D5.1 Pledger Applications for the use cases

Document Identification			
Status	Final	**Due Date**	31/07/2021
Version	1.0	**Submission Date**	30/07/2021

Related WP	WP5	**Document Reference**	5.1
Related Deliverable(s)	D2.1, D2.2, D2.3	**Dissemination Level (*)**	PU
Lead Participant	i2CAT	**Lead Author**	August Betzler (i2CAT)
Contributors	HOLO, i2CAT, FILL	**Reviewers**	Antonio Castillo Nieto (ATOS)
			Orfeas Voutyras (ICCS)

Keywords:
Use Case Applications, Development, Integration, Testing, Validation Setup

(*) Dissemination level: **PU**: Public, fully open

Document Information

List of Contributors	
Name	Partner
August Betzler	i2CAT
Bruno Cordero	i2CAT
Estela Carmona	i2CAT
Verena Stanzl	FILL
Carina Pamminger	HOLO
Nour Fendri	HOLO
Alex Werlberger	HOLO

Document History			
Version	Date	Change editors	Changes
0.0	23/02/2021	i2CAT	Table of Contents shared with partners
0.1	01/06/2021	i2CAT	First version with UC2 as an example
0.2	28/06/2021	i2CAT, FILL	First round of inputs received (UC2, UC3)
0.3	09/07/2021	i2CAT, FILL, HOLO	Second round of inputs received (all UCs)
0.4	11/07/2021	i2CAT, HOLO	Version to be reviewed internally
0.5	17/07/2021	ICCS	Internal review
0.6	19/07/2021	ATOS	Internal review
0.7	22/07/2021	i2CAT	Final draft
0.8	27/07/2021	ATOS	Quality assurance review
1.0	30/07/2021	ATOS	Final Version

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable Leader	August Betzler (i2CAT)	22/07/2021
Quality Manager	Carmen San Roman Alonso (ATOS)	27/07/2021
Project Coordinator	Lara Lopez Muñiz (ATOS)	30/07/2021

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
3D	three-dimensional
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AR	Augmented Reality
ARES / AR3S	AR Engineering Space
BSC	Barcelona Super Computing
BTP	Basic Transport Protocol
CAD	Computer-Aided Design
CAM	Cooperative Awareness Message
C-ITS	Cooperative Intelligent Transport Systems
CNC	Computer Numerical Control
COTS	Custom of the shelve
CPU	Central Processing Unit
DENM	Decentralized Environmental Notification Message
DLT	Distributed Ledger Technology
DSS	Decision Support System
Dx.y	Deliverable number y belonging to WP x
FPGA	Field-programmable Gate Array
GPS	Global Positioning System
GPU	Graphics Processing Unit
HMD	Head-Mounted Displays
HTTP	Hypertext Transfer Protocol
HW	Hardware
IP	Internet Protocol
MB	Megabyte
MR	Mixed Reality
MRTK	Mixed-Reality Toolkit
MQTT	Message Queuing Telemetry Transport
NC	Numerical Control
OBU	On Board Unit
OCB	Outside of the context of a basic service set
PM	Project Month
QoE	Quality of Experience
QoS	Quality of Service
QR code	Quick Response code
PLC	Programmable Logic Controller
PoE	Power over Ethernet
RAM	Random Access Memory

Abbreviation / acronym	Description
RAN	Radio Access Network
RDNS	Risk Detection and Notification System
RTC	Real-Time Communication
SBC	Single-Board Computer
SDK	Software Development Kit
SLA	Service-Level Agreement
SLAM	Simultaneous Localization and Mapping
SSD	Solid State Disks
Tx.y	Task number y belonging to WP x
TPV	Time, Positioning and Velocity
UC	Use Case
UDP	User Datagram Protocol
UI	User Interface
UPER	Unaligned Packed Encoding Rules
V2X	Vehicle-to-Everything
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VR	Virtual Reality
VRU	Vulnerable Road User
WebRTC	Web Real-Time Communication
WP	Work Package
XER	XML Encoding Rules
XR	Cross-Reality

Executive Summary

This deliverable documents the status of T5.1 “Application development”, describing the work done on the application development of the 3 use cases (UCs) that are deployed in the Pledger project and in particular, the prototype versions of the UC applications. The applications developed in this task implement the functionalities of each UC, namely mixed reality applications on the edge, the enhancement of global road safety for vulnerable road users, and the manufacturing of data mining with edge infrastructures, as per UC1, UC2, and UC3, respectively. The basic application description and relationships with the Pledger platform have been presented in WP2 deliverables, which serve as the basis for the work done in T5.1.

This document details the design of each application and how the development of each module that composes the applications has progressed up to the time of its submission. Furthermore, the document provides the description of the initial testing and validation setups for each UC, which are used for testing and integration purposes, but also for initial demos to showcase a subset of the final functionalities of each UC and a first integration with platform components.

This document is the first of two iterations of the deliverable which are delivered in M20 and M33, respectively, and is directed towards anyone who is interested in the details of the application development for the three Pledger use cases, as well as the common design principles relevant to the integration with the Pledger platform.

1 Introduction

1.1 Purpose of the document

This deliverable describes the initial work performed in T5.1 "Application development", presenting the design of the UC applications and their development status. The design of each application is carried out by its respective UC leader, who is leading the effort to design and implement the application.

Since the application development across the UCs is mostly independent from each other and applications might vary a lot on a functional level, but also in their requirements (Computing resources, underlying operating system, etc.), and how they are virtualized, an effort has been made to define a common terminology to be able to describe and categorize how applications are designed and implemented in the context of Pledger. The understanding of the nuances of the modules that compose each application is key to understand how the applications can be embedded in the Pledger ecosystem and its features and how the integration with other components can be handled.

Another important aspect of the application development addressed in this deliverable is the definition of the application metrics and statistics that can be later used during the testing and validation to identify the performance and correct operation of the application, but also provide further insights into metadata that might be interesting for the service providers. To ensure optimal performance, the placing of the application is very important and as such, recommendations or even requirements are given for the placing and scaling of those application modules.

Another important topic covered in this deliverable is the definition and preparation of plans to validate and test the UC applications, as well as the definition of demo setups and scenarios that can be used to showcase the status of each application development. The outcomes of a set of initial validations are presented in this document.

1.2 Relation to other project work

The UC applications have been initially defined in WP2 of the project. This initial definition covers the basic functionality to be implemented, as well as requirements to be fulfilled. In this deliverable, the implementation and final design choices for the so far implemented parts of the UC applications are documented.

The integration of the applications with the rest of the Pledger platform is a work performed in T5.2 "Integration and Demonstrators Setup" and as such any related concepts are only explained briefly for the sake of completing the picture of the application development process, showing which information developed in T5.1 can be used by T5.2.

1.3 Structure of the document

This deliverable has an overall of 6 Sections, including this introduction (Section 1). The introduction to the key concepts and terminology used to describe the application development and design paradigms is done in Section 2. Sections 3, 4, and 5 are dedicated to UC1, UC2, and UC3, respectively. Each UC describes the development of their application up to the point of writing this deliverable. Conclusions are provided in Section 6.

2 Application design concepts

This Section explains the basic concepts and the terminology used in the development of the different UC applications, serving as the basis to be able to describe how they are built, if and how they are virtualized, which computing resource requirements they have, and which metrics and statistics can be used to measure the applications' performance to validate their functionality. Furthermore, the placement and scaling considerations of the applications are elaborated, considering that the applications will be subject to the scaling and placement mechanisms implemented by the Pledger platform. In the following subsections, each of the aforementioned concepts is introduced in detail.

2.1 Application modularization

Each UC application fulfills a specific functional task (or a series of tasks), as defined in the use case descriptions in WP2 deliverables D2.1 [1], D2.2 [2] and D2.3 [3]. In the early design phase documented in D2.1, flow diagrams depicting the functionality of each UC application were produced. These diagrams split the overall UC functionality into several subfunctions. These distinct functional parts of an application can be grouped and mapped to different modules of an application.

A microservice architecture enabling modularization is a design approach used frequently in application development, as an alternative to a monolithic design which is difficult to maintain and test. Each module can be developed independently, at different tempos and with different degrees of completion, while testing and validation are performed. A modularized application has also the advantage that the submodules can be deployed to different containers, VMs or bare metal services, depending on whether they can be virtualized or not. Independently from the type of virtualization used, a concept that is discussed in detail in Subsection 2.2 (having separate modules that implement microservices) introduces flexibility during the placement of the functional modules, with each module having different requirements and dependencies from other modules.

In Sections 3, 4 and 5, each UC lists and describes the modules that compose the UC application and their dependencies on other modules. It is important to note that each UC provides different levels of details on their application modules. Whether more or less details are presented, depends on whether the code is public or not, if it is part of an asset that might be public or private, whether details can be provided because of confidentiality, etc.

2.2 Virtualization Options for the Applications

Virtualization of applications and services plays an important role in the Pledger project. Virtualization enables the orchestration of services, i.e., lifecycle management, as well as scaling and placement options, which introduce a high degree of flexibility of resource usage. This flexibility can be used to optimally serve the requirements of the virtualized services, by choosing the right physical computing node on top of which virtual services are deployed, taking resources like Random Access Memory (RAM) or virtual Central Processing Units (vCPUs) into consideration, but also other relevant factors like the proximity to the users that interact with the application. Virtualization has also the great advantage of making services independent of the underlying hardware, which instead of specialized hardware can be custom of the shelve (COTS) devices, which are cheap and highly available.

We can find different types of virtualization used in Pledger across the three use cases: Virtual Machines (VMs) and containers. It should also be noted that some functions of the UC applications cannot be virtualized, as they are natively running on bare metal or a transition to a virtualized implementation does not make sense or is technically not possible.

For each software module that is part of an UC application, the virtualization options are the following:

- **Virtual Machine (VM):** The software module is running as a VM that hosts its own operating system and can be hosted by any machine that has a hypervisor. Cloud services like Hetzner [4] or Amazon Web Services [5] offer hosting services on remote servers, where VMs can be deployed. The main alternative to these cloud services is the use of local computing nodes of the UC infrastructure. This could be a data center that offers a pool of computing resources or an edge computing node with maybe less resources, but the benefit of low latency services for the end users. VMs can be orchestrated by orchestrators such as Open Source MANO [6], enabling dynamic lifecycle management. The Pledger orchestrator does not manage VMs directly, but it can be done by managing containers running inside (see below).
- **Virtual Container:** Containers are a more lightweight and dynamic option when it comes to the virtualization of functions. A container packages the key functionalities of an application rather than an entire operating system, as it is the case for VMs. As such, a container leverages the features of the underlying operating system, which makes it more lightweight, but also limited to the capacities of the hosts operating system. Software like Docker [7] allows the packaging and runtime management of containers. Much like VMs, containers can also be orchestrated. A variety of commercial orchestrators for containers (coming from Amazon, Mirantis, Google, etc.) exists, but there are also open source solutions. Probably the most known container orchestrator is being developed by the cloud native foundation, called Kubernetes [8] (K8s). The Pledger platform is capable of orchestrating containers via K8s. Note that K8s worker nodes, i.e., computing nodes where containerized applications can be deployed, apart from being deployable on bare metal computing nodes, they can be deployed within VMs. This option is interesting for infrastructures where a Virtual Infrastructure Manager (VIM) like OpenStack [9] already manages the infrastructure, but where a cloud-native paradigm is to be tested.
- **Bare metal (no virtualization):** Some functions or applications cannot be virtualized, and they run directly within the operating system of a computing node as a native process. There can be several reasons why a software cannot be virtualized: there are strict hardware requirements that cannot be met by COTS hardware, licenses might not allow an application to be packed into a virtual machine or container, the hardware might not form part of the edge or cloud resources and be isolated from the orchestrator (rendering virtualization less useful), etc.

2.3 Resource requirements of the Applications

Each application module has certain requirements on the HW capacities of the computing node it runs on: RAM, vCPUs, and storage on solid state disks (SSDs) represent the three key requirements. They can be used to identify if an application is lightweight or heavy, which is crucial to understand where an instance of the application can be placed. Lightweight applications can be placed both at the edge and in the cloud, as they do not require a considerable number of resources. Heavy applications might have less placement options, since often computing resources at the edge are less than in the cloud. But there are other factors that can determine the placement of an application that might come from the functional requirements, such as being delay-tolerant or requiring small delays, in which case again either running in the cloud or the edge might be a better choice, respectively.

Beyond the three basic computing requirements, there can be further requirements: some applications might require graphics processing units (GPUs) or other specific hardware to do the processing, such as Field-programmable Gate Arrays (FPGAs).

2.4 Application Metrics and Statistics

During its runtime of an application, it can be crucial to understand whether it performs optimally and also whether it is fully operative. Apart from observing the resource usage and performance statistics, which are values provided by the Pledger platform, keeping track of the overall operationality of a service can be achieved by observing a set of measurable metrics or statistics at the application layer. In fact, gathering this information is necessary to validate the UC applications, which is a task that forms part of every application development and is key for the demonstrations and pilots.

Since these measurements at the application layer are done outside of the context of the Pledger platform and are specific to each UC, they are described briefly to better understand their relevance to the different applications and how they are measured or visualized. Typical metrics can be for example the response time to certain events, the delay between two (application) endpoints of the system, or the reliability of a service, measured in number of successfully processed data during a time interval.

2.5 Placement and Scaling Considerations of the Applications

From the software modules that build a UC application, some might be flexible in terms of their placement location, e.g. edge vs. cloud, or they might support scaling. Not every computing location might be suited though, as the resource utilization, the (physical) distance to other sevices, etc. might impact on the performance of the different application modules. As such, if an application module supports either scaling or placement - or both -, we indicate any relevant aspects that might need to be considered by the decision support system (DSS) core module, the module that based on certain triggers, decides when to scale or move a container that is part of an application. A prerequisite to enable placement and scaling of the applications is the definition of the metrics and thresholds for SLA monitoring purposes that are handed over to Pledger's Configuration Manager and then used by the SLA Manager module to detect breaches and notify the DSS. These three tools (Configuration Manager, SLA Manager, DSS) work together to assure the right placement and necessary scaling operations to meet the SLAs. The initially considered (and yet to be evaluated in-depth) SLA metrics to be used for the UC application modules are reported here.

