworthy computation & communication infrastructure as indicated Figure 1.

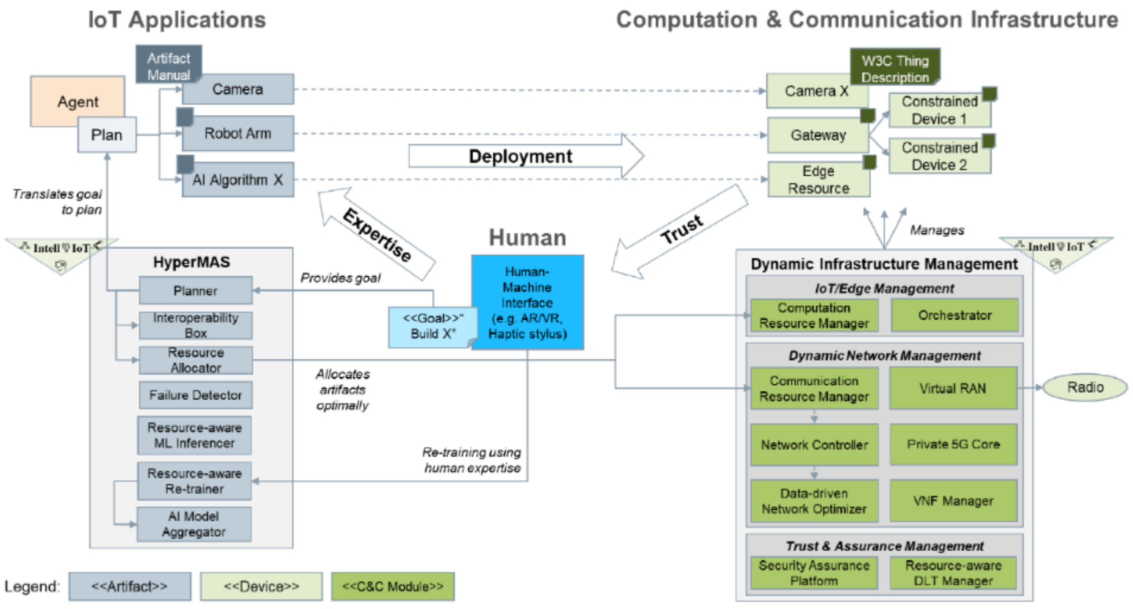


Figure 1: The IntellIoT framework – example artifacts and devices are shown in the upper, deployment part.

The relation to other Tasks and Deliverables of the IntellIoT project is the following:

In T2.3, the architecture specification & interoperability the components were defined and described in D2.3 "High Level Architecture". In the first versions of the deliverables of the tasks of WP3 (distributed self-aware IoT applications) and WP4 (Efficient, reliable, and trustworthy computation & communication infrastructure) the technological components making up the IntellIoT framework have been described in detail. T5.2 is following T5.1 in the project's progress. It will deploy, test, and demonstrate the IntellIoT framework components and UC applications in the demonstration environment, which means we will conduct the demonstrations for the 3 UCs in real-life settings by incorporating, e.g., an actual tractor, real patients, and actual manufacturing devices and machinery.

1.1 Outline of this Deliverable

To present the work of Task 5.1 on integrating the components, the remainder of this deliverable is organized as follows:

- Chapter 2 details the integration efforts on the different components of the IntellIoT framework.
- Chapter 3 describes the use case specific implementations, particularly, the implemented applications and services.
- Chapter 4 concludes this deliverable and outlines future work.

2 COMPONENT INTEGRATION

In D2.6 the final high-level architecture of *IntellIoT* was described. Components were identified and clustered into five core component groups (see Figure 2, which is described in D2.6 as *Figure 13)*. In the deliverables of WP3 and WP4 the functionality and design of those components is described. Before the deployment of these components to UC1-3 is done in T5.2, they are integrated in the developer environment. Therefore, in the following sections the test environment, inputs, and evaluation of the outputs are described. The sections represent the first four core component groups, and we describe the individual components within these groups in the respective subsections. The fifth core component group (the use case specific implementations) is covered separately in Chapter 3.

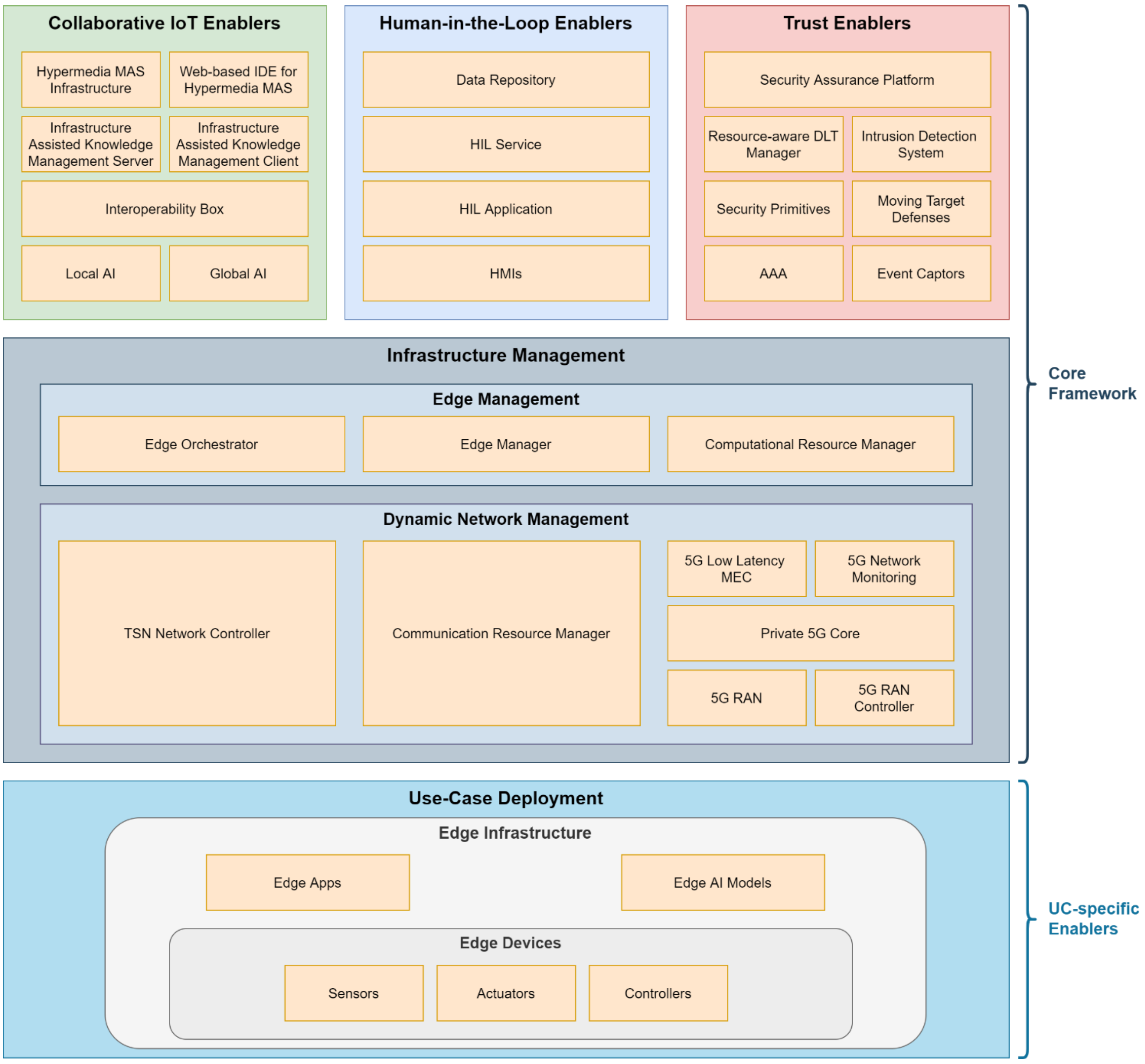
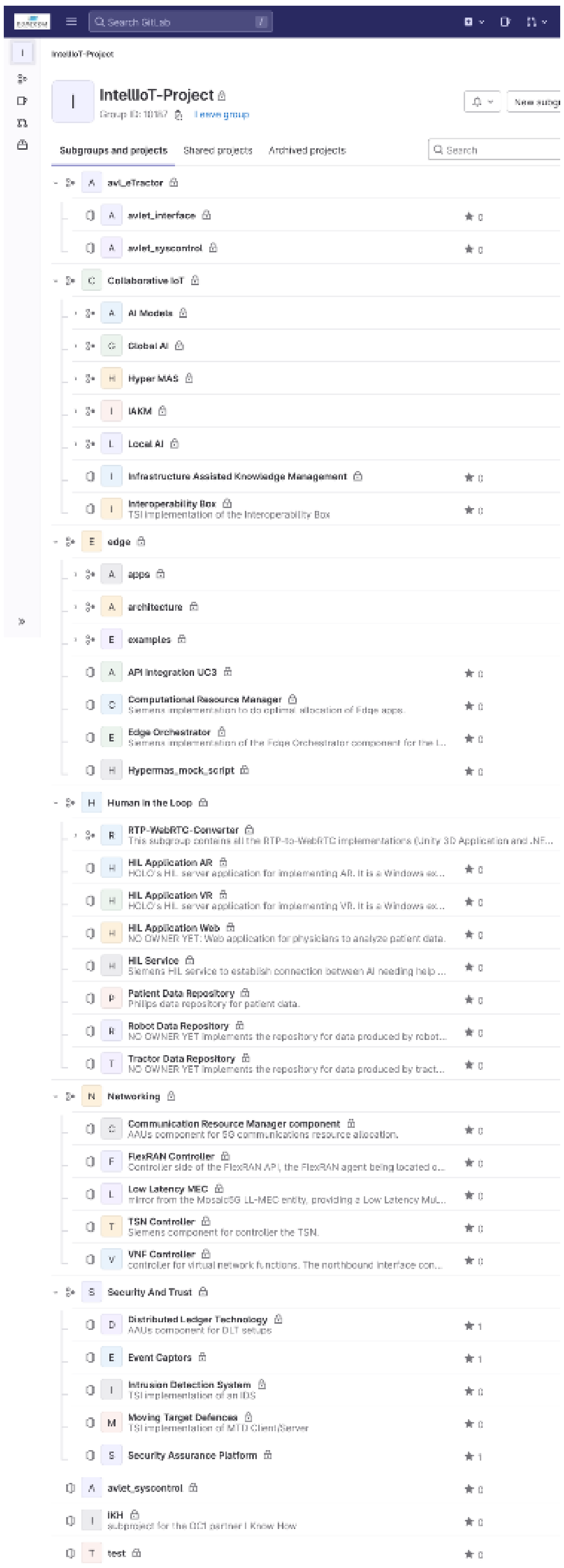


Figure 2: High-level view of IntellIoT's logical architecture (Source: D2.6, Figure 13)

2.1 Overview of Integration and Implementation Methodology

Integration of components (deployment is content of follow-up task T5.2), is done in different developer environments. For Continuous Integration (CI) a private *GitLab group* was set up with sub-groups for the pillars of *IntellIoT* containing further sub-groups and projects for the components (see Figure 2).



IntellIoT-Project

IntellIoT-Project
Group ID: 10167 Leave group

- avl_eTractor
  - avlet_interface ★ 0
  - avlet_syscontrol ★ 0
- Collaborative IoT
  - AI Models
  - Global AI
  - Hyper MAS
  - IAKM
  - Local AI
  - Infrastructure Assisted Knowledge Management ★ 0
  - Interoperability Box ★ 0
    TSI implementation of the Interoperability Box
- edge
  - apps
  - architecture
  - examples
  - API Integration UC3 ★ 0
  - Computational Resource Manager ★ 0
    Siemens implementation to do optimal allocation of Edge apps.
  - Edge Orchestrator ★ 0
    Siemens implementation of the Edge Orchestrator component for the I...
  - Hypermas_mock_script ★ 0
- Human In the Loop
  - RTP-WebRTC-Converter
    This subgroup contains all the RTP-to-WebRTC implementations (Unity 3D Application and .NE...
  - HIL Application AR ★ 0
    HOLO's HIL server application for implementing AR. It is a Windows ex...
  - HIL Application VR ★ 0
    HOLO's HIL server application for implementing VR. It is a Windows ex...
  - HIL Application Web ★ 0
    NO OWNER YET: Web application for physicians to analyze patient data.
  - HIL Service ★ 0
    Siemens HIL service to establish connection between AI needing help ...
  - Patient Data Repository ★ 0
    Philips data repository for patient data.
  - Robot Data Repository ★ 0
    NO OWNER YET Implements the repository for data produced by robot...
  - Tractor Data Repository ★ 0
    NO OWNER YET Implements the repository for data produced by tract...
- Networking
  - Communication Resource Manager component ★ 0
    AAUs component for 5G communications resource allocation.
  - FlexRAN Controller ★ 0
    Controller side of the FlexRAN API, the FlexRAN agent being located o...
  - Low Latency MEC ★ 0
    mirror from the Mosaic5G LL-MEC entity, providing a Low Latency Mul...
  - TSN Controller ★ 0
    Siemens component for controller the TSN.
  - VNF Controller ★ 0
    controller for virtual network functions. The northbound interface con...
- Security And Trust
  - Distributed Ledger Technology ★ 1
    AAUs component for DLT setups
  - Event Captors ★ 1
  - Intrusion Detection System ★ 0
    TSI implementation of an IDS
  - Moving Target Defences ★ 0
    TSI implementation of MTD Client/Server
  - Security Assurance Platform ★ 1
- avlet_syscontrol ★ 0
- IKH ★ 0
  subproject for the OC1 partner I Know How
- test ★ 0

Figure 3: Private GitLab group of the IntellIoT project.

From May 2021 to July 2023, 48 bi-weekly WP5 online meetings including T5.1 topics were held. When, after Covid lockdowns, the consortium members finally could meet face to face in Munich at the consortium meeting #5 (Nov 2nd & 3rd 2021), it was easier to align integration issues of the 3 UCs. A result was e.g., the UC1 demo setup slide in Figure 4.

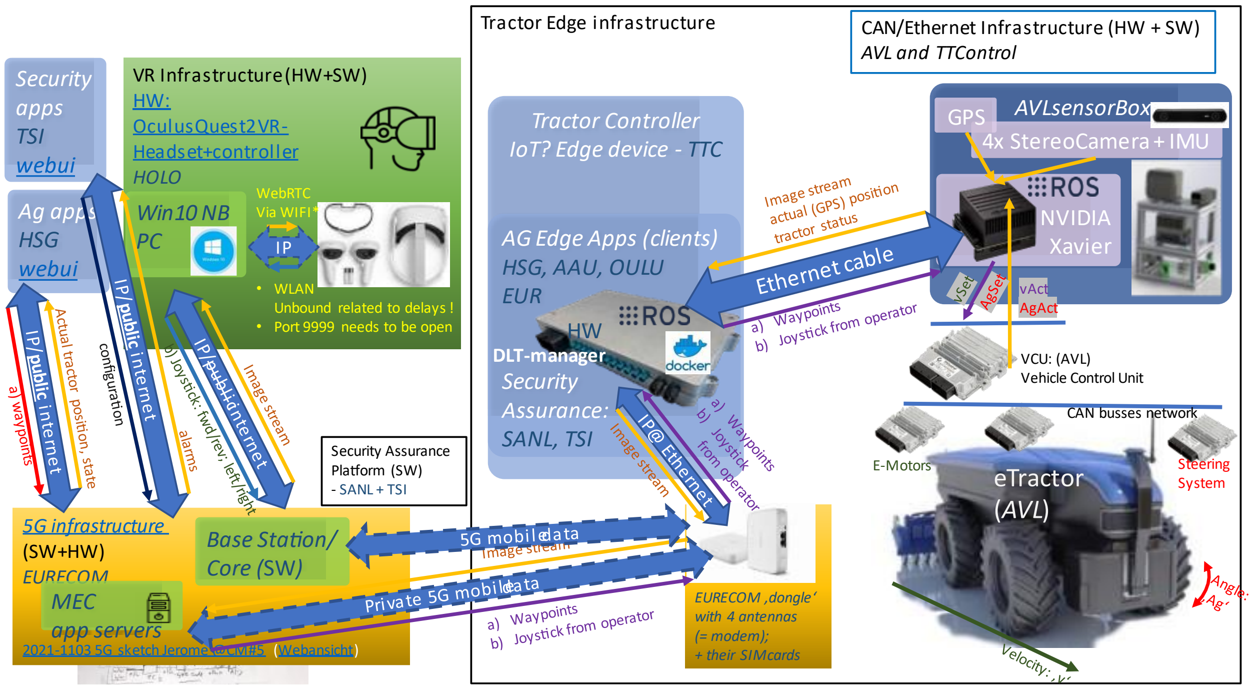


Figure 4: UC1 demo setup worked out at Consortium Meeting #5.

2.2 Collaborative IoT Enablers

2.2.1 HYPERMEDIA MAS INFRASTRUCTURE

The Hypermedia MAS Infrastructure[1] is a platform that is used to create and deploy hypermedia environments that are based on the Agents & Artifacts Meta-Model. Its key abstractions are "workspaces", and "artifacts", and "agents". The "workspaces" and "artifacts" are modelled as W3C WoT Things and expose their functional interfaces using W3C WoT Thing Description[2]. While agents are not modelled as W3C WoT Things, they also provide a REST API. The infrastructure furthermore permits Agents to observe resources in the hypermedia environment through a W3C WebSub Hub. Since the Hypermedia MAS Infrastructure is described in detail in Section 3.1 of IntellIoT Deliverable D3.5, we only give a brief account of its capabilities for contextualization in this integration-oriented document.

A version of the Hypermedia MAS Infrastructure is deployed on HSG's servers[3] and is reachable world-wide. Together with locally deployed instances of the MAS Infrastructure, this is used in the IntellIoT use cases. To machine clients, it then provides the functionality to create, populate, and update hypermedia environments containing workspaces, artifacts, and agents. Relevant devices that are deployed in HSG's laboratories (e.g.,

[1] https://github.com/Interactions-HSG/yggdrasil
[2] These refer to the World Wide Web Consortium's (W3C) Web of Things (WoT) standardization group that has recently standardized Thing Descriptions (TDs) that describe the metadata and interfaces of WoT Things.
[3] https://yggdrasil.interactions.ics.unisg.ch/

mobile robots[4] and stationary robots[5]) that mock devices from IntellIoT partners (in this case, AVL's autonomous tractors and Siemens' UR5 robots) as well as selected devices that are deployed with IntellIoT partners (e.g., the UC3 Mock Engraver[6]) are registered with the infrastructure using W3C WoT TD. Given this setup, for the registered (hypermedia) artifacts (i.e., the robots, engraver, etc.), the infrastructure exposes the W3C WoT TD-described hypermedia controls to permit Agents to interact with this artifact and to permit Domain Experts to configure Agents to autonomously interact with it (see Section 2.2.2, Web-based IDE for Hypermedia Multi-Agent systems). In addition, artifact TDs can be updated, and clients can subscribe to receive notifications through the infrastructure's W3C WebSub Hub. Furthermore, the infrastructure exposes semantic containment relationships using the EVE vocabulary (see [1] and [2]) for the current version and the HMAS[7] vocabulary for the version in development workspaces and environments that permit the crawling of all registered resources and, thus, the efficient searching for artifacts that might be required by a client.

2.2.1.1 INFRASTRUCTURE ASSISTED KNOWLEDGE MANAGEMENT (IAKM)

The IAKM module has a dual task of assisting in the identification of potential AI models required by an agent according to a specific semantic, as well as assisting in the training of an AI model according to the model's required environment semantics. Accordingly, the IAKM provides two main functions to other IntellIoT components:

- An HTTP API to subscribe to train or use a required model.
- A JSON-based semantic for AI models uniquely describing the AI model or the AI environment required to use, respectively to train.

2.2.1.2 IAKM INTERNAL FUNCTIONS

The IAKM has two components; *IAKM backend* and *IAKM client*, securely connected and provided as a multi-microservice docker container.

The *IAKM backend* is a set of microservices provided as a multi-docker container and providing the infrastructure side of knowledge management. It consists of various complementary microservices:

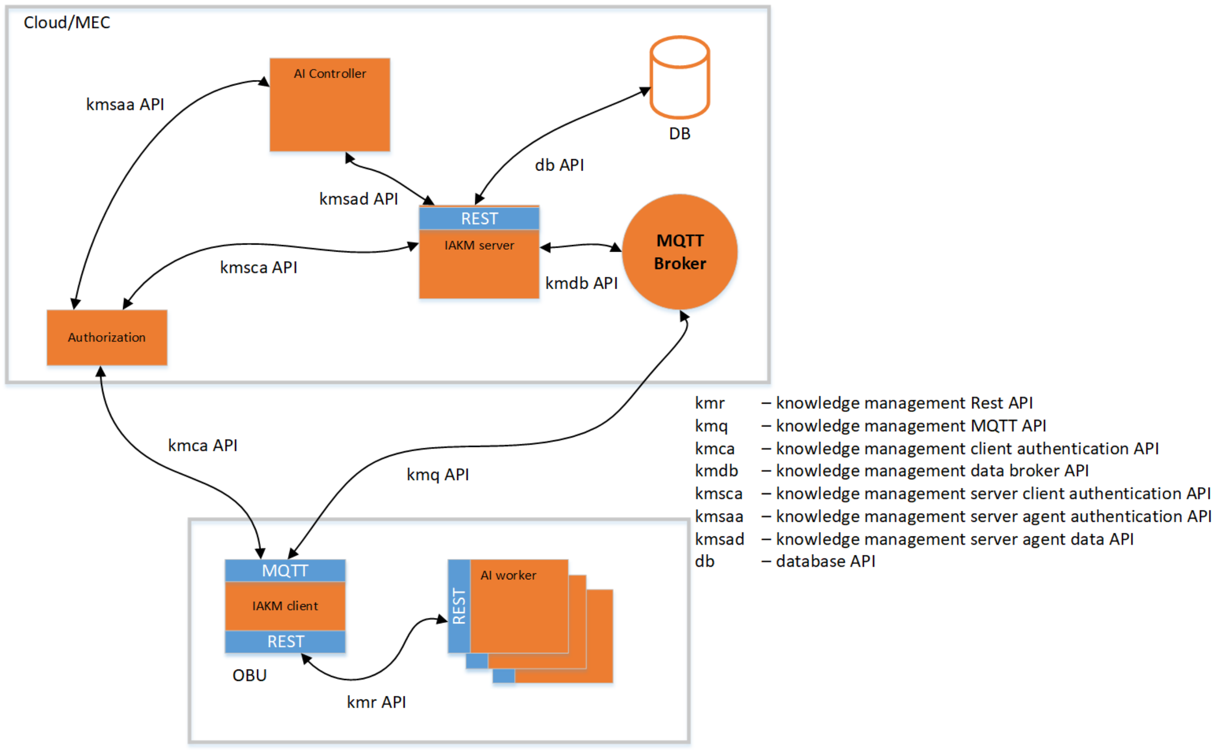
- *Databroker – It is implemented as an MQTT broker, with secured subscription and message passing between the IAKM agents and the IAKM server microservice.*
- *IAKM server* - its main objective is to handle the authentication of the external microservices, such as the IAKM clients pr the global AI component. Its second objective is to act as a Web server and expose an HTTP API to the AI entity.
- *Database* – it is implemented as a MongoDB and is connected to the AIKM server to save and retrieve knowledge models if locally available.

The *IAKM client* is a set of microservices provided on the client side and connected over an HTTP API to local AI workers. It has two main interfaces:

- *Northbound API – it is a MQTT API to send and retrieve data from the databroker. It also subscribes to the AIKM server.*
- *Southbound API – it is a HTTP API to expose subscribe and publish functions.*

[4]https://yggdrasil.interactions.ics.unisg.ch/environments/intelliot/workspaces/uc-agriculture/artifacts/spockbot
[5] http://yggdrasil.interactions.ics.unisg.ch/environments/intelliot/workspaces/uc-industry/artifacts/leubot1
[6] https://yggdrasil.interactions.ics.unisg.ch/environments/intelliot/workspaces/uc-industry/artifacts/engraver
[7] https://ci.mines-stetienne.fr/hmas/core#

The architecture of the IAKM (back end and client) is available in Figure 5.



kmr	– knowledge management Rest API
kmq	– knowledge management MQTT API
kmca	– knowledge management client authentication API
kmdb	– knowledge management data broker API
kmsca	– knowledge management server client authentication API
kmsaa	– knowledge management server agent authentication API
kmsad	– knowledge management server agent data API
db	– database API

Figure 5: Component architecture of the IAKM (server back end and client).

2.2.1.3 IAKM MESSAGE FLOW AND INTEGRATION WITH AI COMPONENTS

Figure 6 illustrates the message flow and integration of the IAKM client and backend with the Global AI component. In this example, we considered one AI infrastructure as a FML controller and two AI agents as FML workers for two functions:

- AI knowledge retrieval for usage – one or more AI agents require the assistance of the IAKM to locate a particular AI model. In that function, the AI infrastructure is not required. If the IAKM has the requested model in its local DB it will provide it to the AI agent. If not, it will contact the external IAKM backend entities to locate the AI model.
- AI knowledge training assistance – the IAKM does not train AI models but provides assistance in the training in the form of the identification of the most efficient AI agents to train a particular AI model.

Although IAKM internal message exchanges are either based on MQTT or plain socket APIs, the IAKM only exposes HTTP APIs to subscribes to a particular IAKM service, to post or get particular AI models.

The upper part of Figure 6 (for usage) shows that both FML controller and both AI agents subscribe to an IAKM service. Note that specific callback functions are implemented on the IAKM agent and server to provide a REST subscription API. In the example, the AI agents subscribe to a particular AI model according to a specific AI semantic. The IAKM retrieves the AI model and pushes it back to the AI agents.

On the lower part of Figure 6 (for training), we can see that the AI agents subscribe for their availability to train according to a particular environment (transmitted using an AI semantic). In turn, the FML controller subscribes to finding AI agents according to a particular context. The IAKM server makes the match and provides the AI model to the matching AI agents. In turn, after training, both agents send the updated models to the IAKM server, which pushes back to the FML controller. Once merged, the updated AI model is saved on the IAKM database.

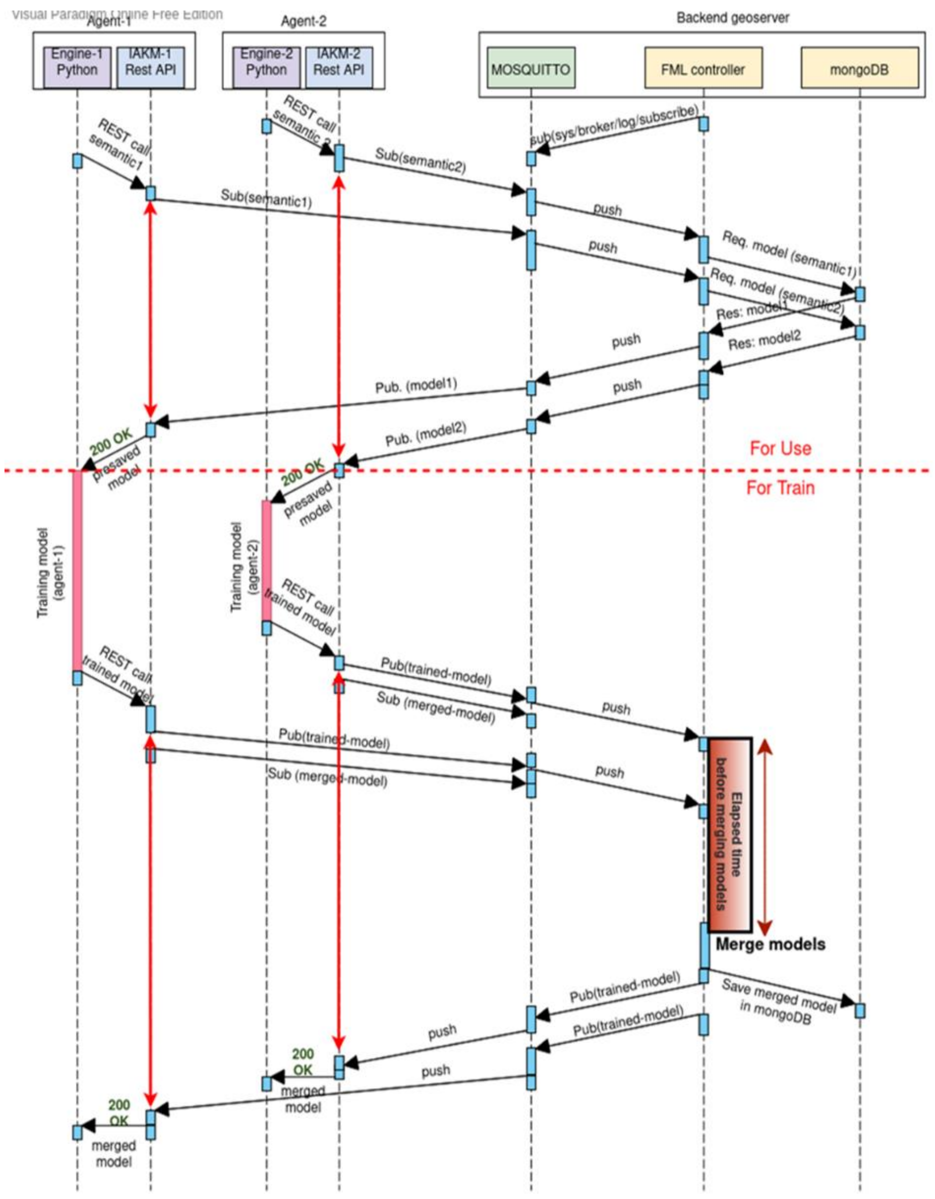


Figure 6: Process flow of IAKM assisted FML training and FML model exchange

2.2.1.4 IAKM HTTP API DESCRIPTION

Figure 7 depicts the subscription flows between the FML controller, the FML agent and the IAKM. The IAKM agent HTTP API provides callback-based subscriptions to three AI functions:

- Train – it is a subscription to ask for the assistance of other AI agents to jointly train a required model. In the case of an infrastructure FML, the IAKM will locate AI agents and provides them to the FML controller. In the case of Gossip FML, the IAKM locates AI agents itself and exchange AI models directly between AI agents.
- Use – it is a subscription to ask the IAKM to retrieve an AI model it needs to use.
- Available – it is a subscription to indicate to the FML controller that the AI agent is available for training according to the current AI context.

The IAKM server provides different services:

- Save/load - saves or loads an AI model from the DB and provides it either to the infrastructure AI or to the AI agents.
- Subscription matching – provides the matching between the subscriptions of the infrastructure AI and the AI agents.

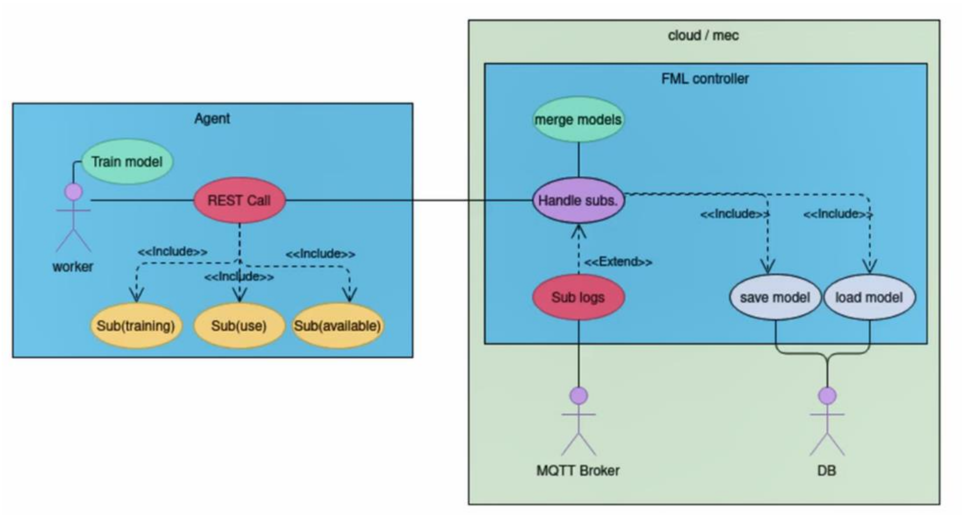


Figure 7: IAKM HTTP interactions for the FML service.

The subscription matching may be better seen on Figure 8. After both FML controller and workers initially subscribe to the IAKM backend (broker), the IAKM server matches both according to a similarity index on the AI semantic and publish the matching AI model.

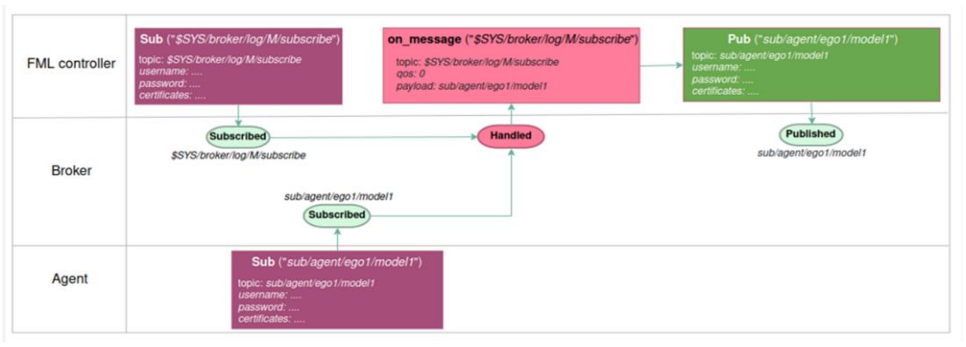
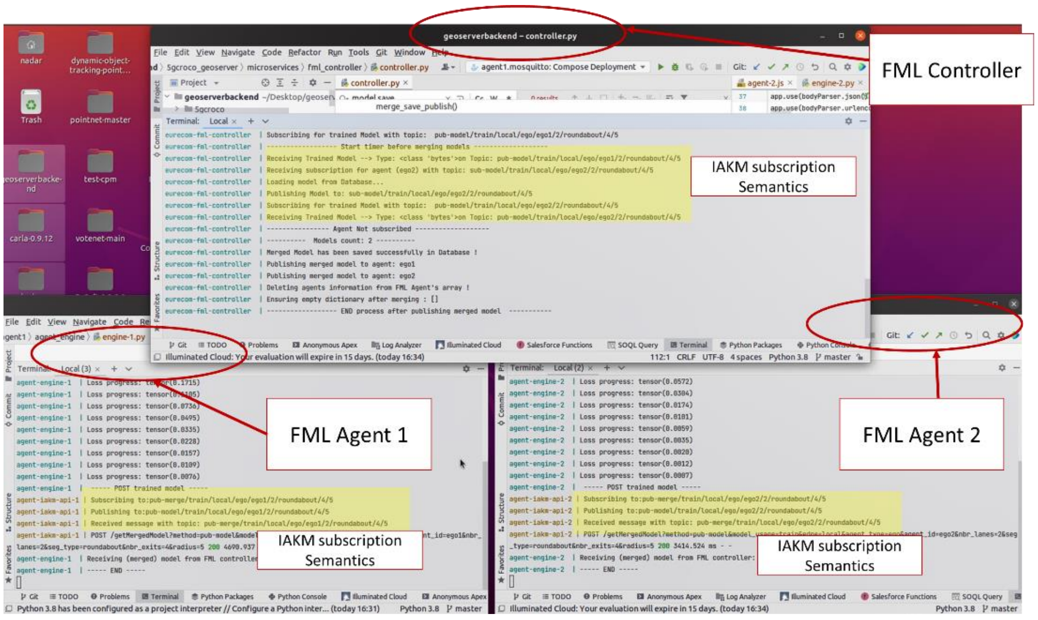

FML controller
Broker
Agent
Sub ("$SYS/broker/log/M/subscribe")
topic: $SYS/broker/log/M/subscribe
username: ....
password: ....
certificates: ....
on_message ("$SYS/broker/log/M/subscribe")
topic: $SYS/broker/log/M/subscribe
qos: 0
payload: sub/agent/ego1/model1
Pub ("sub/agent/ego1/model1")
topic: sub/agent/ego1/model1
username: ....
password: ....
certificates: ....
Subscribed
$SYS/broker/log/M/subscribe
Handled
Published
sub/agent/ego1/model1
sub/agent/ego1/model1
Subscribed
Sub ("sub/agent/ego1/model1")
topic: sub/agent/ego1/model1
username: ....
password: ....
certificates: ....

Figure 8: IAKM subscription/matching process for FML service.

2.2.1.5 IAKM EXAMPLE – FML TRAINING

Figure 9 depicts the output of the IAKM assisting FML. Both FML and the FML workers are implemented are *pytorch* FML code connected to the IAKM via the exposed HTTP APIs. The selected semantics to subscribe to and post AI models are generic at that stage and for the sole purpose of showing the functionality of the back-end services. The IAKM AI semantics are currently being defined.


geoserverbackend – controller.py
FML Controller
IAKM subscription Semantics
FML Agent 1
IAKM subscription Semantics
FML Agent 2
IAKM subscription Semantics

```
eurecom-fml-controller  | Subscribing for trained Model with topic:  pub-model/train/local/ego/ego1/2/roundabout/4/5
eurecom-fml-controller  | ----------------- Start timer before merging models ------------------
eurecom-fml-controller  | Receiving Trained Model --> Type: <class 'bytes'>on Topic: pub-model/train/local/ego/ego1/2/roundabout/4/5
eurecom-fml-controller  | Receiving subscription for agent (ego2) with topic: sub-model/train/local/ego/ego2/2/roundabout/4/5
eurecom-fml-controller  | Loading model from Database...
eurecom-fml-controller  | Publishing Model to: sub-model/train/local/ego/ego2/2/roundabout/4/5
eurecom-fml-controller  | Subscribing for trained Model with topic:  pub-model/train/local/ego/ego2/2/roundabout/4/5
eurecom-fml-controller  | Receiving Trained Model --> Type: <class 'bytes'>on Topic: pub-model/train/local/ego/ego2/2/roundabout/4/5
eurecom-fml-controller  | --------------- Agent Not subscribed ------------------
eurecom-fml-controller  | ---------  Models count: 2 ---------
eurecom-fml-controller  | Merged Model has been saved successfully in Database !
eurecom-fml-controller  | Publishing merged model to agent: ego1
eurecom-fml-controller  | Publishing merged model to agent: ego2
eurecom-fml-controller  | Deleting agents information from FML Agent's array !
eurecom-fml-controller  | Ensuring empty dictionary after merging : []
eurecom-fml-controller  | ----------------- END process after publishing merged model  ----------
```

```
agent-engine-1  | Loss progress: tensor(0.1715)
agent-engine-1  | Loss progress: tensor(0.1105)
agent-engine-1  | Loss progress: tensor(0.0736)
agent-engine-1  | Loss progress: tensor(0.0495)
agent-engine-1  | Loss progress: tensor(0.0335)
agent-engine-1  | Loss progress: tensor(0.0228)
agent-engine-1  | Loss progress: tensor(0.0157)
agent-engine-1  | Loss progress: tensor(0.0109)
agent-engine-1  | Loss progress: tensor(0.0076)
agent-engine-1  |  ----- POST trained model -----
agent-iakm-api-1 | Subscribing to:pub-merge/train/local/ego/ego1/2/roundabout/4/5
agent-iakm-api-1 | Publishing to:pub-model/train/local/ego/ego1/2/roundabout/4/5
agent-iakm-api-1 | Received message with topic: pub-merge/train/local/ego/ego1/2/roundabout/4/5
agent-iakm-api-1 | POST /getMergedModel?method=pub-model&model[illegible]nt_id=ego1&nbr_
lanes=2&seg_type=roundabout&nbr_exits=4&radius=5 200 4690.937 [illegible]
agent-engine-1  | Receiving (merged) model from FML controlle[illegible]
agent-engine-1  | ----- END -----
```

```
agent-engine-2  | Loss progress: tensor(0.0572)
agent-engine-2  | Loss progress: tensor(0.0384)
agent-engine-2  | Loss progress: tensor(0.0174)
agent-engine-2  | Loss progress: tensor(0.0101)
agent-engine-2  | Loss progress: tensor(0.0059)
agent-engine-2  | Loss progress: tensor(0.0035)
agent-engine-2  | Loss progress: tensor(0.0020)
agent-engine-2  | Loss progress: tensor(0.0012)
agent-engine-2  | Loss progress: tensor(0.0007)
agent-engine-2  |  ----- POST trained model -----
agent-iakm-api-2 | Subscribing to:pub-merge/train/local/ego/ego2/2/roundabout/4/5
agent-iakm-api-2 | Publishing to:pub-model/train/local/ego/ego2/2/roundabout/4/5
agent-iakm-api-2 | Received message with topic: pub-merge/train/local/ego/ego2/2/roundabout/4/5
agent-iakm-api-2 | POST /getMergedModel?method=pub-model&model_usage=train&edge=local&agent_type=ego&agent_id=ego2&nbr_lanes=2&seg
_type=roundabout&nbr_exits=4&radius=5 200 3414.524 ms - -
agent-engine-2  | Receiving (merged) model from FML controller:
agent-engine-2  | ----- END -----
```

Figure 9: IAKM FML service demonstration.