2.6 Development and Reporting Methodologies

The development of the UC application has been carried out by each UC leader independently: each partner internally uses different development frameworks, may rely on specific tools or use different programming and development methodologies. It is important to highlight that such internal (and mostly confidential) information is not exposed in T5.1 and as such does not form part of D5.1. Also, it means that each UC might have progressed differently with the application development, as becomes clear from the description of each UC application in the following sections. As an example, UC2 has started with the parallel development of all submodules of their application iterating over these modules with new code and modifications, whereas UC3 has followed an approach where a part of the submodules for their application already have been fully developed and finalized, whereas the development for the rest of modules is still to be initiated. The details on how each UC has tackled the application development are provided in Section 3-5.

While the application development process has been independent for each UC, there is, however, a common set of tools for the application development that are available to the UC leaders and that have emerged from the work done in T5.2 – Integration and Demonstrators setup, the task responsible for the integration tasks. The tools include a project wide Gitlab [10] repository that provides continuous integration and deployment features. A Jenkins [11] server is also provided by Pledger to build, test, and deploy the software. A shared docker registry serves as a common space where application images can be stored and distributed.

3 UC1: Mixed reality applications on the edge

The goal of UC1 is to enhance the capabilities of Augmented Reality (AR) solutions by coupling them with edge computing technologies, and to stream the entire AR application to an immersive Head-Mounted Display (HMD). The *PledgAR Workspace* developed in Pledger enables engineers to visualize and work with 3D computer-aided design (CAD) data in an AR and Mixed Reality environment, with the aim of taking prototyping, factory planning, quality control, and technical education to the next level. Security and data protection play a major role, therefore, methods like secure connections via ledger-based technology will guarantee high security standards. The diagram in Figure 1 shows the interaction between the application logic rendering and the users that consume the 3D-visualized imagery for quality control purposes.

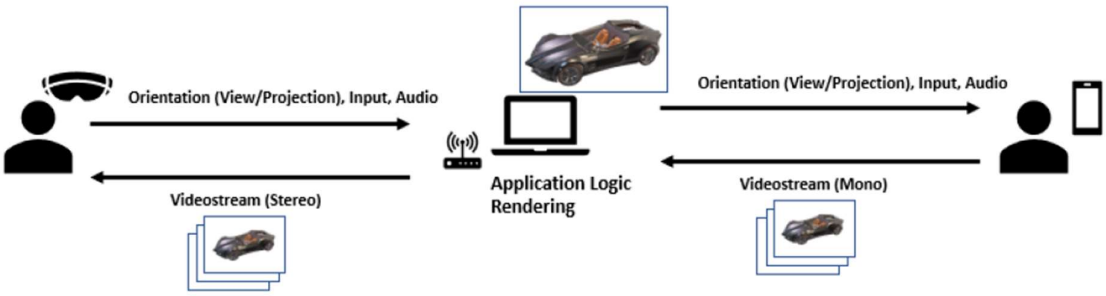

Figure 1: PledgAR Workspace Concept.

3.1 UC Application Description

UC1 will be demonstrated together with FILL, at their facilities. FILL provides an industrial computer numerical control (CNC machine), called Syncromill (see Figure 2).

Figure 2: FILL's Syncromill.

Using the PledgAR Workspace, several scenarios will be developed in UC1. On the Syncromill we focus on the three industrial scenarios: Training, Collaboration, and Service & Maintenance.

3.1.1 Scenario: Enhancing Training & Interaction

A typical service process is to change the shaft (see Figure 3) of a CNC machine. A developed feature in the PledgAR Workspace is to create simple step-by-step instructions that a trainee can follow and execute. This can be done either virtually on a holographic representation of the machine or on the real machine. Each task will be concluded with a brief questionnaire to ensure that the trainee has understood the process completely.

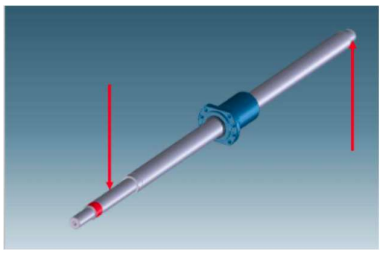

Figure 3: Shaft of CNC machine.

3.1.2 Scenario: Collaborating in Fast Prototyping

The biggest downside of working with holographic 3D models, visible only through smart glasses, is the hindered collaboration. So, in order to remove this problem from the equation, HOLO has worked on a solution that can support the sharing of 3D holographic content through several physically separated devices: an avatar-based multiplayer (see Figure 4) feature, enabling engineers to share and review 3D models. On FILL's facilities HOLO will connect the production engineer stationed on the shop floor with a CAD engineer in the Backoffice where they will share and review changes on some part of the Syncromill's CAD model. Additionally, the collision-recognition feature, which is currently in development, will check if the newly designed parts collide with sections of the machine. The main benefits of this feature are the avoidance of high costs for prototyping and testing and eco-friendliness, as no resources must be wasted.

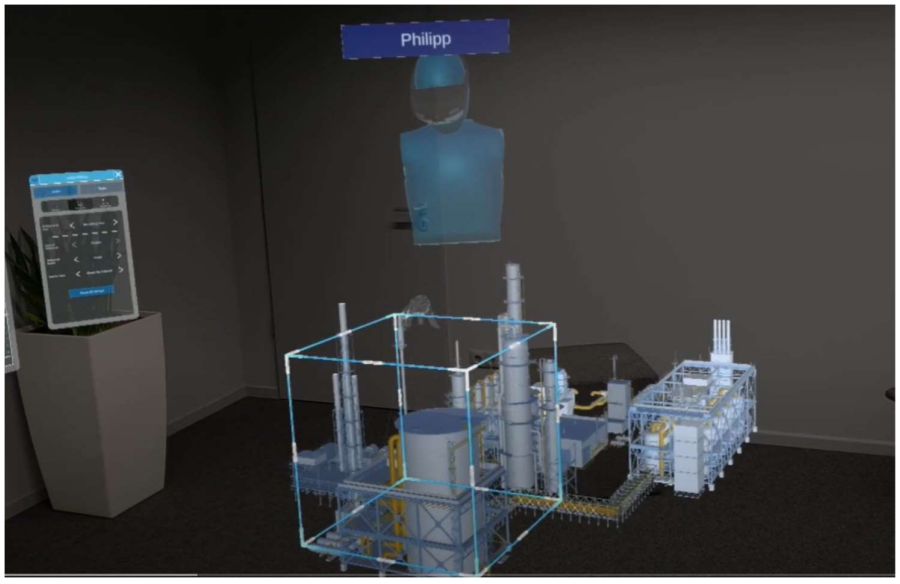

Figure 4: Avatar based Multiuser for collaboration.

3.1.3 Scenario: Assisting in Manufacturing and Service & Utility procedures

A CNC machine (like FILL's Syncromill) requires - over its product lifetime - continuous service & maintenance. FILL's backend gathers plenty of machine data and outputs important values concerning the machine's lifetime. To support the responsible employees with their task, a holographic overlay of important data over the real CNC machine will be implemented. A user could, for example, see the history of the last 10 parts produced, the state of the machine (clamping, milling, etc.), progress of the warming up or shutting down phases, and more.

3.2 Application Modules and Dependencies

The UC1 application can be divided into 3 functional submodules that each have a specific role. The combination of them, the developed PledgAR Workspace, will also be addressed. The following subsections detail these modules, as well as the final product.

3.2.1 Module: AR3S

AR3S[1] is the Go-To XR (Cross-Reality) CAD Workspace for the industrial engineers. This software enables engineers to visualize and work with 3D CAD data in an AR and Mixed Reality environment, taking prototyping, factory planning, quality control, remote support, and technical education to an immersive level.

Specific features of this application have been developed (or will be developed throughout the reminder of T5.1) to support the use case. These features have been outlined in WP2 deliverables and are now part of the User Stories (as defined in D2.2).

3.2.1.1 Module implementation details

AR3S as such is a standalone application that runs on the HoloLens 2 [12], the HMD used in UC1. AR3S consists, however, of dozens of submodules that can be combined as needed for a specific use case. The newly created modules are following the overall required structure that allows this pick-and-plug approach.

Although the result is an application, the implementation to the Pledger Framework is not possible at this stage. The orchestrator would not be able to build an application onto the desired hardware, due to hardware restrictions.

3.2.1.2 Module Requirements and Virtualization

AR3S requires at least 500 MB of free storage on the HoloLens 2. This does not include 3D CAD files.

3.2.1.3 Placement and scaling considerations

The placement and scaling considerations do not come into place with this module. The final application, described further below, will address what conditions are needed for the integration with Pledger's placement and scaling mechanisms.

3.2.1.4 Inter-Module dependencies and integration considerations

AR3S has several inter-module dependencies that depend on the plugins used. The interaction features depend on the hierarchy feature. File management has an impact on the measurement and multiplayer feature. The tracking feature depends on the camera feed, as it is necessary for QR code scanning.

In order for all required sub-modules to be implemented in the Pledger Framework, another module has to be added, namely the module ISAR, which is described in Section 3.2.2.

3.2.2 Module: ISAR

ISAR is a Software Development Kit allowing developers to stream their entire application from a server, i.e., Laptop, to an XR device with minimal integration effort. Although ISAR was not developed for Pledger, it is necessary to support the integration into the Pledger Framework.

3.2.2.1 Module implementation details

The ISAR Unity3D [13] plugin will be implemented to the PledgAR workspace, enabling the integration with the Pledger Framework. The high-level concept (see Figure 5) to enable this remote application streaming can be summed up as follows:

[1] In all previously created deliverables, this module was called ARES. Beginning of 2021, however, HOLO had to rename it to AR3S.

- Holo-Light provides a code library – an easy-to-use SDK – which enables the remote application streaming capabilities.
- The module uses the WebRTC protocol and is written in C#.
- It is a 2-component solution where the main application - containing all relevant features for the user - are installed on a server, or VM and a small client application is installed on the device.
- The Unity3D plug-in consists of additional APIs for image tracking, QR code tracking, audio transmission, as well as profiles for the Mixed-Reality Toolkit (MRTK) and ARFoundation Framework (both available for Unity3D).

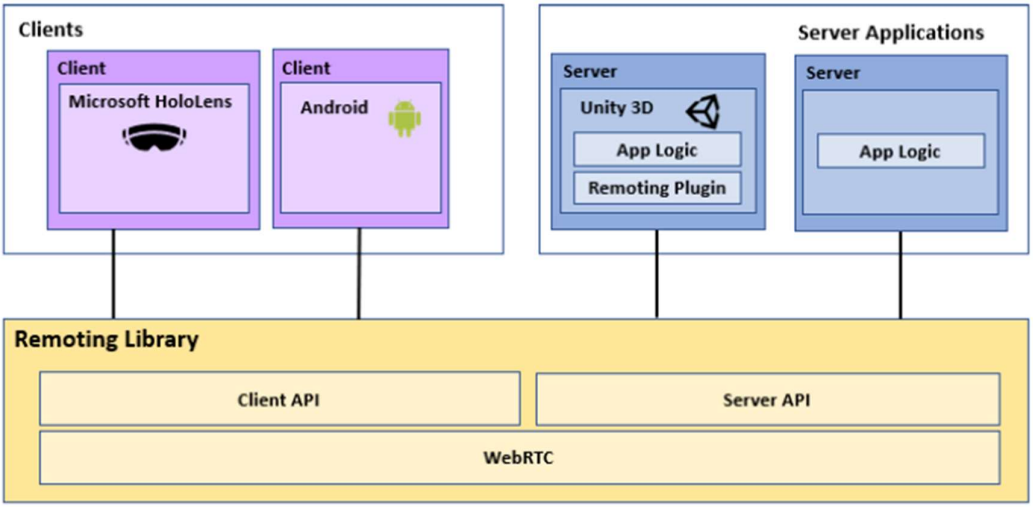

Figure 5: Diagram depicting the ISAR Functionality.

3.2.2.2 Module Requirements and Virtualization

The ISAR module has specific requirements on the clients' user equipment, their wireless connectivity and on the servers (streaming, rendering, signaling) that implement the application logic. The communication between a Simultaneous Localization and Mapping (SLAM) capable XR device and the edge server needs to be wireless. Therefore, the streaming can be established via Wi-Fi (2.4 GHz or 5 GHz) with the following recommended characteristics:

- Network: Wi-Fi operating in the 5 GHz band
- Bandwidth: min. 20 Mbit
- Round Trip Time (latency): max. 50 ms

The client application is installed on the targeted XR device (mobile SLAM-enabled AR/VR device). Once completed, the client app requests an instance from the render server in order to pass input and sensor data to it. In return, it receives and displays the content stream. No additional rendering is performed on the target device. The client application is customizable which enables the implementation of additional services. The following XR device platforms are targeted:

- Microsoft HoloLens 2
- Android – ARCore enabled smartphones and tablets
- iOS – ARKit based smartphones and tablets

The remote application streaming server plug-in is a set of functionalities (packed into a single .dll) that can be integrated into any 3D-capable rendering engine. The plug-in can simultaneously be used with XR Frameworks like Mixed Reality Toolkit (Microsoft HoloLens 2) and AR Foundation (ARKit and ARCore). Current supported graphics backend is DirectX11.

The XR application is hosted and managed on a render server. The render server (VM or personal computer) needs to have a proper hardware setup to run graphically intensive 3D applications, as detailed in Table 1 that shows typical requirements.

Table 1: Requirements Table for ISAR

OS, Specs, etc.	Requirement
OS	Windows 10 or Windows Server 2019
Windows Version	10.0.17763 Build 17763
RAM	min. 16 GByte
CPU	Intel I7 (7. Generation or equivalent)
Graphics Card	**OnPrem**: NVIDIA Geforce 10X0 with min. 5 GB RAM **Virtual Machines**: NVIDIA Tesla with min. 5GB Memory incl. NVIDIA Grid
Ports	Enable Incoming TCP Port 9999 | Enable all outgoing Ports

The Signalling Server manages device requests and builds the peer connection between render server and client. When the Signalling Server receives a request from a client, an instance of the application is started on the server. As soon as the instance runs, the peer connection is released, and the rendering process is initialized. The client sends input and sensor data to the server, which renders the scene based on that data and sends it back to the client.

3.2.2.3 Placement and scaling considerations

The placement and scaling considerations do not come into place with this module. Instead, ISAR is the necessary tool that will allow the integration of the Orchestrator, scaling and placement tools and the DLT component.

3.2.2.4 Inter-Module dependencies and integration considerations

ISAR depends on a 3D immersive application. For this use case the 3D application is called "*PledgAR Workspace*". ISAR depends on the ledger-based secure signaling between server and AR client and it depends on the orchestration module, in order to fast start a VM, when user needs them.

3.2.3 Module: DLT (WHISPER PROTOCOL)

Before remote rendering is possible, a peer-to-peer connection must be established between the server and the client application, a process called a handshake. In order to allow the handshake between the client (Augmented Reality Head Mounted Device) and the server (Laptop), the nodes running on the blockchain (DLT) require activating the Whisper protocol (shh) provided by Ethereum and performing JSON-RPC requests.

The nodes represent endpoints (IP address + Port) which have to discover and connect to each other before any of the handshake operations start. Since Ethereum only allows the endpoints to use HTTP (which does not implement security), a reverse proxy needs to be setup for each node. The reverse proxy will ensure that the protocol used to exchange data will be HTTPS, thus securing the connection between the nodes and guarantees that no 3rd party could eavesdrop on sensitive information.

Starting up the client and the server; both would connect to the blockchain (corresponding nodes). Then, they would exchange the required handshake data through the Whisper [14] protocol. This protocol allows for complete security and darkness depending on the configuration used. The payloads exchanged are encrypted and can only be decrypted through the corresponding private key. The messages are sent to all the nodes on the blockchain, and they would all receive the messages (receive and broadcast to neighbour) whether it was meant for them or not. That ensures that the destination remains unknown to everyone.

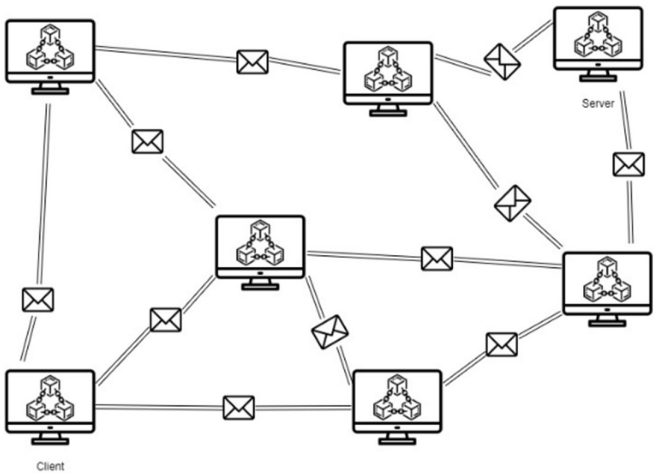

Figure 6: Example network, where Whisper is used for exchanging information between nodes.

3.2.3.1 Module implementation details

The Client and Server application needs an access point to the Pledger DLT. The access details are still being discussed.

3.2.3.2 Module Requirements and Virtualization

This solution is happening only in the backend, hence there will be no virtualization. The Whisper protocol does not have any particular resource requirements, but a series of networking, communication and security requirements:

- The nodes running on the DLT that the client and server are connecting to needs to have the Whisper protocol (shh) activated.
- The nodes (endpoints) should have a public IP address and port to allow incoming requests.
- A reverse proxy and SSH certificate set up for each node (endpoint) to establish an HTTPS connection.

3.2.3.3 Placement and scaling considerations

This module has no placement and scaling dependency.

3.2.3.4 Inter-Module dependencies and integration considerations

This module will be integrated within the DLT Pledger subsystem and will be integrated into ISAR and client library to allow signaling through blockchain.

3.2.4 Module: PledgAR Workspace

HMDs are generally limited in terms of hardware resources and struggle when running a model with millions on polygons. If a user would try loading such a big model, the device would drop in performance to less than 60 fps (frames per second), what creates at best a very bad user experience or worse, it causes headaches. Alternatively, the number of polygons can be lowered to render the objects but that would reduce the model's quality reducing the user experience or usefulness of the application.

To remove this issue and support the full usage of Pledger's components (Orchestrator, DLT, and scaling and placement tools), the main module is a combination of existing AR3S submodules, submodules developed for Pledger and the Unity3D Plugin ISAR.

The *PledgAR Workspace* fixes this and enables the user to enjoy a simple setup of the tool, a smooth experience and high model detail through remote application rendering, when using the developed features.

3.2.4.1 Module implementation details

The application PledgAR Workspace is a two and a half-component solution. The server component (ISAR Client) needs to be deployed on a mobile XR device (e.g., Microsoft HoloLens 2). The server component is provided as an executable (.msi installer), which need to be installed on a Windows 10 based VM and will be part of the snapshot required by the Orchestrator, monitored by the placement and scaling tools. The final part is the blockchain component that ensures a secure connection between both and connects to the DLT.

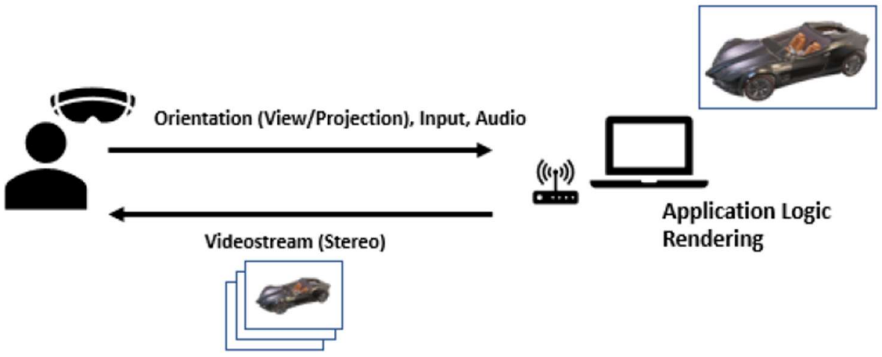

Figure 7: PledgAR Workspace Concept.

3.2.4.2 Module Requirements and Virtualization

The application PledgAR Workspace is hosted and managed on a render server. The render server (Virtual Machine) needs to have a proper hardware setup to run graphically intensive 3D applications. Since PledgAR is based on the ISAR application, the minimum server requirements are the same as listed in Table 1.

3.2.4.3 Placement and scaling considerations

A first feature of the placement and scaling tools is to keep track of the infrastructures' topologies, of the App constraints and those configured by the service provider. Within such constraints, the DSS issues any scaling to ensure the right number of resources to each App service, either reducing if the amount requested is not used for some time and no SLA warnings are received or increasing if the opposite.

UC1 is resource intensive and so it makes sense to have scaling of resources when needed by the App. For the time being, two possible scenarios are proposed:

- Scale memory: The larger the file, the more RAM is being used during the initial loading process of a file. Allowing multithreading or infinite RAM can reduce or remove loading buffers and improve the user experience overall.
- Scale CPU: Constantly updating files, as would be used during simulations, often push the hardware's CPU to its limit. Being able to increase the CPU could bridge that gap.

While using PledgAR Workspace, the memory and CPU usage will be monitored through well-defined metrics and using specific thresholds (the evaluation of these is ongoing). Once the threshold was crossed an upgrade or downgrade event will be sent by the SLA Manager to the DSS. In return the DSS will initiate the interaction with the Orchestrator to launch a new instance of the VM now showing the required resources.

3.2.4.4 Inter-Module dependencies and integration considerations

The Server-side application, which will be deployed on the VM, is built from ISAR & AR3S Modules and therefor inherits all dependencies.

Furthermore, to gain access to the PledgAR Workspace, a user has to request the setup of the VM (Virtual Machine) through the Pledger Orchestrator, following the predefined configurations. This VM will already feature a snapshot of the application. Hence, the remaining step is the installation of the Client application on the HMD.

During the following months it will be analyzed if multiple instances can be setup, which feature different specifications, i.e., Processor, RAM, etc. to support different types of use cases and allow the testing of the scaling operations. It also requires Signaling between client and server therefor the DLT Module.

3.3 Application Metrics and Statistics

Typical continuous technical metrics to ensure proper user experience of the streamed AR3S application are:

- Roundtrip Time of the data packages between XR Client and Rendering Server
- Perceptual latency based on parameters in late stage reprojection
- Running framerate of AR3S on the rendering server
- Server-side allocated network bandwidth (sending & receiving)
- Client allocated network bandwidth (sending & receiving) and complementary package loss

These metrics are and will be used for the validation of the UC1 application functionality.

3.4 Development and Deployment Plan

This section lists the completed, ongoing and upcoming features of the PledgAR Workspace. Many of the features aim to improve the user experience, but also add improved or new tools to the repertoire available to the user. The core functionality of the PledgAR workspace has been implemented. As such, the ongoing and upcoming feature development focuses on improving the quality of experience and key features for users of the PledgAR virtual experience.

Completed Features

- *3D file loading* and *interacting* has been completed: during this process HOLO created their own format (HLCAD) to allow faster file sharing during multiplayer sessions.
- *QR code tracking* features have been implemented, allowing precise object tracking.
- *Avatars* will be displayed during multiplayer sessions, to allow for an improved collaboration experience.
- The *file importers* have been improved upon, hence the current application supports *.obj, .fbx, .jt (v 8.1 and v 9.5)*, as well as *.gltf* file formats.
- *Saving an AR workspace* was implemented. This feature was a basic requirement for the still ongoing development of the content creation part.

Ongoing Feature Development

- The multiplayer function has partially been developed. Currently, up to 3 people can join a session. The main focus for the upcoming 5 months is optimization and stability.
- First results were made with content creation. The application now supports taking measurements in the AR space. During the upcoming 6 months, features such as angle measurements and more will be developed.
- 1st implementation of Blockchain technology in the PledgAR Workspace was successful. During the next phase it has to be optimized and fully integrated with the DLT component.
- The first iteration of the step-by-step instruction feature was completed, but it still requires optimization. The upcoming months will be dedicated to improving the UI and simplifying the instructions.

Upcoming Feature Development

Between month 23 to 26 HOLO will develop the feature allowing the active highlighting of collisions in the AR workspace. Month 27 and further will be used for testing and optimization.

The following months until 33 will be used to optimize all components in combination.

3.5 Initial UC Application Validation and Testing Outcomes

First basic tests have been conducted with the PledgAR Workspace application. These tests were carried out in HOLO's office space and not yet on the FILL's site.

Sample 3D files of a factory and a car were fully loaded in the selected 1:100 (factory) or 1:1 (car) scale. Interactions such as scaling, moving, rotating, worked for the complete file, as well as specific parts, without issues in a single player session, as well as for the host in multiplayer sessions. Nevertheless, issues were encountered with the interaction of the client with the 3D model. Hence this feature will not be demonstrated in PM 20.

Measurements were taken successfully, when point A and point B were selected with near or far interaction. Issues occurred with .NET framework updates. These have been addressed but could return with the next Windows Update.

A basic step-by-step instruction feature was added, but still requires optimization, to allow a better user experience. Test subjects complained about the number of buttons to click and responsiveness of buttons in the AR space.

3.6 Demonstration and Trial plan

The first demonstration shown during the interim review will be a video recording of a training execution at FILL's site. This video will show a recorded training session, where a single user is guided through the required steps to replace a broken part of the Syncromill machine in AR. This session will be held during a day-long visit in early July 2021. The steps are based on the actual instruction manual provided by FILL.

During the final review demo, Pledger will show (live or as video recording) an extended version of the training, as well as the other scenarios that require currently not completed features, such as multiplayer, collision detection and the integration of FILL's Dashboard to display meta-data for the "Maintenance" case.

Most core features have been completed in the PledgAR Workspace. M17 began with the concept creation of the UC execution, focusing on the training scenario. First trials of the training scenario are conducted during M19. Following that, more frequent visits to FILL's site will have to be scheduled, allowing the completion of the roadmap for the outstanding scenarios.

Additionally, a workshop is planned to be held on FILL's site in November 2021 and a possible collaboration with the FutureLab of FILL may take place.

4 UC2: Edge Infrastructure for Enhancing the Safety of Vulnerable Road Users

UC2 is deployed in Barcelona, exploring mechanisms to enhance the safety of vulnerable road users (VRUs) in the vicinity of public transport stations, where risky situations can emerge due to the concurrent layout of the roads, train rails, pedestrian walks, and bicycle lanes. UC2 makes a proposal for enhancing safety for both pedestrians and users of bicycles or electric scooters that are transiting a tram station. Figure 8 shows the scenario on top of which the UC is built. The bicycles (or electric scooters; from here on only bicycles are mentioned) transiting this area use a bicycle lane that separates the tram lane from the pedestrian lane. This implies that bicycles are forced to cross an area that is potentially used by pedestrians that are entering or leaving a tram. Considering that the tram station can potentially block the sight, the risk gets even bigger.

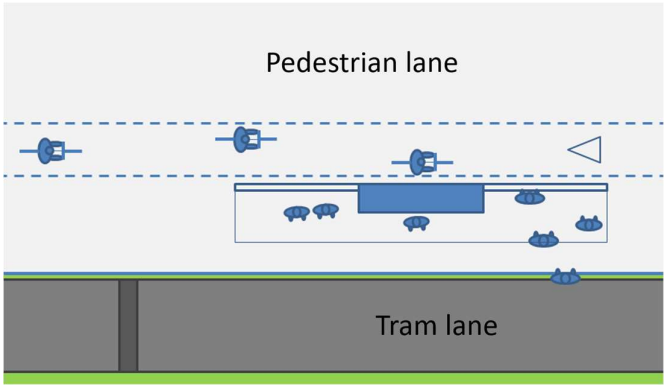

Figure 8: Layout of tram station with the tram lane (bottom), the tram station with pedestrians waiting for the tram, the bicycle lane (between dashed lines), and the pedestrian lane (top).