2.2.2 WEB-BASED IDE FOR HYPERMEDIA MULTI-AGENT SYSTEMS

The Web-based IDE for Hypermedia MAS enables Domain Experts to configure Hypermedia MAS by integrating Agent-oriented Programming with a no-code environment that permits programming Agents' procedural knowledge as well as the organizational setup of a Hypermedia MAS. This component is described in detail in Section 3.2 of the IntellIoT Deliverable D3.5 and we therefore only give a brief account of the capabilities of the currently implemented system.

Using the Web-based IDE for Hypermedia MAS, Domain Experts can select an operating workspace from the workspaces provided by the Hypermedia MAS Infrastructure, and thereby gain access to the artifacts in that workspace via W3C WoT TDs. Based on these machine-readable descriptions of the artifact interfaces, the Web-based IDE synthesizes Blockly no-code programming blocks that are tied to programming abstractions in the Jason agent programming language. By combining these blocks, Domain Experts may configure Agents to consume information from artifacts, reason upon it, and issue commands to artifacts based on their reasoning. Agents may also be configured to communicate with each other. The Web-based IDE can also create organizations by using graphical abstractions to represent the structural, functional, and normative components of an organization. The Web-based IDE can also attribute a role to an agent running on the Hypermedia MAS Infrastructure. Using the current version of the Web-based IDE for Hypermedia MAS, Domain Experts may program multiple Agents simultaneously using the blocks abstraction (including authentication workflows when using interaction affordances of devices and services, which is currently limited to key-based authentication methods [Basic, API-Key]), persist and load Agent code, define and store MAS configurations, submit MAS configurations for execution, and add and remove Agents to/from a currently running MAS.

2.2.3 GOAL SPECIFICATION FRONTEND

The Goal Specification Frontend is a UC-specific component, described in Section 3.3 in Deliverable D3.5. This component enables the end user to define a goal to send it to an agent running on the Hypermedia MAS Infrastructure.

This component can be integrated within an existing interface for end users. For example, in UC1, the Goal Specification Frontend was implemented by adding a button to a farming management system that was provided by one of the IntellIoT Open Call partners. Using this button, the user sends a generated goal to an agent running on the Hypermedia MAS Infrastructure whose URL is known by serializing the goal into the JSON format that the agents can parse. In UC3, the component was created from scratch since no interface for end users was provided in this use case.

2.2.3 INTEROPERABILITY BOX

The Interoperability Box (IO Box) has been described in Deliverable D3.5. As a component in the IntellIoT framework, it aims to interface heterogeneous devices that have either their own private semantics or are unable (from a computational point of view as these are limited resource devices) to communicate with the Hypermedia MAS and the related communication mechanisms it uses. The Interoperability Box is deployed in an intermediate node (assuming the role of a gateway) bridging IoT devices and the IoT application.

The gateway has the necessary physical interfaces to communicate with those devices and the software stack necessary to bridge these devices with the IoT application. The software stack has two main components. The first one involves the device-specific low-level drivers that represent how the local system views and accesses the device. The second component (called IO Box) provides a translation between this local view and a higher level W3C WoT TD description that the remote IoT application may use to communicate with the device. In this way, devices with protocols that do not yet have a W3C WoT TD binding become available. Once the device description has been advertised to the HyperMAS, the communication API is defined, and the IO Box implements the specified HTTP REST API to enable the handling of the device from the IoT application. Figure 10 demonstrates this architecture.

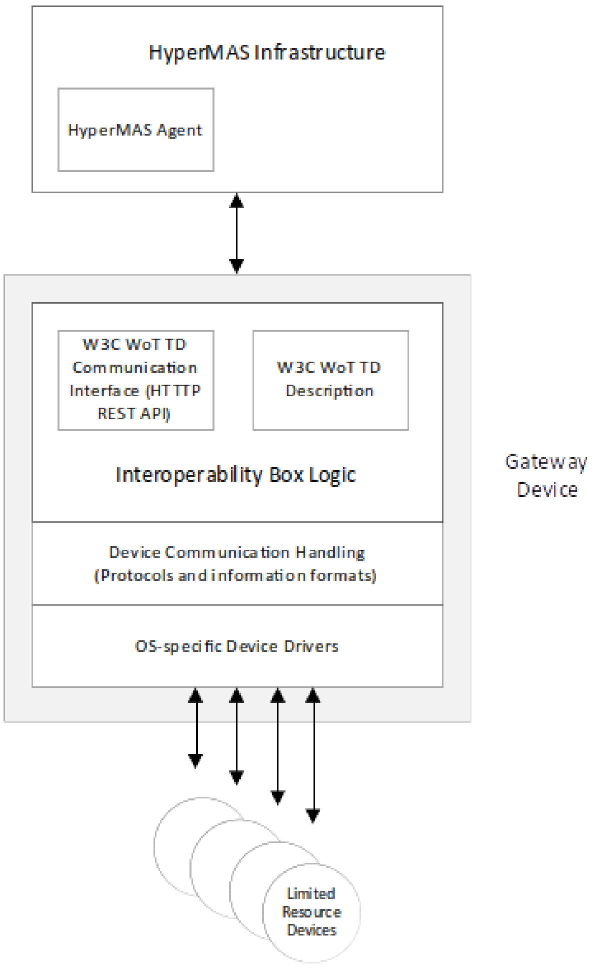

HyperMAS Infrastructure
HyperMAS Agent
W3C WoT TD Communication Interface (HTTP REST API)
W3C WoT TD Description
Interoperability Box Logic
Gateway Device
Device Communication Handling (Protocols and information formats)
OS-specific Device Drivers
Limited Resource Devices

Figure 10: High-level, logical architecture of Interoperability Box

The Interoperability Box requires a Linux-capable device, with proper physical interfaces with all devices that have limited resources and need its services to communicate with the IoT application. A common example of such a device is the Raspberry Pi, which is the device chosen for demonstration purposes in IntellIoT.

The latest version of the Interoperability Box can be found on the public repository at https://gitlab.eurecom.fr/intelliot/iobox .

It is developed in Python and can run in any system that supports recent Python versions. The provided functionality by the current version is communication with a MiroCard [5] over BLE (used as a demonstrator in the Manufacturing use case), and generic serial devices to facilitate the second Open Call partners. Within IntellIoT the Interoperability Box runs on a Raspberry Pi device provided by HSG.

When the Interoperability Box is executed using the Manufacturing use case configuration, it provides the following endpoints:

- /
- /devices
- /devices/ble
- /devices/ble/mirocard_testbed/temperature
- /devices/ble/mirocard_testbed/humidity

All of the above use GET requests. Other types of requests can be added when that is meaningful for the underlying device (for example when the underlying device is an actuator). The first three requests are not tied to a specific device. '/' is used as a health check endpoint. '/devices' will list the number of configured devices per interface, for example:

```
{
    "ble": 1,
    "serial": 2
}
```

And for each interface type listed above, one can also request the configured device details using for example the '/devices/ble' endpoint:

```
{
    "mirocard_testbed": {
        "mac": "60:77:71:57:17:c9",
        "module": "ble",
        "name": "testbed",
        "type": "mirocard"
    }
}
```

In the MiroCard case, we can check the values measured but cannot send commands to it. The Interoperability Box will always store the latest MiroCard measurement and provide that as a response. Measurement value is always accompanied by the measurement timestamp so the requesting party can take more meaningful actions about the value it received. For example, a GET request at the '/devices/ble/mirocard_testbed/temperature' endpoint, yields a response like the following:

```
{
    "temperature": 21.58,
    "timestamp": 1678460287
}
```

2.2.4 GLOBAL AND LOCAL AI

In collaboration, global and local AI establish a distributed AI system, implementing Federated Learning. The main tasks of the Global AI Component are model aggregation and model evaluation. The main two roles of the Local AI are (I) to train the local AI model using training over local data and communicating with the Global AI, and (II) to make local inference utilizing the knowledge acquired throughout the network. The tasks defined under inference is use case-specific, which are described separately in Chapter 3. In that section, we also briefly describe the integration of the Global AI with the Local AI Component and the IAKM Components.

2.2.4.1 RESOURCE AWARE SCHEDULER

In UC1 and UC3, the Global AI Component perform an additional function of resource-aware scheduling: Depending on the available computation and communication resources, it determines which Local AI components need to be scheduled to carry out local model training for a given training iteration. Here, the objective is to ensure that the accuracy of the model is improved over scheduling a subset of Local AI components.

Input: For a given training iteration, the size of the local datasets, the estimates of local computing power available at the devices running Local AI components, the estimated channels between the local devices and Global AI component, and available communication resources (resource blocks) are used as the input for the Resource Aware Scheduler.

Output: A Boolean vector corresponding to a subset of Local AI Components that needs to carryout local model training and model sharing during the selected training iteration.

Test Environment: This is tested over a simulated, which is a Python development environment running on a Windows workstation. Here, the local computing power availability is modelled as a Poisson arrival process [3] and the dynamic wireless channel are generated according to Rayleigh distribution [4].

2.2.4.2 MODEL AGGREGATOR

The role of Model Aggregator is to update the Global AI Model using the inputs received from the Local AI components.

Input: For each training iteration, the size of the local datasets along the Local AI Models are used as the input.

Output: The Local AI Models parameters are averaged in terms of dataset size as their weights to generate the parameters of the Global AI Model, which is the output of the Model Aggregator.

Test Environment: This is implemented and tested over simulated environments that typically run on development PCs, prior to integration.

2.2.4.3 MODEL EVALUATION

As part of the federated learning process, new AI models with the aggregated updates from the Local AI need to be evaluated against the base or current Global AI model. This evaluation is performed by computing an appropriate model performance metric (accuracy, mean squared error, amongst others) on a curated dataset available at the Global AI. The evaluation will determine if the new AI model should be sent to the Local AI and replace the current Global AI model, also considering the personalization of Local AI models in certain use cases.

Input: A base or current Global AI model and a new, updated AI model as well as a curated dataset that serves as a reference dataset to ensure that model performance is preserved.

Output: The model performance metric from both models described in INPUT, computed on the curated dataset. A researcher will determine if there is an improvement, in which case there is a new Global AI model. The researcher will also determine if the new model should be sent to the Local AI. We will as well explore local customizations of the model and evaluate approaches towards out of distribution detection to automatically reject updates that are based on erroneous or low-quality data and ground truth. We will explore if this method can reject updates based on data collected for instance with wrongly positioned devices or other type of technical errors.

In UC2, considering that in production the models will be subject to strict regulatory requirements, we separate the training and validation processes from the deployment of new models in production and their use for inference. Similarly, the exploration of model customization to local data, without updating the global model, will be explored as a research concept but not automatically rolled out from the training and validation workflow into the production/model inference workflow.

Test Environment: As part of the Global AI, deployed on use case specific, accessible location, like Amazon cloud, or 5G MEC.

2.2.4.4 INTEGRATION WITH THE LOCAL AI

As described in Deliverable D3.6, the federated learning framework used in IntellIoT distinguishes between a Global AI and a Local AI. The Global AI will receive model updates sent by the Local AI through a protected and secure API and a HTTP over TLS connection on a 5G network. As was described in Deliverable D2.6, pending model aggregation and evaluation, updated models with improved performance will be sent to the Local AI through the same API and connections. Once the Global AI determines that an improved model is available, it will push it to the Local AI as either an update to the model itself or as an update to the whole Local AI.

It is envisioned that the training as part of the federated learning process at the Local AI will be initiated from the Global AI, allowing it to pass to the Local AI various training configuration parameters. It will be left to the (integrated) local AI, to decide to participate in a training cycle, depending on constraints of the Edge device, like battery level, connection to metered network, etc.

2.2.4.5 INTEGRATION WITH THE IAKM

For UC1 and UC3, the AI components will communicate with the IAKM, as described in Deliverables 2.2 and 2.6 and in this deliverable Section 4.2.2.

2.3 *Human in the Loop Enablers*

2.3.1 HIL SERVICE

The HIL Service is an IntellIoT specific service which mediates between HyperMAS and HIL application. Figure 11 gives an example how the HIL Service, HyperMAS, Edge Orchestrator, Robot Controller, and multiple instances of the HIL application could interact on a help request raised by an AI in the manufacturing use case. Please note that these interactions are based on the actual interface definitions in the TDs that are resolved at run time. Figure 11 provides examples of what requests might be sent by the components, which are defined by the component vendors and described in Swagger specifications. The HIL service itself solely exposes the "help request" interface, which is used by HyperMAS in the example in Figure 11 and depicted in Figure 12. See https://gitlab.eurecom.fr/intelliot-project/human-in-the-loop/hil-service/-/blob/master/hil-service.yaml for details. All other interfaces referred to in Figure 11 are exposed by the components surrounding the HIL Service and defined in the corresponding component descriptions.

One special thing about the HIL service is the MQTT request which, via a MQTT broker, goes out to all available operators, i.e., directly to the Hololenses or the operators. The operator who accepts the request first gets the job, later acceptances from other operators will be refused. The HIL service will now establish the realtime-connection between the operator and the HIL application.

To test the HIL Service, we are using an HTTP client, e.g., *Postman*, to trigger the help request interface. For the interaction with other services, we have been using simple stubs, which can be generated from the swagger descriptions or W3C WoT TDs and filled with simple code to provide the expected responses. These stubs are by and by replaced by the real services where available.

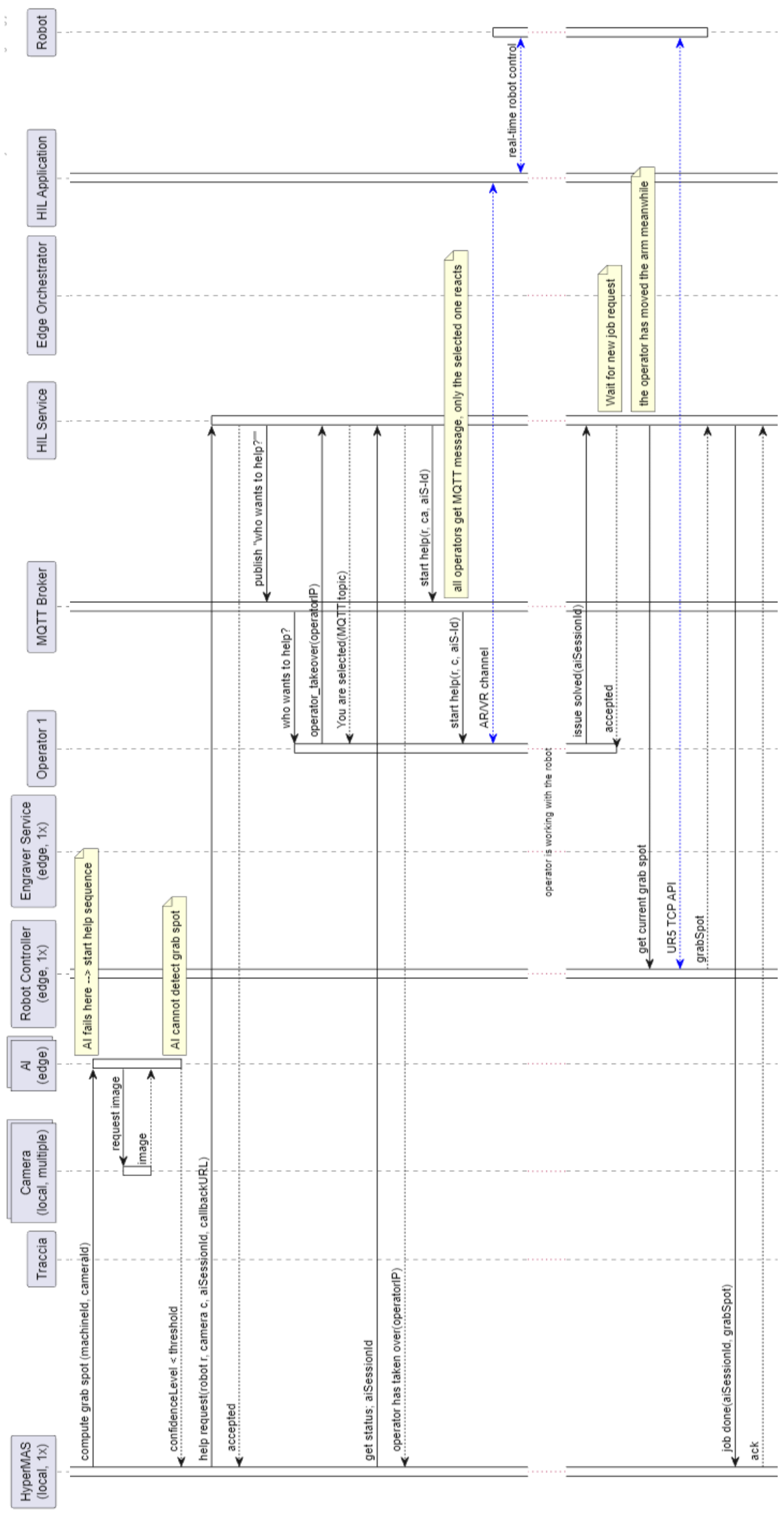
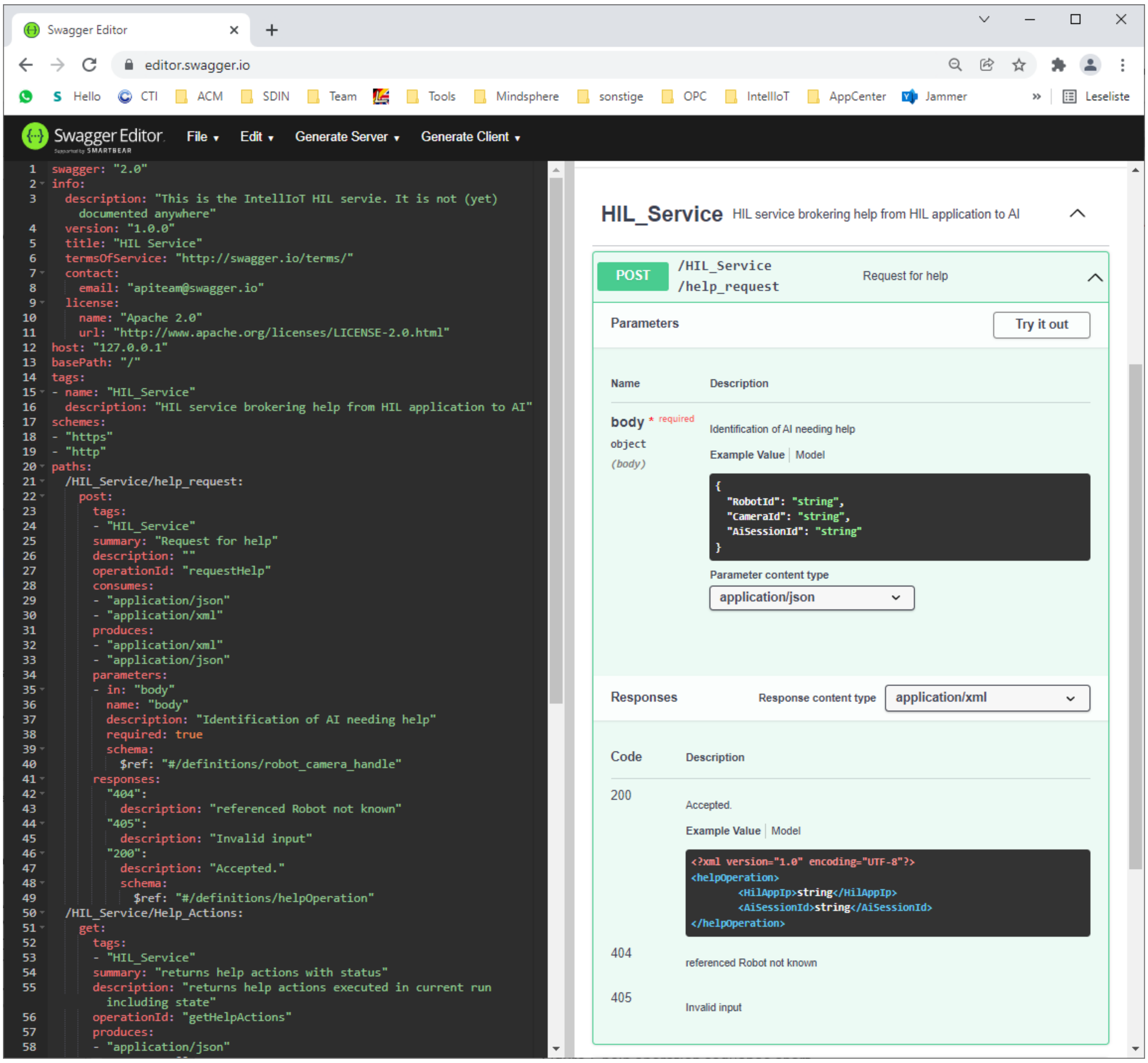

HyperMAS (local, 1x)
Traccia
Camera (local, multiple)
AI (edge)
Robot Controller (edge, 1x)
Engraver Service (edge, 1x)
Operator 1
MQTT Broker
HIL Service
Edge Orchestrator
HIL Application
Robot
compute grab spot (machineId, cameraId)
AI fails here --> start help sequence
request image
image
AI cannot detect grab spot
confidenceLevel < threshold
help request(robot r, camera c, aiSessionId, callbackURL)
accepted
publish "who wants to help?"
who wants to help?
operator_takeover(operatorIP)
You are selected(MQTT topic)
get status: aiSessionId
operator has taken over(operatorIP)
start help(r, ca, aiS-Id)
all operators get MQTT message, only the selected one reacts
start help(r, c, aiS-Id)
AR/VR channel
real-time robot control
operator is working with the robot
issue solved(aiSessionId)
accepted
Wait for new job request
the operator has moved the arm meanwhile
get current grab spot
UR5 TCP API
grabSpot
job done(aiSessionId, grabSpot)
ack

Figure 11: HIL support operation sequence chart.



```
swagger: "2.0"
info:
  description: "This is the IntellIoT HIL servie. It is not (yet)
    documented anywhere"
  version: "1.0.0"
  title: "HIL Service"
  termsOfService: "http://swagger.io/terms/"
  contact:
    email: "apiteam@swagger.io"
  license:
    name: "Apache 2.0"
    url: "http://www.apache.org/licenses/LICENSE-2.0.html"
host: "127.0.0.1"
basePath: "/"
tags:
- name: "HIL_Service"
  description: "HIL service brokering help from HIL application to AI"
schemes:
- "https"
- "http"
paths:
  /HIL_Service/help_request:
    post:
      tags:
      - "HIL_Service"
      summary: "Request for help"
      description: ""
      operationId: "requestHelp"
      consumes:
      - "application/json"
      - "application/xml"
      produces:
      - "application/xml"
      - "application/json"
      parameters:
      - in: "body"
        name: "body"
        description: "Identification of AI needing help"
        required: true
        schema:
          $ref: "#/definitions/robot_camera_handle"
      responses:
        "404":
          description: "referenced Robot not known"
        "405":
          description: "Invalid input"
        "200":
          description: "Accepted."
          schema:
            $ref: "#/definitions/helpOperation"
  /HIL_Service/Help_Actions:
    get:
      tags:
      - "HIL_Service"
      summary: "returns help actions with status"
      description: "returns help actions executed in current run
        including state"
      operationId: "getHelpActions"
      produces:
      - "application/json"
```

HIL_Service HIL service brokering help from HIL application to AI

POST /HIL_Service/help_request — Request for help

Parameters — Try it out

Name	Description
body * required object (body)	Identification of AI needing help Example Value | Model

```
{
  "RobotId": "string",
  "CameraId": "string",
  "AiSessionId": "string"
}
```

Parameter content type: application/json

Responses — Response content type: application/xml

Code	Description
200	Accepted. Example Value | Model
404	referenced Robot not known
405	Invalid input

```
<?xml version="1.0" encoding="UTF-8"?>
<helpOperation>
	<HilAppIp>string</HilAppIp>
	<AiSessionId>string</AiSessionId>
</helpOperation>
```

Figure 12: HIL service HTTP interface

2.3.2 HIL APPLICATION

The HIL Application allows humans to play a direct and interventive role during the pilot deployments of UC1 and UC3. In the second cycle of this deliverable, a specific HIL Application was developed for each of these use cases. The core functionalities of the HIL Application are generally consistent across the two, with some differences in their current extents of HIL Service integrations (see below) and their input/output sources (e.g., HoloLens 2 versus Oculus Quest 2). The HIL Application is based in Unity3D, allowing it to provide immersive two- and three-dimensional visualizations. These visualizations allow humans to view important state information about IntellIoT IoT devices, such as the robot in UC3 or the tractor in UC1, for example. Importantly, the HIL Application has integrations with other IntellIoT module components which allow humans to become "in the loop" and control dedicated IoT devices directly. The HIL Application is a Windows application and is hosted locally on a PC/laptop, rather than instantiated on an edge or cloud-based virtual machine. While detailed information on the HIL Application can be found in D3.7 ("Human in the loop in Intelligent IoT environments"), several important details will be mentioned here regarding integrations as they pertain to deployment implementations. For the HIL Application in both use cases, the application is enabled with HOLO's Interactive Streaming for Augmented Reality (ISAR) plugin technology. The ISAR-enabled HIL Application as a result is streamed to client augmented and virtual reality (AR/VR) devices. ISAR streaming of the HIL Application is mediated by WebRTC and allows the streaming virtual content, such as e.g., the digital twin of the UC3 robot or the tractor camera information from UC1, respectively. ISAR streaming is not unidirectional however, and here ISAR streams also permit the return of sensor data from the AR/VR devices. Sensor data from these devices includes important event and movement information which is used when the human-in-the-loop attempts their remote operation of their respective IoT devices. Facilitating this return of sensor data to the HIL Application is a dedicated ISAR client application on each AR/VR device. In UC1, the Oculus Quest 2 VR and its associated VR controllers are used, whereas UC3 utilizes the HoloLens 2 and the Holo-Stylus pen for this purpose. Information from these controllers and the Holo-Stylus used during interaction with the digital twin or visualized video stream is used to remotely operate the tractor and robot. To make informed and situationally relevant operations, each HIL Application receives video streams and other state information about the tractor or robot, which is then visualized to the operator. In UC3, video streams from the robot are sent via WebRTC to the HIL Application. In UC1, video streams of the tractor are converted from RTP to WebRTC. This conversion previously occurred in the NVIDIA Jetson TTC Motion but now occurs within the HIL Application itself. UC3 has additional information, that is consumed by the HIL Application, which is the joint position of the robot which is sent through TCP/IP. Control of the tractor and robot occur through the sending of motion commands via TCP/IP or API for the robot and tractor, respectively. These commands end up at the controller of the respective device. Additionally, these commands are also sent to a Data Repository where local AI can utilize it for future learning. Figure 12 depicts the HIL application and its general interfaces.

In UC2, Human-In-the-Loop (HIL) plays a crucial role in ensuring the effectiveness and safety of the IntellIoT system for remote monitoring and caring of cardiovascular patients. The system utilizes Machine Learning (ML) algorithms, to predict clinically relevant parameters based on sensor measurements and health-related data. The HIL Application in UC2 may not directly involve augmented or virtual reality (AR/VR) devices as in UC1 and UC3, however, it still plays a crucial role in the healthcare context. The Vida24© mobile app collects important data about the patient, reflecting their health status and progress regarding the assigned care plan. This data is then used by physicians to make informed decisions about the patient's condition, care plan adjustments, and interventions.

The HIL approach in UC2 serves two main purposes. First, it serves as a safeguard for patients, protecting them from potentially harmful actions. No AI system is perfect, and there's always a possibility of erroneous predictions or recommendations. To mitigate this risk, every recommendation generated by the Local AI can be examined by a clinician before being sent to the patient. This additional layer of human oversight helps protect patients from potential harm. Secondly, it generates information for AI model training and personalization. Clinicians play a significant role in collecting and annotating data for AI model training. Their input gives meaning to the sensor-generated data, which is then used to train the base model in the Global AI component and personalize the ML models in the Local AI setting. Apart from automatic data annotation, clinicians also provide additional annotations when inspecting the outputs of trained ML models. They mark certain predictions as unreliable, and the Local AI utilizes this information to re-train the model, giving special attention to these erroneous examples. This process improves the performance of both the Local and Global AI, as the latter aggregates the models from all patients.

In summary, the Human-in-the-Loop approach in UC2 plays a pivotal role in ensuring patient safety and enhancing the performance and personalization of the AI models used for remote monitoring of cardiovascular patients. By incorporating clinician expertise into the system, the IntellIoT framework can deliver more accurate and reliable healthcare solutions.

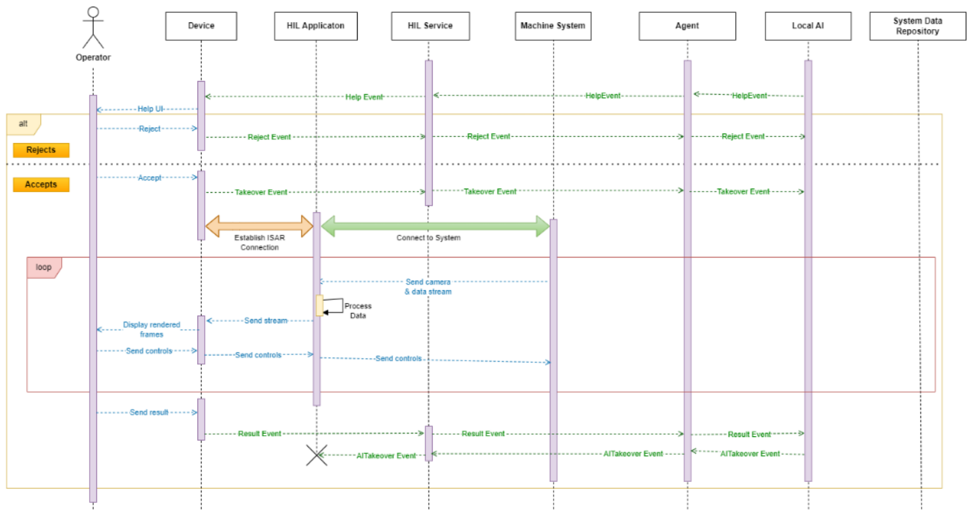
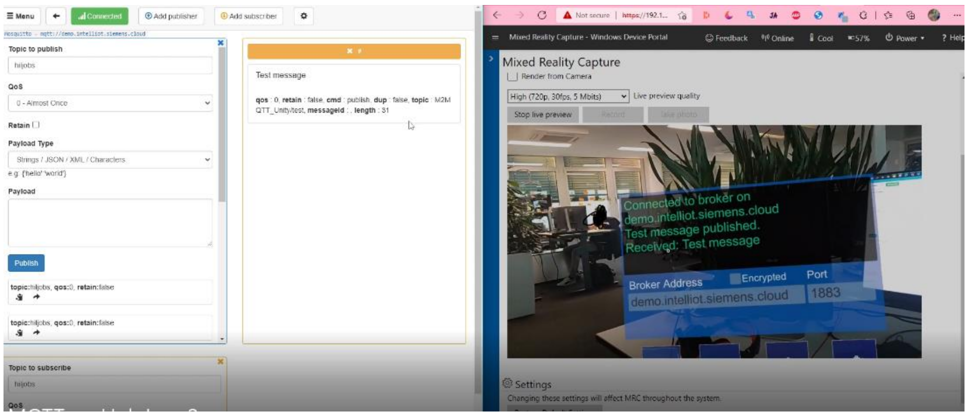


Figure 13: Human in the Loop Interfaces.

The role of the HIL Service is to broker a connection between the human-in-the-loop and the AI of the IoT devices. In doing so, the HIL Service allows the human to be notified in the event that the AI cannot take action and needs a human to step in to remotely operate. A connection between the MQTT of the HIL Service exists and human-in-the-loop components exists for UC3 and UC1 directly. Through a publish-subscribe method, the MQTT Message Broker sends messages when the AI throws an incident and needs help. For example, a dedicated client application on the HoloLens 2 receives these messages (Figure 13). When an operator wearing the HoloLens 2 receives this message, they can confirm via a user interface interaction to signal to the HIL Service that they are able to step in and assist remotely. Afterwards, through an API call, connectivity protocol information (e.g., IP address) are provided from the HIL Service, which allows the HIL Application to connect with the robot controller. Currently, such a mechanism is in place for UC1 and UC3 on the Oculus Quest 2 and HoloLens 2, respectively.



Test message

qos : 0, retain : false, cmd : publish, dup : false, topic : M2MQTT_Unity/test, messageId : , length : 31

Connected to broker on demo.intelliot.siemens.cloud
Test message published.
Received: Test message

Broker Address: demo.intelliot.siemens.cloud
Encrypted
Port: 1883

Figure 14: Unity3D application for HoloLens 2 and MQTT communication.

As mentioned above, the ISAR technology used here operates using the WebRTC protocol. In doing so, it establishes a connection between the HIL Application ("Server") and the AR/VR devices ("Client"). This connection is a peer-to-peer connection. The streaming of the entire HIL Application to the Client devices is critical for ensuring that the human-in-the-loop can accurately and appropriately operate the IoT devices when needed (resulting as mentioned above from the HIL Service). Low latency and fast streaming of the video streams from the tractor and robot are required to ensure this. Furthermore, interaction in AR/VR space must be free of lag.

Maintaining these parameters during HIL operations is permitted by the high-quality streaming between the client and server devices mediated by ISAR. The specific aspects of ISAR processing and connectivity have been described in the first version (see Figure 14, 15) of this deliverable and have remained unchanged over the second cycle here.

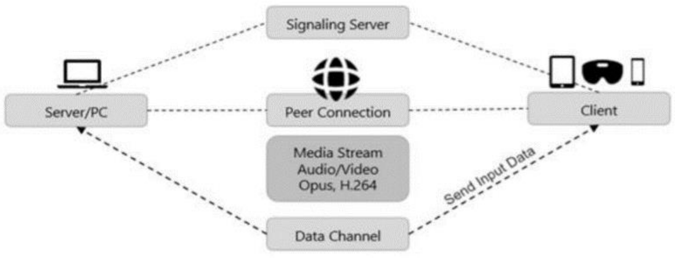


Figure 15: ISAR - Server to Client interface.

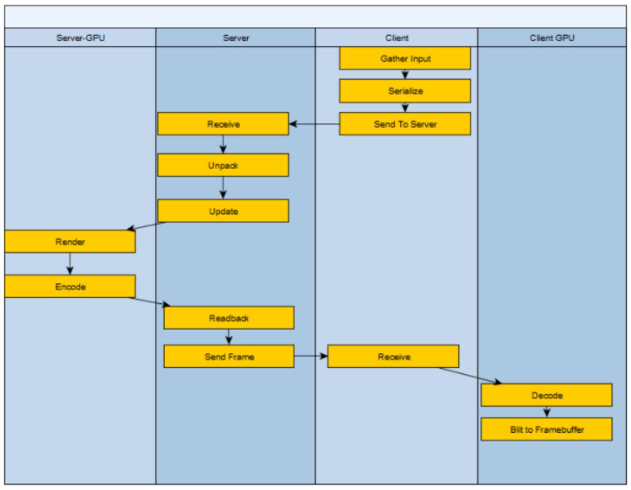


Figure 16: ISAR Interaction Diagram.

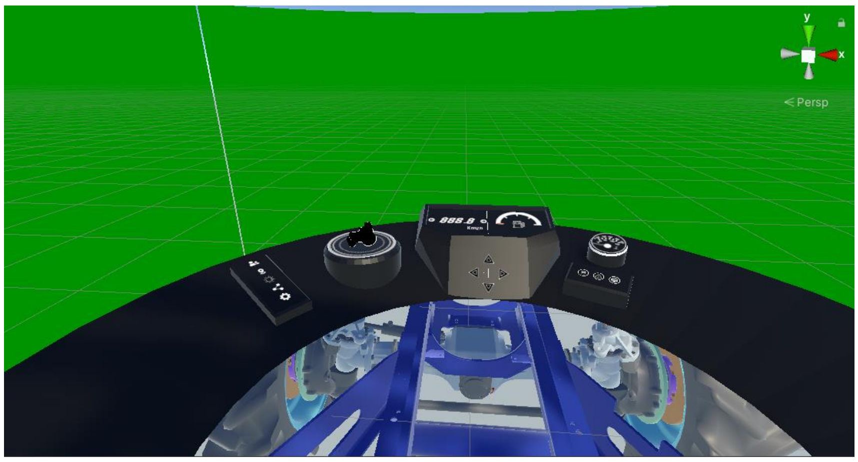


Figure 17 : HIL Application: Machine Data.

2.3.3 END USER GOAL SPECIFICATION INTERFACES

As part of *IntellIoT's* T3.1 and specific to UC1 (Agriculture) and UC3 (Manufacturing), we are developing and integrating graphical interfaces for end users. These components permit end users (i.e., farmers and customers) to specify the goal to be achieved by the IntellIoT system (e.g., "Fertilize Field 5"; "Engrave the text 'IntellIoT' in a wooden work piece") and transmit specified goals for processing to an Agent in the Hypermedia MAS Infrastructure that has been configured using the no-code configuration environment.

The end user goal specification interfaces are simple Web applications that exchange JSON-LD payloads with the Hypermedia MAS Infrastructure according to agreed-upon schemas. For UC1, the payload contains the ID and entry coordinates of the field as well as a description of the desired process to be executed, i.e., the implement to use (e.g., a "Sprayer"), the specification of the lines to move along the field (e.g., 30 lines that are spaced 5m apart) and, optionally, further parameters for the implement (e.g., flow rate). For UC3, the payload contains the desired work piece material and process (e.g., "Laser") along with a list of drawings (each with properties "Text", "position", and "size").

2.3.4 REMOTE OPERATOR DEVICES

Enabling the remote operation for the human-in-the-loop are specific operating devices. In both UC1 and UC3, a powerful edge device is used to host the HIL Application. The edge device used here is a personal computer/laptop, with specifications that permit high quality VR/AR streaming and visualization. The following specifications are the minimum requirements of the laptop/edge device for this purpose:

- Operating System: Windows 10 (10.0..17763 Build) or Windows 11
- Memory: 16 GB
- CPU: Intel i5 8th Gen. 6 Cores
- GPU: NVIDIA GTX 1070Ti
- Storage: Solid State Drive

These features are critical for rendering processes to be possible, as described and graphically visualized in 2.3.2.

The network connectivity between the HIL Application, AR/VR devices, and other IntellIoT components is envisioned to be 5G. Currently, the HoloLens 2 and associated HIL Application support 5G connectivity for this purpose. In the next cycle of this task, the Oculus Quest 2 in UC1 will be investigated for technical solutions which enable 5G connectivity. Irrespective of network connectivity mediums (e.g., Wi-Fi versus 5G), open port requirements are still present. For the HIL Application to function (through ISAR streaming) a Port of 9999 is still required. In addition, incoming and outgoing WebRTC streams need associated ports as well and will be used here.

The details about use case specific remote operating devices are given in Section 3.1.3 and Section 3.3.4 for UC1 and UC3, respectively.

2.4 Trust Enablers

The Trust Enablers pillar of IntellIoT, as defined in Deliverable D2.3 - "High level architecture (first version)", comprises innovative building blocks that individually, but also through their tight integration, realise the project's vision to provide security, privacy, and trust by design (see also IntellIoT's Objective 4: *"Enable security, privacy and trust by design with continuous assurance monitoring, assessment and certification as an integral part of the system, providing trustworthy integration of third party IoT devices and services"*). The high-level, logical view of the components within said pillar is provided in Figure 18.