As explained in the following subsection in detail, the risk detection and notification system (RDNS) developed in this UC builds upon different technologies and methodologies. Vehicular communications, image processing and a risk detection logic running on top of a dedicated radio and computing infrastructure form the base to implement the security enhancement service. This section presents the details about how these components form part of the overall UC application and how they are used to implement a service that delivers the safety enhancement mechanisms.

4.1 UC Application Description

The RDNS application implements a logic that can determine when VRUs might be exposed to a risk that could lead to an accident between bicycles and pedestrians. By knowing the location of bicycles that are approaching a tram station, and by being aware when a tram is entering the tram station, the RDNS application can estimate possible collisions or risky situations between the VRUs. If such a situation is detected, the drivers of the affected bicycles need to be made aware of it via an audio-visual notification, indicating to be on alert and to reduce speed when approaching the tram station. This type of alert could also be notified to other endpoints, such as a small device installed in the tram station or the tram itself, as long as these endpoints are connected to the UC deployment. Note that at the time of writing, only the notification for bicycles can be considered for the UC as any other infrastructure access (tram station/tram) is currently not yet confirmed.

Having explained the basic functionality of the UC, a series of required functions or services can be derived: the UC application has to be able to determine the location of bicycles and there has to be a way to communicate with them in order to send out alerts. Both requirements can be fulfilled at once with the use of vehicular communications relying on the IEEE 802.11p standard. This standard enables

a low delay, high reliability communication channel that is used to transport location information and notifications between the infrastructure where the UC application runs and the bicycles.

Also, the UC application needs to be able to detect when a tram arrives, which is the event that eventually results in the transition of Pedestrians from the tram to the pedestrian lane and vice-versa. For this UC deployment, the detection of the tram is handled by an image recognition system that processes a live video taken from cameras on street and can detect when a tram approaches the station and when it leaves it.

Finally, the information provided by these two sources (the bicycle location coming collected over the vehicular communications and the information about arriving trams coming from the image processing function) have to be processed by the application logic. Based on this premise, the UC application can split into three functional modules: The vehicular communications (V2X) stack, the image processing application and the RDNS.

Beyond these UC application-specific modules, the i2CAT infrastructure also uses two further modules that are used to create dedicated end-to-end slices, including radio and computing resources, in complex infrastructure where Pledger is to be deployed. These two core Pledger modules are the RAN Controller and the Slicing and Orchestration Engine. While not forming part of the UC application logic per se and being developed as a WP3 activity, these elements are only present in the i2CAT infrastructure and as such they are tightly related to UC2. Since they form part of the UC2 tools and are used to deploy the UC, we include the two modules in the UC application description, in order to provide the reader with a better understanding on the responsibilities of the modules and to explain their role in the Barcelona infrastructure. More details on these software modules will be provided in the second iteration of the WP3 deliverables.

The following Subsection 4.2 provides the details on each of the aforementioned software modules that take part of UC2.

4.2 Application Modules and Dependencies

The UC2 application can be broken down into 3 functional modules: the V2X stack, the image processing service and the RDNS. Furthermore, the RAN Controller and the Slicing and Orchestration Engine core modules are directly related to the UC deployment, implemented by two modules that are independent from the application modules and used to configure the underlying infrastructure in a day 0 configuration. Also, although not an application module, an important element for UC2 is the hardware used in the vehicular communications: the elements that go on the bicycles and the lamp posts are also being designed, developed and tested as part of the work done in this task. The following subsections present each of the software modules and the vehicular hardware.

4.2.1 Module: V2X Stack

A core element of the UC application are vehicular communications, in particular, vehicle-to-everything (V2X) which enables the exchange of information between vehicles and an infrastructure. In UC2, the Barcelona infrastructure hosts Road Side Units (RSUs) that are equipped with IEEE 802.11p technology. Bicycles with on board units (OBUs) using the same radio technology can connect to the RSUs to share and receive information. In this UC in particular, the OBUs share their position and can receive information about hazards or alerts from the services running in the Pledger infrastructure. I2CAT refers to the software necessary to implement the IEEE 802.11p communications the V2X stack.

The V2X stack developed in UC2 is based on the Vanetza [15] implementation, which has been improved and adapted for the use case by i2CAT. The result is a more lightweight, streamlined version of the stack that provides only the required functionality. In this implementation, the V2X stack can be broken down into two submodules that implement two core aspects: the V2XCOMM module and the Message Queuing Telemetry Transport (MQTT) Broker.

4.2.1.1 Module implementation details

The V2XCOM module takes care of handling GeoNetworking and Basic Transport Protocol (BTP) for standardized and previously encoded V2X messages - following SAE DSRC J2735 [16] definitions -, which sometimes can be accompanied with an extra header of UDP/IP. To do so, the BTP-Geonet module takes care of assembling the BTP/GeoNetworking headers given a geodesic position coming from a GPS service daemon, a configuration which states the destination along with any other configurable parameters that BTP and GeoNetworking headers require and the payload that has to be sent in bytes. The aforementioned relationships between the different submodules are shown in Figure 9.

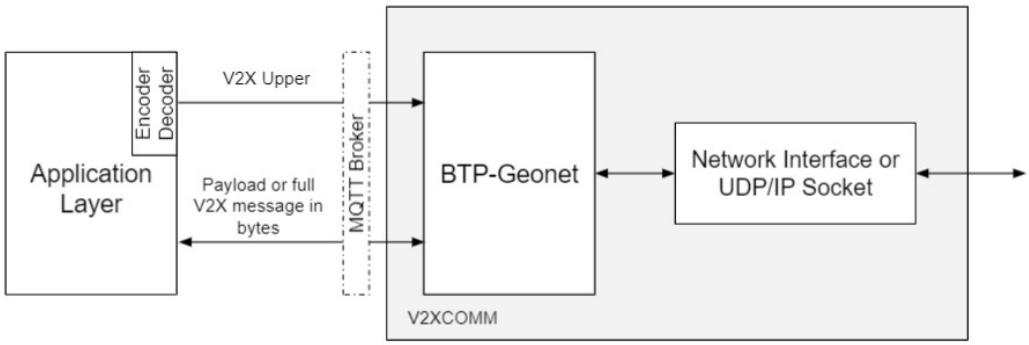

Figure 9. V2XCOM and Application Layer simplified view through the MQTT Broker.

The communication between the different modules is done through MQTT queues handled by a broker. Before going deeper, it is necessary to understand how a V2X message, e.g., a Cooperative Awareness Message (CAM), is generated, encoded and sent. The journey of a CAM starts at the application layer, where it is generated in an XML format. The XML format is friendly and readable not just for any program but also for any person. After that, the application layer sends the XML to the "Encoder" function. The "Encoder" encodes the CAM message according to the standards, so turns the XML into an encoded Unaligned Packet Encoding Rule (UPER) byte array which is automatically sent to the "BTP-Geonet" module using the corresponding MQTT queue. The BTP-Geonet module, depending on the queue from which has received the byte array of the "encoded" CAM, will construct the headers based on a previously given configuration. Also, depending on this configuration, it will sign additionally the packet with the regarding certificate. Once the header is generated and the packet signed – if applies -, is automatically sent to the preconfigured network interface or UDP/IP socket.

As it can be seen in Figure 9, V2XCOMM module simplifies the development of V2X applications with just having to code the XML handling of the given V2X messages. On the other side, the whole process that has been just described is completely symmetric. If a new CAM is received from the network interface, then the "BTP-Geonet" module will pre-process it, by checking the headers and dropping it if necessary. When the headers of a received packet are correct, the "BTP-Geonet" module uploads the packet payload to the "Encoder" which automatically will decode the UPER byte array to the corresponding XML version, following ASN.1[2] codification rules. Additionally, there is also a 'fast track' for non-V2X messages that don't have to be coded or decoded. Also, any array of bytes can be sent with a BTP and GeoNetworking header when wanted.

Inside the V2XCOMM, the BTP-Geonet module takes care of all the logic and implementation regarding the transport and network layer in a given Cooperative Intelligent Transport Systems (C-ITS) system. To do so, as stated above, it receives the payload of the messages to be sent, a configuration of where those messages are to be sent (for example, if they are GeoBroadcasting or Single-Hop Broadcast,

[2] Described by Rec. ITU-T X.680 | ISO/IEC 8824-1, Rec. ITU-T X.681 | ISO/IEC 8824-2, Rec. ITU-T X.682 | ISO/IEC 8824-3, Rec. ITU-T X.683 | ISO/IEC 8824-4.

the certificate that has to be used to sign them, or the area on which are to be sent), the current GPS position and a connection to the lower-layer network interface or the UDP/IP socket if wanted.

To be completely adaptable, the module requires dynamic and flexible configurability. Also, it is taken for granted that multiple instances of the BTP-Geonet module will be running at the same time. Not just because it will require ensemble messages considering different locations, but also because the use case might require handling different interfaces/sockets (outputs) for the crafted V2X messages. So, each instance, when launched, will be configured to be hooked to a specific position (that will be prefixed or constantly scrapped from a GPSD), a network interface or UDP/IP socket (as the output for the generated messages) and the name of the instance (which will be used later at the MQTT queues naming convention).

Once the static characteristics of the configuration are settled at launch, the more dynamic ones (BTP port, GeoNet destination...) require a different treatment. For that reason, the V2X stack handles each configuration for sending V2X messages with the BTP and Geonetworking protocol as a "socket" which can be opened through passing the new socket configuration in JSON format through MQTT at the queue named "<instance_name>/open" (where the "<instance_name>" is the previously configured name for the instance). As an example, for opening a CAM socket the JSON would look like as follows:

```
{
        "btp" : {
                "port": 2001
        },
        "geonet":{
                "certificate": "/files/ticket.cert",
                "key": "/files/ticket.key",
                "cert_chain": ["/files/root.cert", "/files/aa.cert"],
                "mode" : "shb",
        },
        "general":{
                "timeout": 600,
                "name" : "cam",
                "type": "cam"
        }
}
```

The BTP port is configured with the 2001, the packet signed certificate is specified and also the name of the socket at the "general -> name" configuration parameter. So, all the communication to the recently opened socket will be done through the queues which name starts with "<instance name>/<socket name>". One of the examples of what can be done with the socket is closing it. To do so there just has to be sent anything to the queue "<instance_name>/<socket_name>/close" and it will automatically close.

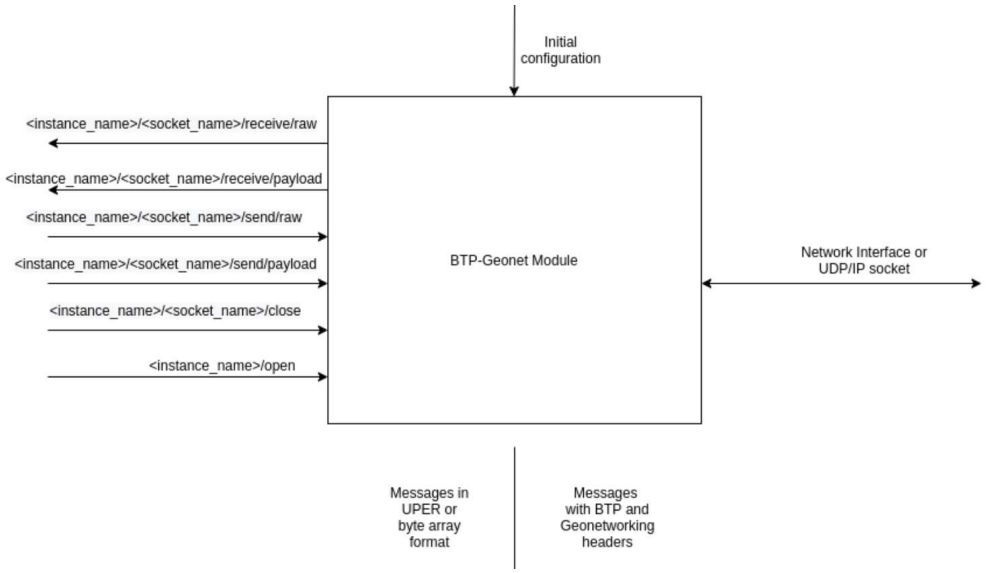

Figure 10. BTP-Geonet Module with MQTT Broker queues

To send a V2X message, the payload of the message just needs to be published at the queue "<instance_name>/<socket_name>/send/payload" and the BTP-Geonet module will ensemble the headers and send the message. To forward an entire message (already signed) which doesn't need the crafting of more headers, there just has to be published at the "<instance_name>/<socket_name>/send/raw" queue. Finally all V2X messages received will be automatically published at the queues: "<instance_name>/<socket_name>/receive/ raw/[#]" and "<instance name>/<socket name>/receive/payload/[#]". Where at the payload with "raw" the entire byte array (including headers) will be published and the "payload" one just the payload. Finally, the "[#]" will be a number identifier for the packet, so there is a way of relating payloads to the whole message. Figure 10 resumes the inputs/outputs of the module.

The architecture is completely symmetrical so if the socket receives a CAM message, it will publish it automatically to the queues: "instance1/cam/receive/raw/1" and "instance1/cam /receive/payload/1" (considering that's the case of the first CAM published). Finally, the socket will be closed when publishing anything to the queue "instance1/cam/close".

Given the Pledger requirements, Decentralized Environmental Notification Message (DENM) are more suitable along the use case to be transmitted and received among the nodes. While CAM just informs other road users about the presence of the transmitter, DENM contains information related to a road hazard or an abnormal traffic conditions, such as its type and its position so the driver is then able to take appropriate actions to react to the situation accordingly.