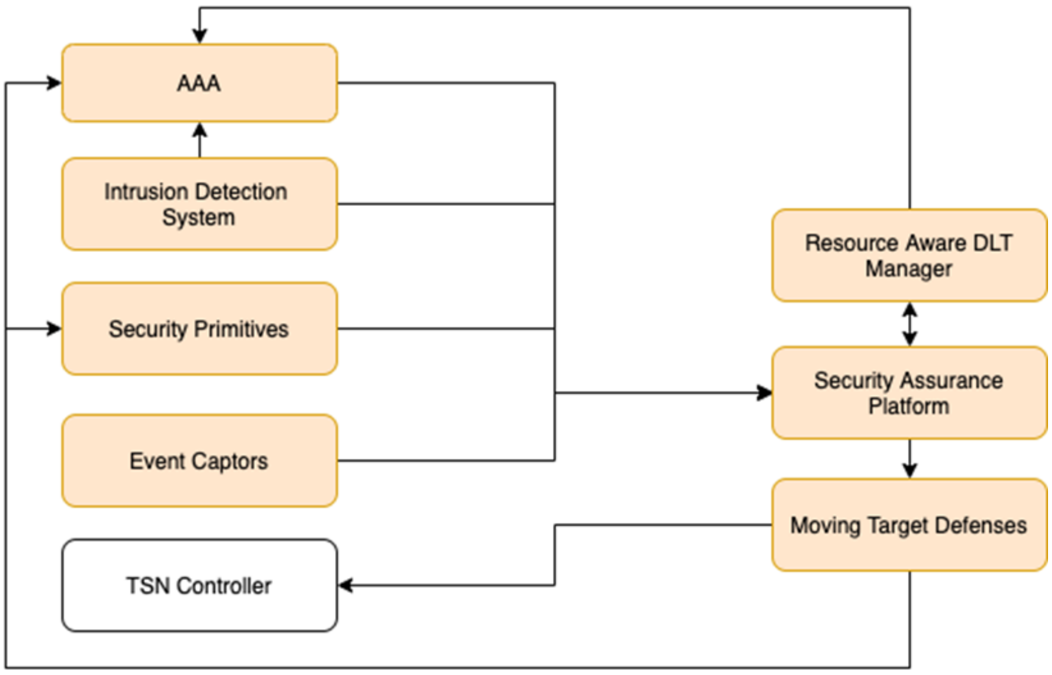

AAA
Intrusion Detection System
Security Primitives
Event Captors
TSN Controller
Resource Aware DLT Manager
Security Assurance Platform
Moving Target Defenses

Figure 18. Trust Enablers UML logical diagram.

The Trust IDS continuously monitors network traffic and analyses it for anomalies. It is a decentralised component installed on every node of the IoT system. Each instance computes a local trust value for each other node that it communicates with. When the trust value for a node drops below a threshold, it generates a warning. These warnings are left to be processed by the rest of the system to determine if an action is needed to be taken against offending nodes.

Similar to the IDS, the Event Captors (ECs) are software modules responsible for capturing and aggregating pertinent evidence from multiple sources related to the operation of individual components, as well as the overarching processes where these components are involved in, thus enabling the real-time, continuous assessment of the security posture of the IntellIoT system. ECs can be deployed across all layers of the IntellIoT architecture, from the robot arm (e.g., to monitor telemetry that may indicate abnormal activities) to device operating systems (e.g., to monitor running processes) and backend storage databases (e.g., to parse access logs or calculate uptime), due to the fact of their minimal footprint while transmitting their monitoring evidence to the Security Assurance Platform (SAP). In addition, Event Captors can collaborate within ElasticSearch data pools, leveraging their capturing and aggregating capabilities.

The SAP is central to the trustworthiness components and has its own queue in the message broker pipeline. SAP digest messages respectively from the IDS queue and EC queue while at the same time through appropriate REST APIs it can communicate with the Distributed Ledger Trustworthiness (DLT) manager.

Finally, the MTD components are responsible for prevention and mitigation actions. MTDs are divided into two distinct components: the server MTD installed on a central node, and the client MTDs installed in every node of the IoT deployment. The first has its own queue (MTD server queue) while MTD clients consume messages from the latter queue. The MTD server will generate different network configurations for the overall deployment either periodically (in random time intervals) as an attack prevention mechanism or reactively once the SAP receives an Intrusion Warning from the IDS. In the latter case, the MTD system is able to isolate offending nodes from the IoT network by updating the network configuration without propagating these updates to these specific nodes.

While details on the design and implementation of the individual components is provided in Deliverable D4.8 – "Trust mechanisms (final version)", the subsections below will provide additional details on the testing and testbed integration of the components, but also their overall integration within the Trust pillar. The latter follows the overall integration as sketched (from a logical, process, development, and deployment perspective) within D2.3, and re-iterates and expands upon the testing and integration aspects presented in D4.8.

2.4.1 OVERALL TRUST ENABLERS INTEGRATION

As detailed in Section 6 of D4.8, a key characteristic that maximises the efficacy of IntellIoT's Trust Enablers is their tight integration and continuous communication through the dedicated Trust Broker. An overview of this integration is provided in Figure 19, while for details on the integration (e.g., technologies, message structure and sample messages) we refer the reader to D4.8 (Section 6).

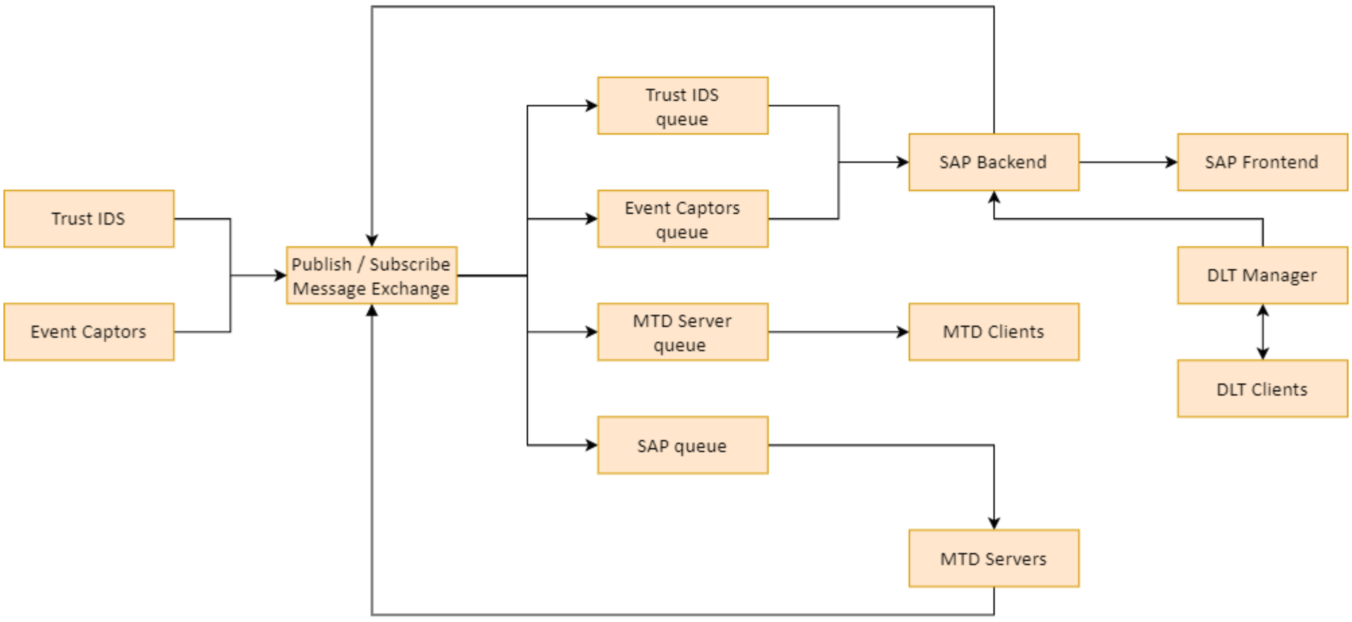


Figure 19. Overview of Trust Enablers' integration via shared Trust Broker.

From a physical (deployment) perspective, the local integration testbed used in SANL's premises is shown in Figure 20.

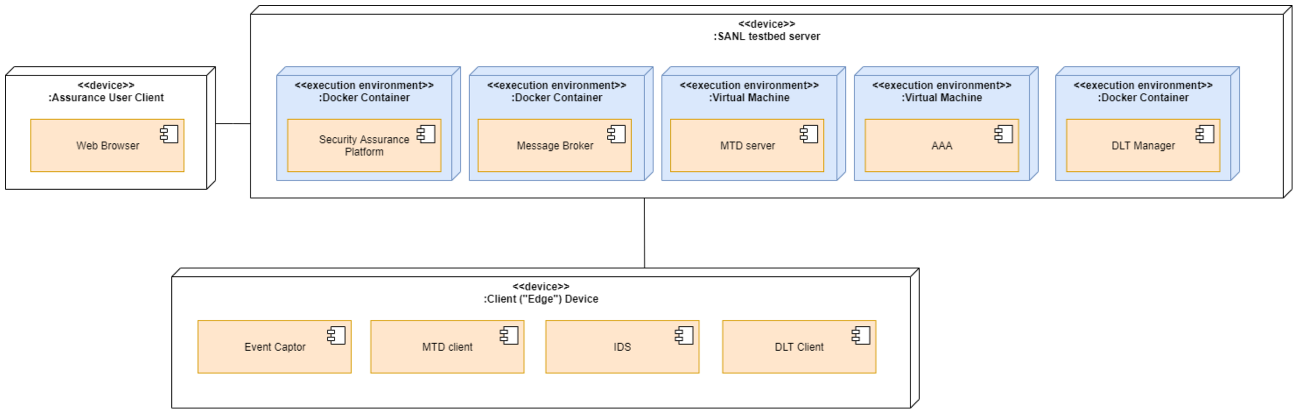


Figure 20. Deployment view of local Trust Enablers' integration testbed.

2.4.1.1 SAP-DLT INTEGRATION TESTING

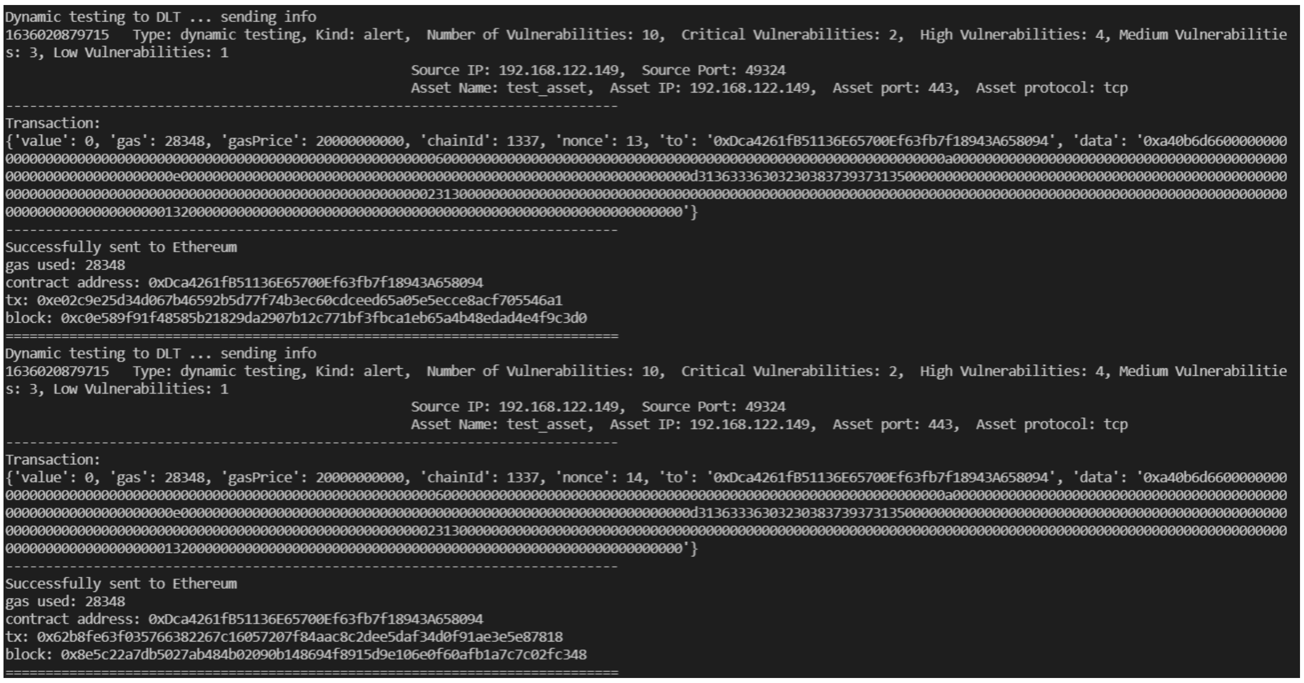
Through interaction between the SAP and the DLT Manager, based on the necessary APIs developed, evidence is recorded in the Ledger in an automated manner, and the entries (transaction and block IDs) are returned to the

SAP, to be provided to the SAP operator for verification (e.g., in the case of any audit). This provides an additional layer of trust for said evidence aggregated at the SAP.

Our testing of these components encompassed all different types of evidence that these interactions may involve, including:

(i) evidence generated internally at the SAP (e.g., vulnerability or dynamic testing assessment results);
(ii) evidence the SAP collects from monitoring and interacting with other Trust Enablers (e.g., Trust IDS alerts, or triggered MTD strategies), and;
(iii) monitoring evidence, i.e., reasoning results that are triggered from monitoring the Event Captors deployed across the various layers of IntellIoT and the protected deployment.

A screenshot of a successful message exchange between the SAP and the DLT Manager, whereby SAP Assessment Results (in this case Dynamic Testing results) are recorded in the Ledger, can be seen in Figure 21.



```
Dynamic testing to DLT ... sending info
1636020879715   Type: dynamic testing, Kind: alert,  Number of Vulnerabilities: 10,  Critical Vulnerabilities: 2,  High Vulnerabilities: 4, Medium Vulnerabilitie
s: 3, Low Vulnerabilities: 1
                                                   Source IP: 192.168.122.149,  Source Port: 49324
                                                   Asset Name: test_asset,  Asset IP: 192.168.122.149,  Asset port: 443,  Asset protocol: tcp
-----------------------------------------------------------------------------
Transaction:
{'value': 0, 'gas': 28348, 'gasPrice': 20000000000, 'chainId': 1337, 'nonce': 13, 'to': '0xDca4261fB51136E65700Ef63fb7f18943A658094', 'data': '0xa40b6d6600000000
000000000000000000000000000000000000000000000000000060000000000000000000000000000000000000000000000000000000000000a00000000000000000000000000000000000000000
0000000000000000000000e00000000000000000000000000000000000000000000000000000000000000d31363336303230383739373135000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000023130000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000132000000000000000000000000000000000000000000000000000000000000'}
-----------------------------------------------------------------------------
Successfully sent to Ethereum
gas used: 28348
contract address: 0xDca4261fB51136E65700Ef63fb7f18943A658094
tx: 0xe02c9e25d34d067b46592b5d77f74b3ec60cdceed65a05e5ecce8acf705546a1
block: 0xc0e589f91f48585b21829da2907b12c771bf3fbca1eb65a4b48edad4e4f9c3d0
=============================================================================
Dynamic testing to DLT ... sending info
1636020879715   Type: dynamic testing, Kind: alert,  Number of Vulnerabilities: 10,  Critical Vulnerabilities: 2,  High Vulnerabilities: 4, Medium Vulnerabilitie
s: 3, Low Vulnerabilities: 1
                                                   Source IP: 192.168.122.149,  Source Port: 49324
                                                   Asset Name: test_asset,  Asset IP: 192.168.122.149,  Asset port: 443,  Asset protocol: tcp
-----------------------------------------------------------------------------
Transaction:
{'value': 0, 'gas': 28348, 'gasPrice': 20000000000, 'chainId': 1337, 'nonce': 14, 'to': '0xDca4261fB51136E65700Ef63fb7f18943A658094', 'data': '0xa40b6d6600000000
000000000000000000000000000000000000000000000000000060000000000000000000000000000000000000000000000000000000000000a00000000000000000000000000000000000000000
0000000000000000000000e00000000000000000000000000000000000000000000000000000000000000d31363336303230383739373135000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000023130000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000132000000000000000000000000000000000000000000000000000000000000'}
-----------------------------------------------------------------------------
Successfully sent to Ethereum
gas used: 28348
contract address: 0xDca4261fB51136E65700Ef63fb7f18943A658094
tx: 0x62b8fe63f035766382267c16057207f84aac8c2dee5daf34d0f91ae3e5e87818
block: 0x8e5c22a7db5027ab484b02090b148694f8915d9e106e0f60afb1a7c7c02fc348
=============================================================================
```

Figure 21: Recording of SAP Assessment Results in the ledger, via interaction with DLT Manager (request & response).

Similar to the above, the integration was successfully tested for all envisioned exchanges between the two components (e.g., recording to the Ledge evidence such as Trust IDS events, Monitoring events, vulnerability assessment results).

2.4.1.2 IDS, MTD & SAP INTEGRATION VIA BROKER

As mentioned, the Trust-based IDS and MTD components communicate through a dedicated Trust broker based on RabbitMQ. In the following, a description of a demo will be presented to showcase the integration of the IDS, MTD server and several MTD clients with a RabbitMQ broker. In this example, a RabbitMQ broker is initiated along with an MTD server and two MTD clients. Another node carrying an IDS component is also in the network. Upon completing the initialization and node registration phases, the node with the IDS detects that a node's trust falls below a certain threshold, and this triggers a warning message to the broker. This warning is relayed to the MTD server that isolates the offending node by sending a new network configuration message to the healthy node. This new configuration is not relayed to the offending node and therefore this node is unable to communicate with other nodes (effectively it is isolated from everyone else).

2.4.1.3 TESTING RESULTS

To realize this demo, since all software components are dockerized, we employ docker-compose and bring up four nodes:

1. Node 1 with IP address 192.168.176.2, where a RabbitMQ broker is hosted
2. Node 2 with IP address 192.168.176.3, where the MTD server runs
3. Node 3 with IP address 192.168.176.4, where a Trust IDS instance is running along with an MTD client
4. Node 4 with IP address 192.168.176.5 has the same configuration as Node 3 plus a network traffic generator

IDS monitors network traffic. To simulate a misbehaving node, a traffic generator is executed on Node 4. IDS is configured with a suitable packet rate threshold. Every time this threshold is exceeded, trust is decreased for the misbehaving node. When trust goes below 25/100, a warning is triggered.

During the demo, Node 4 starts generating traffic towards Node 3. IDS instance in Node 3 will trigger a warning message to the broker concerning Node 4 as soon as its trust value falls below the 25/100 threshold. Then the broker relays a new network configuration to Node 3 in order to drop Node 4. Part of the logs during the demo follows.

IDS in Node 3 lowers its trust toward Node 4:

```
PacketRateFilter window timeout

192.168.176.5, trust: 23

Send warning to broker
```

The broker delivers the warning to MTD server. The MTD server drops Node 4:

```
Received warning:
  {
    "Type":"error",
    "Action":"drop",
    "Body":"client2_mtd",
    "Time":{
      "seconds":1639752428,
      "nanos":612924426
    }
  }
dropping mtd.config.client2_mt
```

This triggers a network configuration change that is only delivered to Node 3:

```
mtd.config.client_mtd -> &{
  CIDR:10.0.1.1/24
  Port:30000
  Protoc:udp4
  CipherKey:38336737335274704...
  CipherType:4
  ExtIP:192.168.176.4
  LocIP:10.0.1.1
  RoutesToAdd:map[10.0.1.1:192.168.176.4]
  RoutesToRemove:[10.0.0.1]
}
```

And now Node 4 is successfully isolated.

2.4.2 SECURITY ASSURANCE PLATFORM

As detailed in D4.8, the Security Assurance Platform (SAP) is an integrated framework of models, processes, and tools to enable the continuous assurance and certification of the security properties of assets across the IntellIoT infrastructure.

The initial (Cycle 1) version of SAP has been deployed locally in SANL's testbed, along with a set of credentials, an "IntellIoT" organization and associated project needed to enable testing the correct operation of the different capabilities that said platform will have to support (as a standalone component) for the first version of the IntellIoT integrated platform. It should be noted that the same testbed was used for the initial integration (and testing of said integration) with the other Trust Enablers, as detailed in 2.4.1 above.

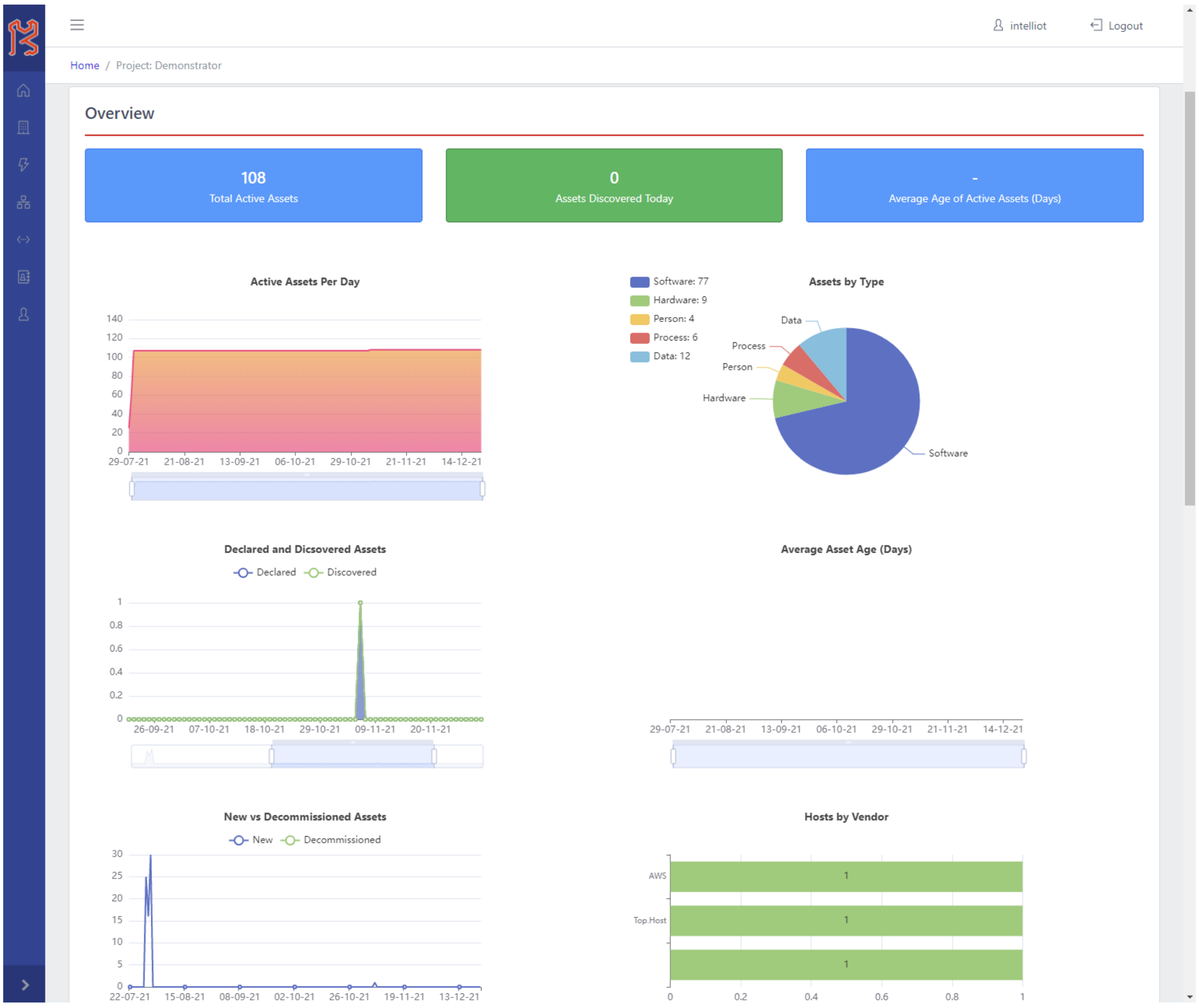
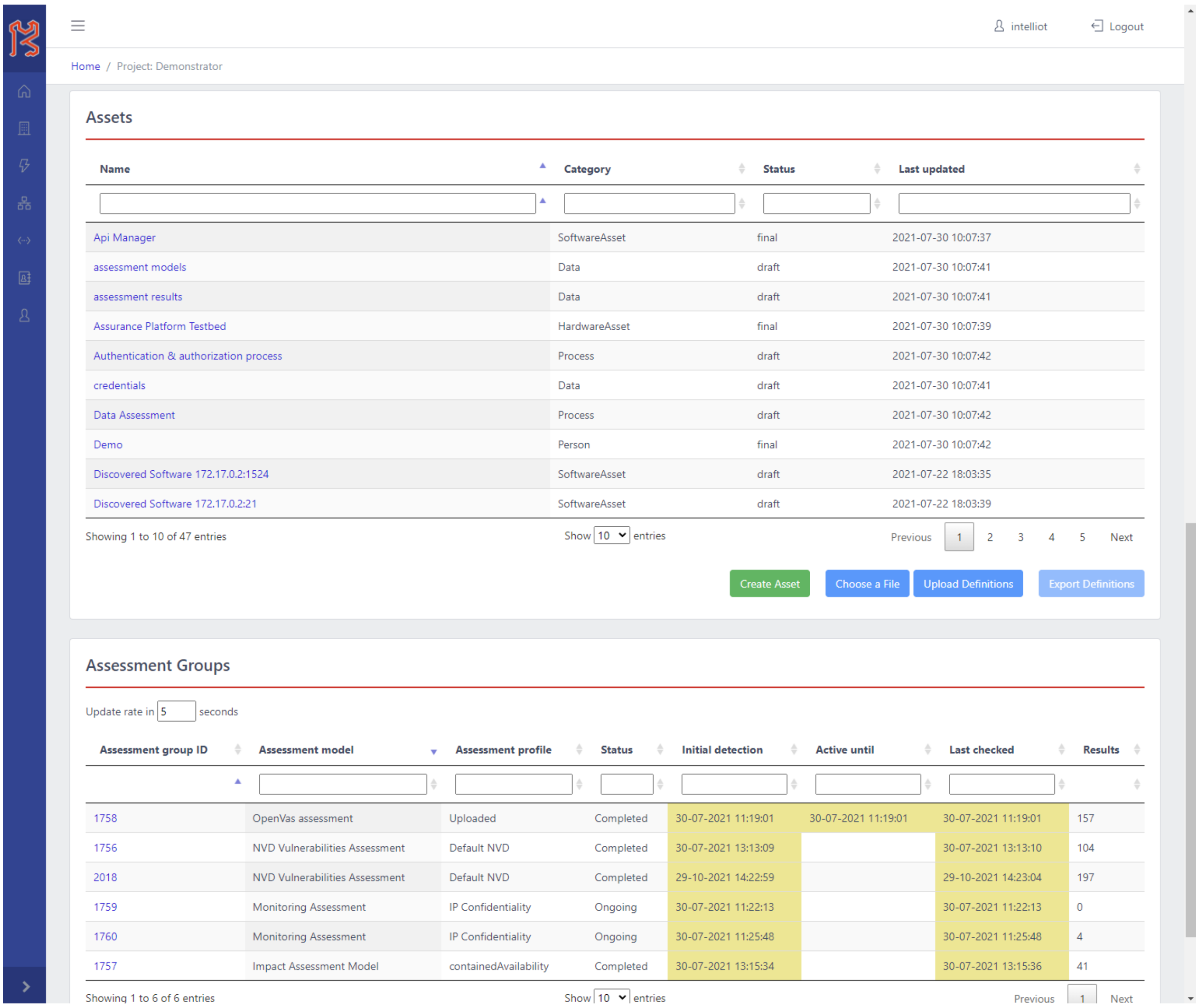


Figure 22. Overall view of "Demonstrator" testing project and related KPIs within the SAP front end.

Within the "Demonstrator" project an overview of the assets, assessment results, and other key performance indicators are correctly shown (see various KPIs in Figure 22 and the list of assets and assessments carried out during testing in Figure 23).

A first step in the use of SAP is the definition of the IntellIoT assurance model (see Section 2.3 of D4.8). For testing purposes, a corresponding testing model was created, including assets of all types that reflect the actual testbed environment.



Assets

Name	Category	Status	Last updated
Api Manager	SoftwareAsset	final	2021-07-30 10:07:37
assessment models	Data	draft	2021-07-30 10:07:41
assessment results	Data	draft	2021-07-30 10:07:41
Assurance Platform Testbed	HardwareAsset	final	2021-07-30 10:07:39
Authentication & authorization process	Process	draft	2021-07-30 10:07:42
credentials	Data	draft	2021-07-30 10:07:41
Data Assessment	Process	draft	2021-07-30 10:07:42
Demo	Person	final	2021-07-30 10:07:42
Discovered Software 172.17.0.2:1524	SoftwareAsset	draft	2021-07-22 18:03:35
Discovered Software 172.17.0.2:21	SoftwareAsset	draft	2021-07-22 18:03:39

Showing 1 to 10 of 47 entries Show 10 entries Previous 1 2 3 4 5 Next

Assessment Groups

Update rate in 5 seconds

Assessment group ID	Assessment model	Assessment profile	Status	Initial detection	Active until	Last checked	Results
1758	OpenVas assessment	Uploaded	Completed	30-07-2021 11:19:01	30-07-2021 11:19:01	30-07-2021 11:19:01	157
1756	NVD Vulnerabilities Assessment	Default NVD	Completed	30-07-2021 13:13:09		30-07-2021 13:13:10	104
2018	NVD Vulnerabilities Assessment	Default NVD	Completed	29-10-2021 14:22:59		29-10-2021 14:23:04	197
1759	Monitoring Assessment	IP Confidentiality	Ongoing	30-07-2021 11:22:13		30-07-2021 11:22:13	0
1760	Monitoring Assessment	IP Confidentiality	Ongoing	30-07-2021 11:25:48		30-07-2021 11:25:48	4
1757	Impact Assessment Model	containedAvailability	Completed	30-07-2021 13:15:34		30-07-2021 13:15:36	41

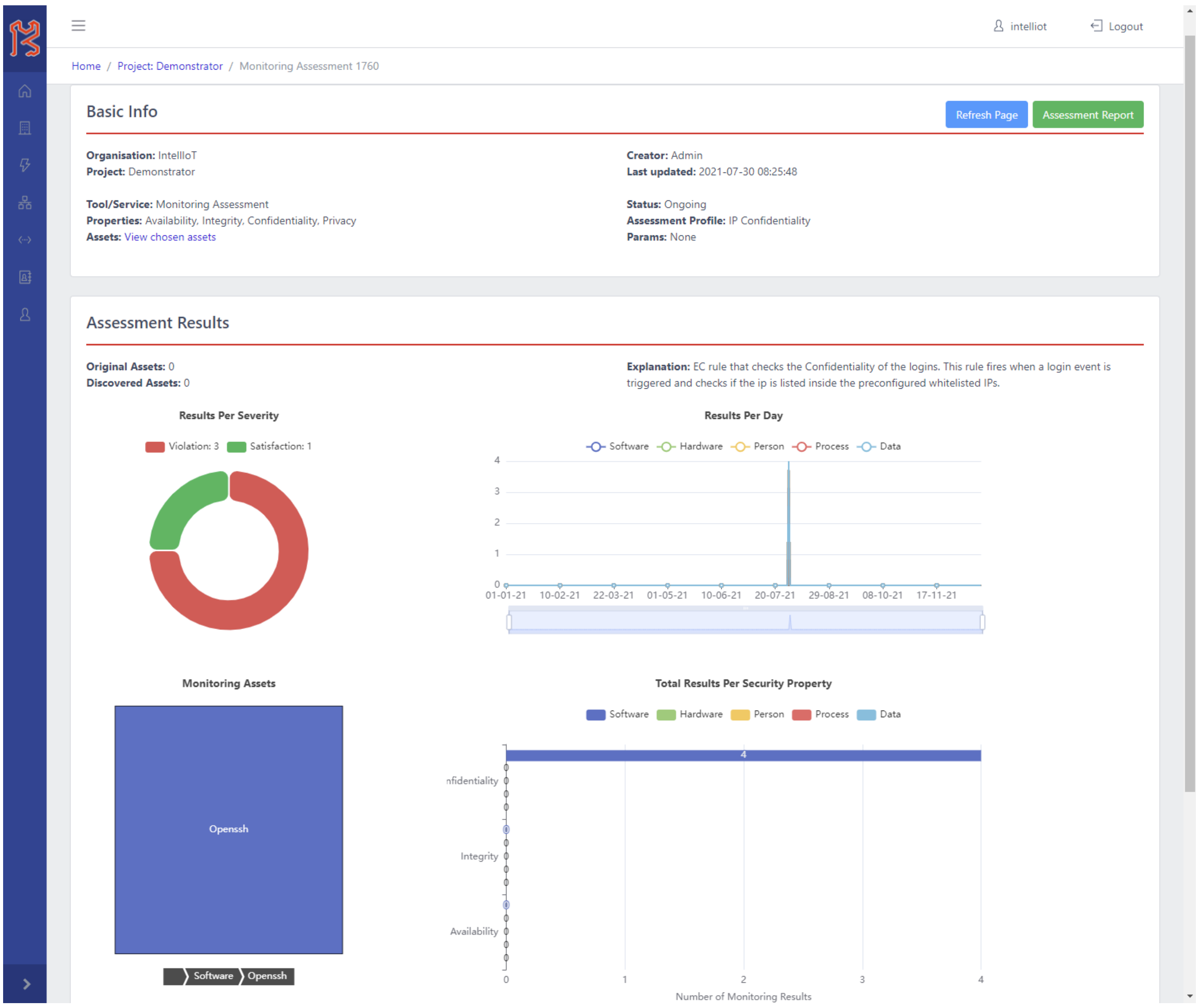
Showing 1 to 6 of 6 entries Show 10 entries Previous 1 Next

Figure 23. View of defined assets and executed assessments during SAP testing.

2.4.2.1 ASSESSMENT TESTING

Furthermore, all types of assessments that will be delivered for Cycle 2 where successfully tested within the local testbed environment. These, as detailed in Section 2.4 of D4.4 & D4.8, included:

(i) Monitoring Assessment with satisfaction and violation of pre-defined testing rules that are based on Drools specification language (see results in Figure 24)
(ii) Vulnerability Analysis Assessment on the testbed assets (see Figure 25 and Figure 26), and
(iii) Dynamic Testing Assessment of the testbed assets (see Figure 27 and Figure 28).


Home / Project: Demonstrator / Monitoring Assessment 1760

Basic Info

Organisation: IntellIoT
Project: Demonstrator

Tool/Service: Monitoring Assessment
Properties: Availability, Integrity, Confidentiality, Privacy
Assets: View chosen assets

Creator: Admin
Last updated: 2021-07-30 08:25:48

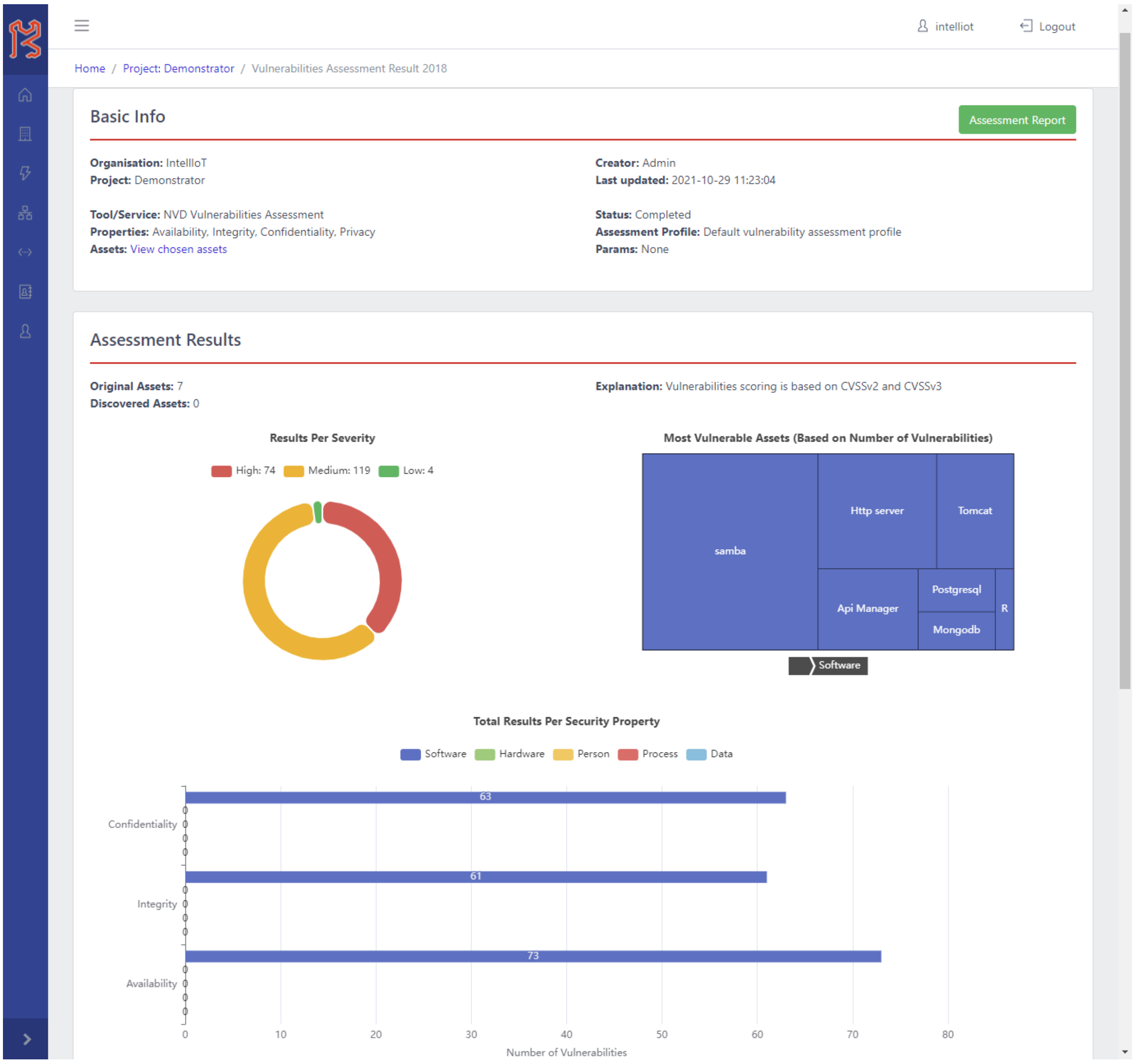
Status: Ongoing
Assessment Profile: IP Confidentiality
Params: None

Assessment Results

Original Assets: 0
Discovered Assets: 0

Explanation: EC rule that checks the Confidentiality of the logins. This rule fires when a login event is triggered and checks if the ip is listed inside the preconfigured whitelisted IPs.