In this use case, a cyclist will receive an alert warning about users on the bike lane. Following the ETSI EN 302 638-3 v1.2.1 standard [17], below is listed the type of cause used within this scenario, being 1 and 2 the codes of sub cause to identify the warning and critical situations, respectively (see Table 2).

Table 2: Alert description for VRU risk detection in the warning/critical areas of the deployment.

Cause code description	Direct cause code	Mapping with TPEG-TEC	Sub cause code	Sub cause description
Human presence on the road.	12	Specified as people on roadway in tec002 of clause 9.2 in TISA TAWG11071 [i.10]	0	Unavailable.
			1 to 3	As specified in tec112 of clause 9.21 in TISA TAWG11071 [i.10].

4.2.1.2 Module Requirements and Virtualization

The V2XCOM module and the MQTT module that form the V2X stack are implemented as separate functions. Further, a distinction can be made between the V2XCOM module that runs in the Pledger infrastructure and is used for the RSUs and the V2XCOM module that is running in the OBUs. The V2XCOM module used in the OBUs differentiates slightly in the functionality, but also the virtualization used: for the OBUs, the module runs on bare metal and is not virtualized, whereas the same module for the RSUs running in the infrastructure is containerized. The MQTT module, much like the V2XCOM module for the RSUs, is containerized.

Table 3: V2X Stack Requirements

Module Name	RAM	vCPUs	Storage (SSD)	Virtualization
V2XCOM (OBU)	4 GB	2	8 GB	Bare metal
V2XCOM (RSU)	4 GB	2	8 GB	Docker Container
MQTT Broker	1 GB	1	1 GB	Docker Container

4.2.1.3 Placement and scaling considerations

The placement and configuration of the V2X stack is done in a day 0 configuration, as it forms part of the infrastructure setup and there is no dynamicity for scaling or placement of the modules (as such ther are no dynamic triggers for scaling/moving the application). However, the location where the V2XCOM modules for the RSUs are placed is flexible and it can affect the performance of the V2X stack. The requirement for an optimal performance is to run the V2XCOM modules as close as possible to their respective IEEE 802.11p radio devices in order to minimize the delay between them. For the case of the Barcelona infrastructure, this means that the V2XCOM needs to run at the edge, which is automatically done during the setup phase, a procedure that lies outside of the actual scope of the UC and is part of the infrastructure configuration.

4.2.1.4 Inter-Module dependencies and integration considerations

As described in Section 4.2.1.1, the V2XCOM module will communicate with upper layers through the MQTT protocol, by using the queue names specified on the same section.

4.2.2 Module: Tram Detection Service

For the detection of the tram, several options have been considered during the design phase of the UC, such as using radars or lasers, vibration sensors installed on the rails at the tram station, or the use of cameras. Eventually, a solution based on image processing techniques will be used, which is provided by Barcelona Super Computing (BSC) [18], who in a collaborative effort with the UC2 partners will provide this functionality.

Live video streams obtained from several cameras on-street that are recording the tram station are fed to an application that is capable of recognizing trams. The image processing goes even a step further

and allows to determine the location of the tram, as it approaches, reaches and leaves the tram station, which is a piece of information necessary to be able to detect possible risks efficiently. The information can be then fed to the RDNS module. The tram recognition service runs on dedicated hardware which is not part of the Pledger infrastructure and exposes the information to UC2 via a dedicated API. As such, neither the computing requirements, nor the type of virtualization matters, as well as any placement and scaling considerations.

4.2.2.1 Module implementation details

As mentioned in Section 4.2.2, a dedicated WebSocket or API REST between the Tram Detection Service and the RDNS module is needed to maintain a data stream with real-time Tram positions. From the external hardware capable of recognizing trams, the RDNS will require – at least – basic information about time, positioning and velocity (TPV) data. The final data models are not yet decided, as the integration with the external tram detection service is still work in progress. Below an example of a TPV message that could be used for feeding RDNS is shown, which includes the key information, such as timestamp, GPS position, speed, etc.:

```
{"class":"TPV","device":"/dev/pts/1",
  "time":"2005-06-08T10:34:48.283Z","ept":0.005,
  "lat":46.498293369,"lon":7.567411672,"alt":1343.127,
  "eph":36.000,"epv":32.321,
  "track":10.3788,"speed":0.091,"climb":-0.085,"mode":3}
```

4.2.2.2 Module Requirements and Virtualization

Since the service runs externally, there are no requirements on the Pledger HW resources and virtualization details are not relevant.

4.2.2.3 Placement and scaling considerations

Since the service is not orchestrated by the Pledger platform, there are no placement or scaling considerations.

4.2.2.4 Inter-Module dependencies and integration considerations

The tram detection service has to be able to send out notifications to the RDNS. For that a WebSocket or REST-based API is used with interfaces on both these modules.

4.2.3 Module: RDNS Application

The main UC2 logic is executed in the RDNS application. Its task is to correlate the information provided by the V2X stack about the location of bicycles and the information from the tram detection module in order to recognize risky situations. Such a situation results whenever a tram is arriving at the station or is currently stationed there, and a bicycle approaches the station. If these conditions are met, the application triggers an alert. This alert can be used to warn both the bicycle and the pedestrians. The bicycles are warned via the same V2X stack that is used to gather location information, whereas for the notification of the pedestrians at or close to the tram station remote calls are to be implemented. At the time of writing, there are still options to be evaluated on how to generate the warning signal: an infrastructure element is provided by the Tram company that can send out an audiovisual signal. This would be an IP-endpoint that has an API to allow the triggering of an alert with which the RDNS application can interact. Another option is that the tram itself is equipped with a prototype of a portable device that could emit such a notification, which could be connected over IEEE 802.11p or another wireless technology. Discussions on this integration are being held with Barcelona Tram.

4.2.3.1 Module implementation details

The elements involved in the solution and their behavior can be summarized as shown in Figure 11:

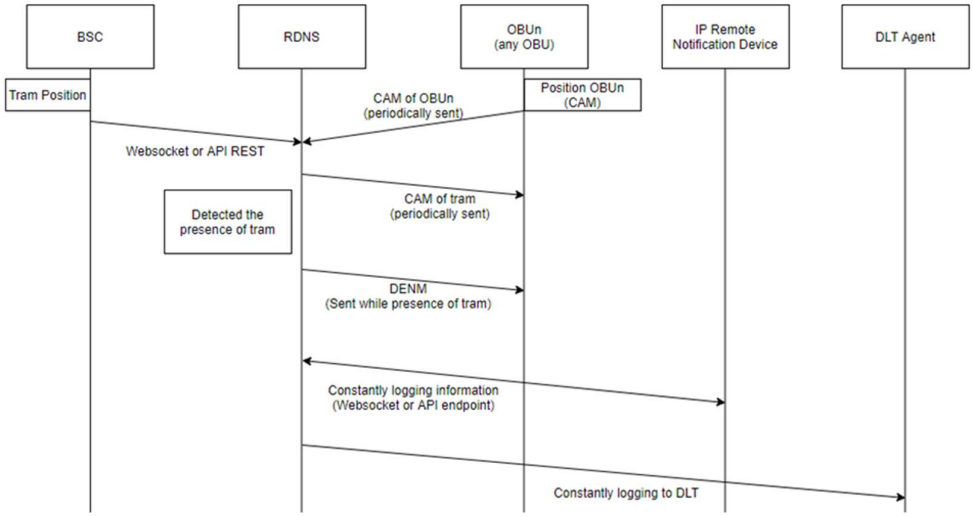

Figure 11. Communication workflow among all modules.

There are two different parts of the application: the one running in the computing resources of the city infrastructure (far-edge, edge or main DC) and the one running on each OBU. Here are the inputs and behavior of each part:

- OBU side:
  - Sends the CAMs revealing the position and parameters of the vehicle constantly.
  - Is listening to any GeoBroadcasted DENM. If a DENM is received - which by the standard will mean that the vehicle is within the awareness area – it will trigger a "warning" or a "critical warning" depending on the message received. A first approach to these two levels of warning depending on the geographical area are shown in Figure 19, where an example of the IEEE 802.11p coverage region received during the first trials is also plotted.
  - The "critical warning" area is within the just "warning" area which means that both types of DENMs are received within the area. To solve this issue the "critical warning" will always have priority.

- MEC side:
  - Incoming data to be considered:
    - The position of the tram (provided by the BSC solution).
    - The CAMs received by the respective OBUs (position of the bikes).
  - Outgoing communications:
    - The DENMs sent (to the OBUs) every time the tram is within the depot area.
    - The CAMs revealing the position of the tram.
    - The API endpoint that allows the "Awareness Module" to fetch the data for the frontend.
    - Logging to DLT.

4.2.3.2 Module Requirements and Virtualization

The RDNS application is fully containerized, so it can be deployed by the Pledger orchestrator on any of the k8s worker nodes. On the VRU, the application runs on a Docker container. These are the computing requirements of the application:

Table 4: Risk Detection and Notification Service Requirements

Module Name	RAM	vCPUs	Storage (SSD)	Virtualization
RDNS Application	4 GB	2	8 GB	K8s deployment

4.2.3.3 Placement and scaling considerations

The RDNS application has full flexibility in terms of placement and can be deployed anywhere in the K8s cluster. However, a placement close to the V2X stack running at the edge is desirable. As such, the preference for the initial deployment of this application module should be the edge. If the availability of resources should at some point drop considerably (CPU and RAM are the targeted metrics), the CPUs usage of the physical nodes should ever exceed a certain limit that could impact the performance of the application, or packet losses are observed on any of the network interfaces, the application module should be placed somewhere else to avoid starvation or even an eviction of the process.

4.2.3.4 Inter-Module dependencies and integration considerations

Since the RDNS module is gathers inputs from different modules and sends out notifications to other endpoints of the UC deployment, it has a series of dependencies and integration considerations.

A first overview of the inter-module integration and how all of them communicates with each other is briefly introduced in Section 0. From the RDNS point of view, three data flows can be defined:

- BSC (external tram detection) – RDNS: described in Section 4.2.2.1, a TPV message is expected from the BSC module by the RDNS to compare all node positions within designated areas.
- RDNS – OBU: XML messages, coded following the ASN.1 XER format, will be exchanged between V2XCOM modules both in RDNS and all OBUs.

  RDNS – IP Remote Notification Device: an API message having information about the Tram, all OBUs and their risk level will be sent to the remote IP node. Formatted in JSON, an example output of this message is attached below:

```
-       {
-       "tram": {
-         "position": {
-           "lat": 12.34,
-           "lon": 12.34
-         }
-       },
-       "obus": [
-         {
-           "station_id": 1,
-           "position": {
-             "lat": 12.34,
-             "lon": 12.34
-           },
-           "risk": "NORISK|WARNING|CRITICAL"
-         },
-         {
-           "station_id": 2,
-           "position": {
-             "lat": 12.34,
```

```
-                "lon": 12.34
-              },
-              "risk": "NORISK|WARNING|CRITICAL"
-            }
-          ]
-        }
```

4.2.4 Module: RAN Controller

As a core component forming part of the Barcelona deployment, yet not the UC application per se, the RAN controller module is responsible for the setup and configuration of the V2X radio infrastructure in UC2. The RAN controller has visibility of all radio nodes connected to the infrastructure and can talk to them for the physical and logical configuration.

An infrastructure that runs the RAN controller (together with the slicing and orchestration engine, see Section 4.2.5), can apply the concept of slicing, allowing to allocate dedicated resources and network connectivity to a project or use case by reserving and isolating a share of the overall set of radio and computing resources that are available. In Barcelona, a dedicated slice for the UC2 application is allocated that includes the radio nodes that provide IEEE 802.11p connectivity, along with the computing resources in which the K8s cluster is deployed and where the application modules can be deployed.

In this subsection, it is briefly explained how the RAN controller operates and in which way it supports vehicular communications based on IEEE 802.11p.

4.2.4.1 Module implementation details

The RAN controller is composed of several building blocks. An OpenDayLight [19] (ODL) instance is required for the management of the data plane of the wireless devices. The radio nodes run an openvswitch [20] instance on which the SDN controller connects to in order to deploy VLAN tunnels to manage and route the data plane traffic. Another building block is the NETCONF-Manager [21] (custom software) which is responsible for the configuration and management of the radio devices and its resources, i.e. the physical configuration and functional settings. This last module talks to a NETCONF agent (Netopeer[3]) that runs on the radio nodes.

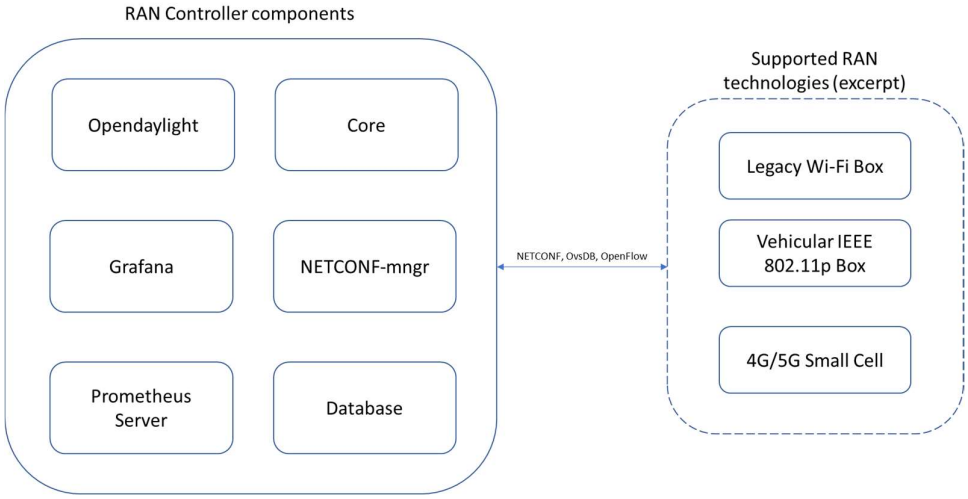

Figure 12: RAN Controller architecture.