Figure 24. SAP testbed monitoring assessment results.


intelliot
Logout
Home / Project: Demonstrator / Vulnerabilities Assessment Result 2018
Assessment Report

Basic Info

Organisation: IntellIoT
Project: Demonstrator

Tool/Service: NVD Vulnerabilities Assessment
Properties: Availability, Integrity, Confidentiality, Privacy
Assets: View chosen assets

Creator: Admin
Last updated: 2021-10-29 11:23:04

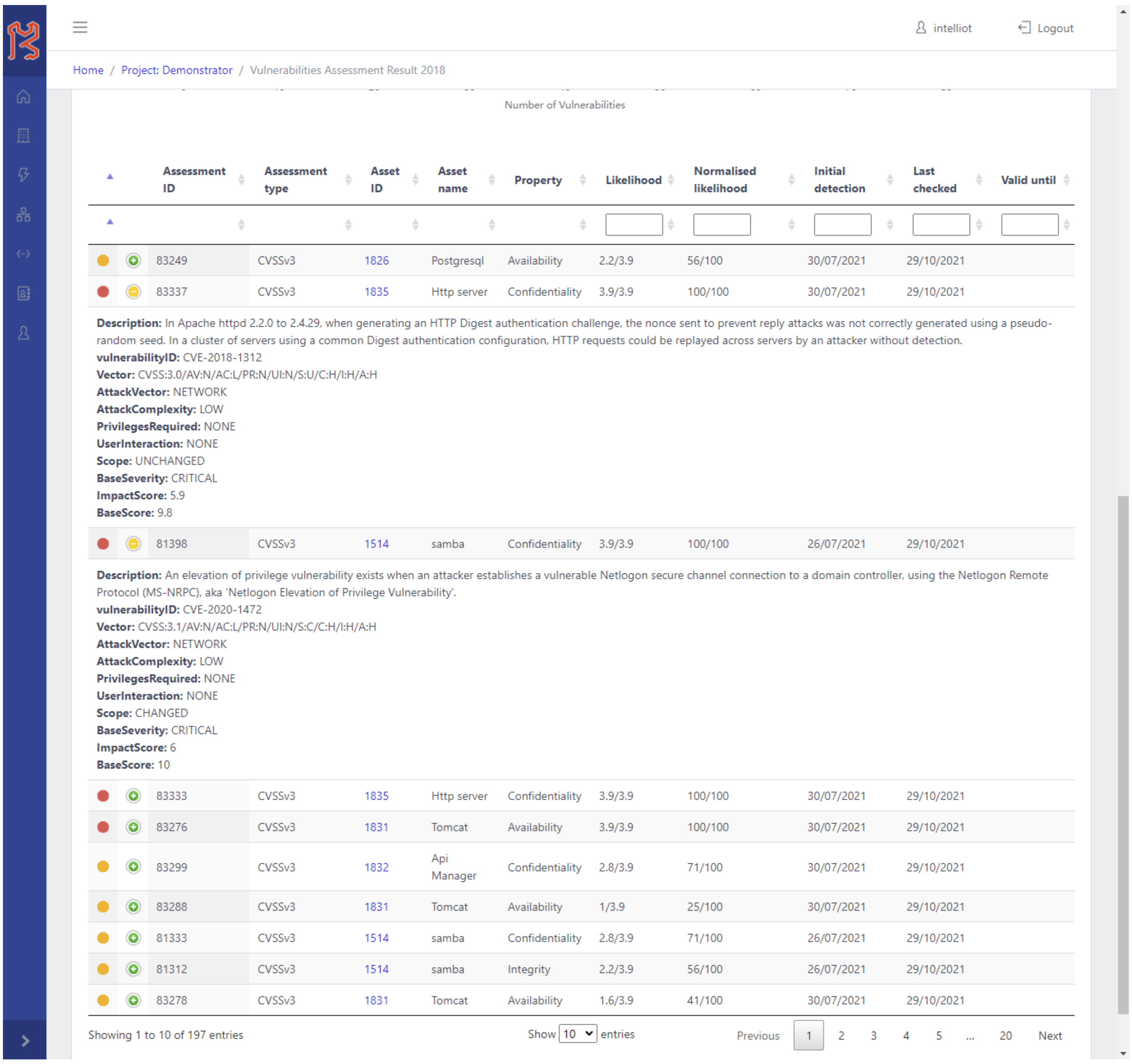
Status: Completed
Assessment Profile: Default vulnerability assessment profile
Params: None

Assessment Results

Original Assets: 7
Discovered Assets: 0

Explanation: Vulnerabilities scoring is based on CVSSv2 and CVSSv3

Figure 25. SAP Vulnerability Assessment carried out on testbed setup (Overview, KPIs).


intelliot
Logout
Home / Project: Demonstrator / Vulnerabilities Assessment Result 2018
Number of Vulnerabilities

		Assessment ID	Assessment type	Asset ID	Asset name	Property	Likelihood	Normalised likelihood	Initial detection	Last checked	Valid until
🟠	⊕	83249	CVSSv3	1826	Postgresql	Availability	2.2/3.9	56/100	30/07/2021	29/10/2021	
🔴	⊖	83337	CVSSv3	1835	Http server	Confidentiality	3.9/3.9	100/100	30/07/2021	29/10/2021	

Description: In Apache httpd 2.2.0 to 2.4.29, when generating an HTTP Digest authentication challenge, the nonce sent to prevent reply attacks was not correctly generated using a pseudo-random seed. In a cluster of servers using a common Digest authentication configuration, HTTP requests could be replayed across servers by an attacker without detection.
vulnerabilityID: CVE-2018-1312
Vector: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
AttackVector: NETWORK
AttackComplexity: LOW
PrivilegesRequired: NONE
UserInteraction: NONE
Scope: UNCHANGED
BaseSeverity: CRITICAL
ImpactScore: 5.9
BaseScore: 9.8

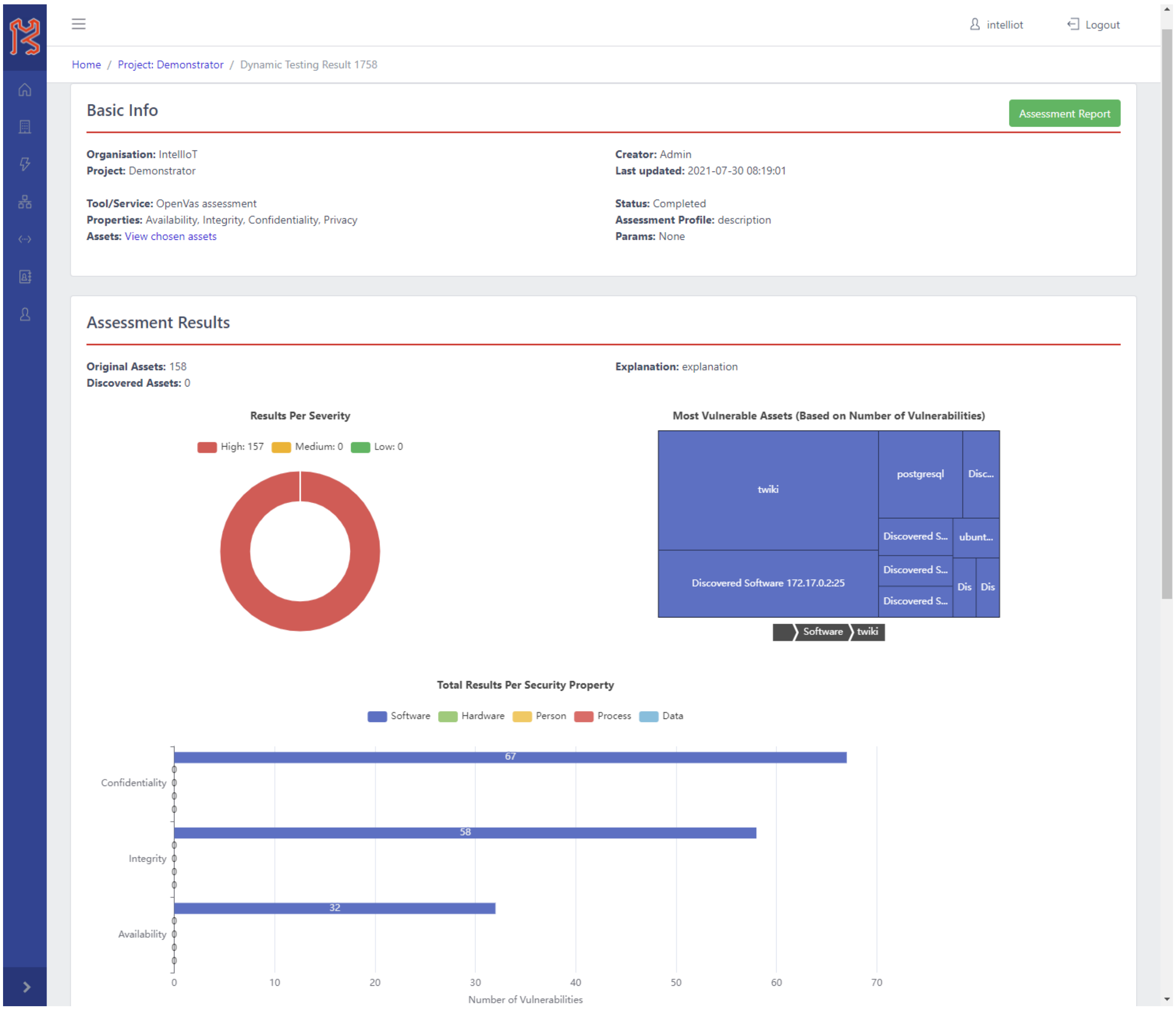
		Assessment ID	Assessment type	Asset ID	Asset name	Property	Likelihood	Normalised likelihood	Initial detection	Last checked	Valid until
🔴	⊖	81398	CVSSv3	1514	samba	Confidentiality	3.9/3.9	100/100	26/07/2021	29/10/2021	

Description: An elevation of privilege vulnerability exists when an attacker establishes a vulnerable Netlogon secure channel connection to a domain controller, using the Netlogon Remote Protocol (MS-NRPC), aka 'Netlogon Elevation of Privilege Vulnerability'.
vulnerabilityID: CVE-2020-1472
Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H
AttackVector: NETWORK
AttackComplexity: LOW
PrivilegesRequired: NONE
UserInteraction: NONE
Scope: CHANGED
BaseSeverity: CRITICAL
ImpactScore: 6
BaseScore: 10

		Assessment ID	Assessment type	Asset ID	Asset name	Property	Likelihood	Normalised likelihood	Initial detection	Last checked	Valid until
🔴	⊕	83333	CVSSv3	1835	Http server	Confidentiality	3.9/3.9	100/100	30/07/2021	29/10/2021	
🔴	⊕	83276	CVSSv3	1831	Tomcat	Availability	3.9/3.9	100/100	30/07/2021	29/10/2021	
🟠	⊕	83299	CVSSv3	1832	Api Manager	Confidentiality	2.8/3.9	71/100	30/07/2021	29/10/2021	
🟠	⊕	83288	CVSSv3	1831	Tomcat	Availability	1/3.9	25/100	30/07/2021	29/10/2021	
🟠	⊕	81333	CVSSv3	1514	samba	Confidentiality	2.8/3.9	71/100	26/07/2021	29/10/2021	
🟠	⊕	81312	CVSSv3	1514	samba	Integrity	2.2/3.9	56/100	26/07/2021	29/10/2021	
🟠	⊕	83278	CVSSv3	1831	Tomcat	Availability	1.6/3.9	41/100	30/07/2021	29/10/2021	

Showing 1 to 10 of 197 entries

Figure 26. SAP Vulnerability Assessment carried out on testbed setup (viewing details on results).



Home / Project: Demonstrator / Dynamic Testing Result 1758

Basic Info

Organisation: IntellIoT
Project: Demonstrator

Tool/Service: OpenVas assessment
Properties: Availability, Integrity, Confidentiality, Privacy
Assets: View chosen assets

Creator: Admin
Last updated: 2021-07-30 08:19:01

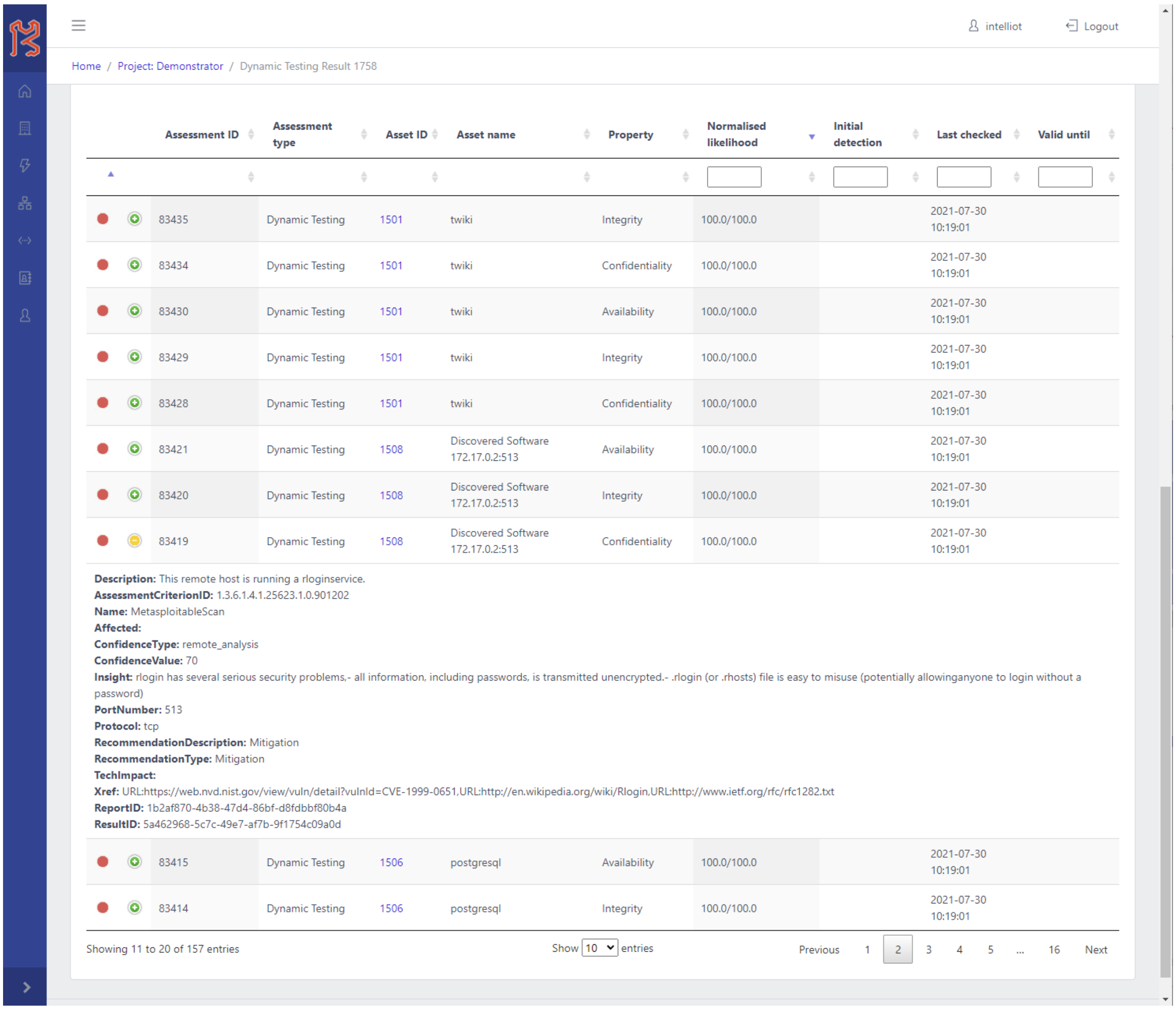
Status: Completed
Assessment Profile: description
Params: None

Assessment Results

Original Assets: 158
Discovered Assets: 0

Explanation: explanation

Figure 27. SAP Dynamic Testing assessment carried out on testbed setup (Overview, KPIs).



		Assessment ID	Assessment type	Asset ID	Asset name	Property	Normalised likelihood	Initial detection	Last checked	Valid until
●	⊕	83435	Dynamic Testing	1501	twiki	Integrity	100.0/100.0		2021-07-30 10:19:01	
●	⊕	83434	Dynamic Testing	1501	twiki	Confidentiality	100.0/100.0		2021-07-30 10:19:01	
●	⊕	83430	Dynamic Testing	1501	twiki	Availability	100.0/100.0		2021-07-30 10:19:01	
●	⊕	83429	Dynamic Testing	1501	twiki	Integrity	100.0/100.0		2021-07-30 10:19:01	
●	⊕	83428	Dynamic Testing	1501	twiki	Confidentiality	100.0/100.0		2021-07-30 10:19:01	
●	⊕	83421	Dynamic Testing	1508	Discovered Software 172.17.0.2:513	Availability	100.0/100.0		2021-07-30 10:19:01	
●	⊕	83420	Dynamic Testing	1508	Discovered Software 172.17.0.2:513	Integrity	100.0/100.0		2021-07-30 10:19:01	
●	⊖	83419	Dynamic Testing	1508	Discovered Software 172.17.0.2:513	Confidentiality	100.0/100.0		2021-07-30 10:19:01	

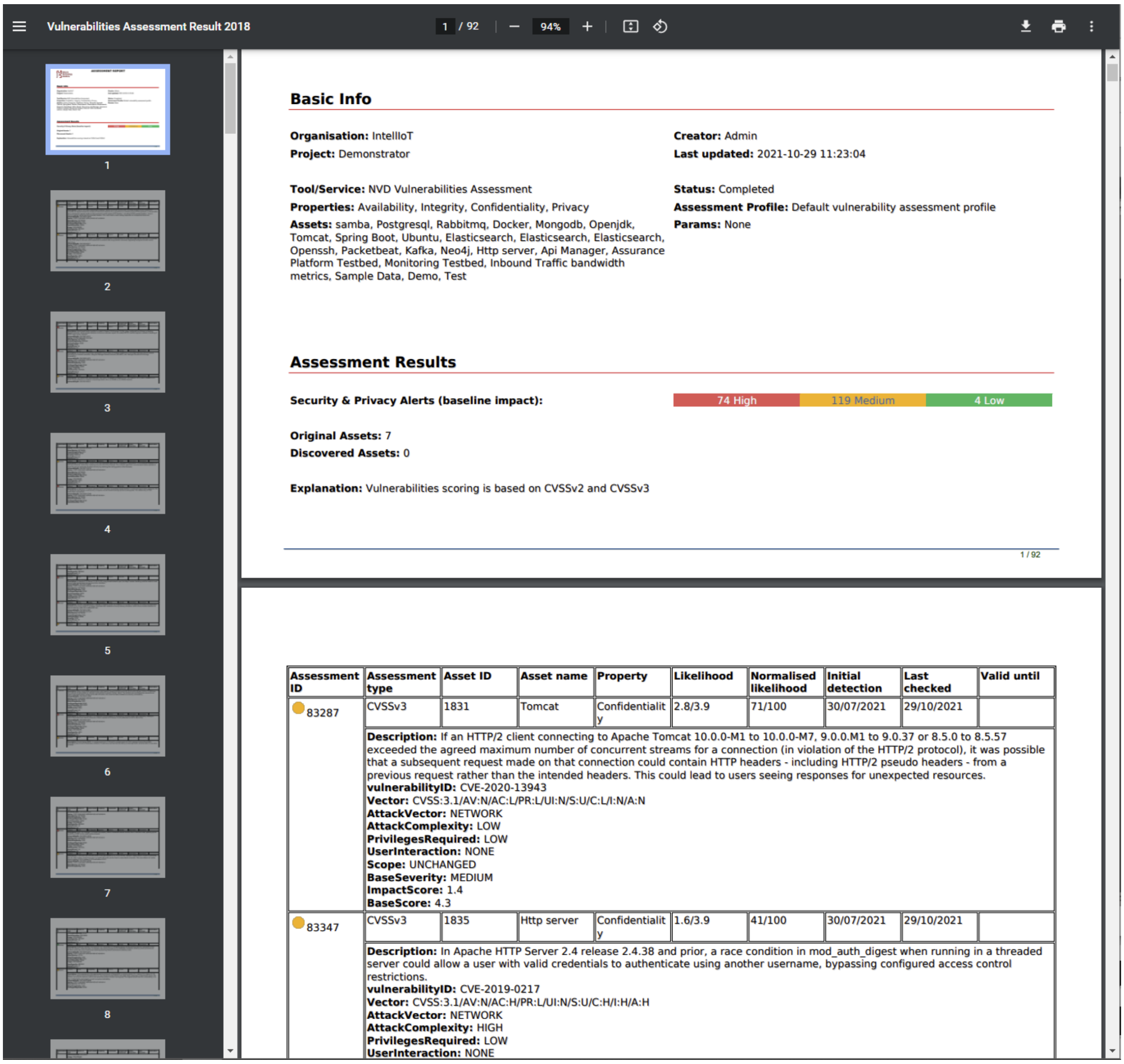
Description: This remote host is running a rloginservice.
AssessmentCriterionID: 1.3.6.1.4.1.25623.1.0.901202
Name: MetasploitableScan
Affected:
ConfidenceType: remote_analysis
ConfidenceValue: 70
Insight: rlogin has several serious security problems,- all information, including passwords, is transmitted unencrypted.- .rlogin (or .rhosts) file is easy to misuse (potentially allowinganyone to login without a password)
PortNumber: 513
Protocol: tcp
RecommendationDescription: Mitigation
RecommendationType: Mitigation
TechImpact:
Xref: URL:https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-1999-0651,URL:http://en.wikipedia.org/wiki/Rlogin,URL:http://www.ietf.org/rfc/rfc1282.txt
ReportID: 1b2af870-4b38-47d4-86bf-d8fdbbf80b4a
ResultID: 5a462968-5c7c-49e7-af7b-9f1754c09a0d

●	⊕	83415	Dynamic Testing	1506	postgresql	Availability	100.0/100.0		2021-07-30 10:19:01	
●	⊕	83414	Dynamic Testing	1506	postgresql	Integrity	100.0/100.0		2021-07-30 10:19:01	

Showing 11 to 20 of 157 entries

Figure 28. SAP Dynamic Testing assessment carried out on testbed setup (viewing details on results).

Finally, the horizontal capability of SAP to export reports of its assessment results in PDF format (e.g., to provide in written form the certification evidence required) was also successfully tested, as shown in Figure 29.



Basic Info

Organisation: IntellIoT
Project: Demonstrator
Creator: Admin
Last updated: 2021-10-29 11:23:04

Tool/Service: NVD Vulnerabilities Assessment
Properties: Availability, Integrity, Confidentiality, Privacy
Assets: samba, Postgresql, Rabbitmq, Docker, Mongodb, Openjdk, Tomcat, Spring Boot, Ubuntu, Elasticsearch, Elasticsearch, Openssh, Packetbeat, Kafka, Neo4j, Http server, Api Manager, Assurance Platform Testbed, Monitoring Testbed, Inbound Traffic bandwidth metrics, Sample Data, Demo, Test
Status: Completed
Assessment Profile: Default vulnerability assessment profile
Params: None

Assessment Results

Security & Privacy Alerts (baseline impact): 74 High | 119 Medium | 4 Low

Original Assets: 7
Discovered Assets: 0

Explanation: Vulnerabilities scoring is based on CVSSv2 and CVSSv3

1 / 92

Assessment ID	Assessment type	Asset ID	Asset name	Property	Likelihood	Normalised likelihood	Initial detection	Last checked	Valid until
● 83287	CVSSv3	1831	Tomcat	Confidentiality	2.8/3.9	71/100	30/07/2021	29/10/2021	
	Description: If an HTTP/2 client connecting to Apache Tomcat 10.0.0-M1 to 10.0.0-M7, 9.0.0.M1 to 9.0.37 or 8.5.0 to 8.5.57 exceeded the agreed maximum number of concurrent streams for a connection (in violation of the HTTP/2 protocol), it was possible that a subsequent request made on that connection could contain HTTP headers - including HTTP/2 pseudo headers - from a previous request rather than the intended headers. This could lead to users seeing responses for unexpected resources. **vulnerabilityID:** CVE-2020-13943 **Vector:** CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N **AttackVector:** NETWORK **AttackComplexity:** LOW **PrivilegesRequired:** LOW **UserInteraction:** NONE **Scope:** UNCHANGED **BaseSeverity:** MEDIUM **ImpactScore:** 1.4 **BaseScore:** 4.3								
● 83347	CVSSv3	1835	Http server	Confidentiality	1.6/3.9	41/100	30/07/2021	29/10/2021	
	Description: In Apache HTTP Server 2.4 release 2.4.38 and prior, a race condition in mod_auth_digest when running in a threaded server could allow a user with valid credentials to authenticate using another username, bypassing configured access control restrictions. **vulnerabilityID:** CVE-2019-0217 **Vector:** CVSS:3.1/AV:N/AC:H/PR:L/UI:N/S:U/C:H/I:H/A:H **AttackVector:** NETWORK **AttackComplexity:** HIGH **PrivilegesRequired:** LOW **UserInteraction:** NONE								

Figure 29. Exported SAP assessment report in PDF format (in this case, of a vulnerability assessment).

Further testing of the SAP that took place focused on its integration with the rest of the Trust components (as mentioned in D2.3, D4.4 & D4.8, SAP is a core component that all other Trust enablers interact with), as was presented in Section 2.4.1.

2.4.2.2 ORCHESTRATION, AUTOMATION & INCIDENT RESPONSE COMPONENT

CACAO security playbooks can be executed manually or automatically using the Sphynx Incident Response (IR) platform. The latter can be configured as a standalone system or as a module attached to the Sphynx Security & Privacy Assurance Platform (SAP) and has been thoroughly described in section 2.5 of D.4.8 The IR platform can import, export, and run CACAO security playbooks as a standalone solution by using the REST API of each playbook. The IR platform functions as a module of the SPAP and can be used to carry out security playbooks prompted by the SPAP's components, such as EVEREST and CRISIS, and to make use of data gathered by the Asset Model and the CTI component. The SPAP's components can also be coordinated via the IR platform. The Incident Response platform is composed of five main components as presented in Figure 29.

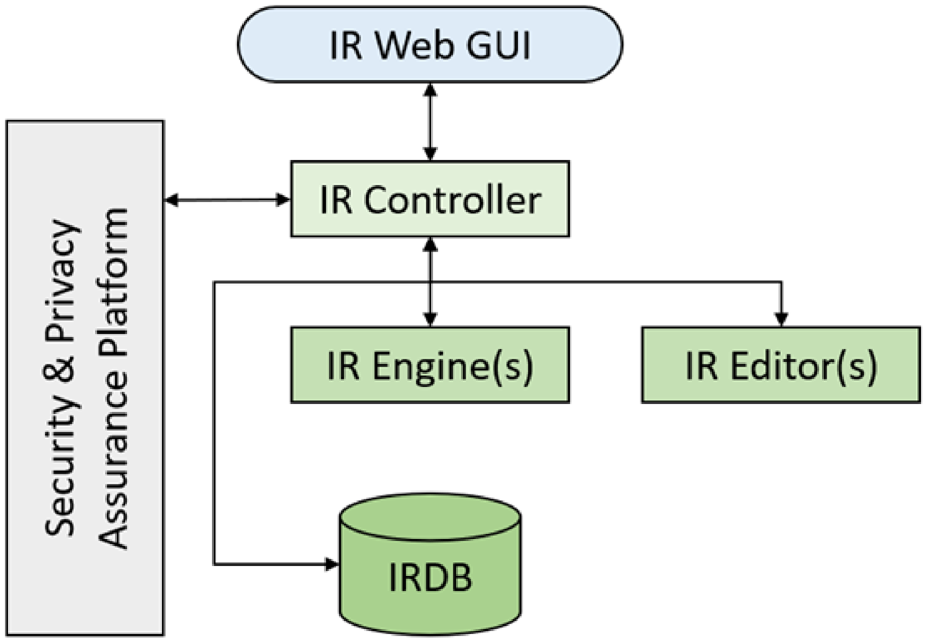


Figure 29. *IR Platform architecture overview.*

In accordance with the CACAO specification, the system also provides a graphical drag-and-drop interface for building and changing CACAO security playbooks, which may then be executed or exported as CACAO JSON files. The IR platform also features an interactive dashboard that delivers real-time views of the system's state and the execution of security playbooks in addition to high- and low-level logs, KPIs, user notification, and other data.

The standalone CACAO engine installation assumes a Linux-based environment with the following tools present:

- `git`
- `npm`
- `node` (Supported: >8.x, Recommended: 14.x)
- `maven`
- `jdk11-openjdk`

The basis for the CACAO engine is Node-RED, and the engine is built on top of it as a module. The engine supports all Node-RED versions >1.x but we recommend using version 3.0.x.

2.4.3 TRUST-BASED IDS

The Trust-based Intrusion Detection System (IDS) has been described in Deliverable D4.8. It is a software component that resides on each network node of the IoT application (in the tractor, wearable device, robotic arm in IntellIoT's use cases for example) and monitors network traffic. For every node of the network that there is communication with, it builds a trust value based on certain criteria. Depending on the computed trust value, nodes are characterised as trustworthy or not and this information (warnings) is relayed to the security components of the network so that action may be triggered. More details about the software component, the rules upon which it operates, and computes trust values and its implementation architecture may be found in Deliverable D4.8, however for clarity and readability purposes Figure 30 reiterates the architecture of the tool.

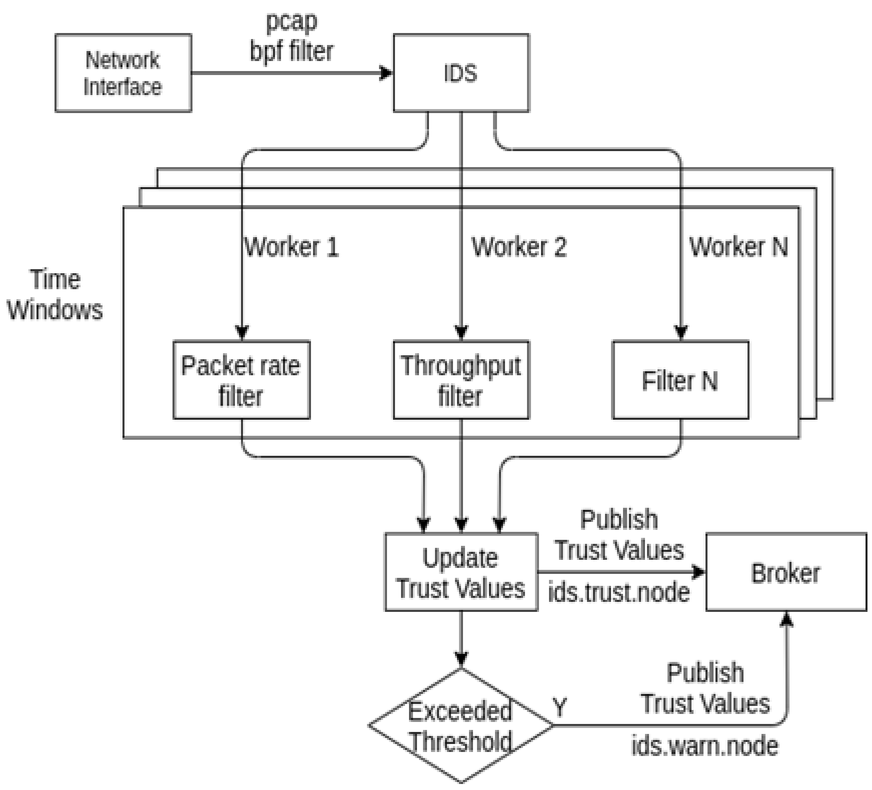


Figure 30. Architecture of the Trust-based IDS tool.

Concerning the integration of the tool with the other software components that is the focus of the current deliverable, the Trust-based IDS, as can be seen from Figure 30, uses a secure communication channel. This channel is an AMQP broker, that is used to send the trust values and warning messages to the other security components (the MTD module and the SAP). AMQP is an application layer messaging protocol that allows for various routing topologies, such as point-to-point and publish-subscribe, with TLS support to provide secure communication. This communication scheme is depicted in Figure 31.

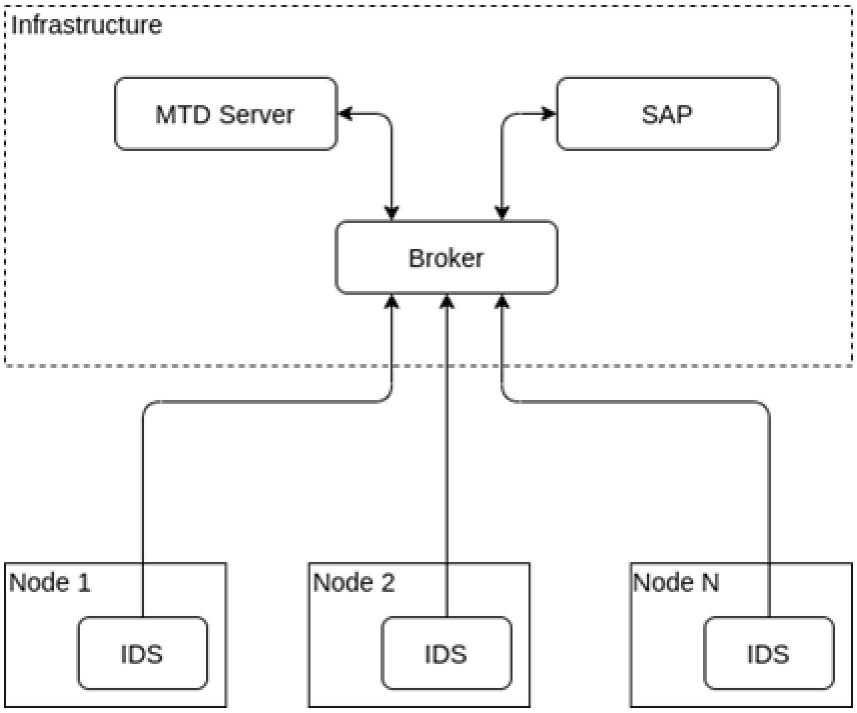


Figure 31: Communication between IDS components and the MTD and SAP modules

2.4.3.1 INTEGRATION OF IDS WITH OTHER TRUST ENABLERS

IntellIoT employs a dedicated Trust broker based on RabbitMQ and using the AMQP binary protocol as mentioned above. Through the broker, the IDS component communicates with the SAP and the MTD components.

IDS exposes two topics, in JSON format, to the broker for the rest of the security components to consume and take appropriate actions. The first one is the trust topic `'ids.trust.[node id]'`, where each IDS instance publishes its trust values for the nodes it has communicated with. An example payload is:

```
{
  "10.0.0.1": 0.0,
  "10.0.0.2": 1.0,
  ...
}
```

The second topic is the `'ids.warn.[node id]'`, where each IDS instance publishes a list of nodes that are considered untrustworthy. An example payload is:

```
{
  "nodes": [
    "10.0.0.1"
  ]
}
```

2.4.4 MOVING TARGET DEFENCES

The Moving Target Defences (MTD) component has been presented in Deliverable D4.8. The MTD comprises of several components that aim to provide a dynamic and constantly shifting configuration to the network infrastructure that they try to defend. This is done either proactively (to prevent an attack) or reactively (to mitigate an ongoing attack).

While MTD can be applied in several ways, in IntellIoT, we focus on managing the network configuration and dynamically changing it to fit the needs of the infrastructure. Examples of our strategies include the periodic shuffle of IP addresses and communication ports, as well as the encryption of network traffic while changing in different time intervals (or as a response to certain events) of the encryption keys and the encryption algorithms used to counter sniffing and Man-In-The-Middle attacks.

The MTD components that have been developed are a client-server set of software components that manage the network configuration of the edge network. The MTD server is responsible for managing all the clients, handling events like warnings from the IDS or actions from the SAP and generating new configurations. The MTD clients are responsible for applying the configuration sent by the server, and based on that configuration, encrypt the traffic, and transmit it through tunnels to avoid packet sniffing and Man-In-The-Middle attacks.

The server and clients communicate by exchanging the network configuration. Parts of the network configuration are the same for all clients, and parts are specific to each client.

The MTD server proactively generates a new configuration at fixed intervals to shift the attack surface, and when a warning is received, it reactively generates a mitigation configuration based on predefined strategies to mitigate the possible attack. It is also possible for the HIL service to send actions (e.g., *isolate node X*) through the Security Assurance Platform. Configuration changes focus on network layer 3 and above, and do not interfere with the TSN controller and the resource reservations it maintains. The MTD clients are responsible for managing the network configuration, maintaining an encrypted connection between each other using the routing table sent by the MTD server and applying any changes the server sends.

At start up, each client has to register with the server by sending a registration request containing the client name and IP address through the broker. The client name is a combination of the username of the user that owns this device and the devices client ID in the format *<username>_<clientID>* (e.g. ,TSI_sensor1, TSI_sensor2, SANL_tractor etc).

2.4.4.1 INTEGRATION OF MTD WITH OTHER TRUST ENABLERS

MTD exposes the following topics:

`mtd.registration`

`mtd.config.<client name>`

`mtd.keepaliveReq.<client name>`

`mtd.keepaliveResp.<client name>`

`mtd.alert`

`mtd.trigger`

2.4.4.1.1 MTD AND SAP

The MTD client and server applications communicate over a secure channel provided by the Trust Message Broker. More specifically, the MTD Server consumes messages generated from IDS and takes mitigation actions when this is needed. When such an action is taken, the server needs to notify the Security Assurance Platform, as it is the component providing a holistic view of the current security and privacy posture of the system to the operators. This happens through the 'mtd.alert' topic.

The reverse case is also needed. The Security Assurance Platform, through the event captors, can also identify attacks and trigger a mitigation action using the 'mtd.trigger' topic.

Both of these follow the same payload logic as the 'ids.warn' *topic:*

```
{
  "nodes": [
    "10.0.0.1"
    ]
}
```

2.4.5 AUTHENTICATION – AUTHORISATION – ACCOUNTING (AAA)

Authentication, Authorization, and Accounting (AAA) in IntellIoT is provided through a framework that allows legitimate users or applications to gain access to protected resources, thus ensuring network security. It can enforce policies (e.g., define multiple permission levels) and audit usage.

As described in Deliverable D4.8, the solution selected for IntellIoT is *Keycloak*. *Keycloak* comprises a dedicated server that centrally controls user access. This decouples local security configuration considerations and allows for better scalability and less administrative tasks. Furthermore, the servers available on the network are not tasked with storing user credentials locally, which is a preferable approach from both the security and administrative perspectives.

Authorization is the process of evaluating to what extent a user can access a resource. With *OAuth 2.0*, the following four roles are defined:

(1) ***Resource Owner*** (i.e., any entity that owns a number of protected resources, such as files)

(2) ***Resource Server*** (e.g., the actual server that keeps the files)

(3) ***Client*** (e.g., an application that requests access to a resource from the server is a client)

(4) ***Authorization Server*** (the Keycloak server itself in this case)

When a client wants to access a resource on behalf of an Owner in the Server, it first contacts the Authorization Server. The Authorization Server issues a JSON Web Token (JWT) that gives limited access to the resources, which can then be used to access the Resource Server. JWT tokens are also a standard format based on JSON, supported by libraries for various programming languages. ***Authentication*** is the process of identifying a user. OpenID Connect defines three roles:

(1) ***End User*** (i.e., entity that needs to be authenticated)

(2) ***Relying Party*** (the application/server that asks for a user to authenticate himself/herself before accessing it)

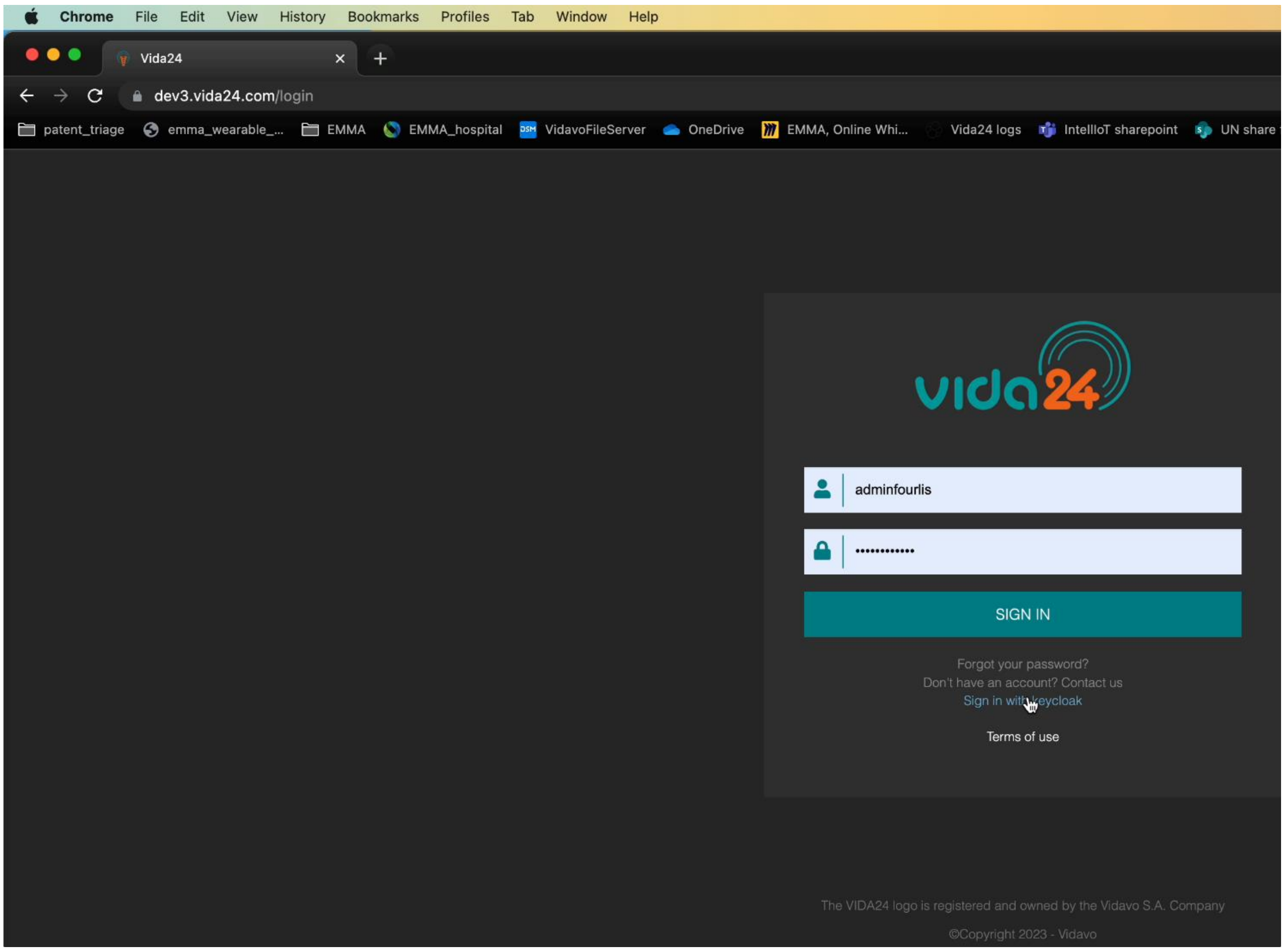
(3) ***OpenID Provider*** (the server providing the actual authentication, i.e., Keycloak in this case)

Similarly, to the authorization case, a JWT token is used. In this scenario it contains additional information in order to signify the authentication process. ***Accounting*** is the last functionality provided which is about monitoring authentication and authorization events for auditing. These are clustered in two categories: the login events and the admin events. Login events are generated during normal operation and cover all authentication and authorization aspects. Admin events are generated during Keycloak administration and cover any action that can be performed from within the administration web UI. Every action can be recorded and stored in the Keycloak database as well as the system log.