[3] https://github.com/CESNET/netopeer2

A part of the work done in WP3 is the addition of vehicular communications to the repertoire of supported radio technologies for the RAN controller, in order to support vehicular use cases like UC2. The IEEE 802.11p standard requires a different usage of the radio transceiver as an RSU, compared to legacy Wi-Fi access points (APs) that operate in a/b/g/n/ac-mode. For the V2X communications used in Pledger, the radio transceivers rely on the so-called *Outside of the context of a basic service set* (OCB) mode. In this mode, the transceiver works on specific channels and bandwidths that are based on the IEEE 802.11p standard, supporting the vehicular communication packet formats. As such, if a radio node of the infrastructure is to be used for vehicular communications, it needs to be configured to operate in OCB mode and marked as such on the RAN controller. All of this is supported by the RAN controller with the additions made in Pledger.

4.2.4.2 Module Requirements and Virtualization

The RAN controller is virtualized and can run in a VM instance. For the use in the Pledger Barcelona infrastructure, a dedicated VM is instantiated in the cloud during the day 0 configuration. This instance is expected to run permanently, as it forms part of the infrastructure setup. The RAN controller has the following requirements:

Table 5: RAN Controller Requirements

Module Name	RAM	vCPUs	Storage (SSD)	Virtualization
RAN Controller	16 GB	8	40 GB	VM

4.2.4.3 Placement and scaling considerations

The RAN controller is a static instance that forms part of the infrastructure modules and cannot be moved or scaled.

4.2.4.4 Inter-Module dependencies and integration considerations

The RAN controller integrates on its northbound with the Slicing and Orchestration engine, the core module that issues the generation of slices and the configuration of the radio devices. For this interaction, a dedicated RESTful API is used. On its Southbound, the RAN controller communicates with the radio devices over the NETCONF protocol and OpenFlow [22]. NETCONF is used for the configuration of devices, whereas OpenFlow is used to configure the data plane connectivity of the traffic coming and going to the radio devices that form part of the slice. Both the northbound and southbound API are internal and not exposed to any external modules.

4.2.5 Module: Slicing and Orchestration Engine

The Slicing and Orchestration Engine (SOE), like the RAN controller, is a core module deployed in the Barcelona infrastructure. Although it is not a component of the UC application, it is briefly introduced here in the context of UC2. The SOE is responsible for the creation and lifecycle management of end-to-end network slices and the orchestration of vertical services. Whenever the creation of a slice is requested, the SOE breaks the request down into tasks and assigns each task to other components of the framework.

In the Barcelona infrastructure, the SOE serves the purpose of creating a static slice where UC2-related applications are deployed. During the slice set-up process, the SOE enables the reservation of some radio and computing resources available in the infrastructure, and the generation of a logical slice on top of these resources. Further, the orchestration capabilities of the SOE enable the deployment of services and the management of use cases and verticals such as UC2 services.

In UC2, the slice created by the SOE is used to deploy a K8s cluster. During the first iteration of the integration with the core Pledger components, the Pledger orchestrator will communicate directly with the K8s master node in order to deploy UC2 applications. In a later iteration of the integration with

Pledger's core components, the Pledger orchestrator will communicate with the SOE through a RESTful API.

4.2.5.1 Module implementation details

The slice manager is capable of managing shared computing, network and radio resources. It manages the creation and life cycle of resource chunks and slices. In addition, it triggers the instantiation of vertical services on slices, including supporting actions such as DNS provisioning. The SOE is formed by several inter-connected modules, as detailed below.

The SOE leverages the use of OpenStack and K8s to set up computing and network chunks. Radio access nodes are managed by the RAN controller. All resource chunks are stored on the slice manager.

OSM MANO is used as the network functions virtualisation orchestrator, which is used to deploy new network slice instances and to instantiate and terminate vertical apps. Network slice instance objects are stored on the multi-tier orchestrator and the slice manager.

The communication among SOE modules is achieved through RESTful API interfaces. The interactions between SOE and other modules deployed in the Barcelona infrastructure is shown in Figure 13.

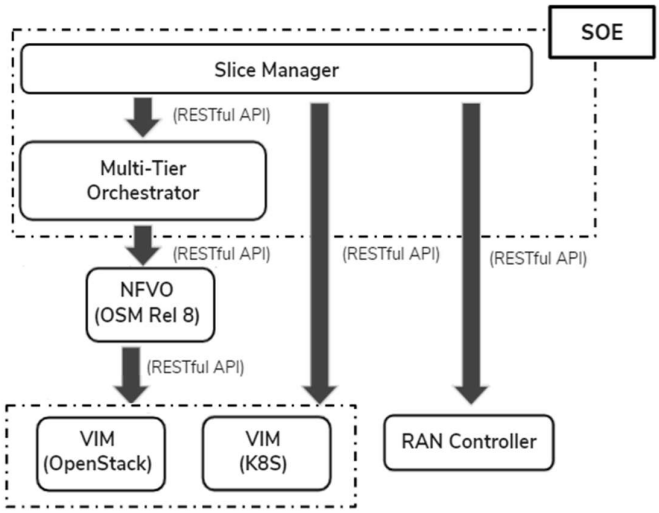

Figure 13: Overview of the SOE and dependencies with other modules.

4.2.5.2 Module Requirements and Virtualization

After the design and implementation phases are complete, the SOE will be able to run, as a containerised application, on the same K8s cluster as OSM MANO.

OpenStack is installed in a bare metal server. OSM MANO is installed in the Kubernetes cluster, and the latest supported release is OSM R8. The SOE instance is expected to run permanently, as it forms part of the infrastructure setup. The SOE has the following minimum requirements:

Table 6: SOE Requirements.

Module Name	RAM	vCPUs	Storage (SSD)	Virtualization
Slice Orchestration Engine	8 GB	4	40 GB	VM

The deployment of an SOE instance can be automated through the use of Terraform and Ansible playbooks. Terraform modules support the management of cloud infrastructure as code, whereas Ansible playbooks are used to automate installation tasks.

4.2.5.3 Placement and scaling considerations

The SOE is a static instance that forms part of the infrastructure modules and cannot be moved or scaled.

4.2.5.4 Inter-Module dependencies and integration considerations

On the Southbound, the SOE uses a RESTful API to communicate with the RAN Controller (Section 4.2.4), and OpenStack whenever a network slice with radio resources is requested.

In order to create network slice instances, the SOE communicates with the multi-tier orchestrator on the southbound through a RESTful API. Then, in order to deploy the network slice instance, the multi-tier orchestrator then communicates with OSM MANO through a southbound RESTful API and, last, OSM MANO communicates with OpenStack and K8s through a southbound RESTful API.

After the implementation phase is complete, the SOE will communicate on the northbound with the Pledger orchestrator through a RESTful API, whereas for now the API is called with other tools, such as Postman or by creating the calls manually. The Pledger orchestrator will be capable of deploying application instances on the UC2 K8s cluster through an API call.

4.2.6 Vehicular Hardware: OBUs and RSUs

In this section, initial prototypes, experimental and final devices are introduced to get a first approach to the hardware involved in the Pledger project and which form part of the UC application.

For both OBUs and RSUs, the PC Engines APU single board [23] is being currently used. Since the initial ALIX model, PC Engines has continued its series of SBCs through the APU and APU2 equipment. Although there are different configurations (oriented towards 3G, LTE or wireless connections), the focus for this UC lies on the APU2.E4 model [24]. This latest model has an integrated AMD G Series GX-412TC x64 processor working at 1GHz with 64k (instruction cache + data) per core and 2 to 4 GB of DDR3-1333Mhz RAM depending on the model.

Unlike the Raspberry PI, it does not incorporate on-board wireless connectivity for Wi-Fi or Bluetooth, but instead it becomes a non-factory-equipped SBC. It is capable of using solid hard drives as main storage unit instead of SD cards, through its mSATA slot which significantly reduce bottlenecks derived from this component. As for its connections, it offers three 1 Gbps Ethernet ports, two external USB 3.0 and two internal USB 2.0, two miniPCI-E slots (one of them equipped with a SIM card slot) and LPC, GPIO and I2C buses. Regarding its consumption, it has a 2.5 mm jack to power it with 12V DC, using 6W to 12W depending on the processor load.

Figure 14. Picture of an APU SBC.

It is a device with higher performance, moderate dimensions (152.4 mm x 152.4 mm), and low-cost compared to e.g. more popular Raspberry PI products, although the technical characteristics it offers perfectly cover the slight increase in the final price. In this sense, by offering an x64 architecture, three available connectivity slots (1x mSSD and 2x mPCI-E) and a processor with sufficient computing power to set the architecture to stress tests, this SBC model can be a strong candidate for the deployment of an

OBU. Not only does it cover the needs that Raspberry PI 4 was incompatible with, but it continues to keep a form factor light enough for deployment on infrastructure sites.

On the RSU side, the current model in use is the Gateworks Newport GW6400 (Figure 15). Equipped with a Cavium OcteonTX Quad Core ARMv8 processor working at 1.5GHz, it has 2 GB of DDR4 memory and 8 GB of Flash eMCC memory. With a four mPCI-E slots and dimensions of 140x100 mm, it has five 1Gbps Ethernet ports, two USB 2.0/.03 and USB OTG, a CAN 2.0 serial port and an RS232 serial port (with optional RS485).

Regarding its energy consumption, it consumes an average of 7W in a normal operating state. This board can be powered by Power over Ethernet (PoE) thanks to its compatibility with IEEE802.3af; alternatively, an 8 to 60V DC power source can be used.

Figure 15: Picture of a Gateworks 6400 Newport SBC.

On the other hand, the Venice board is being actively tested to replace the APU SBC. The Venice GW7200 model from Gateworks [25], the seventh generation of the Venice family aimed at a wide range of applications, both indoors and outdoors. Equipped with an NXP i.MX8M mini Quad Core processor working at 1.6GHz, it has 4 GB of LPDDR4 memory and 8 GB of Flash eMCC memory. With a capacity of two mPCI-E slots, it can support any combination of 802.11ac/b/g/n, 4G, WiMax, 3G CDMA / GSM cards and any other type of connectivity given in this format. In terms of connectivity, it has two 1Gbps Ethernet ports, two USB 2.0/.03 and USB OTG, a CAN 2.0 serial port and an RS232 serial port (with optional RS485). Additionally, it also includes a ZOE-M8 GNSS module.

Regarding its energy consumption, it consumes an average of 6 W in a normal operating state. One of the advantages of this model is that it can be powered by Power over Ethernet (PoE) thanks to its compatibility with IEEE802.3af; alternatively, an 8 to 60V DC power source can be used. Its dimensions (70x100mm) make it also tentative in multi-field deployments, especially if it is used via Power Over Ethernet (POE).

Figure 16: Picture of a Venice GW7200 SBC.

Finally, in an experimental status, the Venice GW 7100 is also being actively tested for future releases, replacing the current OBU hardware. With a NXP i.MX8M processor, a 1 GB of RAM and only one mPCIe, its size (35x100 mm) makes it ideal for VRUs.

Figure 17. Picture of a Venice GW 7100 SBC.

All these SBCs listed above, combined with the WLE200NX wireless card – equipped with an Atheros9K chipset – makes possible to modify the pertinent Kernel and driver files to operate on the 5.9 GHz frequency in OCB mode.

To do so, Figure 18 shows all the modules involved during the process and their context from the Linux wireless architecture:

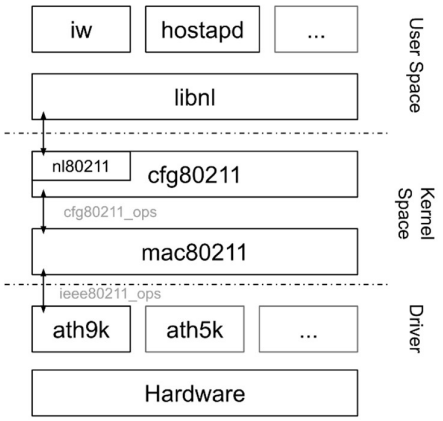

Figure 18. Linux wireless architecture depiction.

A brief introduction to key elements involved is listed below:

- mac80211: Framework needed for the development of Soft-MAC Wireless card drivers. Unlike Full-MAC, first ones can handle the entity where the MAC state machine is stored (also known as IEEE 802.11 Mac Sublayer Management Entity or MLME) via software, while Full-MAC will handle this sub element through the hardware.
- cfg80211: Configuration API for IEEE 802.11 devices, acting as a link between the user space and the drivers. Additionally, it also provides support for regulatory compliance through the use of wireless-regdb and CRDA.
- nl80211: Used to configure cfg80211 through the Netlink socket, enabling user and kernel space communication.
- iw: Command-line interface application based on nl80211, used for configuring wireless devices on Linux-based systems.
- wireless-regdb: Regulatory database used by the Central Regulatory Domain Agent (CRDA), where frequency allocation and acceptable transmission power levels for each country is specified.
- CRDA: Located in the user space, the CDRA is in charge of uploading the wireless regulatory domain into the kernel.

Although specific instructions to modify Kernel and driver files are beyond the scope of this document, interface requirements according to ETSI EN 302 663 v1.2.0 – Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band will be held. For instance, next table shows the ITS Road Safety Applications channel requirements:

Table 7: Details on the channels and radio settings supported in UC2 for communications over IEEE 802.11p.