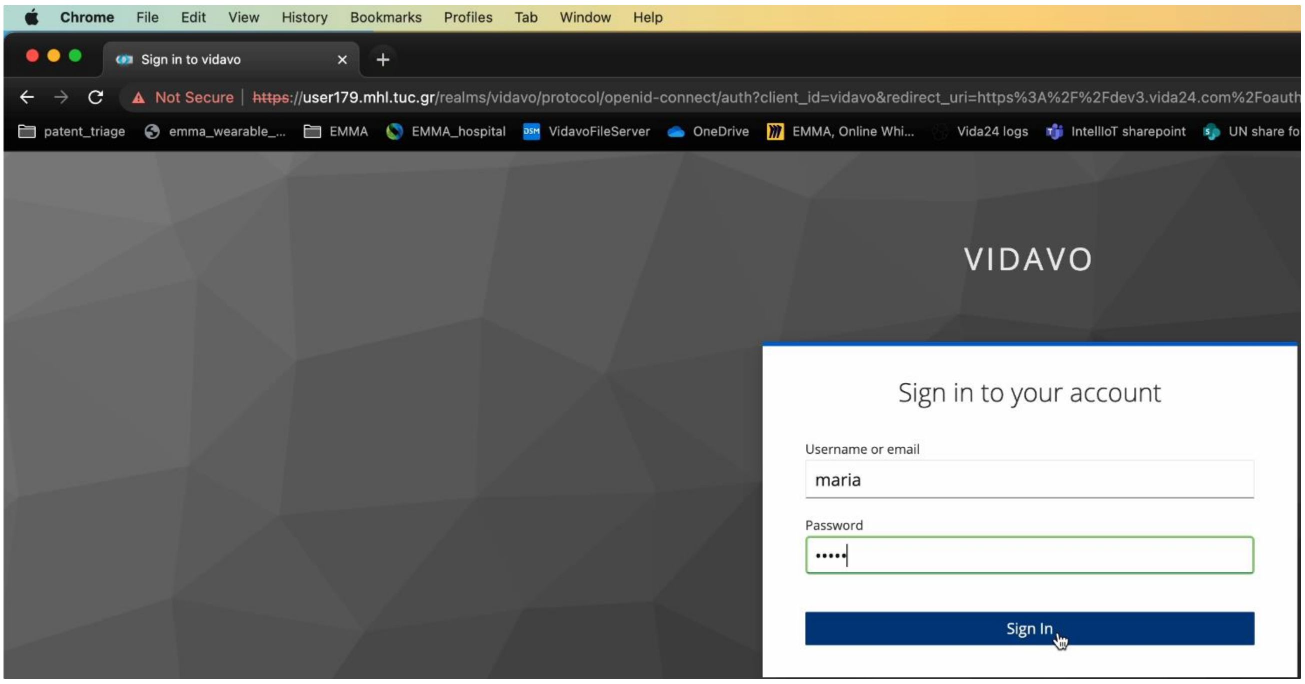
In the following two subsections, two approaches for the deployment of Keycloak in IntellIoT are presented. The first one is a direct deployment of Keycloak where a service can access it, as long as it talks OAuth 2.0. The second one uses an intermediate reverse proxy node in front of a service that requires authentication and authorization but does not support OAuth 2.0. In this way users can authenticate and access a service from their browser. Additional work is in progress to allow automated access without a need for a browser.

2.4.5.1 USE CASE 2 INTEGRATION

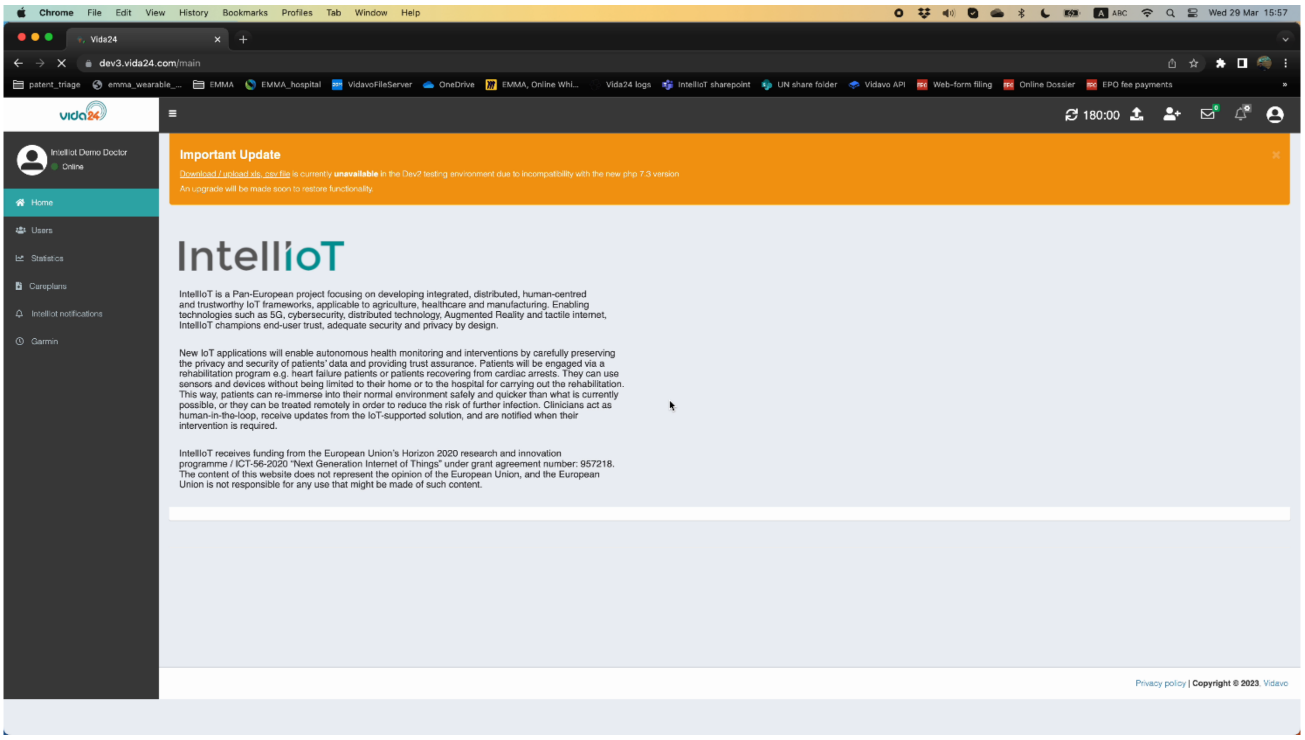
For the Healthcare Use Case we have integrated Keycloak with Vidavo web interface, Vida24. A user can select to login via Keycloak as shown below:


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Vida24
dev3.vida24.com/login
patent_triage emma_wearable_... EMMA EMMA_hospital VidavoFileServer OneDrive EMMA, Online Whi... Vida24 logs IntellIoT sharepoint UN share
vida24
adminfourlis
SIGN IN
Forgot your password?
Don't have an account? Contact us
Sign in with keycloak
Terms of use
The VIDA24 logo is registered and owned by the Vidavo S.A. Company
©Copyright 2023 - Vidavo

In that case, the user will first be redirected to the Keycloak instance for authentication:


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Sign in to vidavo
Not Secure | https://user179.mhl.tuc.gr/realms/vidavo/protocol/openid-connect/auth?client_id=vidavo&redirect_uri=https%3A%2F%2Fdev3.vida24.com%2Foauth
patent_triage emma_wearable_... EMMA EMMA_hospital VidavoFileServer OneDrive EMMA, Online Whi... Vida24 logs IntellIoT sharepoint UN share fo
VIDAVO
Sign in to your account
Username or email
maria
Password
•••••
Sign In

A legitimate user will be redirected back to Vida24 site and be successfully logged in:



Important Update

Download / upload xls, csv file is currently **unavailable** in the Dev2 testing environment due to incompatibility with the new php 7.3 version
An upgrade will be made soon to restore functionality.

IntellIoT

IntellIoT is a Pan-European project focusing on developing integrated, distributed, human-centred and trustworthy IoT frameworks, applicable to agriculture, healthcare and manufacturing. Enabling technologies such as 5G, cybersecurity, distributed technology, Augmented Reality and tactile internet, IntellIoT champions end-user trust, adequate security and privacy by design.

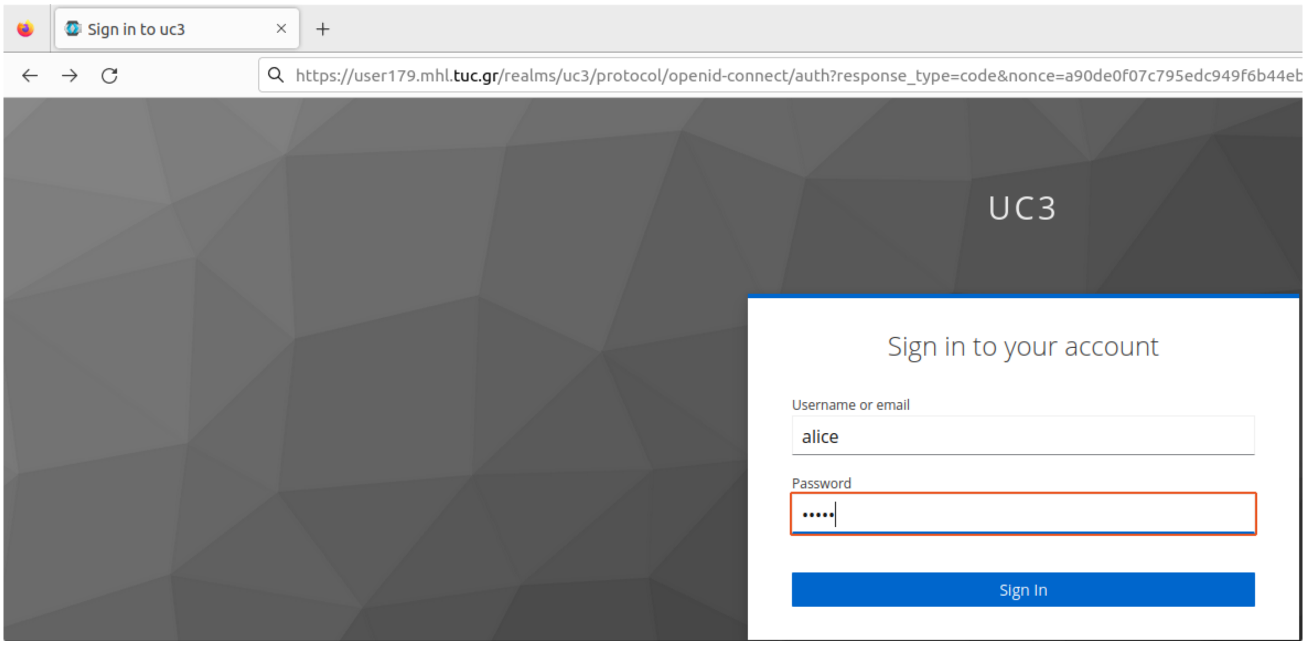
New IoT applications will enable autonomous health monitoring and interventions by carefully preserving the privacy and security of patients' data and providing trust assurance. Patients will be engaged via a rehabilitation program e.g. heart failure patients or patients recovering from cardiac arrests. They can use sensors and devices without being limited to their home or to the hospital for carrying out the rehabilitation. This way, patients can re-immerse into their normal environment safely and quicker than what is currently possible, or they can be treated remotely in order to reduce the risk of further infection. Clinicians act as human-in-the-loop, receive updates from the IoT-supported solution, and are notified when their intervention is required.

IntellIoT receives funding from the European Union's Horizon 2020 research and innovation programme / ICT-56-2020 "Next Generation Internet of Things" under grant agreement number: 957218. The content of this website does not represent the opinion of the European Union, and the European Union is not responsible for any use that might be made of such content.

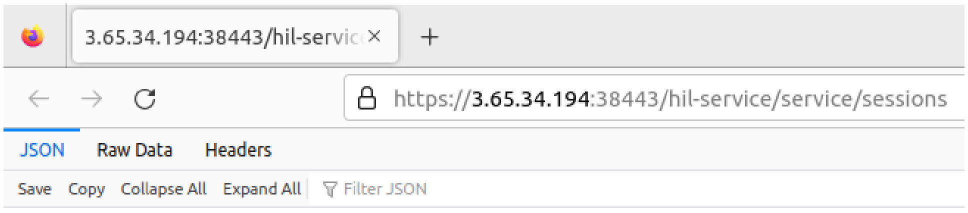
Privacy policy | Copyright © 2023. Vidavo

2.4.5.2 USE CASE 3 INTEGRATION

In Use Case 3, Keycloak provides authentication for users that need access to services running on Edge Devices. A reverse proxy installed on the Edge Device handles the communication with the Keycloak instance. It only allows requests from authenticated users. A custom port is used for SSL connections towards the reverse proxy. Trying to access a service while not authenticated redirects to the Keycloak sign in page:


Sign in to uc3
https://user179.mhl.tuc.gr/realms/uc3/protocol/openid-connect/auth?response_type=code&nonce=a90de0f07c795edc949f6b44eb
UC3
Sign in to your account
Username or email
alice
Password
•••••
Sign In

If the user is authenticated and authorized to access the requested service, the reverse proxy passes the request, and a response is given back. In the following example we have requested the hil-service sessions. The response, although just an empty JSON, shows that we have proper communication in place:


3.65.34.194:38443/hil-servic
https://3.65.34.194:38443/hil-service/service/sessions
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

2.4.6 DISTRIBUTED LEDGER TECHNOLOGIES

There are various types of Distributed Ledger platforms including public DLTs such as *Ethereum*, *Bitcoin*, *IOTA*, *Solana* and private DLTs such as *Quorum* and *Hyperledger Fabric*. A detailed comparison and evaluation are given in Deliverables D3.4 and D3.8.

In a cross-domain project like IntellIoT, we exploit the Ethereum Blockchain platform for the use cases due to the flexibility and convenience to integrate with different components in heterogeneous environments. Ethereum is a distributed public blockchain network that focuses on running the programming code of any decentralized application. Specifically, *Ethereum* is a platform for sharing information across the globe that cannot be manipulated or changed. Ethereum has its own cryptocurrency, called Ether (ETH), and its own programming language, called Solidity. The decentralized applications on the network are called *Dapps*. Practically, Ethereum provides a convenient platform for development and smart contracts system to integrate with FL. We run the Ethereum network via *Ganache* which is a personal blockchain for rapid Ethereum distributed application development for first phase integration.

The general communication workflow of DLT network is shown in Figure 32:

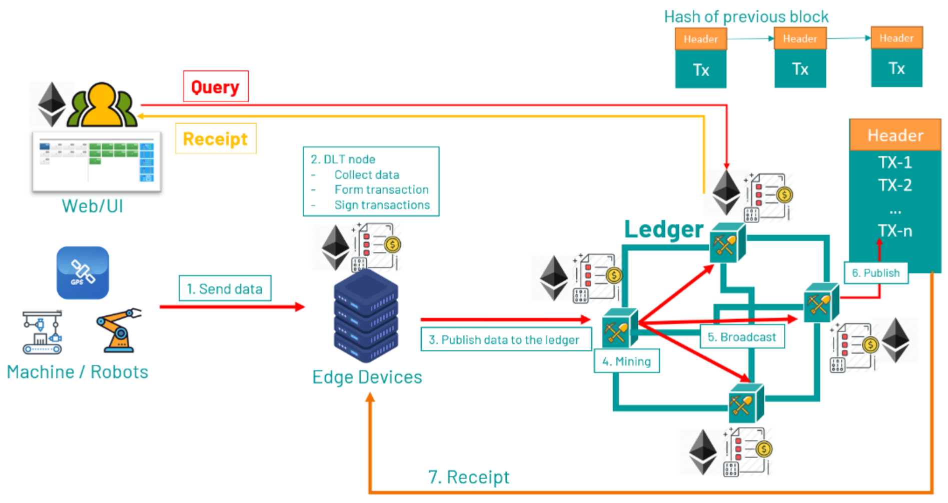


Figure 32 General communication architecture of Ethereum DLT network.

The implementation includes two parts called *DLT-manager* and *DLT-clients*. The DLT-managers require more resources (e.g., storage and computing capacity) to do verification and validation than *DLT-clients*, so that DLT-mangers usually are implemented in edge devices or powerful servers. Meanwhile, the *DLT-clients* are implemented in resource-constrained devices such as IoT devices, and sensors.

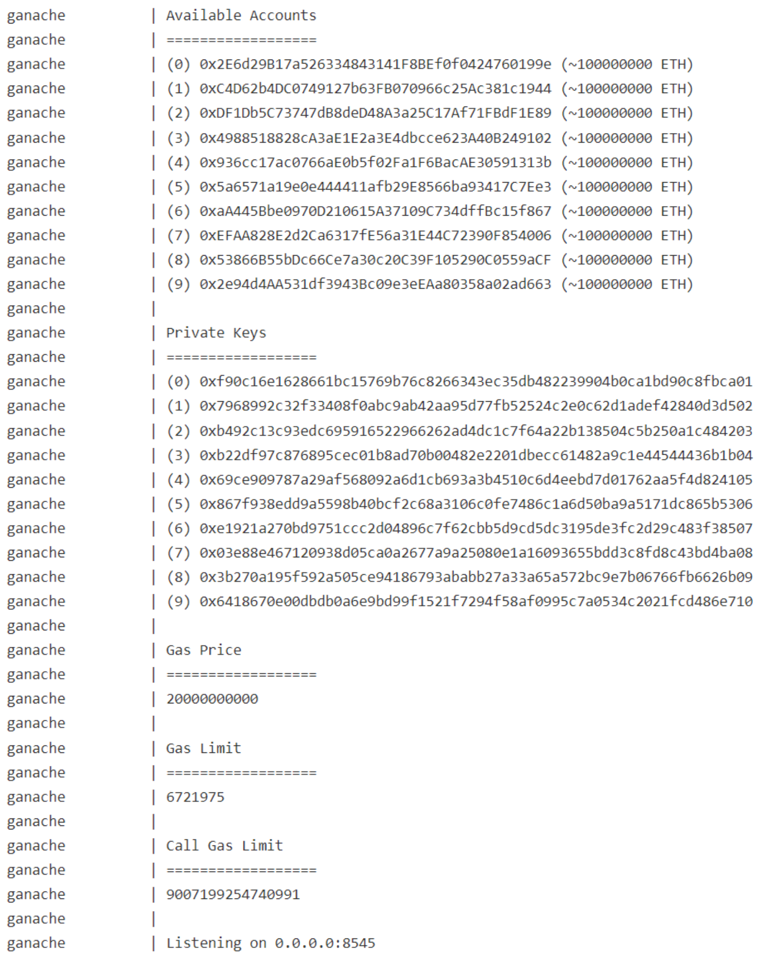
The minimum implementation of DLT-managers and clients are described as below:

	DLT Managers	***DLT clients***
Storage	4 GB RAM minimum with an SSD	1 GB RAM minimum with an SSD
Computing	CPU with 2+ cores	N/A
Connectivity	Wire/Wireless	Wire/Wireless
Interfaces	JSON RPC and REST API	JSON RPC and REST API
Operating System	Linux, Windows	Linux, Windows
Smart Contracts	Solidity	N/A

Table 1: Minimum requirement for installation of DLT components.

In the first phase of integration, we built a simple framework to integrate with components from different partners, for example, the security and information of vulnerable data from TSI, GPS data from sensor of tractors and record these data to tractor controllers, and activities of UR5 Robot arms. We implement an Ethereum private Blockchain by using *Ganache* as *DLT-managers* to simulate a distributed ledger. For Cycle 2, further updates took place in the integration between SAP & the DLT Manager, to support the final (refined) Use Case scenarios, as detailed in deliverable D2.4 – "Use Case specification & Open Call definition (final version)". These updates include richer (REST-based) interfacing capabilities, as well as further work on the common deployment of key DLT components through the same process (i.e., alongside) the rest of the trust enablers. Moreover, we have progressed in the successful integration with the different type of distributed applications for various domains and partner platforms, e.g., with the HyperMAS, the Edge Orchestrator, and the tractor controller. Details of these integrations are reported in D3.8 "Decentralized trust via secure interaction and contracts (final version)".

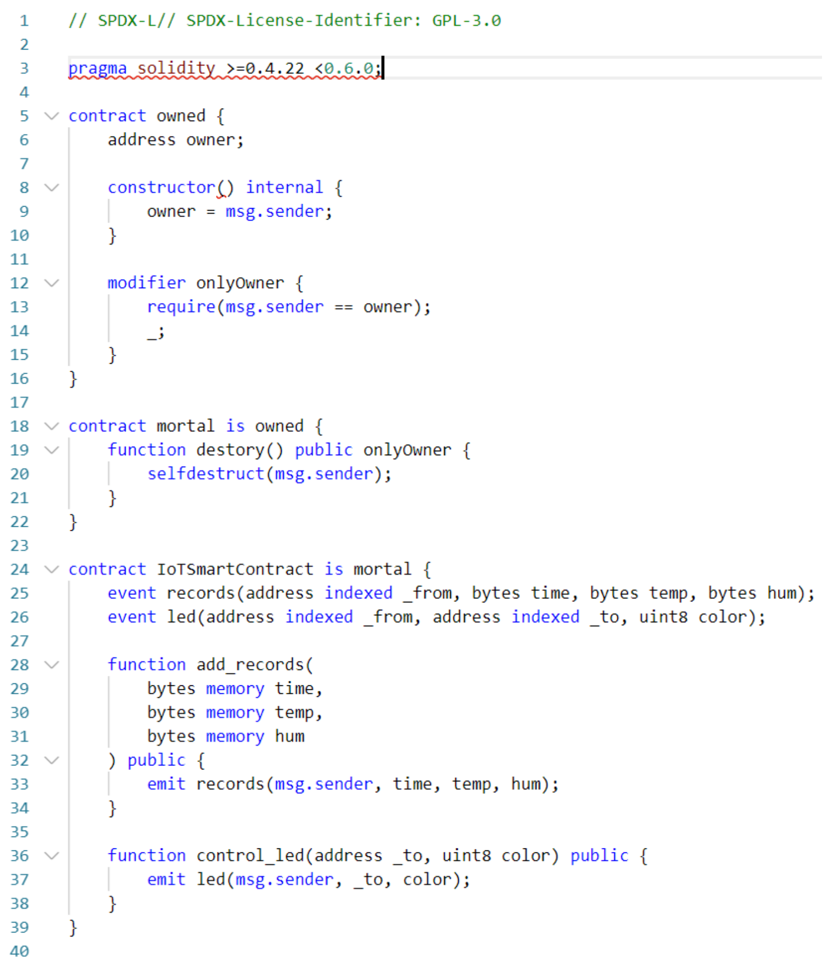
One example of successful implementation of Ethereum DLT private network is demonstrated as Figure 33 below. Each component in the DLT network is assigned an account including a private key and public key. Public key stands for digital identity and broadcast for other nodes, while the private key is used to sign transactions and kept secret. The "gas fee" is in this context *fee* charged by miners for executing a transaction on the Ethereum Blockchain.



```
ganache          | Available Accounts
ganache          | ==================
ganache          | (0) 0x2E6d29B17a526334843141F8BEf0f0424760199e (~100000000 ETH)
ganache          | (1) 0xC4D62b4DC0749127b63FB070966c25Ac381c1944 (~100000000 ETH)
ganache          | (2) 0xDF1Db5C73747dB8deD48A3a25C17Af71FBdF1E89 (~100000000 ETH)
ganache          | (3) 0x4988518828cA3aE1E2a3E4dbcce623A40B249102 (~100000000 ETH)
ganache          | (4) 0x936cc17ac0766aE0b5f02Fa1F6BacAE30591313b (~100000000 ETH)
ganache          | (5) 0x5a6571a19e0e444411afb29E8566ba93417C7Ee3 (~100000000 ETH)
ganache          | (6) 0xaA445Bbe0970D210615A37109C734dffBc15f867 (~100000000 ETH)
ganache          | (7) 0xEFAA828E2d2Ca6317fE56a31E44C72390F854006 (~100000000 ETH)
ganache          | (8) 0x53866B55bDc66Ce7a30c20C39F105290C0559aCF (~100000000 ETH)
ganache          | (9) 0x2e94d4AA531df3943Bc09e3eEAa80358a02ad663 (~100000000 ETH)
ganache          |
ganache          | Private Keys
ganache          | ==================
ganache          | (0) 0xf90c16e1628661bc15769b76c8266343ec35db482239904b0ca1bd90c8fbca01
ganache          | (1) 0x7968992c32f33408f0abc9ab42aa95d77fb52524c2e0c62d1adef42840d3d502
ganache          | (2) 0xb492c13c93edc695916522966262ad4dc1c7f64a22b138504c5b250a1c484203
ganache          | (3) 0xb22df97c876895cec01b8ad70b00482e2201dbecc61482a9c1e44544436b1b04
ganache          | (4) 0x69ce909787a29af568092a6d1cb693a3b4510c6d4eebd7d01762aa5f4d824105
ganache          | (5) 0x867f938edd9a5598b40bcf2c68a3106c0fe7486c1a6d50ba9a5171dc865b5306
ganache          | (6) 0xe1921a270bd9751ccc2d04896c7f62cbb5d9cd5dc3195de3fc2d29c483f38507
ganache          | (7) 0x03e88e467120938d05ca0a2677a9a25080e1a16093655bdd3c8fd8c43bd4ba08
ganache          | (8) 0x3b270a195f592a505ce94186793ababb27a33a65a572bc9e7b06766fb6626b09
ganache          | (9) 0x6418670e00dbdb0a6e9bd99f1521f7294f58af0995c7a0534c2021fcd486e710
ganache          |
ganache          | Gas Price
ganache          | ==================
ganache          | 20000000000
ganache          |
ganache          | Gas Limit
ganache          | ==================
ganache          | 6721975
ganache          |
ganache          | Call Gas Limit
ganache          | ==================
ganache          | 9007199254740991
ganache          |
ganache          | Listening on 0.0.0.0:8545
```

Figure 33 Implementation of DLT managers.

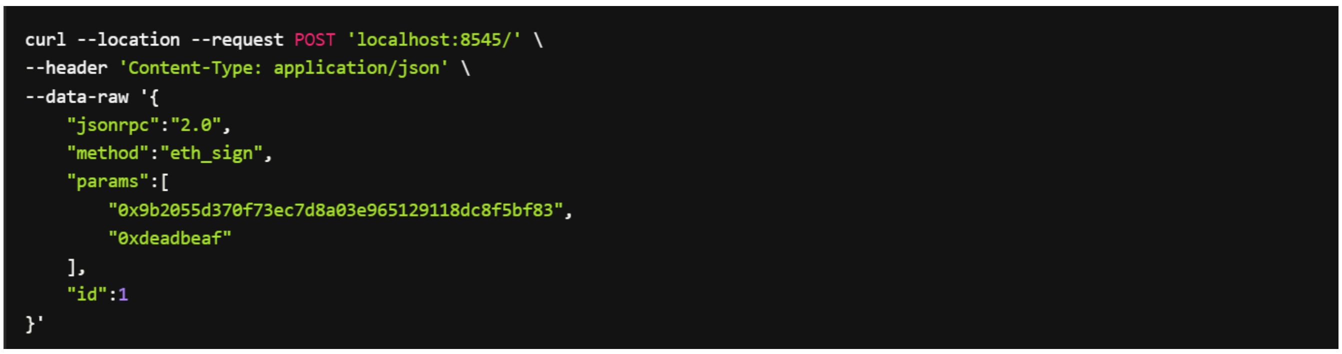
After building the DLT infrastructure for recording data and making transactions, we need to build distributed applications which execute autonomously in the network to satisfy the constraints and agreement of involved participants. The smart contract is written in the Solidity language and running in the DLT network. An example of a smart contract is shown in Figure 34 below:



```
// SPDX-L// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.4.22 <0.6.0;

contract owned {
    address owner;

    constructor() internal {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
}

contract mortal is owned {
    function destory() public onlyOwner {
        selfdestruct(msg.sender);
    }
}

contract IoTSmartContract is mortal {
    event records(address indexed _from, bytes time, bytes temp, bytes hum);
    event led(address indexed _from, address indexed _to, uint8 color);

    function add_records(
        bytes memory time,
        bytes memory temp,
        bytes memory hum
    ) public {
        emit records(msg.sender, time, temp, hum);
    }

    function control_led(address _to, uint8 color) public {
        emit led(msg.sender, _to, color);
    }
}

```

Figure 34: Smart contract of recording sensing data.

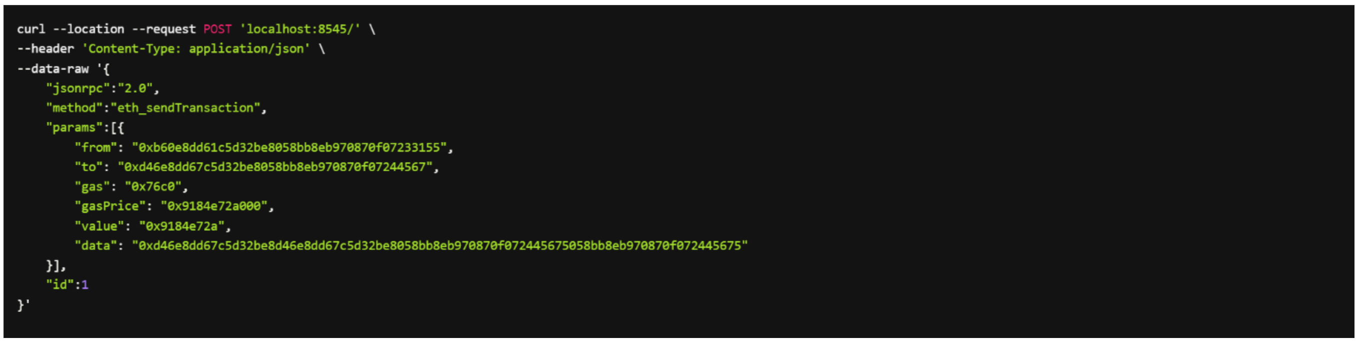
A typical smart contract consists of various information, for example, the owner of the devices, data, and predefined rules by system administrators, agreements between involved components in the system. Based on the scenario and specific requirements, the content of a smart contract will be written and run autonomously in the network. communicate with smart contracts, Ethereum provides an API which allows DLT clients to connect and communicate with smart contracts based on the predefined rules. DLT-nodes which want to publish data to the distributed ledger or instruct smart contracts for doing a task, first need to generate a DLT transaction format and sign it with JSON-RPC as:



```
curl --location --request POST 'localhost:8545/' \
--header 'Content-Type: application/json' \
--data-raw '{
    "jsonrpc":"2.0",
    "method":"eth_sign",
    "params":[
        "0x9b2055d370f73ec7d8a03e965129118dc8f5bf83",
        "0xdeadbeaf"
    ],
    "id":1
}'
```

Figure 35: Signing a transaction.

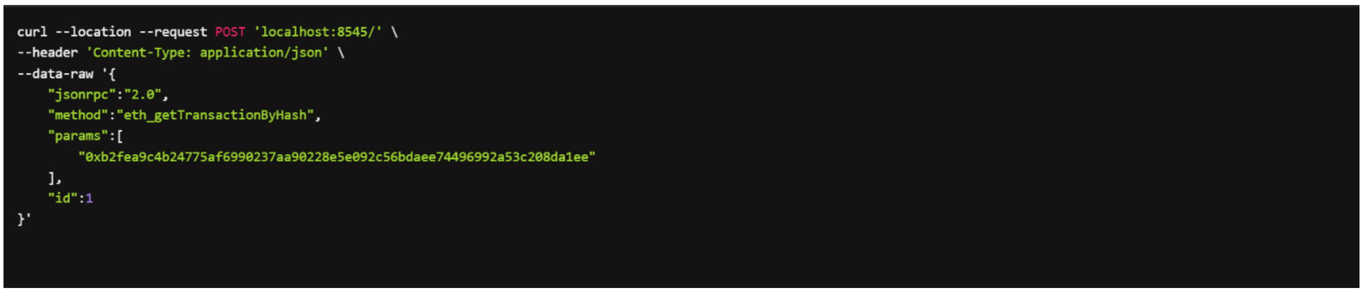
After signing, the transaction can be sent to the distributed ledger via *sendTransaction* as below:



```
curl --location --request POST 'localhost:8545/' \
--header 'Content-Type: application/json' \
--data-raw '{
    "jsonrpc":"2.0",
    "method":"eth_sendTransaction",
    "params":[{
        "from": "0xb60e8dd61c5d32be8058bb8eb970870f07233155",
        "to": "0xd46e8dd67c5d32be8058bb8eb970870f07244567",
        "gas": "0x76c0",
        "gasPrice": "0x9184e72a000",
        "value": "0x9184e72a",
        "data": "0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8eb970870f072445675"
    }],
    "id":1
}'
```

Figure 36: Sending a transaction from clients to distributed ledger.

The metadata of transaction shows the address of sender and destination, the gas fee to make the transaction, and the (hashed) data. Any human customer of the system (e.g., the one asking for a manufacturing task in UC1 or the one renting the tractor in UC3) can query the data recorded in the distributed ledger by using the transaction hash or transaction ID as well as the whole block ID if needed.



```
curl --location --request POST 'localhost:8545/' \
--header 'Content-Type: application/json' \
--data-raw '{
    "jsonrpc":"2.0",
    "method":"eth_getTransactionByHash",
    "params":[
        "0xb2fea9c4b24775af6990237aa90228e5e092c56bdaee74496992a53c208da1ee"
    ],
    "id":1
}'
```

Figure 37 Querying transaction details.

Integration with the SAP is shown in 2.4.1.

2.5 Infrastructure Management Components

2.5.1 EDGE MANAGEMENT

2.5.1.1 EDGE ORCHESTRATOR

The Edge Orchestrator is described in D4.1. It is a software component, which uses multiple other components. In the following, we describe the test and integration concepts for those components.

The Edge Orchestrator itself follows a microservice architecture. The single services provide specific system functionality. For example, we have services for hosting the API, the database, the input processing and many other system services. We are currently not following a strict test process to guarantee the functionality, however, we support different test cases, which include relevant components tests, so the integrity of the system can be checked. How these are integrated into a CI/CD pipeline is not yet decided.

The API, which is relevant to integrate the Edge Orchestrator with other systems within IntellIoT, is available as an OpenAPI specification. Client applications can use this document to generate client code for various programing platforms. The specification can also be used to generate test servers to facilitate the development of client applications. The software suite of swagger.io provides different tools to enable integrations. For example, with swagger codegen code can be generated for client as well as server applications. If required, a mock server of the Edge Orchestrator can be deployed as a test platform. It has to be kept in mind that a productive system, accessing physical devices, cannot be provided for general tests. Accordingly, we will schedule live tests during IntellIoT's integration workshops, e.g., during a plenary meeting.

Other components, as the Edge Manager, will be exclusively used by the Edge Orchestrator in a remote-controlled fashion. As this component is not used by external systems, no distinct integration or test plans are foreseen. The testing will be covered by the test concepts of the Edge Orchestrator.

Edge Devices are managed by the Edge Manager and won't need specific integration and testing concepts either.

Edge Apps are specific components of IntellIoT. Test concepts of Edge Apps will be the responsibility by the Edge App owners. However, we provide an Edge Deployment Tool, which takes specifications as input and outputs a configuration, which can be directly deployed within Edge Infrastructure. This means, if Edge Apps are on-boarded or updated, the process of deployment on a live system is widely automatized.

2.5.1.2 EDGE MANAGER

The Edge Manager is an off-the-shelf component used by the Edge Orchestrator. The component itself has product-grade and accordingly need not to be tested explicitly. We focus on the testing of the integration of the Edge Orchestrator with the Edge Manager. Here, we implemented regression tests to identify potential issues, which could lead us to incompatibilities due to software updates of the edge manager or the operating system of the edge devices.

2.5.2 DYNAMIC NETWORK MANAGEMENT

2.5.2.1 TSN CONTROLLER

The TSN controller is an edge application realized mainly in Python and JavaScript. It receives so-called communication service requests, also known as stream requests, via its northbound interface. In IntellIoT's manufacturing use case, this HTTP-based interface will mainly be used by the edge orchestrator. For test purposes, any HTTP client, e.g., Postman[8], can be used, see e.g., Figure 38. Additionally, a Web UI is available for convenient manual testing, see e.g., Figure 40 and Figure 41. A description of the interface is given in in https://gitlab.eurecom.fr/intelliot-project/networking/tsn-controller and valid requests are available as a Postman collection and in Swagger.

To test the TSN controller, a testbed is available at the Siemens site in Munich Perlach. The TSN controller gathers topology information via LLDP and netconf and presents topology information in the Web UI (Figure 39) and via its HTTP interface (Figure 40, Figure 41). In both cases, information about available resources and resources already reserved for embedded streams are given. As a first step to test the TSN controller, we manually check the correctness of this information in the Web UI and via the HTTP interface.

[8] https://www.postman.com/

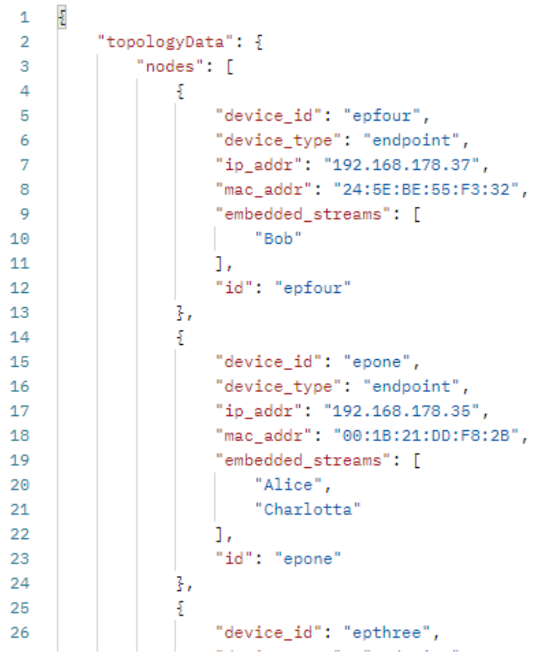
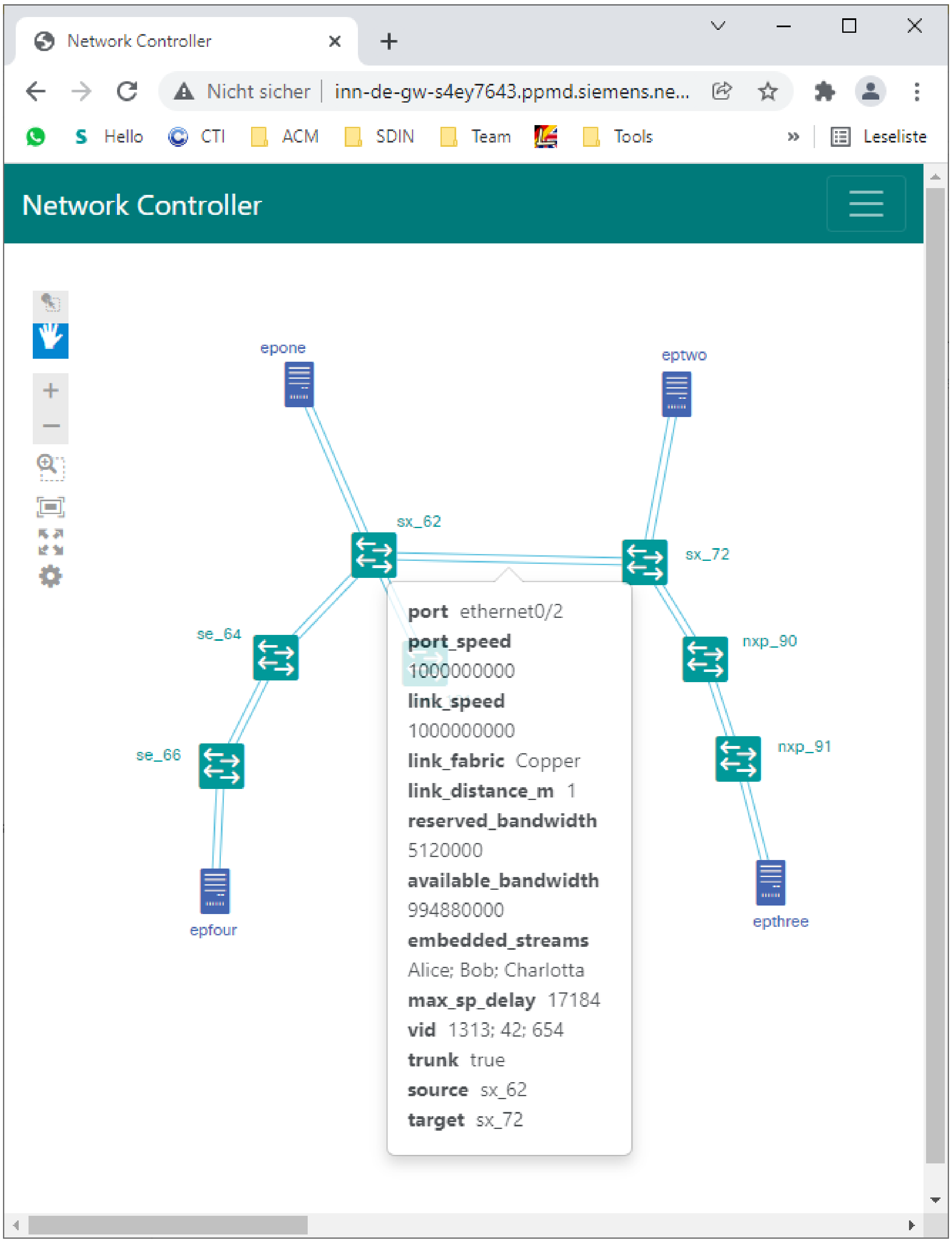

```
{
    "topologyData": {
        "nodes": [
            {
                "device_id": "epfour",
                "device_type": "endpoint",
                "ip_addr": "192.168.178.37",
                "mac_addr": "24:5E:BE:55:F3:32",
                "embedded_streams": [
                    "Bob"
                ],
                "id": "epfour"
            },
            {
                "device_id": "epone",
                "device_type": "endpoint",
                "ip_addr": "192.168.178.35",
                "mac_addr": "00:1B:21:DD:F8:2B",
                "embedded_streams": [
                    "Alice",
                    "Charlotta"
                ],
                "id": "epone"
            },
            {
                "device_id": "epthree",
```

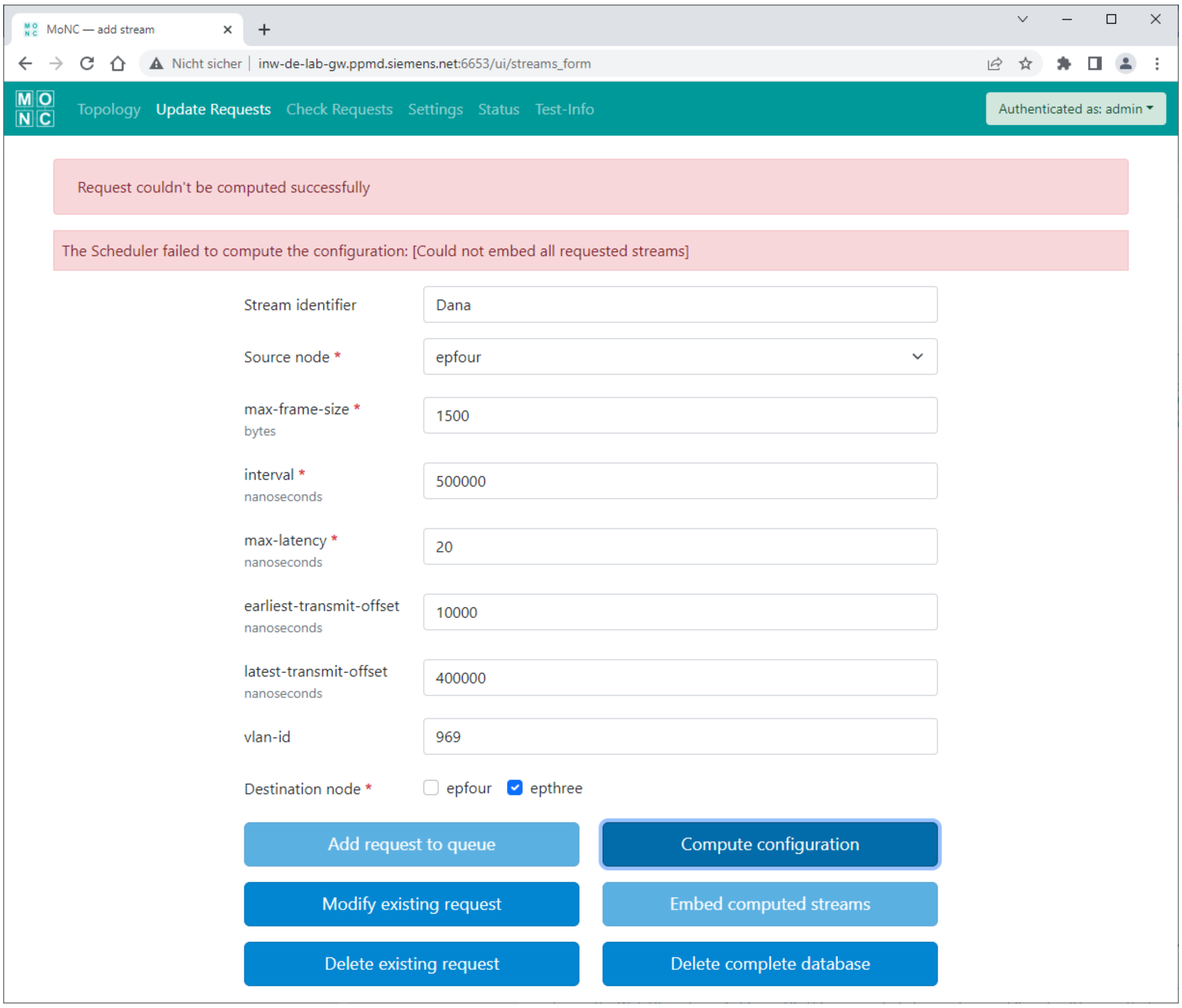
Figure 38: Topology view with stream information



port ethernet0/2
port_speed 1000000000
link_speed 1000000000
link_fabric Copper
link_distance_m 1
reserved_bandwidth 5120000
available_bandwidth 994880000
embedded_streams Alice; Bob; Charlotta
max_sp_delay 17184
vid 1313; 42; 654
trunk true
source sx_62
target sx_72

Figure 39: Topology view with stream information in Web UI.

For a standalone test, we send communication service requests, i.e., requests to embed streams, to the TSN controller via its HTTP API and Web UI, see Figure 40. We then request the TSN controller to compute the schedule and embed it to the devices. We will send feasible and infeasible requests, like e.g., the request for a stream called "Dana" in Figure 40, where we requested an infeasible end-to-end latency of only 20 nanoseconds to demonstrate that the feasibility checks function as intended.



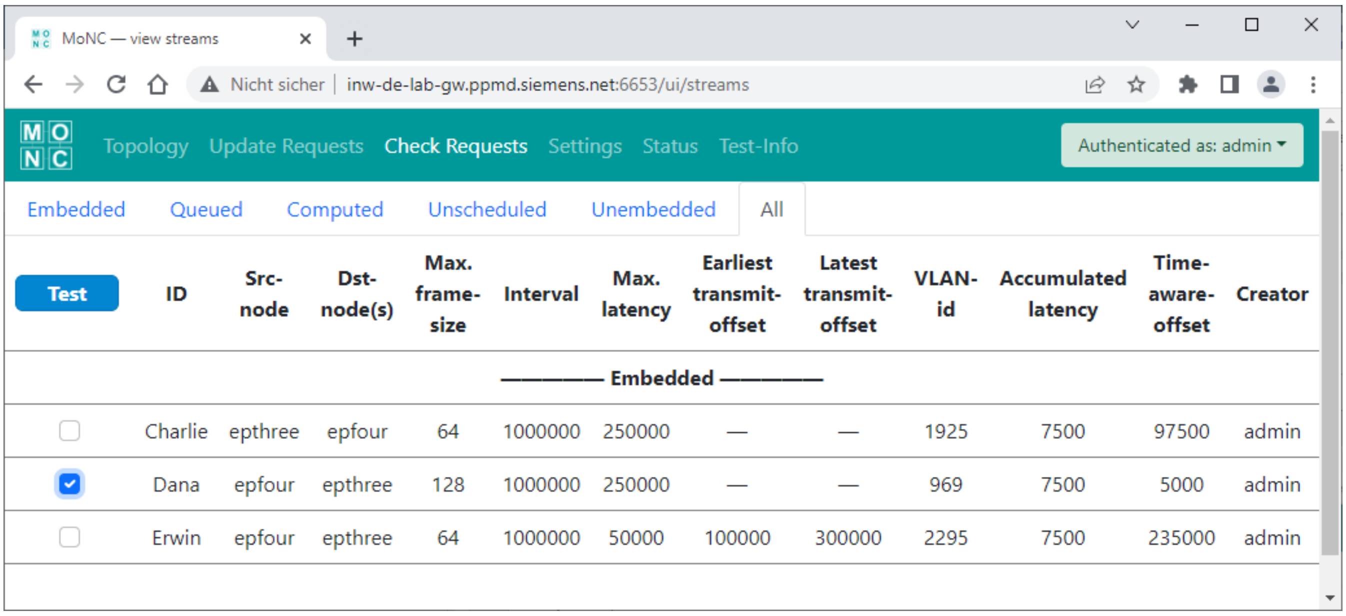
Request couldn't be computed successfully

The Scheduler failed to compute the configuration: [Could not embed all requested streams]

Stream identifier	Dana
Source node *	epfour
max-frame-size * bytes	1500
interval * nanoseconds	500000
max-latency * nanoseconds	20
earliest-transmit-offset nanoseconds	10000
latest-transmit-offset nanoseconds	400000
vlan-id	969
Destination node *	☐ epfour ☑ epthree

Figure 40: Communication service request via the Web UI.

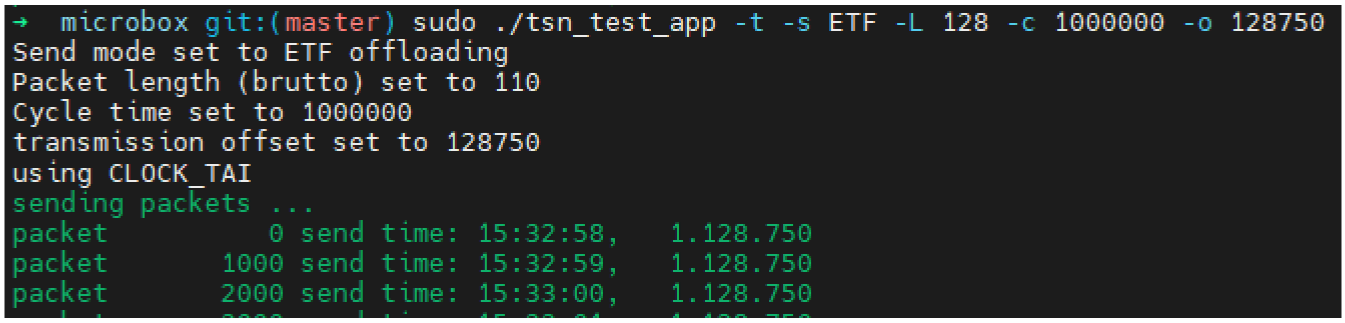
For feasible requests, we first check if the computed QoS parameters are within the requested ranges, e.g., using the Web UI as shown in Figure 41.



Test	ID	Src-node	Dst-node(s)	Max. frame-size	Interval	Max. latency	Earliest transmit-offset	Latest transmit-offset	VLAN-id	Accumulated latency	Time-aware-offset	Creator
						———— Embedded ————						
☐	Charlie	epthree	epfour	64	1000000	250000	—	—	1925	7500	97500	admin
☑	Dana	epfour	epthree	128	1000000	250000	—	—	969	7500	5000	admin
☐	Erwin	epfour	epthree	64	1000000	50000	100000	300000	2295	7500	235000	admin

Figure 41: Communication service details in Web UI.

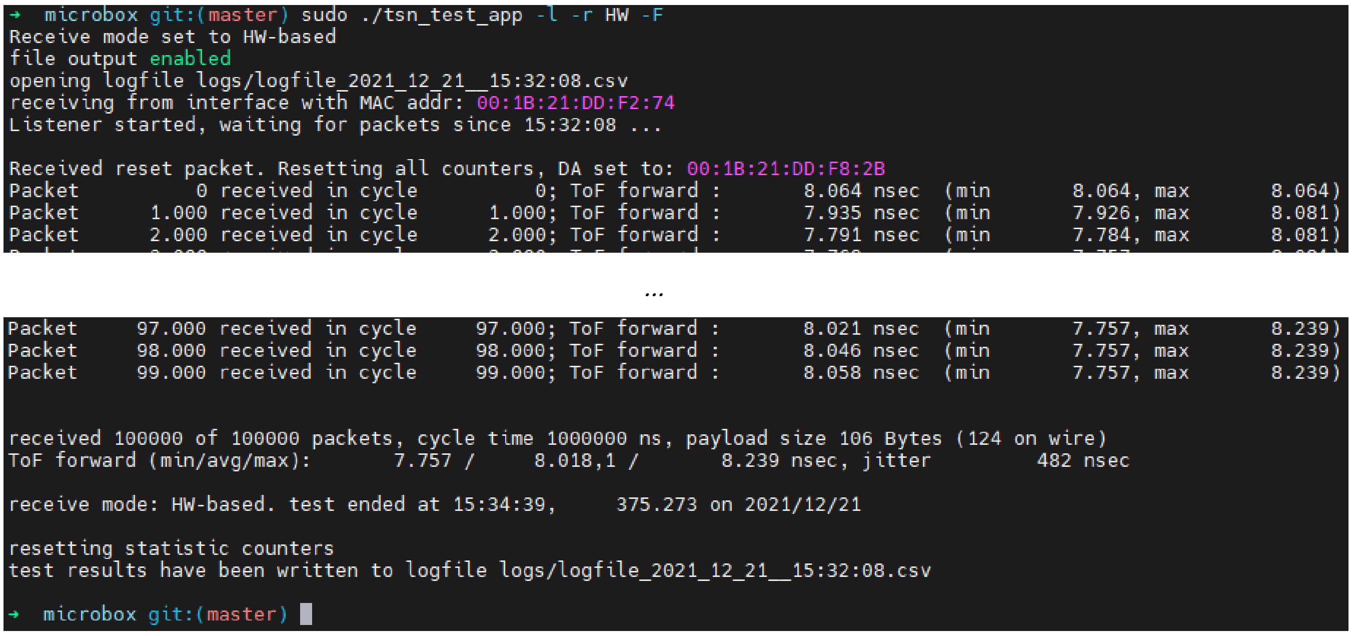
To check whether the computed schedules were configured in the devices properly, we use the Siemens-proprietary "TSN test app" to measure the real end-to-end delay for computed and embedded streams. For this manual test, we start the TSN test app with the parameters given in the stream request, e.g. cycle time 1ms and frame size 128 Bytes for the stream called "Dana" in Figure 41, and the additional parameters computed by the TSN controller, e.g. the transmission offset on the end station sending the stream, see Figure 42. Instead of manually transferring the parameters to the test app, we can start the test app on all concerned end points in so-called agent mode, mark the stream(s) to test in the web-UI and klick the "Test"-button, see Figure 41. The network controller will then send the required parameters to the test app agents via a socket connection and start the test.



```
➜  microbox git:(master) sudo ./tsn_test_app -t -s ETF -L 128 -c 1000000 -o 128750
Send mode set to ETF offloading
Packet length (brutto) set to 110
Cycle time set to 1000000
transmission offset set to 128750
using CLOCK_TAI
sending packets ...
packet          0 send time: 15:32:58,    1.128.750
packet       1000 send time: 15:32:59,    1.128.750
packet       2000 send time: 15:33:00,    1.128.750
```

Figure 42: TSN test app, sender side.

The TSN test app on the receiver side Figure 42 provides statistics about delay, jitter, and packet loss of the test packets sent according to the configured stream parameters. With this we can easily check if the actual QoS parameters of the requested streams match the requests. Please note that the displayed frame length seems incorrect at the receiver, because the Linux network stack has removed the VLAN tag before the test app receives the packet and computes the packet length.



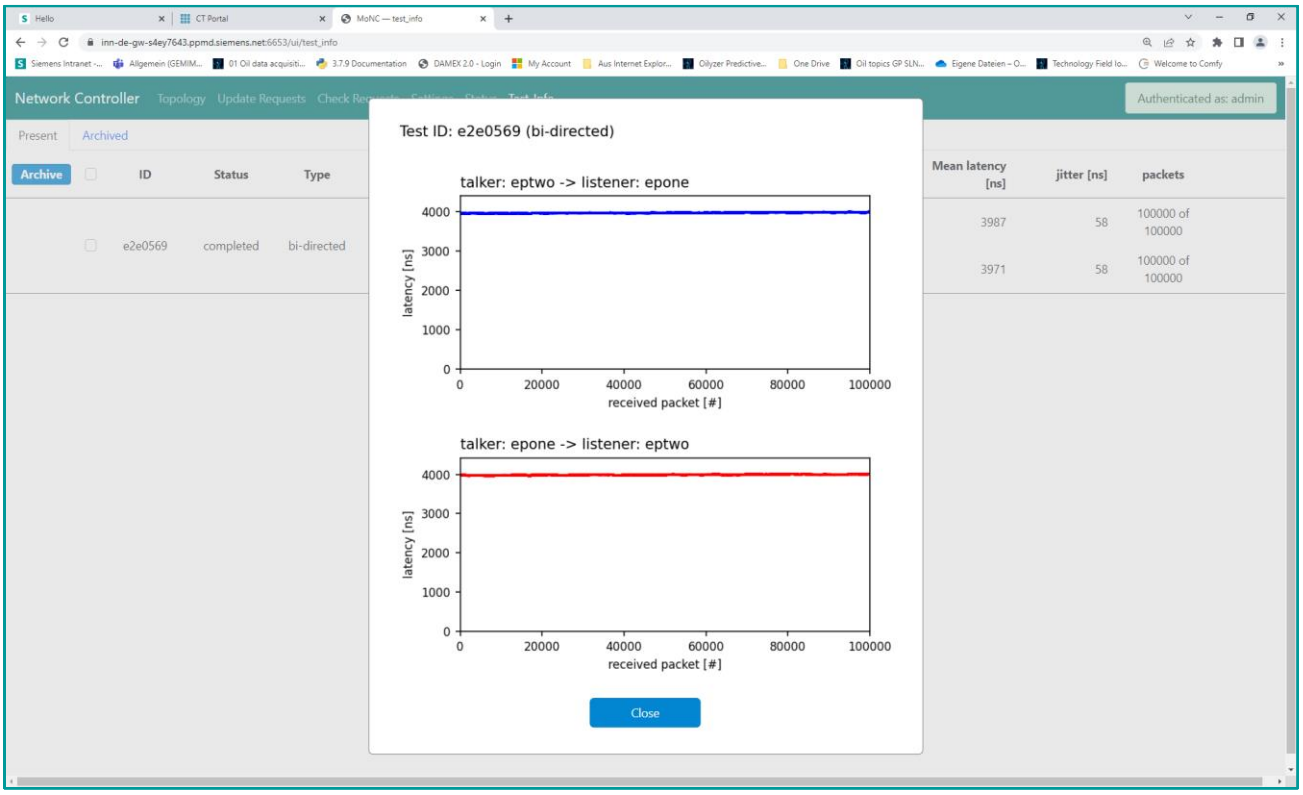
```
➜  microbox git:(master) sudo ./tsn_test_app -l -r HW -F
Receive mode set to HW-based
file output enabled
opening logfile logs/logfile_2021_12_21__15:32:08.csv
receiving from interface with MAC addr: 00:1B:21:DD:F2:74
Listener started, waiting for packets since 15:32:08 ...

Received reset packet. Resetting all counters, DA set to: 00:1B:21:DD:F8:2B
Packet          0 received in cycle          0; ToF forward :       8.064 nsec  (min       8.064, max       8.064)
Packet      1.000 received in cycle      1.000; ToF forward :       7.935 nsec  (min       7.926, max       8.081)
Packet      2.000 received in cycle      2.000; ToF forward :       7.791 nsec  (min       7.784, max       8.081)
```

...

```
Packet     97.000 received in cycle     97.000; ToF forward :       8.021 nsec  (min       7.757, max       8.239)
Packet     98.000 received in cycle     98.000; ToF forward :       8.046 nsec  (min       7.757, max       8.239)
Packet     99.000 received in cycle     99.000; ToF forward :       8.058 nsec  (min       7.757, max       8.239)


received 100000 of 100000 packets, cycle time 1000000 ns, payload size 106 Bytes (124 on wire)
ToF forward (min/avg/max):       7.757 /     8.018,1 /       8.239 nsec, jitter         482 nsec

receive mode: HW-based. test ended at 15:34:39,     375.273 on 2021/12/21

resetting statistic counters
test results have been written to logfile logs/logfile_2021_12_21__15:32:08.csv

➜  microbox git:(master) █
```

Figure 43: TSN test app, receiver side.

If the test has been started via the TSN controller's web-UI, the test results can also be observed directly in this web-UI, including a graphical representation of the results, see Figure 44.



Test ID: e2e0569 (bi-directed)

ID	Status	Type	Mean latency [ns]	jitter [ns]	packets
e2e0569	completed	bi-directed	3987	58	100000 of 100000
			3971	58	100000 of 100000

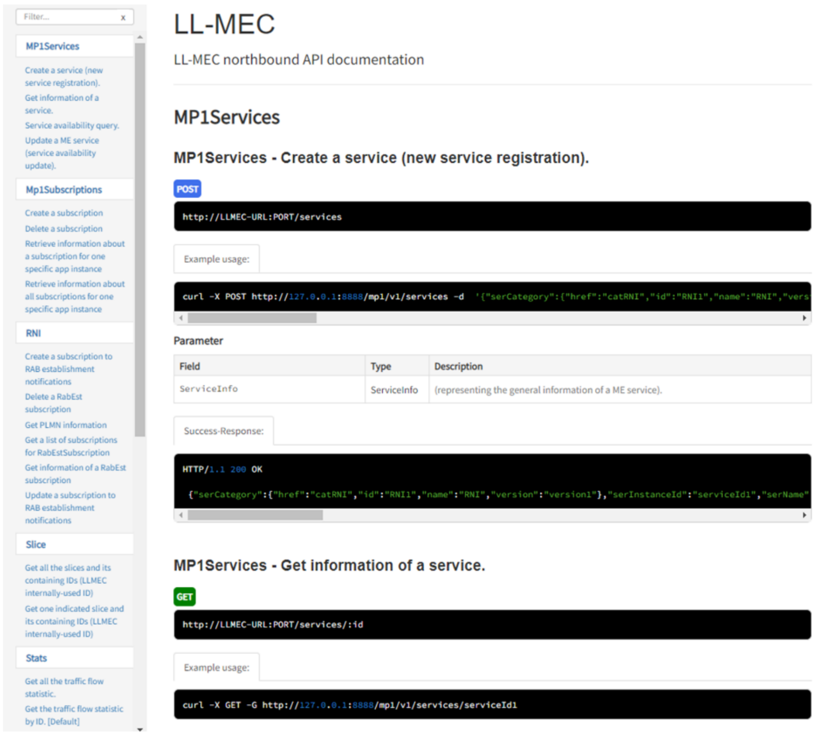
Figure 44: Test results displayed in TSN controller's web-UI

An additional feature to exclude malicious nodes from any communication has been developed and is available on supported devices.

2.5.2.2 COMMUNICATION RESOURCE MANAGER

The 5G Communication Resource Manager is an IntellIoT component aiming at controlling the communication resources allocated by the 5G network. Following the architecture description of the Mosaic5G FlexRIC and FlexCN, the 5G Communication Resource Manager is an xApp and controls the FlexRAN/FlexRIC and LL-MEC/FlexCN components to enable dynamic 5G slice creation, resource optimization as well as dynamic adaptation. It is deployed by the Edge Orchestrator component in the context of UC1. Specifically, it is the responsible of opening, managing, and closing 5G RAN. The FlexRAN/FlexRIC/LL-MEC/FlexCN as well as xAPPs are software codes operating on any off-the-shelf laptop or PC supporting containerization (docker, kubernetes) available from https://gitlab.eurecom.fr/mosaic5g.

The Communication Resource Manager communicates with the LL-MEC/FlexCN over an HTTP API as illustrated on Figure 45 (for LL-MEC). Similarly, the Communication Resource Manager communicates with the FlexRAN/FlexRIC over an HTTP API as illustrated on Figure 46 (for FlexRAN).


Filter...
x
MP1Services
Create a service (new service registration).
Get information of a service.
Service availability query.
Update a ME service (service availability update).
Mp1Subscriptions
Create a subscription
Delete a subscription
Retrieve information about a subscription for one specific app instance
Retrieve information about all subscriptions for one specific app instance
RNI
Create a subscription to RAB establishment notifications
Delete a RabEst subscription
Get PLMN information
Get a list of subscriptions for RabEstSubscription
Get information of a RabEst subscription
Update a subscription to RAB establishment notifications
Slice
Get all the slices and its containing IDs (LLMEC internally-used ID)
Get one indicated slice and its containing IDs (LLMEC internally-used ID)
Stats
Get all the traffic flow statistic.
Get the traffic flow statistic by ID. [Default]

LL-MEC

LL-MEC northbound API documentation

MP1Services

MP1Services - Create a service (new service registration).

POST

```
http://LLMEC-URL:PORT/services
```

Example usage:

```
curl -X POST http://127.0.0.1:8888/mp1/v1/services -d  '{"serCategory":{"href":"catRNI","id":"RNI1","name":"RNI","vers
```

Parameter

Field	Type	Description
ServiceInfo	ServiceInfo	(representing the general information of a ME service).

Success-Response:

```
HTTP/1.1 200 OK

 {"serCategory":{"href":"catRNI","id":"RNI1","name":"RNI","version":"version1"},"serInstanceId":"serviceId1","serName"
```

MP1Services - Get information of a service.

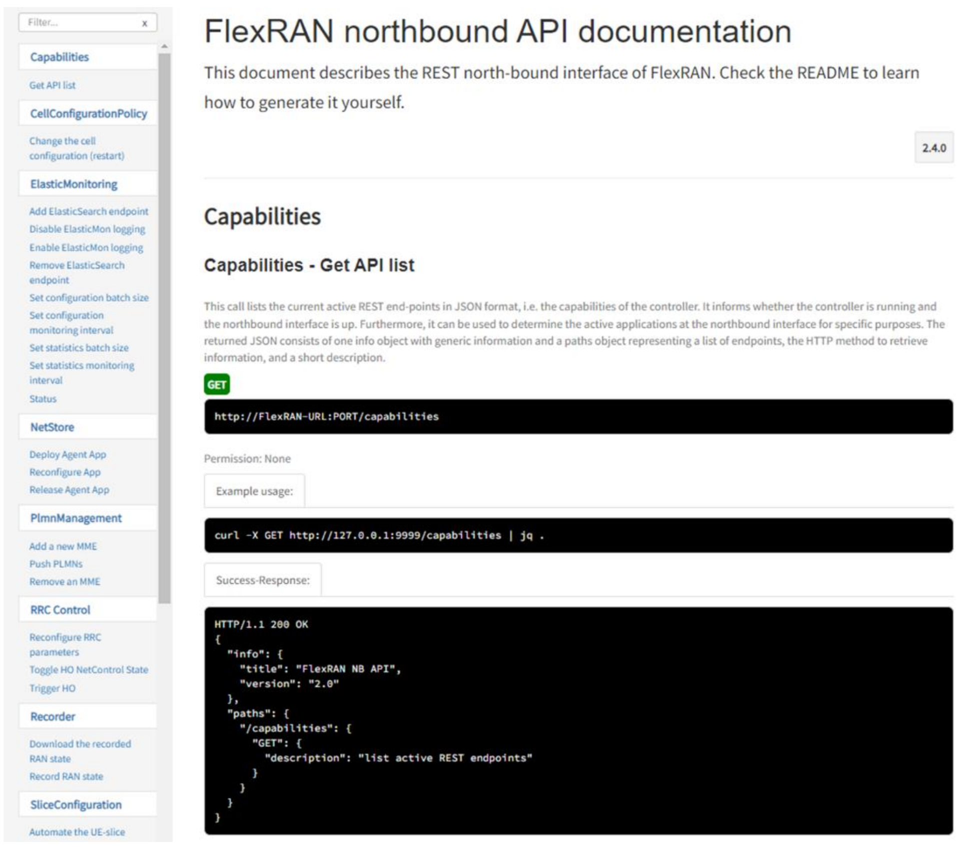
GET

```
http://LLMEC-URL:PORT/services/:id
```

Example usage:

```
curl -X GET -G http://127.0.0.1:8888/mp1/v1/services/serviceId1
```

Figure 45. LL-MEC HTTP API available to the communication resource manager.



Capabilities
Get API list
CellConfigurationPolicy
Change the cell configuration (restart)
ElasticMonitoring
Add ElasticSearch endpoint
Disable ElasticMon logging
Enable ElasticMon logging
Remove ElasticSearch endpoint
Set configuration batch size
Set configuration monitoring interval
Set statistics batch size
Set statistics monitoring interval
Status
NetStore
Deploy Agent App
Reconfigure App
Release Agent App
PlmnManagement
Add a new MME
Push PLMNs
Remove an MME
RRC Control
Reconfigure RRC parameters
Toggle HO NetControl State
Trigger HO
Recorder
Download the recorded RAN state
Record RAN state
SliceConfiguration
Automate the UE-slice

FlexRAN northbound API documentation

This document describes the REST north-bound interface of FlexRAN. Check the README to learn how to generate it yourself.

Capabilities

Capabilities - Get API list

This call lists the current active REST end-points in JSON format, i.e. the capabilities of the controller. It informs whether the controller is running and the northbound interface is up. Furthermore, it can be used to determine the active applications at the northbound interface for specific purposes. The returned JSON consists of one info object with generic information and a paths object representing a list of endpoints, the HTTP method to retrieve information, and a short description.

```
http://FlexRAN-URL:PORT/capabilities
```

Permission: None

Example usage:

```
curl -X GET http://127.0.0.1:9999/capabilities | jq .
```

Success-Response:

```
HTTP/1.1 200 OK
{
  "info": {
    "title": "FlexRAN NB API",
    "version": "2.0"
  },
  "paths": {
    "/capabilities": {
      "GET": {
        "description": "list active REST endpoints"
      }
    }
  }
}
```

Figure 46. FlexRAN HTTP API available to the communication resource manager.

During Cycle 1, a local setup with a simulated radio-level environment has been developed (and available for further testing during Cycle 2) for testing the initial implementation. Initially, the slicing management was supported by the FlexRAN API, available at the beginning of the project, and this has been extended to theFlexRIC API during Cycle 2.

1. There is a default slice that any new UE is subscribed to, and all UEs share these resources equally, in a best-effort manner. Beyond the default slice there will also exist other slices meant to be tailored towards the needs of the UEs. These slices are meant to ensure that UEs with specific needs can have a guaranteed set of resources like bandwidth allocated to them, independent of the number of UEs connected to the network. These slices are referred to as custom slices.
2. The following functionality is available and has been tested in Cycle 2:reating a network slice by requesting a set of resources for a user through choosing its Radio Network Temporary Identifier (RNTI)(i.e. a pseudo-random ID that is generated and granted to a UE). The slice created only applies in the downlink, as uplink slices have not yet been implemented on the gNB. Upon creating a network slice an identifier is returned identifying the slice. It works by creating a HTTP GET request.
3. Removing a network slice. It works by using a HTTP DELETE request. In this form the slice ID must be sent along with the request.
4. Associating a UE, or new UEs to an existing slice by using the slice ID and the RNTI of the UE to add. This will mean that the resources are to be shared between all occupants on a slice. 4. When creating a slice one of the desired inputs has been a requested bitrate that should be available on the slice. This has been implemented by monitoring the radio link quality and allocating a set of physical resources to ensure that. When changes to the radio link occur, a reallocation may occur to ensure the slice guarantee. An example of this is visualised on Figure 32. This is done with a frequency to ensure the guaranteed bitrate is upheld.
5. Taking an existing slice configuration, it can be updated to support a different requested bitrate.

6. Requesting information on a specific slice by inputting its slice ID, or by requesting all the currently active slices that are being supported by the BS.

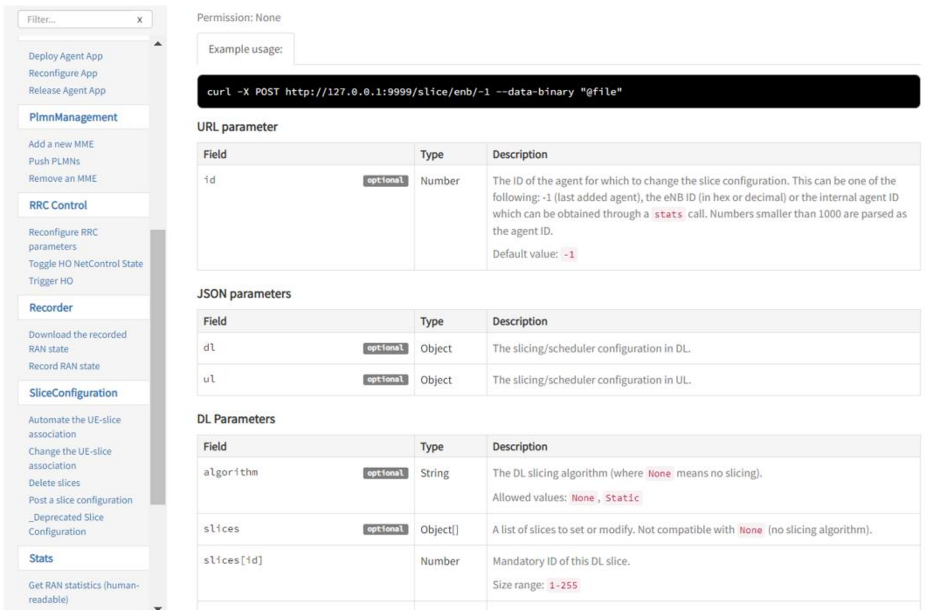
Some of these functionalities are directly exposed to be used through HTTP request, while others happen in the background of actively used for the creation and upholding of a slice definition. The interface with the Communication resource manager has been designed and implemented as a python-flask server and by using swagger for interface design. Interaction is with a set of predefined interfaces that take certain inputs to create, delete, or get information on a network slice.

2.5.2.3 5G RAN CONTROLLER

The 5G RAN controller is implemented in OAI as FlexRAN for 4G+ or FlexRIC for 5G RAN architectures. FlexRAN connects to OAI eNB to retrieve RAN-related data. The FlexRAN agent has a northbound and a southbound API.

The Northbound API is implemented as an HTTP API and provides the following HTTP endpoints. The SouthAPIs is implemented as:

- GetConfig – retrieves 4G/5G RAN configuration, such as UL/DL RAN, DRB (data radio bearers) or measurement configurations.
- GetStat – retrieves 4G/5G RAN statistics, such as Channel Quality Indicator (CQI), SINR/RSSI measurements / Localization or UL/DL performance at specific layer.
- Commands – the most important API, which actuates the 4G/5G RAN according to specific instructions. It is particularly used for slice creation and per-slice internal configuration.
- EventTrigger – the 4G/5G RAN provides to the FlexRAN controller specific information, such as UE attachment, Scheduling Request.
- Control Delegation – a particular function, which delegate control to the 4G/5G RAN to reduce the latency. Typical delegations are new 4G/5G or handover scheduler, or updated parameters of the existing schedulers or handover algorithm.


Filter...
Deploy Agent App
Reconfigure App
Release Agent App
PlmnManagement
Add a new MME
Push PLMNs
Remove an MME
RRC Control
Reconfigure RRC parameters
Toggle HO NetControl State
Trigger HO
Recorder
Download the recorded RAN state
Record RAN state
SliceConfiguration
Automate the UE-slice association
Change the UE-slice association
Delete slices
Post a slice configuration
_Deprecated Slice Configuration
Stats
Get RAN statistics (human-readable)

Permission: None

Example usage:

```
curl -X POST http://127.0.0.1:9999/slice/enb/-1 --data-binary "@file"
```

URL parameter

Field	Type	Description
id optional	Number	The ID of the agent for which to change the slice configuration. This can be one of the following: -1 (last added agent), the eNB ID (in hex or decimal) or the internal agent ID which can be obtained through a `stats` call. Numbers smaller than 1000 are parsed as the agent ID. Default value: `-1`

JSON parameters

Field	Type	Description
dl optional	Object	The slicing/scheduler configuration in DL.
ul optional	Object	The slicing/scheduler configuration in UL.

DL Parameters

Field	Type	Description
algorithm optional	String	The DL slicing algorithm (where `None` means no slicing). Allowed values: `None`, `Static`
slices optional	Object[]	A list of slices to set or modify. Not compatible with `None` (no slicing algorithm).
slices[id]	Number	Mandatory ID of this DL slice. Size range: `1-255`

Figure 47: Mosaic5G FlexRAN HTTP API.

Figure 47 depicts the REST API of the FlexRAN controller for a Control API for Slice creation. A full description of the FlexRAN API is available https://mosaic5g.io/apidocs/flexran/

The southbound interface of the FlexRAN controller is implemented according to a Google ProtoBuffer. It however is an internal API which is not exposed outside the FlexRAN component. The FlexRAN southbound API is available https://mosaic5g.io/apidocs/flexran/flexran_spec_v2.2.3.html.