Channel Type	Frequency Center (MHz)	Frequency Range (MHz)	IEEE Channel Number	Channel Spacing (MHz)	Default Data Rate (Mbit/s)	Tx Power Limit (dBm EIRP)	Tx Power Density Limit (dBm/MHz)
G5-CCH	5900	5895-5905	180	10	6	33	23
G5-SCH1	5880	5875-5885	176	10	6	33	23
G5-SCH2	5890	5885-5895	178	10	12	23	13

For example, to configure an RSU in the G5 Control Channel (G5-CCH) the following commands need to be used:

```
$ ip link set <wirelessInterface> down
$ iw dev <wirelessInterface> interface add ocb0 type
ocb
$ ip link set ocb0 up
$ iw dev ocb0 ocb join 5900 10MHZ
```

4.3 Application Metrics and Statistics

In this UC, metrics and statistics are mainly oriented to capture Tram and VRUs logs, rather than of IEEE 802.11p traces (e.g.: packet loss, power transmission, etc.). In this sense, from the RDNS point of view, application metrics and statistics include:

- Application status: Docker console is configured to prompt current metrics in terms of number of vehicles using the platform, basic information about them (Vehicle ID, positions, and type of vehicle) and the status of the application by itself (service is up and available, a problem has occurred, etc.).
- Vehicle logs: more oriented to capture vehicle's data, it is also possible to dump data about all the connected VRUs and Trams that have been going through the RSU coverage area (and their positioning data) or alert triggering frequency, among others.
- Event logging: the detection of risk situations, possible collisions, VRUs or trams entering the area of the station can all be logged to see if the events are detected correctly. Further, this information can be used to do a post-processing to determine VRU behavior, tram schedule fulfillment, etc.

4.4 Development and Deployment Plan

There is still development to be done for all of the application modules. The V2X module, while operative on a standalone basis and implementing all features, as well as already integrated with the RDNS, still requires to be integrated with the infrastructure core modules (RAN Controller + SOE) to allow for the automated deployment during the day 0 configuration. This, however, does not affect the functionality of the V2X module, as such, the development of the functionality can be considered finished.

The tram detection service module is still to be developed externally and to be integrated with the RDNS module. i2CAT does not participate in its development, but the plan is to integrate with the module before the end of 2021, so that a real detection of a tram can be fed to the RDNS module. The RDNS

module implements all the necessary logic to accept and process the inputs from the V2X stack and the tram module but lacks the integration with the tram module and with the DLT module, a work that will be carried out during the upcoming months of the project.

Regarding the deployment plan, the current development and deployment is all done within the i2CAT premises, where a small indoor lab setup allows the validation of the application software modules. The setup consists of the radio nodes (RSUs) connected over dedicated VLANs with the cloud server, where the infrastructure modules, but also applications can run. This indoor lab environment allows to duplicate the core aspects of the final on-street deployment that is planned to happen between September and end of the year 2021 and to test all the features of the application modules. The goal is to develop and test all modules in the indoor lab environment, and that – once finalized - both hardware and software can be moved to the on-street deployment seamlessly.

The initial testing and validation are especially important for the RSU, since it is a hardware element that will be mounted on the lamp posts, and once mounted, it is costly to unmount in case modifications are necessary. A hybrid deployment that includes both the indoor lab and the on-street deployment is also possible, since both locations are connected, and a dedicated infrastructure slice can be created for Pledger that includes resources from both deployments. This gives a high flexibility for testing purposes and will allow to proceed with the development of the applications even if the hardware is distributed, as the on-street deployment progresses with its own timings. Regarding the development of the hardware, Section 4.2.6 details the state of development and upcoming work to be done. It is currently also being evaluated if far-edge nodes can be used as another type of constrained computing node that can be mounted next to the RSUs on the lamp posts and forms part of the Pledger computing continuum, where small services, such as the V2X stack could run. For this purpose, Odroid-H2+ devices [26] are evaluated.

4.5 Initial UC Application Validation and Testing Outcomes

In order to validate the first prototype of the UC2 application, including the V2X software modules and the prototype RSU and OBU hardware elements, as well as to validate whether the chosen radio technology can deliver the expected performance outdoors where the UC will be deployed, a series of on-street validations were performed.

The setup included an RSU (using the APU SBC) mounted on a lamp post at the Fluvià tram station in the *Avinguda Diagonal* street of Barcelona. The RSU was mounted at a height of around 3m and connected to a laptop over Ethernet, while being powered over a portable power generator. The OBU that connected to this RSU was an APU, as described in Section 4.2.6, powered with a portable battery and mounted on a bicycle.

The first series of tests conducted was focused on determining the range of the radio communications, i.e., in which areas the OBU and RSU would be able to communicate with each other. The outcomes showed that connectivity of up to 300m and more can be achieved, while maintaining a very low packet loss rate of less than 2%, with the errors appearing at the borders of the transmission range. At short and medium distances, a 0% loss rate was measured, even when positioning the bicycle in such a way that a tram and a bus would be stationed between the RSU and OBU, which is the “worst case” scenario in terms of line of sight and with lots of material (metal), but also persons between the two elements. With this communication range and the quality of the communications, i.e., almost lossless, the location, as well as the radio technologies and the V2X stack could be validated and be considered adequate for the UC deployment.

The tests were repeated several times and for the area close to the tram station, a connectivity map was created, as shown in Figure 19: The green area depicts where connectivity was optimal, i.e., no packets were lost, whereas the yellow and orange areas mark proposed “first warning” and “collision imminent/critical” areas that determine when the RDNS triggers notifications.

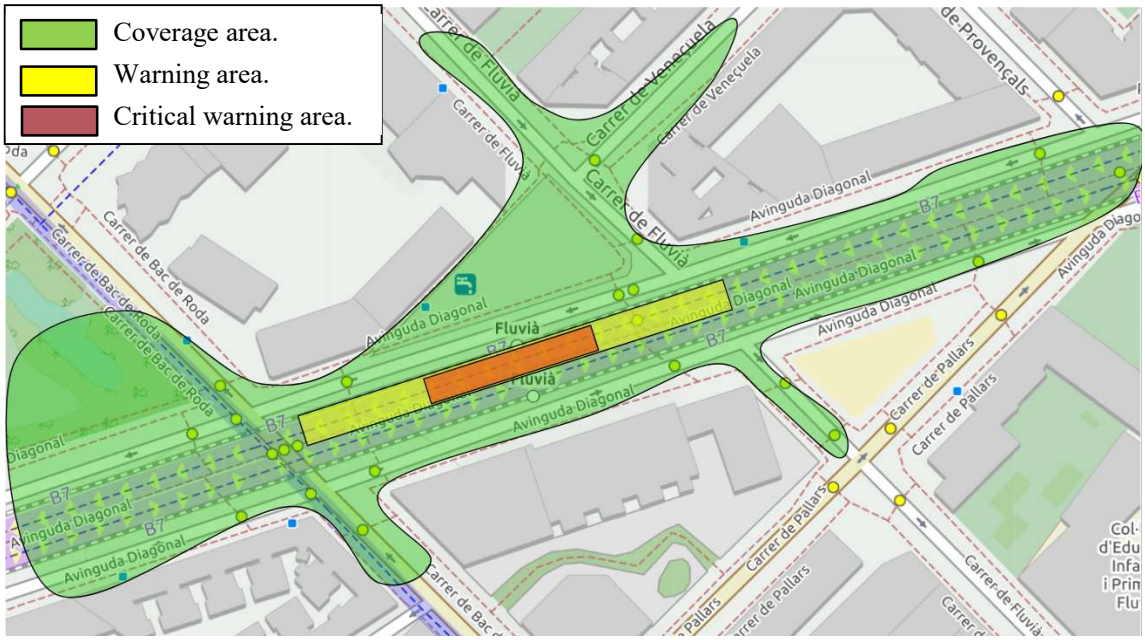

Figure 19. Coverage, warning and critical warning areas around Tram station.

4.6 Demonstration and Trial plan

The first demonstration of the UC2 application will be used to demonstrate the application development status at the intermediate review. The demo consists of 2 parts:

- The first part of the demo shows the status of the prototype development of the RSU and OBU hardware elements, as well as the outcomes of the on-street demo and validations done to showcase IEEE 802.11p and the V2X stack as viable solutions to implement the UC2 scenario. This also validates the physical location as suitable in which the UC will be deployed.
- The second part of the demo shows how the RDNS application reacts to different conditions of a simulated scenario, where GPS positions of OBUs are injected to the system and also a tram arrival is simulated. In this configuration, the V2X stack is fully implemented and operative, while the RDNS application already implements the entire logic that analyzes the available data and detects risky situations, as well as the interface towards the V2X stack and a provisional interface towards the tram detection application.

Both parts of the demo will be recorded and shown at the review of the second year of the project.

In the last year of the project, the trial will take place, where i2CAT and IMI employees will participate in on-street validations of the UC. It will involve the usage of the final infrastructure with all application modules of UC2, as well as real users on bicycles. The final vehicular hardware (OBUs + RSUs) will be used in these tests and validations. Based on what happens during the trial, a video will be recorded that will show the entire workflow of the UC, ranging from the deployment of the application, up to demonstrating key events that are to be showcased during the UC execution (not only the ones specific to the UC application, but also the ones that show the integration with the rest of the architecture).

5 UC3: Manufacturing the data mining on the edge

FILL Cybernetics is the next evolutionary step of digital solutions, adding great value for FILL's customers to their machines. It is an Industrial Internet of Things platform focusing on the transformation of data to smart data and the integration of these smart data in the new smart digital engineering process. Its components are hosted on an edge device to digitize the entire production process. By collecting and recording the data together with analytics services the condition of a machine can be determined, and a complete overview of the production process is given. This enables Cybernetics to increase the profitability and production quality of a factory.

As FILL's edge computing devices face strict customer requirements (use low energy, harsh environments, low costs), FILL is forced to outsource computationally intensive operations to a cloud service. Data transfer and transfer of computational power enables the generation of richer datasets and the computation of more complex machine learning models to improve predictions and analytics. These analysis by the data analysts enable the engineering experts to better evaluate the machine mechanics and to improve the engineering process. Furthermore, they provide valuable insights into the machine for the customers, which enable them to adapt the process adequately.

For this UC, a machine-tool is set-up in the FILL Future Zone, a newly established Center for Research and Development (see Figure 20), where Cybernetics is installed, and analytical services are developed during the project. These applications are described in the following.

Figure 20: Machine tool "SYNCROMILL" set-up in FILL Future Zone.

5.1 UC Application Description

In UC3, applications are developed enabling the analysis of the machine and its underlying processes. This allows the monitoring of the process and the determination of the process stability. Furthermore, advanced algorithms using machine learning can be developed and deployed to facilitate more advanced analytics. In this deliverable, the focus is set to the development of these "Basic Analytics" applications to evaluate the process stability of the machine. In the second and final iteration of this deliverable (due M33), the development details on the "Advanced Analytics" applications will be presented.

The stability of an active process can be determined by two means: the media consumption analysis and the analysis of parts produced. These two methodologies allow to observe and evaluate values over time, giving an estimate about the running process' stability. The measured data is generated in two frequencies (2 Hz and 166 Hz) and pre-processing applications (UDP-To-Broker Manager, HTTP-To-Broker Manager, Programmable Logic Controller (PLC)-To-Broker Manager) are collecting the data from the controllers, Numerical Control (NC) and PLC. These pre-processing applications act as data producers and write Advanced Message Queuing Protocol (AMQP) messages to specified RabbitMQ [27] exchanges (namely *Data.Exchange.fast* for the higher frequency data and *Data.Exchange.slow* for the lower frequency data). From there, the data can be consumed by any analytic application. Further, the results are stored in databases for logging purposes and to enable post-processing of the data. The Cybernetics Backend and Web UI can then access the results and visualize them to the end user. The architectural overview is given in Figure 21 and the cockpit of the Cybernetics UI is shown in Figure 22.

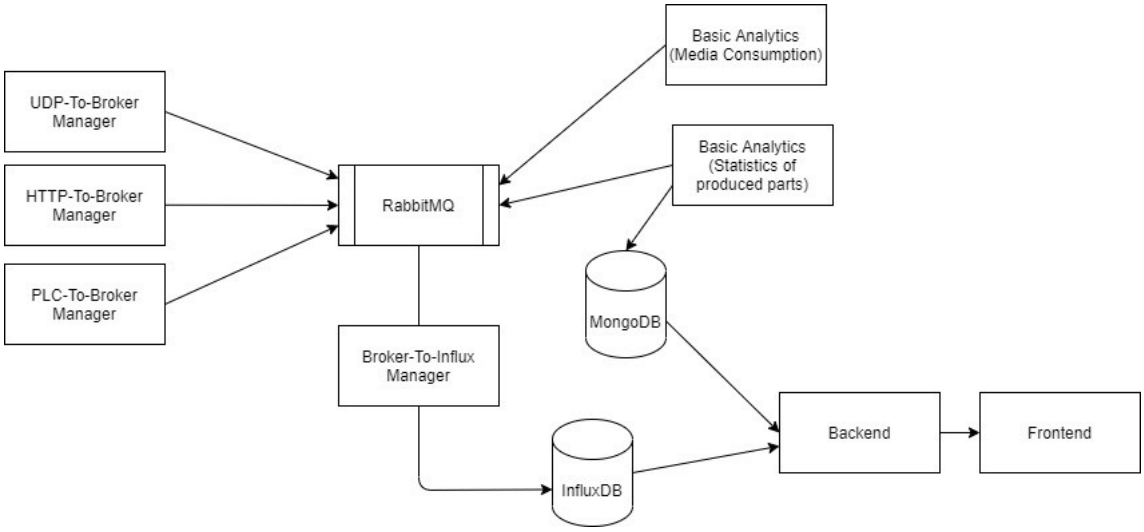

Figure 21: Overview of UC3 architecture.

This microservice architecture was chosen to provide full scalability and flexibility for the different applications, making Cybernetics fully customizable. In the following, three types of these analytical applications which help evaluating the process stability in this UC are described.