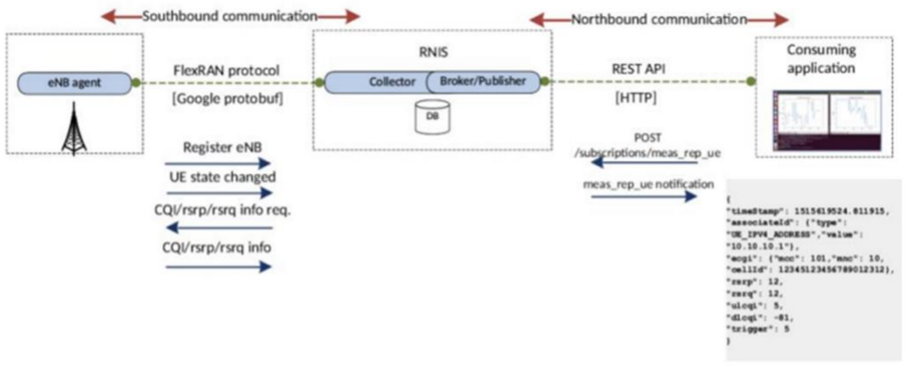
The OAI 5G RAN evolved to support O-RAN E2 agent and the respective APIs. Accordingly, Mosaic5G replaced the FlexRAN agent by a FlexRIC agent, which provides mostly the same HTTP APIs on the northbound interface but replacing the FlexRAN by the E2 API on the southbound interface.

2.5.2.4 5G LOW LATENCY MEC

The main MEC service provided by the 5G low latency MEC is the Radio Network Information Service (RNIS). It is a multi-microservice docker service consisting of three functions:

- RAN data collector – it retrieves data from the 5G RAN over the FlexRAN API. As function of what is requested, it can also actively request specific RAN data.
- Database – the database stores RAN data for faster retrieval.
- Broker – A HTTP server which exposes HTTP APIs to POST or GET specific RNIS data.

Figure 48 illustrates the RNIS service between the 5G RAN and a consuming application, which can be either the Communication Manager or the Edge Controller. FlexRAN is specific to OAI 4G+ architecture and evolved toward a O-RAN compatible E2 agent. However, the RNIS API will not change and will hide the underlying RAN (5G or 4G+).



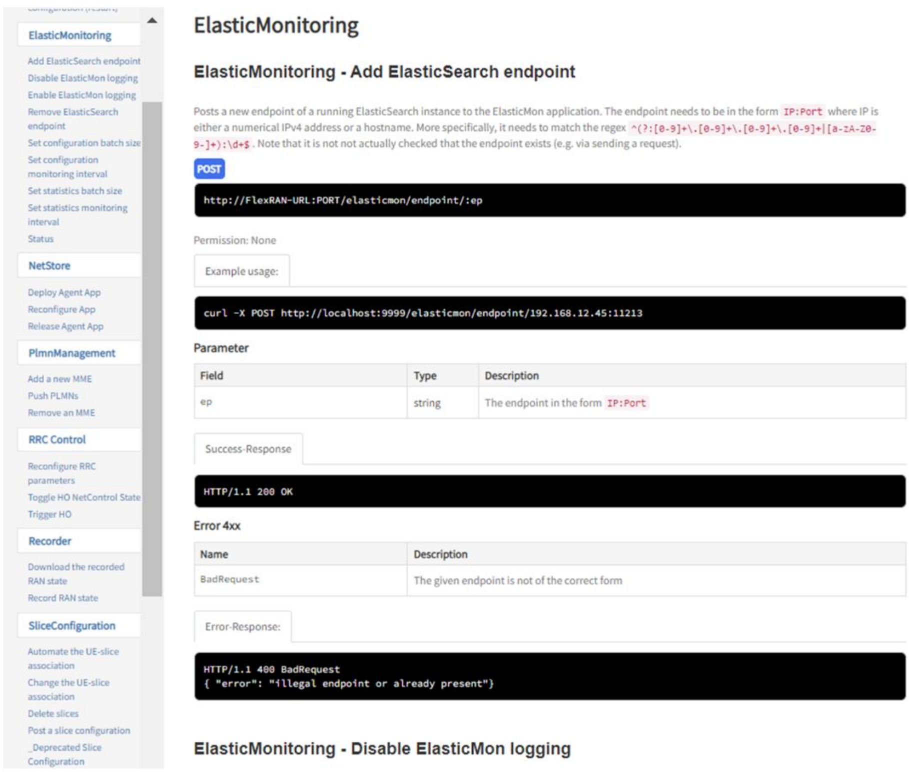
```
{
"timeStamp": 1515619524.811915,
"associateId": {"type":
"UE_IPV4_ADDRESS","value":
"10.10.10.1"},
"ecgi": {"mcc": 101,"mnc": 10,
"cellId": 1234512345678901231 2},
"rsrp": 12,
"rsrq": 12,
"ulcqi": 5,
"dlcqi": -81,
"trigger": 5
}
```

Figure 48: Mosaic5G FlexRAN-based RNIS service.

2.5.2.5 5G NETWORKING MONITORING

The 5G network monitoring can be done either by the RNIS service if monitoring is required as actuation. However, if monitoring is required also for statistics and visualization, elastic search technologies is provided by this component. It corresponds to an Elastic Monitoring component, which exposes Elastic Search APIs to enable or disable an Elastic Search endpoint and connects to Elastic technologies webservices, such as Kibana or Grafana.

The ElasticMonitoring application uses FlexRAN or Flex RIC E2 APIs on a southbound interface to obtain RAN statistics and provide an HTTP API on the northbound API to configure the required statistics and the endpoint to a running ElasticSearch instance.



ElasticMonitoring
Add ElasticSearch endpoint
Disable ElasticMon logging
Enable ElasticMon logging
Remove ElasticSearch endpoint
Set configuration batch size
Set configuration monitoring interval
Set statistics batch size
Set statistics monitoring interval
Status
NetStore
Deploy Agent App
Reconfigure App
Release Agent App
PlmnManagement
Add a new MME
Push PLMNs
Remove an MME
RRC Control
Reconfigure RRC parameters
Toggle HO NetControl State
Trigger HO
Recorder
Download the recorded RAN state
Record RAN state
SliceConfiguration
Automate the UE-slice association
Change the UE-slice association
Delete slices
Post a slice configuration
_Deprecated Slice Configuration

ElasticMonitoring

ElasticMonitoring - Add ElasticSearch endpoint

Posts a new endpoint of a running ElasticSearch instance to the ElasticMon application. The endpoint needs to be in the form `IP:Port` where IP is either a numerical IPv4 address or a hostname. More specifically, it needs to match the regex `^(?:[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+|[a-zA-Z0-9-]+):\d+$`. Note that it is not not actually checked that the endpoint exists (e.g. via sending a request).

POST

```
http://FlexRAN-URL:PORT/elasticmon/endpoint/:ep
```

Permission: None

Example usage:

```
curl -X POST http://localhost:9999/elasticmon/endpoint/192.168.12.45:11213
```

Parameter

Field	Type	Description
ep	string	The endpoint in the form `IP:Port`

Success-Response

```
HTTP/1.1 200 OK
```

Error 4xx

Name	Description
BadRequest	The given endpoint is not of the correct form

Error-Response:

```
HTTP/1.1 400 BadRequest
{ "error": "illegal endpoint or already present"}
```

ElasticMonitoring - Disable ElasticMon logging

Figure 49: Mosaic5G FlexRAN-based ElasticMonitoring HTTP API.

The HTTP API corresponding to enabling ElasticSearch endpoints as shown in Figure 49 is available at https://mosaic5g.io/apidocs/flexran/#api-ElasticMonitoring. A Mosaic5G FlexRIC-based monitoring component, supporting 5G RAN and compatible with O-RAN is under implementation. The North API will be extended with new E2 features from O-RAN.

2.5.2.6 PRIVATE 5G CORE

The private 5G core is a set of microservices representing 3GPP 5G Core functions and provided by OpenAirInterface CN (https://openairinterface.org/oai-5g-core-network-project/). The private 5G core supports both 4G and 5G core functions as depicted on Figure 50. It operates as a standalone entity which connects to the 4G or 5G RAN via the SGW-U/UFP and AMF/SMF/MME entities for user and control plane respectively. It is considered 'Private' first as it can operate without commercial cellular operators and second it can connect to a private data network (DN/PDN) on the premise it is installed, keeping any communications between the UEs and the data network private.

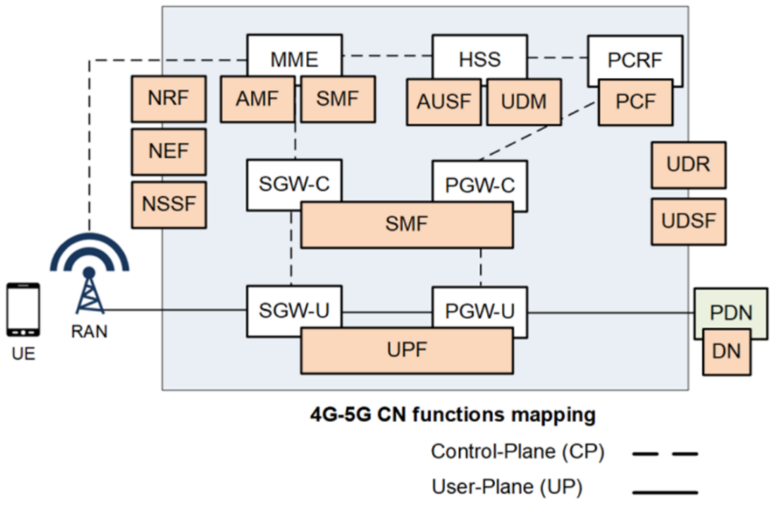
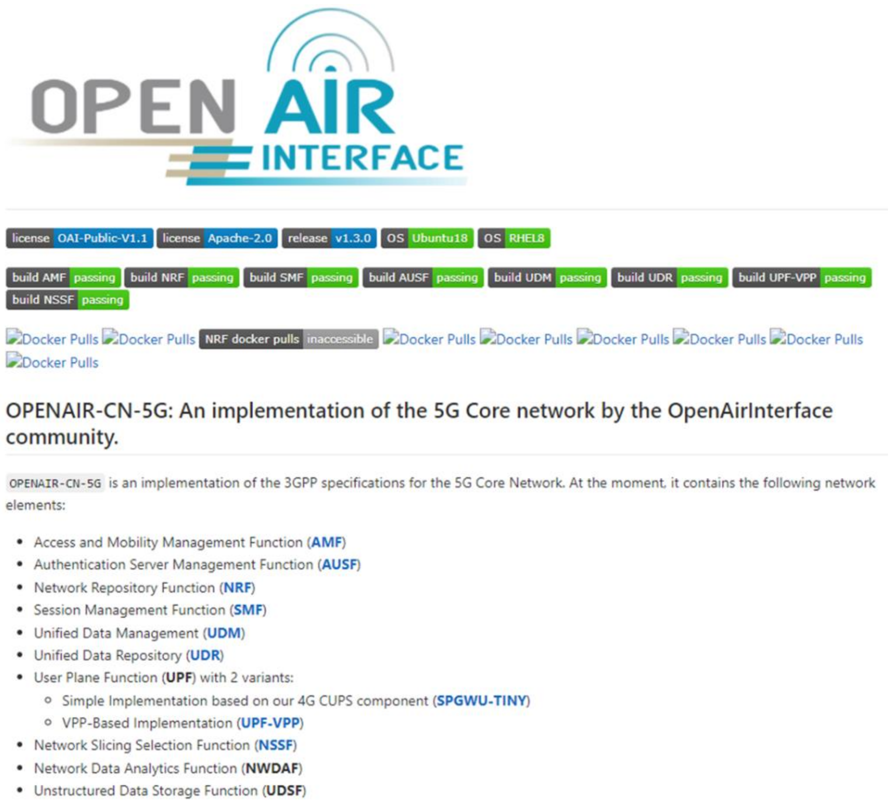


Figure 50: OpenAirInterface 4G/5G Core [source: OAI].

The OAI Private 5G Core can operate on any off-the-shelf PC supporting containerization (docker, kubernetes). The software code can be downloaded from EURECOM OAI GIT repository: https://gitlab.eurecom.fr/oai/cn5g as depicted in Figure 51.



OPENAIR-CN-5G: An implementation of the 5G Core network by the OpenAirInterface community.

`OPENAIR-CN-5G` is an implementation of the 3GPP specifications for the 5G Core Network. At the moment, it contains the following network elements:

- Access and Mobility Management Function (**AMF**)
- Authentication Server Management Function (**AUSF**)
- Network Repository Function (**NRF**)
- Session Management Function (**SMF**)
- Unified Data Management (**UDM**)
- Unified Data Repository (**UDR**)
- User Plane Function (**UPF**) with 2 variants:
  - Simple Implementation based on our 4G CUPS component (**SPGWU-TINY**)
  - VPP-Based Implementation (**UPF-VPP**)
- Network Slicing Selection Function (**NSSF**)
- Network Data Analytics Function (**NWDAF**)
- Unstructured Data Storage Function (**UDSF**)

Figure 51: OpenAirInterface 4G/5G Core software repository.

The Mosaic5G FlexCN/LL-MEC components interface with the Private 5G Core by reaching the dedicated 5G CN microservices (e.g. UPF, SMF, AMF, RNF) to dynamically adjust 5G CN functions to the requirements of higher services and applications.

2.5.2.7 5G RAN

The 5G RAN represents the wireless domain between a 5G User Equipment (UE) and a 5G base station (eNB/gNB). The 5G RAN is provided by OAI 5G RAN (https://openairinterface.org/oai-5g-ran-project/). The 5G RAN can operate either in non-standalone (NSA) or standalone modes, where the 5G RAN (eNB/gNB) would connect to a 4G Core or a 5G Core (respectively), as illustrated on Figure 52.

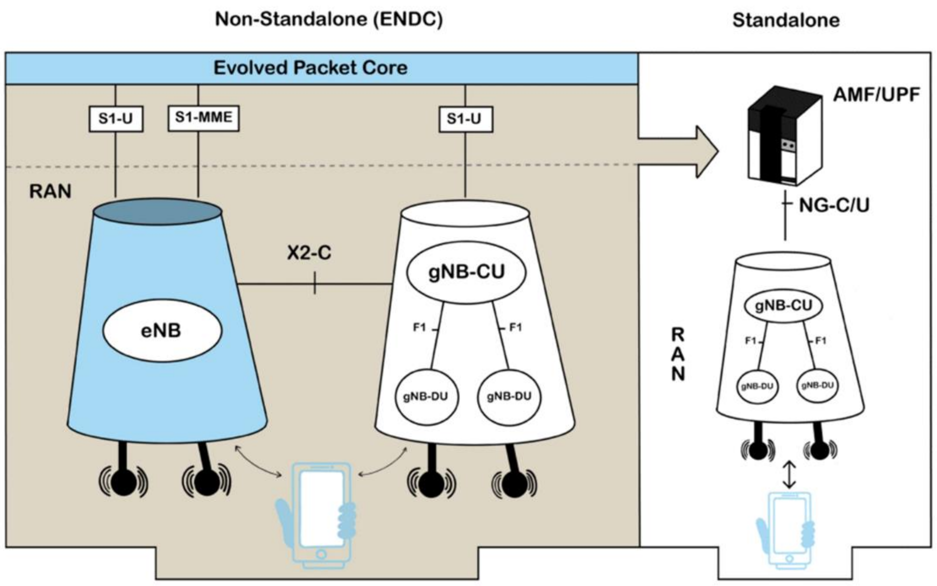


Figure 52: OpenAirInterface 5G RAN NSA and SA architecture (source: OAI).

The 5G RAN is a Software-Defined-Radio (SDR) architecture where RAN functions (RRC, PDCP, RLC, MAC, PHY) are implemented in software and a radio head (RRH) is used to handle the digital/analog and complex signal processing operations, including power amplification. The 5G RAN software contains a UE and a eNb/gNB side, which can be downloaded from EURECOM 5G RAN git repository (https://gitlab.eurecom.fr/oai/openairinterface5g/).

A 5G RAN UE is composed of an off-the-shelf 5G sub-6Ghz radio module connected to an off-the-shelf laptop containing the OAI 5G RAN UE software code, as illustrated on Figure 53.

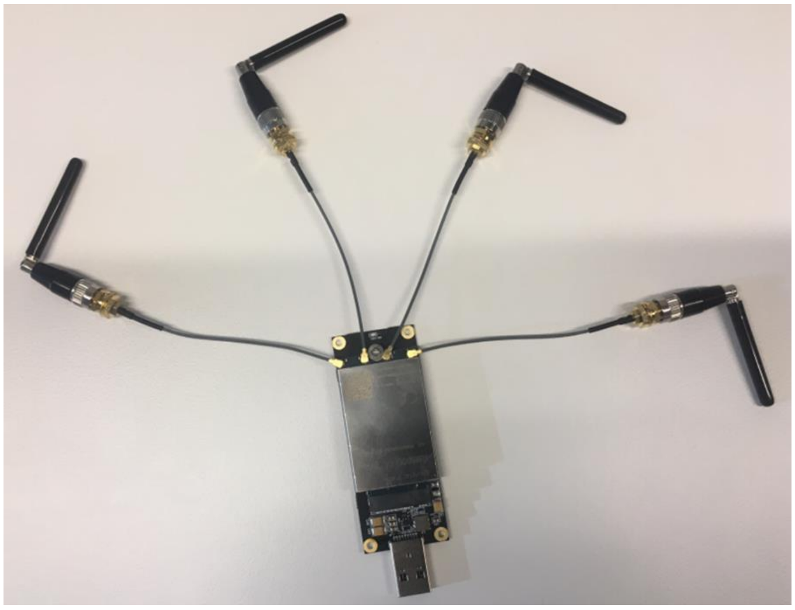


Figure 53:Illustration of the 5G RAN UE radio unit.

The 5G RAN eNb/gNB software code includes in addition to the 3GPP RAN functions (RRC, PDCP, RLC, MAC, PHY) a FlexRAN/E2 agent connected to the FlexRAN/FlexRIC module over a protobuf (FlexRAN) or O-RAN E2 API (FlexRIC). Accordingly, the 5G RAN can be dynamically re-configured or RAN-related information can be reported to the FlexRAN/FlexRIC component for dynamic RAN management.

A 5G RAN gNB is composed of a commercial sub-6GHz radio module (e.g. AW2S RRH - https://www.aw2s.com/RRU.html), connected to a PC containing the OAI 5G RAN gNB software code. 5G omnidirectional antennas are connected to the RRU. Figure 54 illustrates a typical RRU used by OAI 5G RAN.

Figure 54: Illustration of the 5G RAN gNB Remote Radio Unit (RRU) unit.

3 USE CASE SPECIFIC IMPLEMENTATIONS

This chapter corresponds to the 5th core group identified in IntellIoT's architecture (cf. D2.3/ *Figure 12*). We describe here the use case specific implementations of components. The specifications of the three use cases are described in D2.1.

3.1 UC1: Agriculture

3.1.1 TRACTOR CONTROLLER

The tractor controller deployed in the agriculture use case is the main control component on the tractor and hosts multiple edge apps, introduced in the sections above. The tractor controller is a hardware prototype, that is a ruggedized component explicitly designed for the off-highway domain, so it can stand the different environmental conditions these off-highway vehicles (e.g., tractors, harvesters, but also potentially excavators, etc.) are dealing with. The prototype offers two hosts for development, namely a performance host and a safety host. The performance host is a high-performance computing solution, like e.g., an NVIDIA chip. The safety host will mainly host safety applications ensuring the functional safety of the whole system. An example of a safety host can be an Aurix host. The different hosts are combined using a switch, to enable interaction and communication between the two different hosts. A schematic overview of the solution is provided in Figure 55.

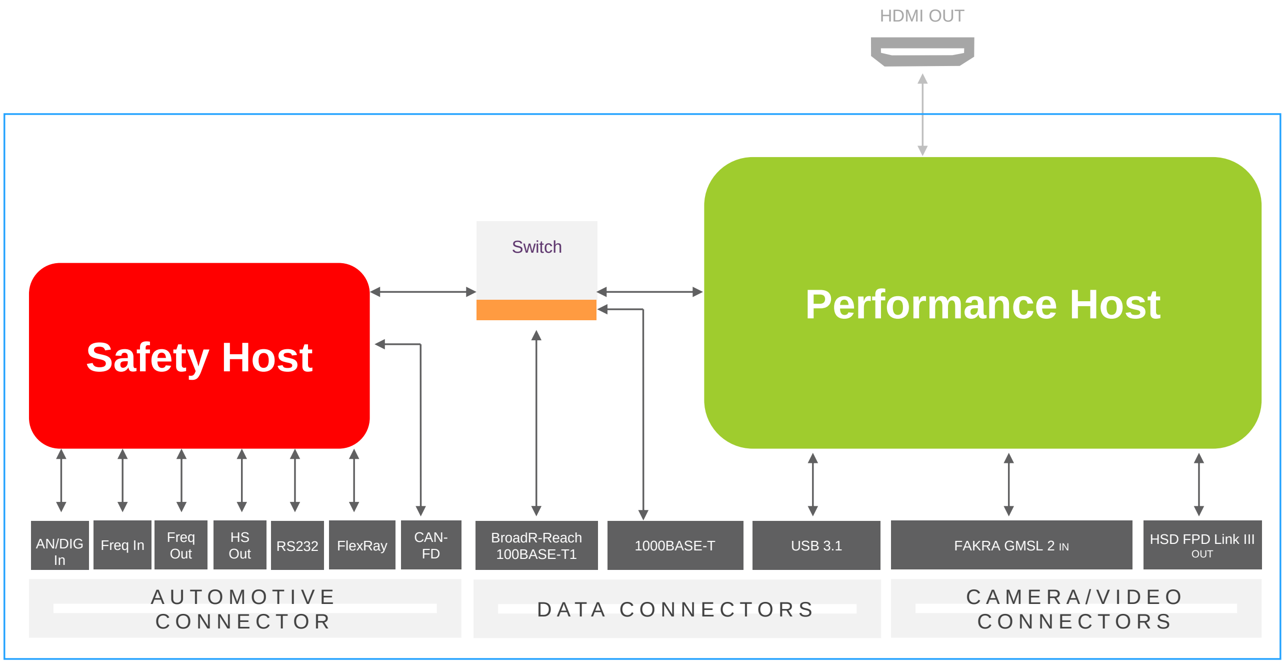


Figure 55: Schematic overview of the Tractor Controller.

The tractor controller prototype supports a Linux Ubuntu platform and will run ROS Melodic for hosting the different apps on the platform. The tractor controller will support a Docker environment, enabling to develop, deploy, and run applications with containers. It provides the ability to package and run an application in a loosely isolated environment on a given host, being in this situation the host on the tractor controller. Docker uses a client-server architecture, where the client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing the Docker container.

To test the tractor controller as an individual component, a test setup has been created at the TTControl site in Vienna. The platform is connected to a power supply and a debug board (via a vehicle connector cable). The debug board enables a connection to the controller and provides the possibility to access all vehicle connector interfaces (like e.g., Ethernet or CAN interfaces). With a remote connection (SSH), the developer has direct access to the platform via a command window. A monitor connected to the controller provides feedback to the developer and debugging possibilities.

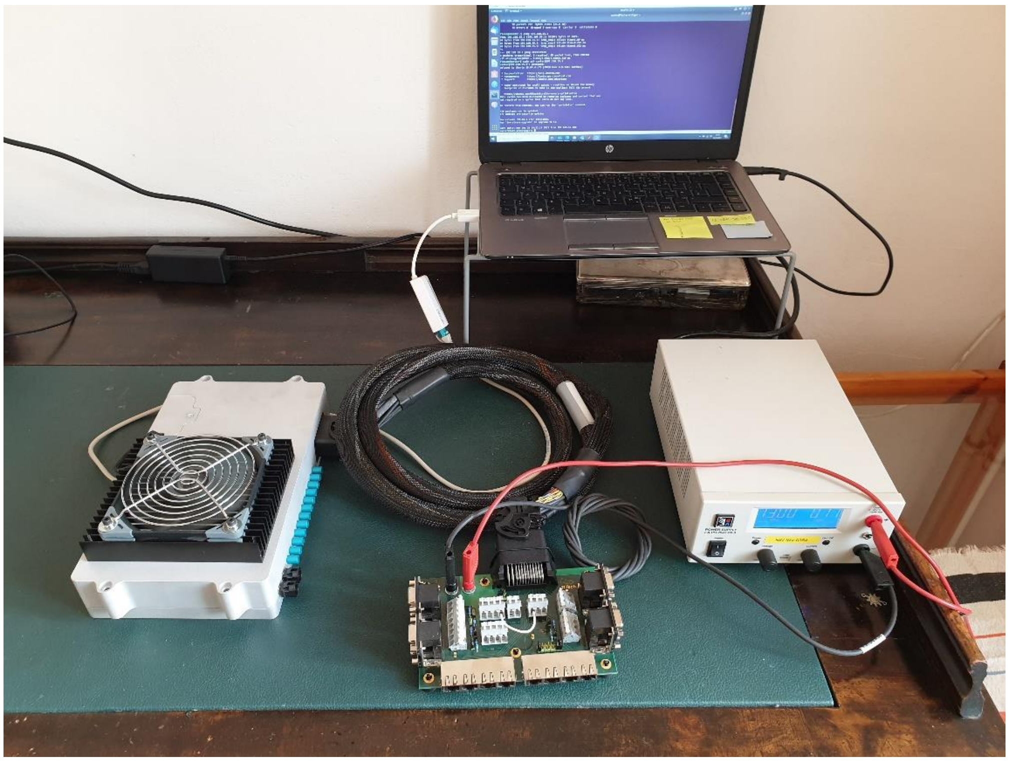
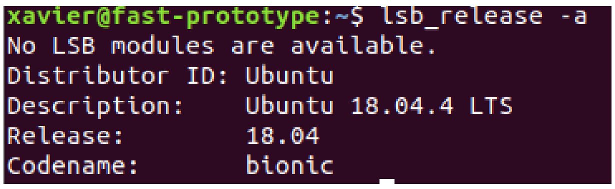


Figure 56: Test setup for tractor controller.

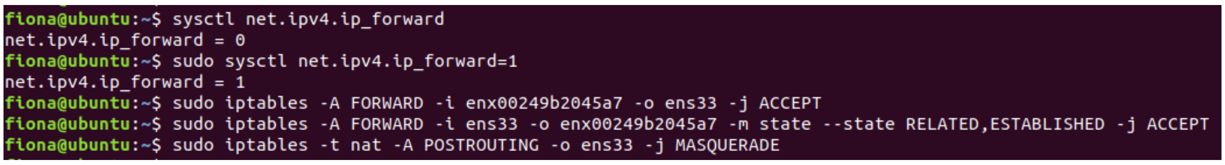
The system is first tested if it is up and running.



```
xavier@fast-prototype:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.4 LTS
Release:        18.04
Codename:       bionic
```

Figure 57: tractor controller Ubuntu bionic console.

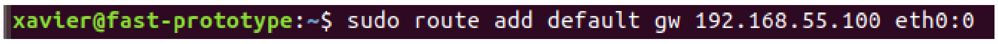
After this, the platform has been connected to the internet, for installing additional packages. The Host controller runs the following commands to enable IP forward (*sysctl net.ipv4.ip_forward=1*), to set up the necessary rules in the iptables.



```
fiona@ubuntu:~$ sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 0
fiona@ubuntu:~$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
fiona@ubuntu:~$ sudo iptables -A FORWARD -i enx00249b2045a7 -o ens33 -j ACCEPT
fiona@ubuntu:~$ sudo iptables -A FORWARD -i ens33 -o enx00249b2045a7 -m state --state RELATED,ESTABLISHED -j ACCEPT
fiona@ubuntu:~$ sudo iptables -t nat -A POSTROUTING -o ens33 -j MASQUERADE
```

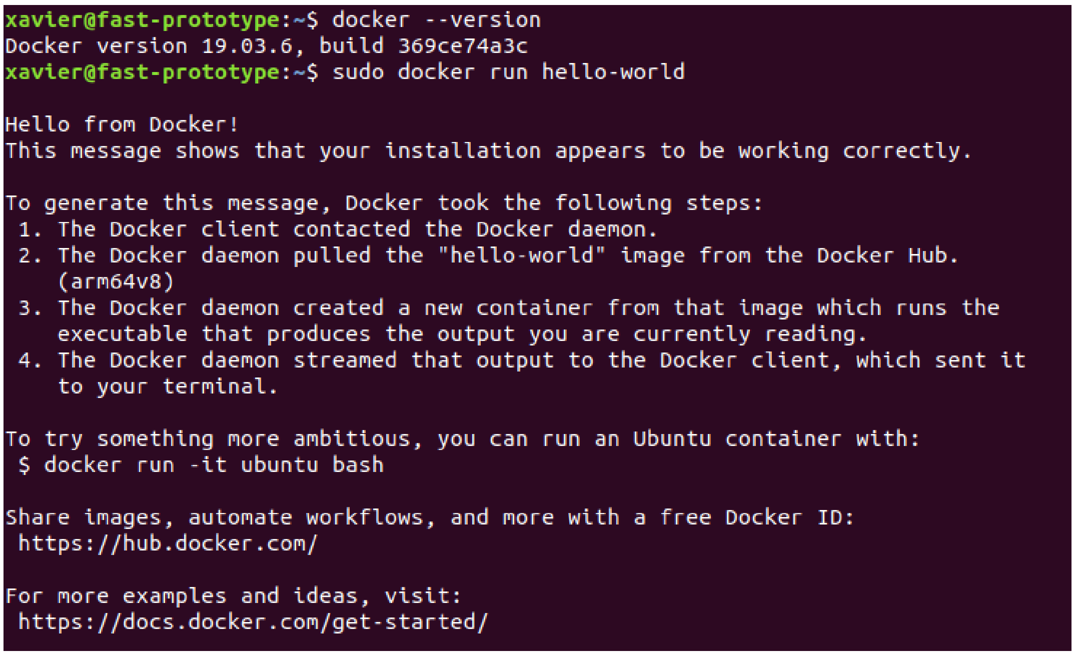
Figure 58: Tractor Controller setup IP tables.

On the platform, it needs to be made sure that the default gateway is set to the eth interface of the connected PC. To test and validate the connection, the system will try to ping the internet (e.g., *ping 8.8.8.8*).



```
xavier@fast-prototype:~$ sudo route add default gw 192.168.55.100 eth0:0
```

The Docker functionality has been tested by checking if it has been correctly installed and a test hello-world application has been deployed and executed to make sure that everything runs correctly and smoothly.



```
xavier@fast-prototype:~$ docker --version
Docker version 19.03.6, build 369ce74a3c
xavier@fast-prototype:~$ sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figure 59: Functionality testing of Docker with hello-world app.

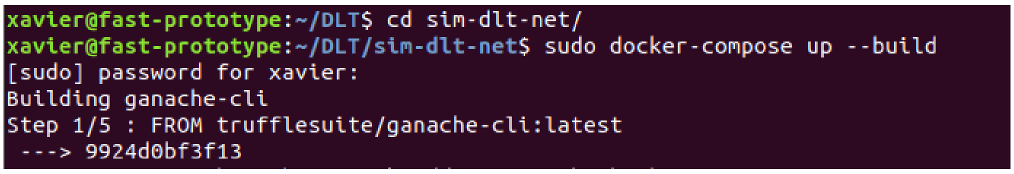
The tests that will be used in the following phase of the use case are integration and functionality tests of the apps that will be deployed and hosted on the controller. Additionally, tests will be performed in the upcoming periods for integrating the controller on the physical tractor and get the required data from the tractor to the apps hosted on the controller. Different interfaces will be required to support all the functionalities of the different apps that will be hosted. These will vary from WebRTC (for streaming video data to the human operator), MQTT for exchanging data between the client apps on the controller and the server apps on the MEC) and potentially other interfaces that are still under definition.

As an example of testing apps on the controller, the DLT app from partner AAU is considered. For running and testing the DLT application on the controller, *docker-compose*, *NodeJS* and *Python3* libraries are required. The missing packages are installed with:

`sudo apt install docker-compose` (respectively *nodejs* and *python3*)

and/or check installed version with *docker-compose -version*

The DLT test application can be found on the IntellIoT Gitlab under: https://gitlab.eurecom.fr/intelliot-project/security/distributed-ledger-technology. To run it on the controller, one has to clone the *sim-dlt-net* directory on the controller and run the DLT manager for testing.



```
xavier@fast-prototype:~/DLT$ cd sim-dlt-net/
xavier@fast-prototype:~/DLT/sim-dlt-net$ sudo docker-compose up --build
[sudo] password for xavier:
Building ganache-cli
Step 1/5 : FROM trufflesuite/ganache-cli:latest
 ---> 9924d0bf3f13
```

Figure 60: tractor controller runDLT manager.

The other applications that will run on the controller, are being validated at the moment and individual tests for the applications are being performed. Later, in the project, these applications will also be integrated on the controller and a complete test of the overall integrated system will be performed.

On the performance host (as depicted in Figure 55), the different applications are deployed. The performance host is equipped with a Linux Ubuntu environment, hosting a Nvidia Jetpack SDK. On this, multiple tools are running that support the hosting and development of applications on the tractor controller. The Robot Operating System (ROS) is also integrated on the platform for controlling the eTractor (see Figure 61).

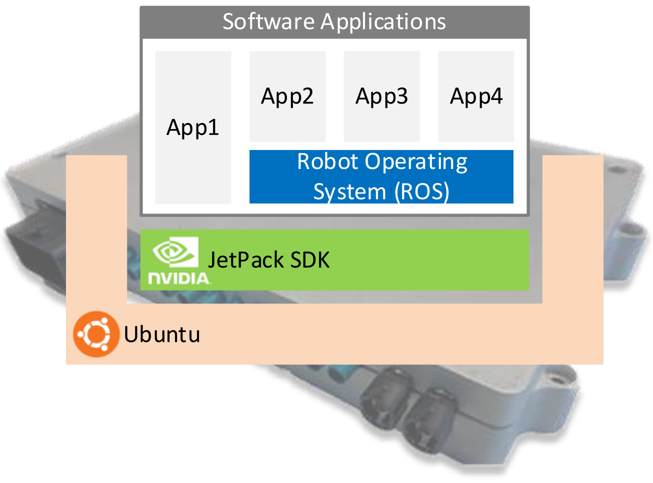


Figure 61: SW Architecture for deploying applications on the tractor controller, supporting ROS

When writing of the deliverable, the following applications are hosted on the tractor controller:

- Intrusion Detection System (IDS) Client, that monitors the network connection between the tractor and the 5G MEC, and potentially also between other different entities in the field. These could for example be other tractors, harvesters or potentially drones.
- Distributed Ledger Technology (DLT) Client, that stores the current position of the tractor (GPS coordinates) using Blockchain technology to track if the tractor actually traversed the trajectory that was defined for it.
- Infrastructure Assisted Knowledge Management (IAKM) Client, which identifies potential AI models required by an agent according to specific semantic, as well as support the training of AI models.

The AI Client, for bypassing (unknown) obstacles in the field, will be included later during the final integration phase.

Furthermore, the controller functions as the gateway between the tractor and the HIL application, where the control messages from the HIL application are sent to the tractor via the tractor controller. The tractor controller supports 5G connectivity, through connecting a 5G dongle directly to the controller and being capable of receiving the messages sent over the 5G infrastructure (see Figure *62*).

All these clients running on the tractor controller interact with their respective servers, which are located on the 5G MEC, exploiting the 5G infrastructure. These individual solutions have been tested separately by the partners who have developed them. After extensive testing of these solutions and confirmation of correct operation, they were made available for integration on the tractor controller. The integration was performed by partner TTC, in close cooperation with the individual partner and test scenarios were developed for each individual solution to guarantee that the solution was working properly on the tractor controller.

Figure 62 *: 5G connectivity to the tractor controller.*

3.1.2 INTEGRATION WITH THE AGRICULTURE AI MODELS

2.2.4The objective of the Agriculture AI model is to provide maneuverer control decisions to the tractor controller upon facing obstacles while the tractor navigates through the waypoints.

Input: The input of the Agriculture AI model is a sequence of visual data from the camera mounted on the tractor. This data consists of 3D point cloud as well as a 2D projection corresponds to RGB video frames and information of the obstacle and tractor positions over time instances. The input is pre-processed to filter out the most informative data to be fed into the AI model.

Output: The inferred output of the AI model corresponds to the tractor control decisions at each time instance that are consist of linear and angular velocity components normalized by predefined maximum velocities. In addition, a confidence measure of the inferred control decisions is calculated to identify the need of retraining and/or escalating to a human operator.

Data Repository: The Agriculture AI model uses supervised training, in which, inputs and outputs are accounted as a labelled dataset. The training dataset corresponds to the traces of observed visual inputs and control decision outputs of each maneuverer sequences is stored on the local storage that can be accessed during Local AI training. The traces are named by a unique identifier corresponding to a timestamp of data collection and control decisions as shown below.

[timestamp]-lm-[linear velocity component]-rm-[angular velocity component].jpg

Test Environment: The training is carried out in a simulated environment and initial testing is carried out in a laboratory setting prior to integration. The simulation environment is a Python development environment running on a Windows workstation. Here, a testing dataset (a fraction of data is partitioned from the original training data to generate training and testing datasets) is stored in a local storage and then randomly selected traces are exposed to the Agriculture AI model. Therein, the inferred outputs of the AI model are compared with the labelled outputs of the selected trace to evaluate the test accuracy of the AI model inference. The lab setting includes an off-the-shelf mobile crawler robot known as "Jetank AI kit" which is powered by Nvidia Jetson nano developer module with 16GB eMMC and 4GB RAM. The robot has two motors that could be controlled individually and is equipped with an on-board camera. The training data collection and validations are carried with the aid of the robot in the laboratory environment illustrated in Figure 63.

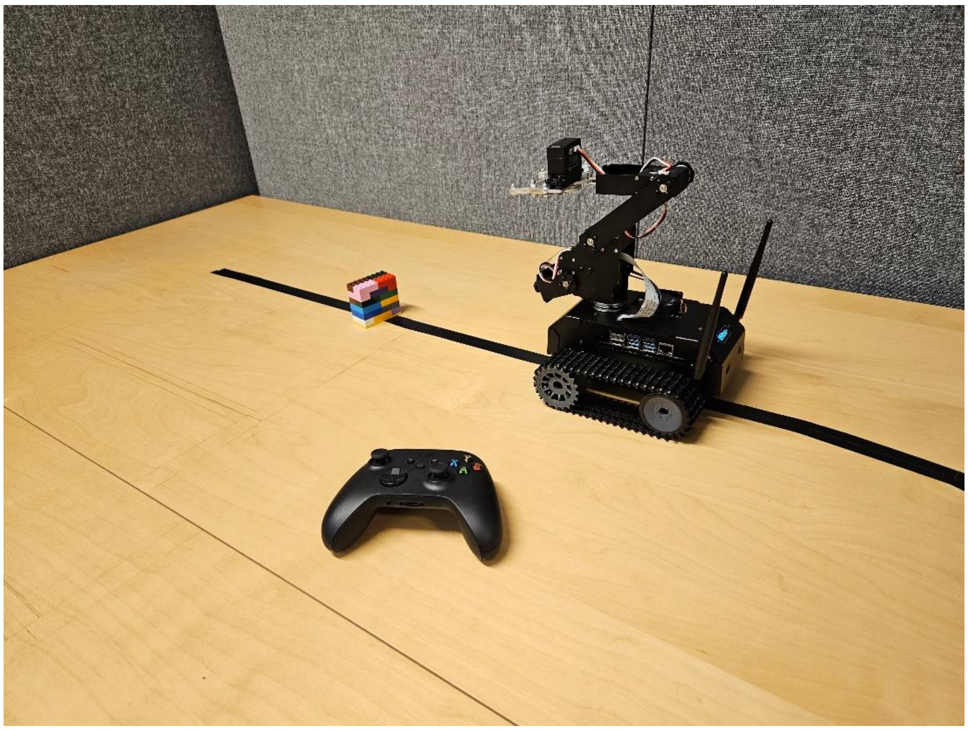


Figure 63: Laboratory setting of obstacle bypassing AI testing.

3.1.3 REMOTE OPERATOR DEVICE

In UC1, the HIL Application connects to the ISAR client application on the VR headset, the Oculus Quest 2. This peer-to-peer connection allows the streaming of the entire HIL Application (and thus virtual content) to the client device. As mentioned in previous iterations of this deliverable, the VR environment is beneficial for this use case as it provides a fully immersive experience for the user, which closely approximates the real experience of piloting a tractor. While described more deeply in D3.3 and D3.7, the movement and control of VR handheld controllers is used to allow the human to remotely operate the tractor. The VR controllers of the Oculus Quest 2 are connected to each other via Bluetooth signaling. Following transmission of Bluetooth signaling, the HIL Application ISAR plug-in allows a sending function which sends the control commands onward to the IntellIoT components which allow movement of the tractor. This occurs with API calls.

On the server side, CustomSend objects in the scene in Unity are enabled. *ServerApi*, *ApiConfig*, *ConnectionCallbacks* and *ConnectionHandle* are initialized on the Start, which are not recommended to be modified unless event handler for receiving incoming messages. Currently, a default message (ping) from Unity is running based on timer *OnTimerElapsed* function in CustomSendExample.cs and can be modified to the use case. The message to be sent can be assigned under HlrCustomMessage defined under ConnectionApi.cs. And through the *ConnectionApi*, *PushCustomMessage* the message can be sent to the client. An example to send "OpenKeyboard" message on a button press event to Client:

```
public void OpenKeyBoard()

{

        if (IsConnected)

        {

        byte[] msg = Encoding.ASCII.GetBytes("ShowKeyboard");
        IntPtr unmanaged = Marshal.AllocHGlobal(msg.Length);
        Marshal.Copy(msg, 0, unmanaged, msg.Length);
        HlrCustomMessage message = new HlrCustomMessage();
        message.Length = (uint) msg.Length;
        message.Data = unmanaged;
        _serverApi.ConnectionApi.PushCustomMessage(_handle, message);
        }

}
```

Figure 64: Code Sample - Custom Send Function - Server Side.

Incoming messages from the client are received under *OnCustomMessageReceived* event handler and can be verified and updated further depending on the use case.

```
OnCustomMessageReceived (in HlrCustomMessage message)
{
        int length = (int)message.Length;
        byte[] managedData = new byte[length];
        Marshal.Copy(message.Data, managedData, 0, length);
        string msg = Encoding.ASCII.GetString(managedData);
        Debug.Log($"Received custom message: {msg}");
}
```

Figure 65: Code Sample - Custom Received Function.

On the Client-Side, the message is received (for reference) under: Remote Rendering in ImmersiveAppView.cpp under Init function in the callback *m_customMessageCallback*. Furthermore, messages can be verified or used to trigger an event by registering in the *register_custom_message_handler* of the *ConnectionApi*.

An Example to open System Keyboard on the HoloLens based on message from the server side:

```
m_customMessageCallback = [ ](HlrCustomMessage* message, void* user_data) {
        auto* immersive = static_cast<ImmersiveAppView*>(user_data);
        uint32_t length = message->length;

        std::string msg {
                message->data, message->data + length};
        };

        if (msg == "ShowKeyboard") {
        immersive->RequestUserInput ();
        }
};

m_isarApi.connection.register_custom_message_handler (
        m_connectionHandle,
        m_customMessageCallback, this
);
```

Figure 66: Code Sample - Custom Send Function - Client Side.

This triggers function RequestUserInput() in the client to open System keyboard on the incoming message. Currently, default message from client (pong) to the server is assigned in the *onConnectionStateChanged* in the ImmersiveAppView.cpp running on timer periodically and can be changed according to the usage.

To send a message back is similar to the Server-Side code and can be achieved by assigning to *HlrCustomMessage* and using ConnectionApi *push_custom_message*.

3.2 UC2: Healthcare

3.2.1 SMART DEVICES

The details of the collection of the data from the smart devices has been described in Deliverable D3.7. Also, the clinical requirements and technical requirements as well as the relevance of the data collected has been described in Deliverable D1.4.

The smart devices (sensors) will connect to the Local AI through Bluetooth Low Energy (BLE) at certain periods as defined by clinical protocol. The integration layer that connects smart devices with the rest projects framework is hosted by Vida24 mobile application. Each smart device exposes BLE services and characteristics and based on manufacturers guidelines, Vida24 mobile application retrieves encrypted bytes and convert them to actual values.

Each smart device (sensor) is regularly connected to the Local AI through Bluetooth Low Energy (BLE), as defined by the clinical protocol. The frequency at which each enrolled patient has been instructed to establish connection between sensor and Local AI for the transmission of biological parameter measurements is device-specific and is described in detail below:

a. Pulse oximeter (oxygen saturation and heart rate): Twice daily at rest, as well as at every exercise session;
b. Blood pressure monitor (systolic/diastolic blood pressure and heart rate): Once or twice daily, depending on each patient's clinical profile (non-hypertensive and hypertensive status, respectively);
c. Weight scale (body weight): Once daily;
d. Thermometer (body temperature): Only at the occurrence of symptoms suggestive of an acute infection;
e. Smartwatch (physical activity in steps / heart rate): At least once daily, for the transmission of aggregated data.

The actual frequency at which connection between sensor and Local AI is established –and thus the actual size of the inputs to the Healthcare AI Models- is additionally dependent on:

a. Enrolled patients' compliance to the aforementioned instructions;
b. Structural and functional integrity of each smart device;
c. Robustness of wireless internet connection at the time of data transmission.

3.2.2 INTEGRATION WITH THE AI MODELS

When it comes to the AI models, two main usages can be identified: (1) development of pre-trained models and (2) post-deployment inference and training on those models.

3.2.2.1 MODEL DEVELOPMENT

AI Models are pretrained by researchers in a development environment that includes python with several relevant libraries, most notable one being the TensorFlow machine learning platform. The researchers have access to a retrospective dataset that was extracted at a specific point in time from the Vida24 backend using its REST interface.

When the models show potential and are deemed ready for further training in the field, they are converted to TensorFlow Lite (TFLite) format and stored to disk. The conversion to TFLite is important because the Android phones that are the deployment target, can only run that framework. Conversion must be performed on the development machine, which can handle both formats.

To ensure that the converted models expose all required interfaces needed for inferencing and training, as well as storing and loading of weights, a wrapper was introduced that takes care of this. It can be used to wrap any TensorFlow model and provide the mentioned functionality. The correct functioning of this wrapper is tested by unit tests.

3.2.2.2 MODEL INFERENCING

Models that are to be used for inference will be deployed by means of packaging them into the Android app. They will explicitly not be downloaded from the global AI component. This is to ensure that at any moment, there can be no confusion about what model version is used to make predictions. Running model inference is facilitated through the Local AI component's TensorflowAdapter class (see Figure 67: Local AI integration into app).

3.2.2.3 TRAINING

The models resulting from the development phase, will be used as starting point for federated learning, as the first model version to be distributed by the global AI component during training. It becomes the first "model to beat". The test part of the development dataset will also be reused, becoming the evaluation dataset that determines if the newly aggregated model weights that result from local training perform better than the old weights.

To ease the use of the TFlite models within an Android app, the TensorflowAdapter class is provided by the local AI component. This class takes care of mapping the various operations to run inference with the model, train it and save and restore its weights. The correct functioning of this convenience class is tested by unit tests.

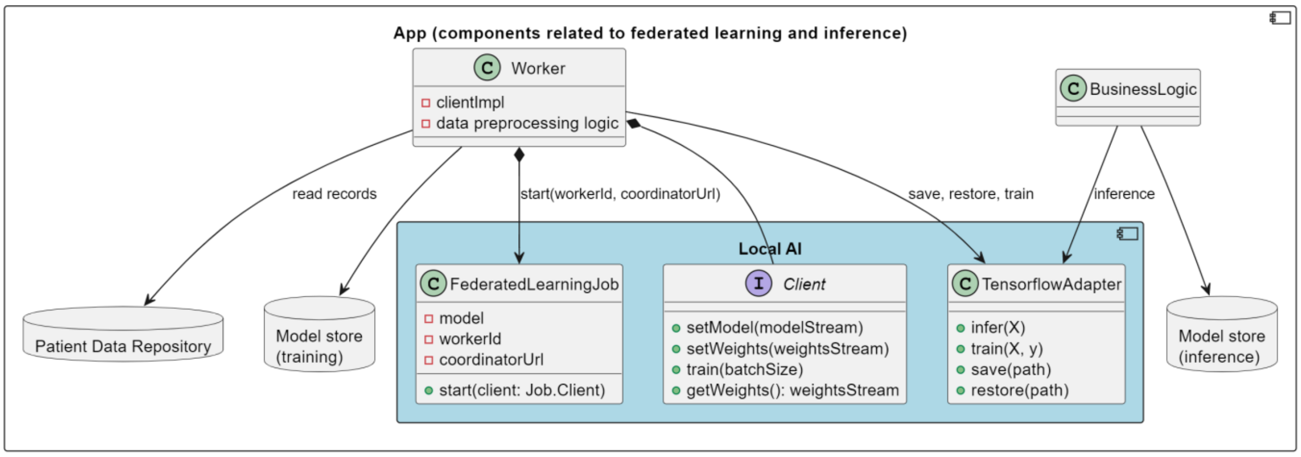


Figure 67: Local AI integration into app.

To participate in one round of training, as defined and initiated by the global AI component, a federated learning workflow is provided through the Local AI component library, through the FederatedLearningJob class. To keep the workflow flexible, it will call upon the integrating app to fill in certain steps of the FL process, e.g., storing of the model weights, initializing the model structure, perform training on the local dataset.

Design decisions and their rationale

Deferring model interactions to the app, rather than putting that directly into the FederatedLearningJob:

- The FederatedLearningJob takes care of the back-and-forth messaging between Local and Global AI, during a round of training. It understands the call order the Global AI expects, as well as the message formats. It's important to note that at this point, no understanding is needed of the underlying machine learning framework (e.g., TensorFlow, PyTorch, ONNX, etc.). This is why any such framework dependencies have been pushed to the app, so that the workflow remains reusable for all.
- Different models and will require different pre-processing steps to be implemented. Any such logic is to be implemented by the app.
- Different apps are expected to make different choices when it comes to data storage. Therefore, the Local AI component defers any data storing and loading to the app.

App provides the execution environment:

- Since apps may want to choose their own (background) execution mechanisms, the Local AI component makes no assumptions about this itself. E.g., an Android app may choose to host the FL job in a background service, use the WorkManager and its advanced scheduling criteria (battery, network state, etc), or implement its own threading model.

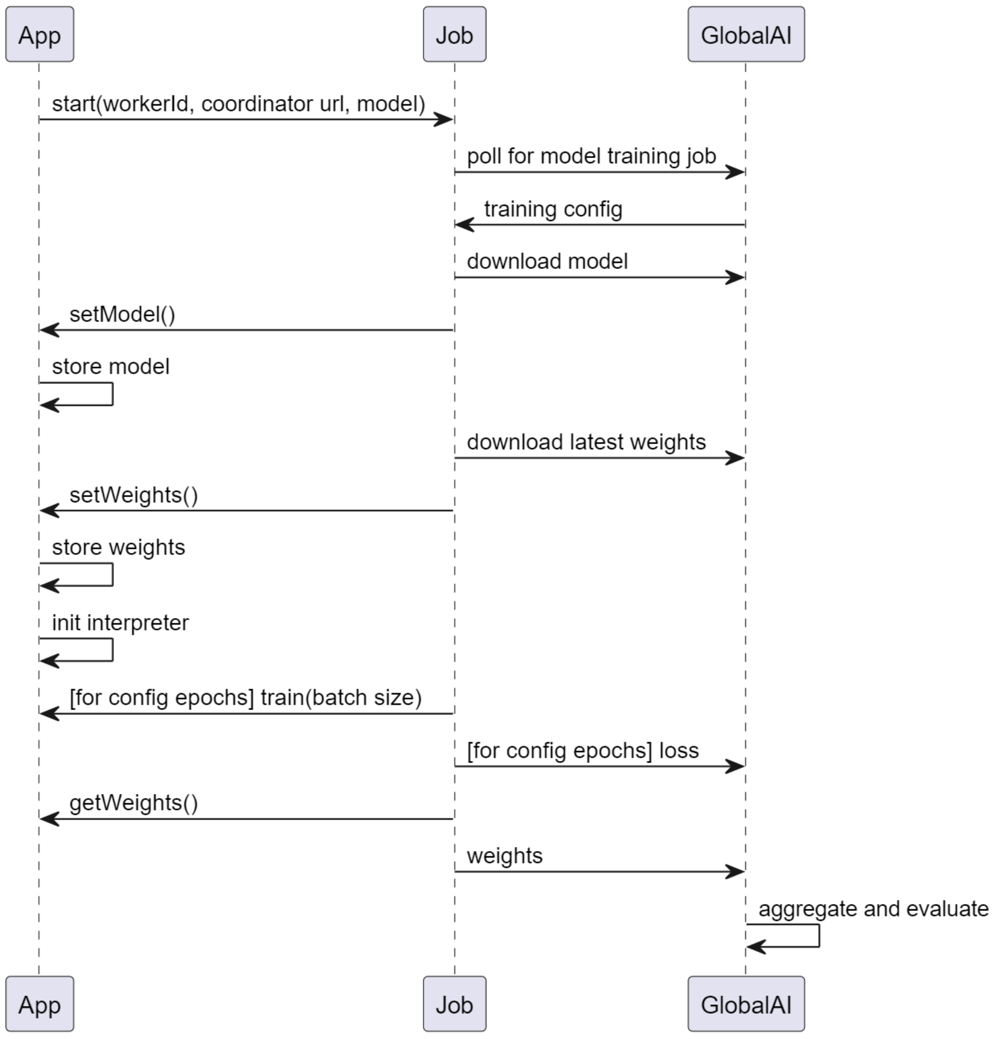


Figure 68: Implemented workflow for FederatedLearningJob.

3.2.3 INTEGRATION WITH THE PATIENT DATA REPOSITORY

The data collected from the UC2 sensors is also needed for analysis by physicians on PAGNI premises. A local deployment of Vida24 cloud application to PAGNI dedicated server is required. The procedure started by creating a secure VPN connection from PAGNI IT department for Vidavo DevOps engineers to connect and install the dependencies on operating system and application level. The next step is to create the correct interfaces to expose Vida24 cloud application to the internet to receive data from the mobile application as well as to allow physicians to have access on patients' data outside of PAGNI premises. Through the pilot study the necessary subset of collected data was sent to the Patients Data Repository through a protected and secure API and a HTTP over TLS connection.

3.3 UC3: Manufacturing

3.3.1 ROBOT CONTROLLER

The robot controller is an abstraction simplifying the control of the very flexible UR5 robot arm to a limited number of fixed tasks necessary for the manufacturing use case demonstrator.

It provides an HTTP interface that allows agents in the HyperMAS and other apps to make the robot perform fixed tasks, move it to predefined positions and read out its status / actual position. It controls the UR5 arm via (non-real-time) TCP commands / URScript. For real-time control of the robot, which is required for the HIL scenario, the HIL application will directly interact with the UR5 robot arm.

The robot's HTTP interface has been defined together with HSG as the main "users" of it in form of an OpenAPI specification, using Swagger. From the Swagger description, a W3C WoT Thing Description has been created to enable agents to be programmed against the robot's affordances.

The latest version of the interface specification can be found in *API Flow for UC 3*[9]; *Robot Controller*[10] contains examples of the API usage.

The implementation of the API has been started using a Node-RED based workflow and it is tested in most parts, using the Swagger-internal functionalities. Implementing the robot controller functionality itself is in progress as well.

3.3.2 ENGRAVER APP

This text addresses the integration of the engraver service edge app with the laser cutter machine as well as the milling machine.

The Engraver App is an edge app, which provides a service interface for controlling the engraving machines. In IntellIoT we support two types of engraving: per laser cutter and per milling machine. Both machine types are different in nature, but, for the purpose of the manufacturing use case, we covered difference by a uniform web interface.

By December 2021, the *Mr Beam* lasercutter[11] is supported. For the support of the Milling Machine, the same service implementation will be used (although it might be instantiated in an own app identity). It is the goal, that the same web API interface towards the client (HyperMAS) is used for both laser cutter and milling machine. Depending on the parametrization the one or the other machine can be selected.

The process of engraving of text should be similar on both machine types. It consists of three phases:

1. preparation
2. engraving
3. postprocessing

Each phase requires service call on the web interface of the engraver service. The web service requests include parameters, which kind of machine is used, and which concrete machine (id) is used. Although, we considered that each machine type has its own type of engraver service and accordingly machine type might be redundant. For the moment, we keep this for flexibility in the future.

Figure 69 and Figure 70 show the UML sequence diagram of production of a workpiece with the Mr Beam laser cutter and respectively with the milling machine. Note that this diagram merely shows one of several possible executions since the concrete HTTP queries are only configured at system set-up time using the Web-based IDE discussed in Section 4.2.2 and in Deliverable D3.1.

[9] https://docs.google.com/document/d/1qWyrCS9p57yd1aCUA-pawPoA8IAqLt7StHaMDn2nrjw/edit
[10] https://app.swaggerhub.com/apis-docs/danaivach/robot-controller/1.1.0
[11] https://www.mr-beam.org/

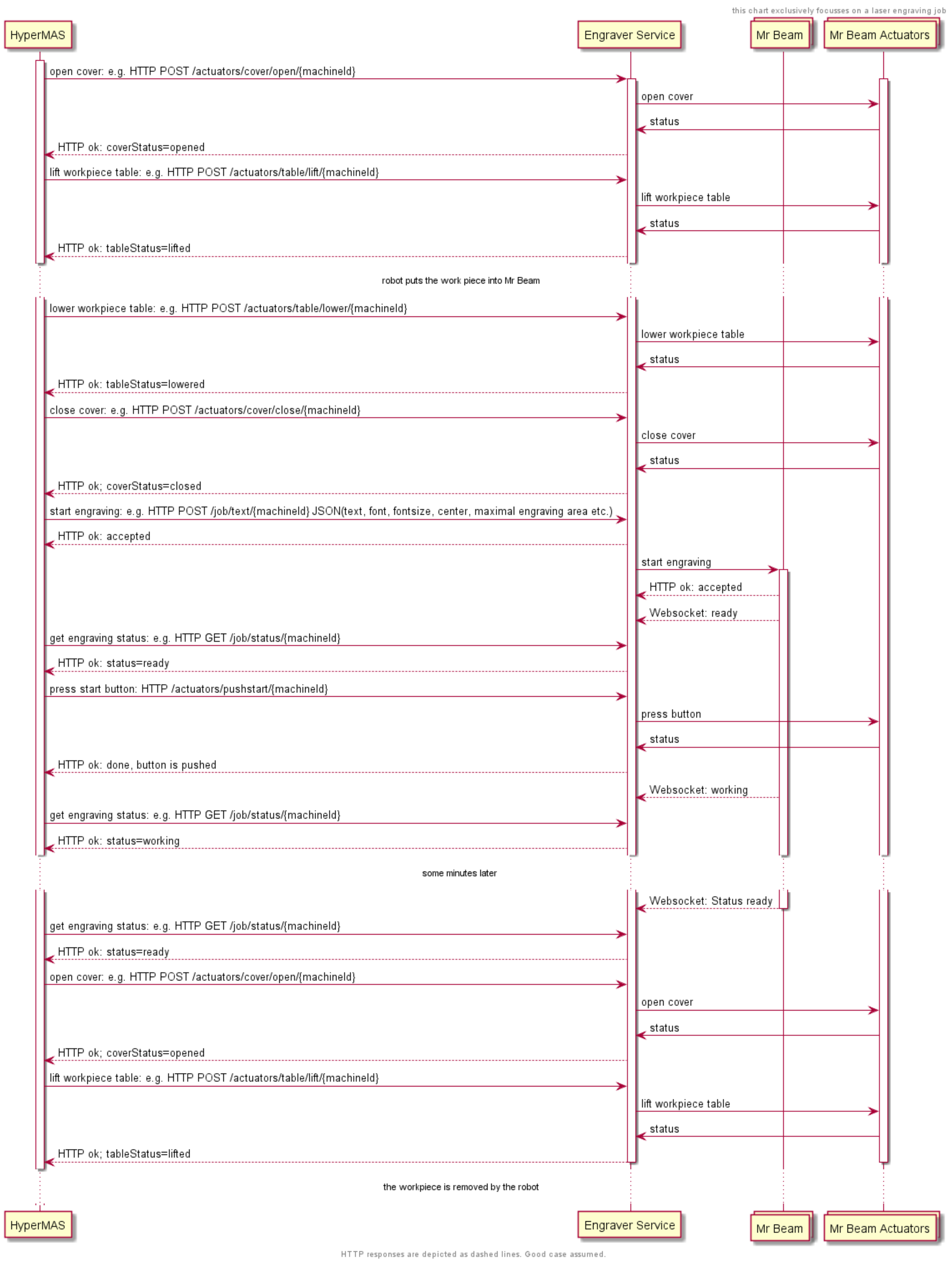

this chart exclusively focusses on a laser engraving job
HyperMAS
Engraver Service
Mr Beam
Mr Beam Actuators
open cover: e.g. HTTP POST /actuators/cover/open/{machineId}
open cover
status
HTTP ok: coverStatus=opened
lift workpiece table: e.g. HTTP POST /actuators/table/lift/{machineId}
lift workpiece table
status
HTTP ok: tableStatus=lifted
robot puts the work piece into Mr Beam
lower workpiece table: e.g. HTTP POST /actuators/table/lower/{machineId}
lower workpiece table
status
HTTP ok: tableStatus=lowered
close cover: e.g. HTTP POST /actuators/cover/close/{machineId}
close cover
status
HTTP ok; coverStatus=closed
start engraving: e.g. HTTP POST /job/text/{machineId} JSON(text, font, fontsize, center, maximal engraving area etc.)
HTTP ok: accepted
start engraving
HTTP ok: accepted
Websocket: ready
get engraving status: e.g. HTTP GET /job/status/{machineId}
HTTP ok: status=ready
press start button: HTTP /actuators/pushstart/{machineId}
press button
status
HTTP ok: done, button is pushed
Websocket: working
get engraving status: e.g. HTTP GET /job/status/{machineId}
HTTP ok: status=working
some minutes later
Websocket: Status ready
get engraving status: e.g. HTTP GET /job/status/{machineId}
HTTP ok: status=ready
open cover: e.g. HTTP POST /actuators/cover/open/{machineId}
open cover
status
HTTP ok; coverStatus=opened
lift workpiece table: e.g. HTTP POST /actuators/table/lift/{machineId}
lift workpiece table
status
HTTP ok; tableStatus=lifted
the workpiece is removed by the robot
HyperMAS
Engraver Service
Mr Beam
Mr Beam Actuators
HTTP responses are depicted as dashed lines. Good case assumed.

Figure 69: Example workflow of production with laser cutter.

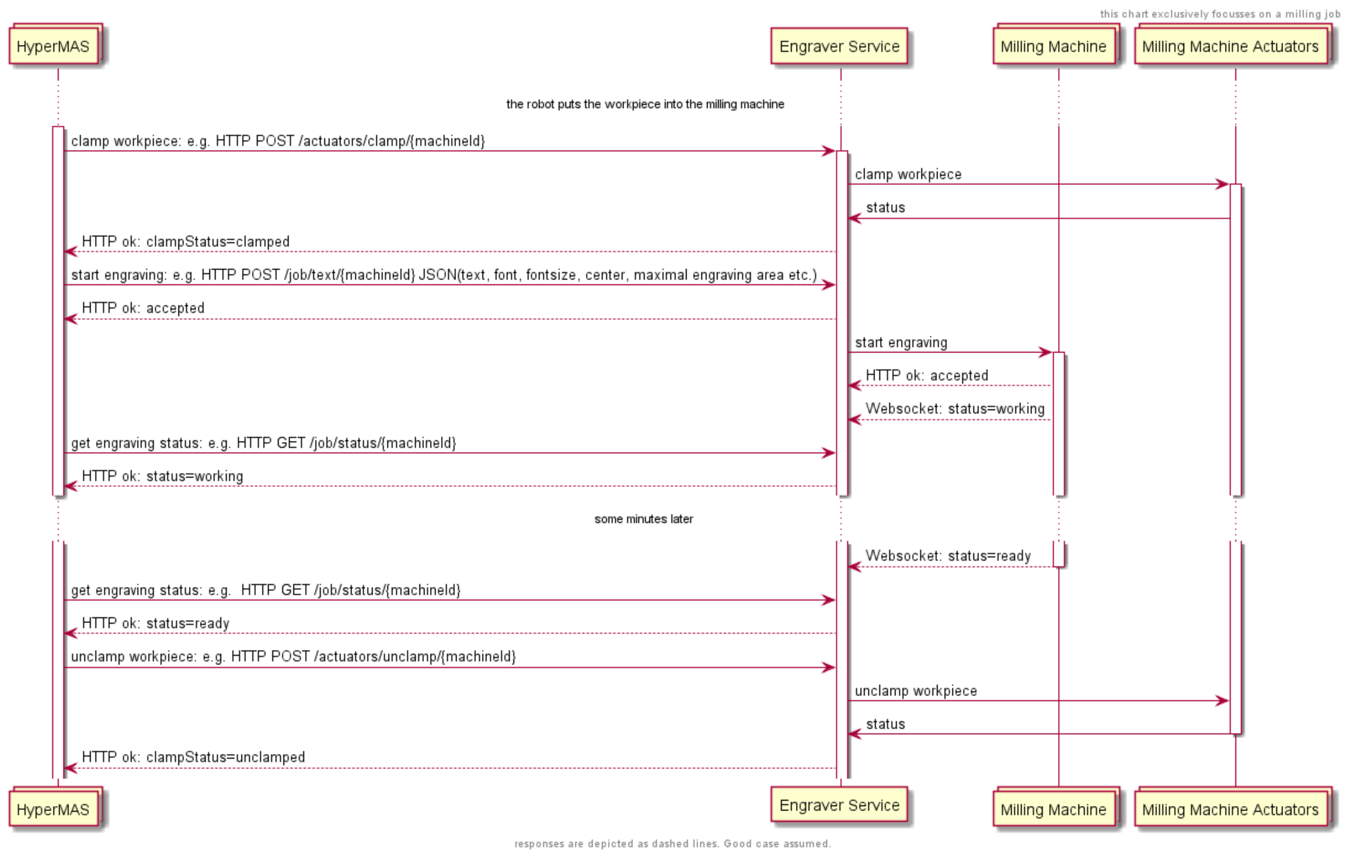

this chart exclusively focusses on a milling job
HyperMAS
Engraver Service
Milling Machine
Milling Machine Actuators
the robot puts the workpiece into the milling machine
clamp workpiece: e.g. HTTP POST /actuators/clamp/{machineId}
clamp workpiece
status
HTTP ok: clampStatus=clamped
start engraving: e.g. HTTP POST /job/text/{machineId} JSON(text, font, fontsize, center, maximal engraving area etc.)
HTTP ok: accepted
start engraving
HTTP ok: accepted
Websocket: status=working
get engraving status: e.g. HTTP GET /job/status/{machineId}
HTTP ok: status=working
some minutes later
Websocket: status=ready
get engraving status: e.g. HTTP GET /job/status/{machineId}
HTTP ok: status=ready
unclamp workpiece: e.g. HTTP POST /actuators/unclamp/{machineId}
unclamp workpiece
status
HTTP ok: clampStatus=unclamped
HyperMAS
Engraver Service
Milling Machine
Milling Machine Actuators
responses are depicted as dashed lines. Good case assumed.

Figure 70: Example workflow of production with milling machine.

3.3.2.1 PHASE 1 AND 3: PREPARATION AND POSTPROCESSING

Phases 1 and 3 include physical actions so that a production process (engraving a text) can be started. This is necessary due to the nature of the manufacturing use case. Here work pieces are put into the machine by a robot arm before the production can start and picked up by the same robot arm, when the production is finished.

The phases are differing whether a laser cutter or a milling machine is used. For the preparation of the production process with the laser cutter, the lid has to be opened, a table has to be lifted and lowered and a button has to be pressed, to start the laser cutter. The post processing includes the same steps in reverse order. In Figure 69, the different action of preparation and postprocessing can be deduced.

For milling machine, the only preparation and postprocessing procedure is to clamp and unclamp the work piece. Figure 70 shows the workflow.

We use actuators on the basis of Tinkerforge elements[12]. Those are devices, which are able to control the mechanics for the actions. They are connected via TCP/IP over Ethernet to the Engraver Service. The Engraver Service, in turn, offers a REST façade to expose their functionality to HyperMAS.

3.3.2.2 PHASE 2: ENGRAVING

To start the engraving job, the interface of the Engraver app includes a distinct service function. Incoming requests are processed by the engraver service in such a way, the engraving can be started by the native interface

[12] https://www.tinkerforge.com

of the machine. This requires service translation, as well as the preparation of the input parameter in such a way, the machine can deal with that. An important capability of the Engraver Service is, that a vector graphics in the SVG format is created out of the ascii text input. Here, the text is rendered into path elements, consisting of lines, arc and Bézier curves.

The typical client will be HyperMAS. The web API for managing the engraving hob will be the same for both milling machine and laser cutter. These APIs will be expressed in a machine-readable way as W3C WoT Thing Descriptions to enable agents to be programmed against them.

To start an engraving job, a HTTP POST on `{{base_url}}/job/text` has to be issued.

An example for the message body is as follows:

```
{
  "machineType": "laser",
  "machineId": "laser-1",
  "text": [
    "here comes the text"
  ],
  "font": "ABeeZee",
  "variant": "regular",
  "fontsize": 76,
  "positionReference": "center",
  "maxWidth": 150,
  "maxHeight": 80,
  "x": 120,
  "y": 220
}
```

Figure 71. /job/text example.

With `machineType`, the Engraver Service learns which implementation is used. Machine types could be laser or milling-machine. With `machineId`, the concrete machine is identified. Although we have just one laser and one miller, we should keep it for the future. The value can be arbitrary for the moment. The value of `font` describes the text font, which is used. `variant` is its corresponding variant (e.g., regular, bold, italics). The font has to be available at the engraver. The values for font and variant must correspond to the file name of the font file. In this case, the file name was `ABeeZee-regular-esDR31xSG-6AGleN6tKukbcHCpE.ttf`. The pattern `{{font}}-{{variant}}-{{whatever}}` is used by all available fonts. We use Google Fonts and have 4500 font/variant combinations available (~500 different fonts). For practical reasons, we currently support only ABeeZee (regular, italics), Abel (regular) and Roboto (regular).

- `fontSize` is the size of the font.
- `x`, `y` is the coordinates of the origin of the engraved text in the coordinate system of the machine.
- `positionReference` is the origin of the engraving. Possible values are currently `northwest`, `center`.

The value of `text` is the content to be engraved. It is encoded in an array. Each array element is a line. Currently only single line (one array element) is supported. If multi line text will be supported in the future is up to be decided. Also, how they are oriented (left, center). The concrete machine has to be configured with respect to its network addresses, its working area and the orientation of the coordinate system.

The interface to the machine is exposed to the engraver service. The engraver service itself is an edge app, which runs on Industrial Edge devices. It is connected via TCP/IP with the engraving machines. The Engraver service will start the engraving via HTTP REST and JSON binding. Accordingly, the milling machine has to provide service functions to start engraving and to provide information about its status. For the start engraving request, the engraver service sends the SVG body plus extra information in a JSON body, which is required for the engraving. The SVG includes the representation of the text encoded in a SVG path (PATH tag). The SVG includes also the definition of a transformation matrix, so that the text is already translated to its position on the work piece. The concrete interface is internal between Engraver Service and milling machine and not further discussed here.

3.3.2.3 TESTING THE ENGRAVING

For integration tests, a test instance is available. This instance includes a simulator of an engraving machine, which can be used to test protocol interaction. The API is documented as OpenAPI specification, which is tracked in IntellIoT's gitlab environment, and a W3C WoT TD has been created from this. Integration tests with the HyperMAS components – both, the HyperMAS Infrastructure and the Web-based IDE for HyperMAS – have been successful.

3.3.3 INTEGRATION WITH THE MANUFACTURING AI MODELS

The Manufacturing AI model carries out two main tasks: detection of engraving area and calculating the grab spot of work pieces. Both of these tasks are facilitated by an additional AI service of detecting crosshair markers.

Input: Despite of the task (detection of engraving area or computing grabs spot), the input of the Manufacturing AI model is an image from the camera mounted over the workbench/engraver/milling machine. This image consists of the top view of the workpiece and special markers (crosshair) placed on the surface that are used to determine the physical dimensions. The input is pre-processed to filter out the most informative data to be fed into the AI model.

Output: The inferred output of the AI model corresponds to one of the following depending on the task: (I) the engraving area corresponds to a circular region (the largest in-circle) with the coordinates of the centre and length of the radius or (II) the grabbing location in terms of a tuple corresponds to x and y coordinates and angle referring to the grabbing arm orientation. In addition, a confidence measure of the inferred output is calculated to identify the need of retraining and/or escalating to a human operator.

Test Environment: The training and testing are carried out in a simulated environment prior to integration. The simulation environment is a Python development environment running on a Windows workstation. Here, a testing dataset (a fraction of data is partitioned from the original training data to generate training and testing datasets) is stored in a local storage and then randomly selected images are exposed to the Manufacturing AI model. Therein, for each selected test image, the inferred outputs of the AI model (both the engraving area and the grab spot separately) are compared with the labelled outputs to evaluate the test accuracy of the AI model inference.

After successful testing, the AI models are deployed in the demo setting as a service that can be accessed using a REST API with following web request.

http://[IP:PORT]/AI_Service/[SERVICE]?storageId=[ID]&cameraHostname=[HOST]&cameraId=[CAMERA]
[IP:PORT] - IP address and the port of the AI service that is run on the edge server (e.g. "127.0.0.1:8080"). [SERVICE] - Defines two tasks computing engraving area and grab spot. • [SERVICE]={compute_engravingArea, GET_grabspot} [ID] - ID of the storage areas. Using negative number returns a visualization for debugging purposes. • Workpiece Storage: ID = {1, 2, 3, 4}. • Laser Engraver: ID = 1 • Milling Machine: ID = 1 • For testing purposes, use ID={1,2,3,4,5,6} with testing keywords given below [HOST] - IP address and the port of the RPi camera (e.g. "127.0.0.1:8080").

- In the Manufacturing setting, currently we are using HOST={camera-storage.fritz.box, camera-milling.fritz.box, camera-engraver.fritz.box}
- Use HOST = testimages for testing with the test image instead of contacting the camera.

[CAMERA] - Defines the camera that connects to the three locations

- CAMERA = {workpieceStorage, millingMachine, laserEngraver}
- Use CAMERA = testimages for testing

Two of exemplary outputs are visualized in Figure 72

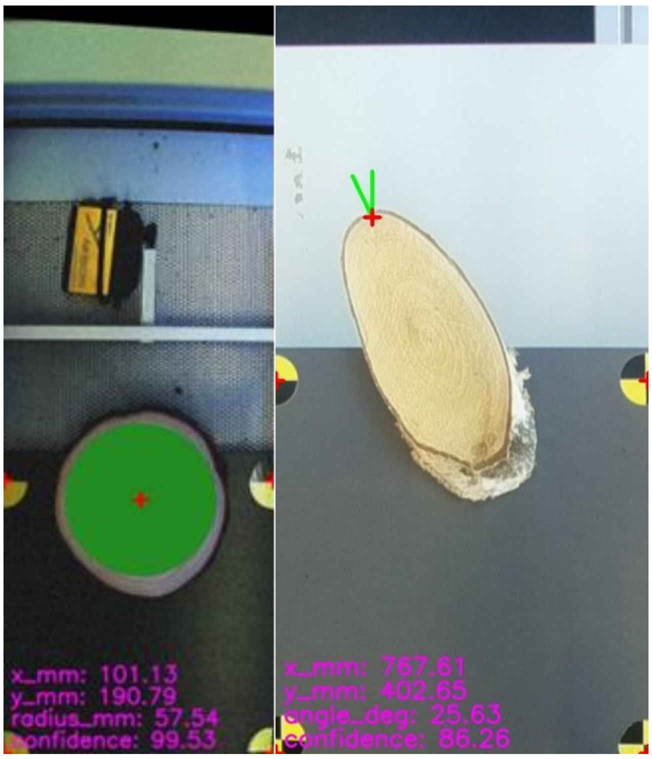


Figure 72. Example illustrative outcomes of area detection AI for a workpiece on the engraver (left) and the grab point AI of a workpiece on the storage (right).

3.3.4 REMOTE OPERATOR DEVICE

UC1 and UC3 utilize similar methods for the operator to remotely operate the IntellIoT targets. The distinguishing differences here are in the hardware used. In UC3 the HoloLens 2 is used to provide an augmented reality environment. An AR environment superimposes virtual content on the real world, and thus allows the visualization of a digital twin of the robot. Additionally, such an environment provides a unique perspective of the robot and its motion in the real world, which provides important information about how the movement of the robot will be reflected in reality. In order to reach high levels of precision and accuracy required to execute movements of the robot, a stylus pen (Holo-Stylus) is used as an AR input device. This device acts as hand gestures would normally but allows a higher degree of accuracy in motion detection and control. The Holo-Stylus here connects via Bluetooth signaling to a head-mounted-unit on the HoloLens 2. The head-mounted-unit can therefore track the motion of the Holo-Stylus to a precise degree, combined with event-based trigger signaling from button presses on the stylus. As in UC1, information from the head-mounted-unit is sent to the HIL Application via ISAR driven WebRTC streams. From there, TCP/IP protocols allow the transfer of control commands from the HIL Application to robot components (e.g., robot controller).

3.3.4.1 TESTING THE REMOTE OPERATION

Test deployments of the remote operation for UC3 and its respective HIL Application and associated components have occurred in the previous cycle. Test deployments for the human-in-the-loop robot were met with success, largely using local Wi-Fi protocols to mediate the connectivity required. Movement of the Holo-Stylus during visualization of the robot enabled precise movements of the robot when needed. Test deployments for the UC1 tractor have not occurred yet but have been planned to occur. These deployments provided (and will provide) valuable feedback regarding specific developments needed for future deployments. Currently this process is ongoing to better refine the integration with IntellIoT IoT components.

4 CONCLUSIONS

In this deliverable D5.4 the integration of the components of the overall IntellIoT framework, as designed in the architecture definition of D2.6, has been presented, and the usage scenarios of this framework of components has been described within the three use cases.

All components were in detail described in the deliverables D3.1-D3.4 and in D4.1-D4.4 as part of the Cycle 1 developments, which focused on the definition, implementation and testing of the components. The Cycle 2 developments of the WP3 and WP4 components were described in deliverables D3.5-D3.8 and in D4.5-D4.8. Overall, the combination of these components as a framework has proven as useful for the implementation of the use cases. Particularly, the communication and computation infrastructure components have provided useful tools to create IoT environments for the three use cases. Then, the trust enabler components have been used in all three use cases to secure the respective IoT environments and allow a trusted engagement of the "human in the loop" through the developed IoT applications. To enable such "human in the loop" scenarios, the human in the loop enabler components could be used and proved their helpfulness. The collaborative IoT enablers were integrated to enable the definition of agent-based IoT applications, enable interoperability, and train and execute AI models on local and global level.

REFERENCES

[1]	Ciortea, A., Boissier, O., Ricci, A. (2019) Engineering World-Wide Multi-Agent Systems with Hypermedia. In: Weyns D., Mascardi V., Ricci A. (eds) Engineering Multi-Agent Systems. EMAS 2018. Lecture Notes in Computer Science, vol. 11375. Springer. https://doi.org/10.1007/978-3-030-25693-7_15	
[2]	Ciortea, A., Mayer, S., Bienz, S., Gandon, F., Corby, O. : Autonomous Search in a Social and Ubiquitous Web. Personal and Ubiquitous Computing, 2020, https://doi.org/10.1007/s00779-020-01415-1	
[3]	Castro, Inma T., Luis Landesa, and Alberto Serna. "Modeling the energy harvested by an RF energy harvesting system using gamma processes." *Mathematical Problems in Engineering* 2019 (2019).	
[4]	T. S. Rappaport, Wireless communications: principles and practice, vol. 2. 1996.	
[5]	https://ieeexplore.ieee.org/document/9797104	