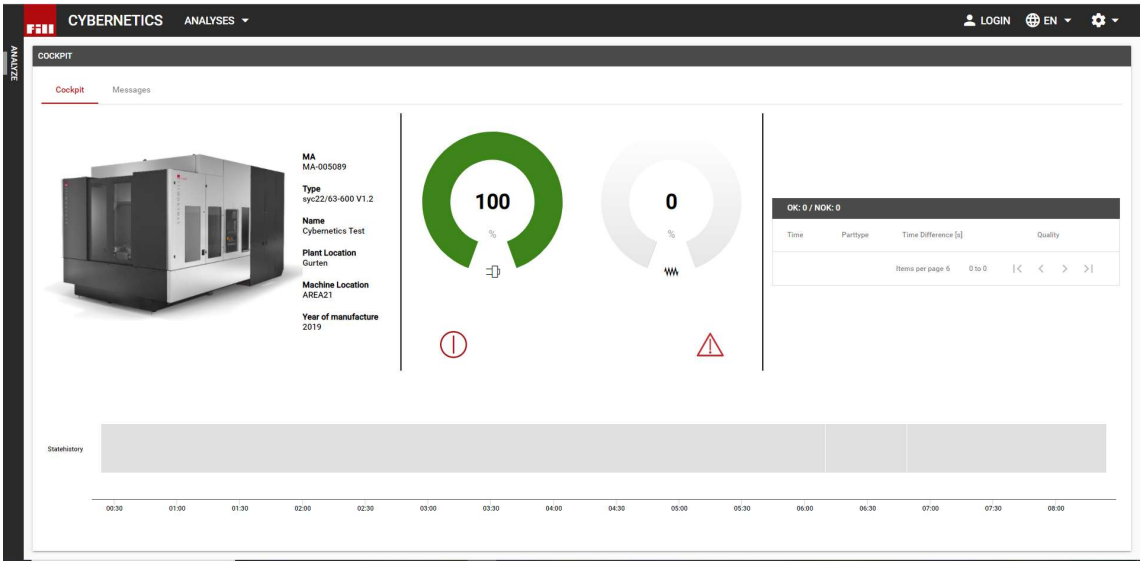

Figure 22: Screenshot of the Cybernetics UI.

5.2 Application Modules and Dependencies

In UC3, three analytical applications are developed by the time of writing this initial iteration of D5.1. The first application analyzes the air consumption of the machine, the second one the energy consumption and the third one gives details about the parts produced by the machine. All data are analyzed batch-wise, and all analytics applications have a three-part structure: The first part of this structure is the "Collect function", responsible for the connection to the message broker (RabbitMQ) and consuming and determining the data required to estimate the boundaries for the batch. Furthermore, the data required for the analysis is collected in this part. The second part performs the actual analytical task. The details about the work performed are given in the corresponding subsections below. The last part of the analytics application consists of a connector to transfer the data to a specified database and acknowledging the messages processed on the message broker. The applications are developed in Python3 using the Python libraries pika [28] and pandas [29]. In the following, details of the applications are given. In Figure 23, the results of this module to estimate the process stability are shown.

5.2.1 Module: Media Consumption (Pneumatic)

This module implements the analysis of the average air consumption, which is one of the interesting properties to determine the process stability of the process. By observing the mean air consumption over time, anomalies can be identified, and the underlying cause can be sought, e.g., leakage or leaky valves, etc.

5.2.1.1 Module implementation details

The application connects to the message broker (RabbitMQ) and consumes the data related to the air flow of the machine. Afterwards the average air consumption is determined, and the result is transferred back to the RabbitMQ. From there on, another consumer ingests the result and transfers it to the InfluxDB.

5.2.1.2 Module Requirements and Virtualization

The module can be virtualized in a Docker container, with low requirements to the computing resources as stated in the following table.

Table 8: Media consumption (Pneumatic) module requirements.

Module Name	RAM	vCPUs	Storage (SSD)	Virtualization
Media Consumption (Air)	50 MB	1	107 MB	Docker

5.2.1.3 Placement and scaling considerations

The application is meant to run on the edge, as lower latency increases the QoS and QoE, so results are available faster than if computed in the cloud taking latency into account. Depending on the utilization of the machine, peaks in the utilization of the computing resources of the edge device may occur, which includes CPU and RAM. These peaks may perdure several minutes. As such, to preserve applications with low-latency requirements as well as applications processing sensitive data, this module can move to the cloud to balance the workload peaks.

5.2.1.4 Inter-Module dependencies and integration considerations

The module Media Consumption (Pneumatic) needs to communicate with RabbitMQ, which is implemented via a Python client (pika). Concerning the integration with the Pledger core system, it needs to be integrated with the Orchestrator to be managed by the Pledger platform. Furthermore, integration with the scaling and placement tools is needed, so the application can be moved to the cloud if necessary. To ensure a smooth execution of the application in the cloud, the access to the required data has to be established. The StreamHandler platform can serve this purpose acting as a bridge between edge and cloud to transfer data to the cloud and to transfer the application's results back to the database on the edge.

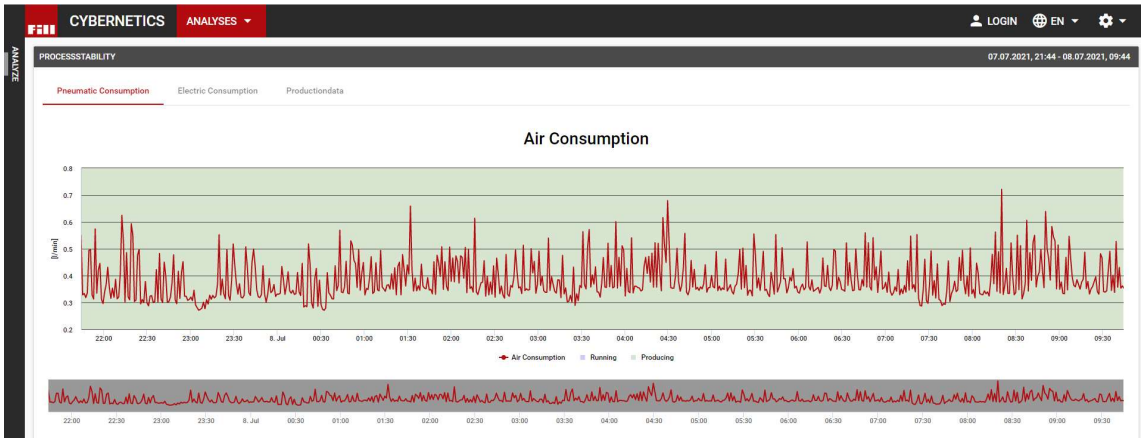

Figure 23: Cybernetics UI showing analytic results of Media Consumption (Pneumatic).

5.2.2 Module: Media Consumption (Electrical)

This module implements the analysis of the average electric consumption of the machine. Together with the average air consumption, the electric consumption has a decisive influence on the stability of the running process. Using the information provided by this module, the machine operator can determine deviations in the process and set corrective actions. In Figure 24, an example showing the results of this module is given.

5.2.2.1 Module implementation details

The implementation of this module is the same as for the module Media Consumption (Air) described in 5.2.1. The only difference is in the data consumed, which for this application is related to energy consumption.

5.2.2.2 Module Requirements and Virtualization

As this application is highly related to Media Consumption (Pneumatic), it has also very similar computational requirements. Furthermore, it is also virtualized in a Docker container. The specific requirements are given in Table 9.

Table 9: Media Consumption (Energy) Requirements.

Module Name	RAM	vCPUs	Storage (SSD)	Virtualization
Media Consumption (Energy)	50 MB	1	107 MB	Docker

5.2.2.3 Placement and scaling considerations

The considerations for placement and scaling are the same as for the Media Consumption (Pneumatic) application. To balance peaks in the workload, it can be executed in the cloud.

5.2.2.4 Inter-Module dependencies and integration considerations

Again, the dependencies and integration considerations overlap with those of the Media Consumption (Pneumatic) in 5.2.1.4.

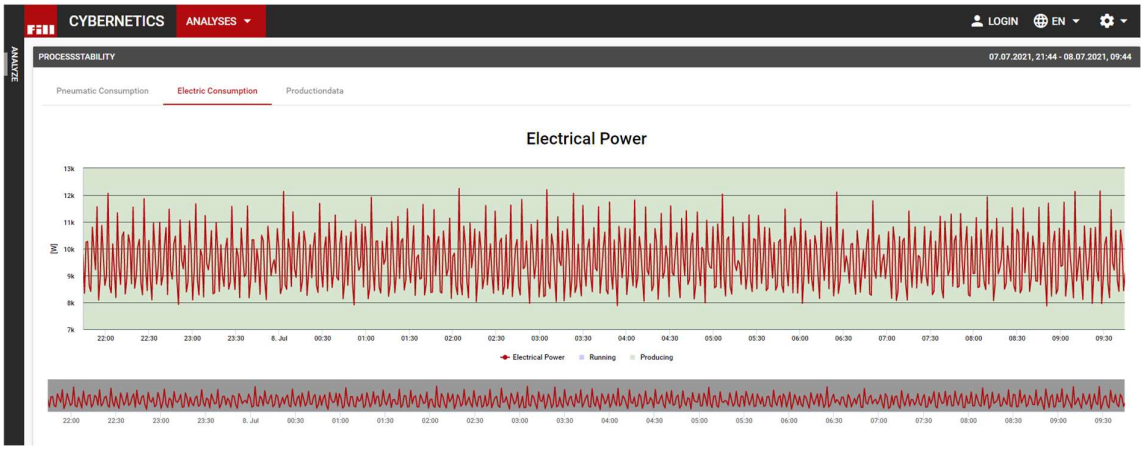

Figure 24: Cybernetics UI showing results of Media Consumption (Electrical).

5.2.3 Module: Analysis of Parts produced

This module implements the analysis of the parts produced by the machine. This includes the evaluation of the cycle time, the different subprocesses (Insertion, Clamping, Milling, Releasing, Unloading), the quality of the part produced, as well as indications of reasons for rejects. This enables the observation of the component process as well as an adequate intervention in case of deviations.

5.2.3.1 Module implementation details

The module is composed of three components: In the first one, data of the slow data stream is analyzed and in the second one, data of the fast data stream is analyzed. Afterwards, the results are merged resulting in one document containing all information for the final result. The structure of the individual components is the same as for the ones described previously. For both, slow data and fast data, information about the subprocess is gathered to determine the borders of the part, starting with inserting the part and ends with unloading. Information about the type and data matrix code is not required in high resolution, therefore the information is gathered on the slow data stream. Usually, several component types are produced on the machine (e.g., different versions of cylinder heads), and these are identified by their designation for the type. For traceability purposes, each component is equipped with a DMC. These values do not change for the duration of the production of a part; therefore, the lower

resolution is sufficient. Information of the subprocesses as well as the estimated quality are gathered on the fast data stream and cycle time, starting and endpoints of the different subprocesses, as well as their durations are evaluated. Furthermore, the quality estimated by the machine and - in case - specified error reasons are determined. In Figure 25, an example for a very stable process according to the cycletimes of the parts is shown.

The results of the individual components are pushed back to the message broker, where another container merges the individual results of slow and fast data into the final result based on the timestamps for insertion. The final result is stored as one document per part in an instance of MongoDB.

5.2.3.2 Module Requirements and Virtualization

The module is virtualized as Docker containers. As the machine tool for this UC can produce 4 parts in parallel, also 4 parts are analyzed in parallel, meaning executing 9 docker containers (4 times 2- one fast and one slow data analysis, plus one merging the results) for this module. The following table shows the computational requirements for one of these containers, where every container has basically the same or very similar requirements:

Table 10: Analysis of Parts Produced Requirements.

Module Name	RAM	vCPUs	Storage (SSD)	Virtualization
Analysis of Parts produced	50 MB	1	N/A	Docker

5.2.3.3 Placement and scaling considerations

The analysis of parts produced by a machine is considered to be sensitive, as it gives details of the running business (workload, number of parts produced, number of rejects, etc.). Therefore, the data necessary, as well as the analysis, has to be kept on edge, leaving no decisions made for the DSS.

5.2.3.4 Inter-Module dependencies and integration considerations

This module depends – like the other modules - on the message broker and the database, which is in this case the MongoDB. Considering the Integration with the Pledger core system, the Orchestrator is needed to instantiate the Module one on the edge. Furthermore, it is considered to store the resulting data on the Blockchain to secure the sensitive information and make it accessible only to authorized parties.

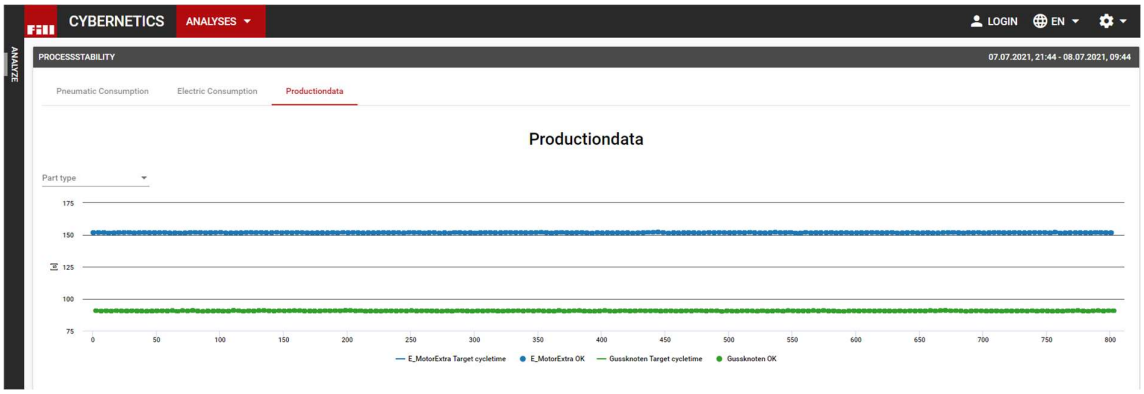

Figure 25: Cybernetics UI showing analytical results of cycle times for parts produced.

5.3 Application Metrics and Statistics

The applications themselves produce many metrics and statistics to specifically monitor the machine. To ensure the smooth running of these applications, the status of the queues at the Message Broker can be monitored (as illustrated in Figure 26), as well as the health status of the Docker containers themselves delivered by the Docker CLI.

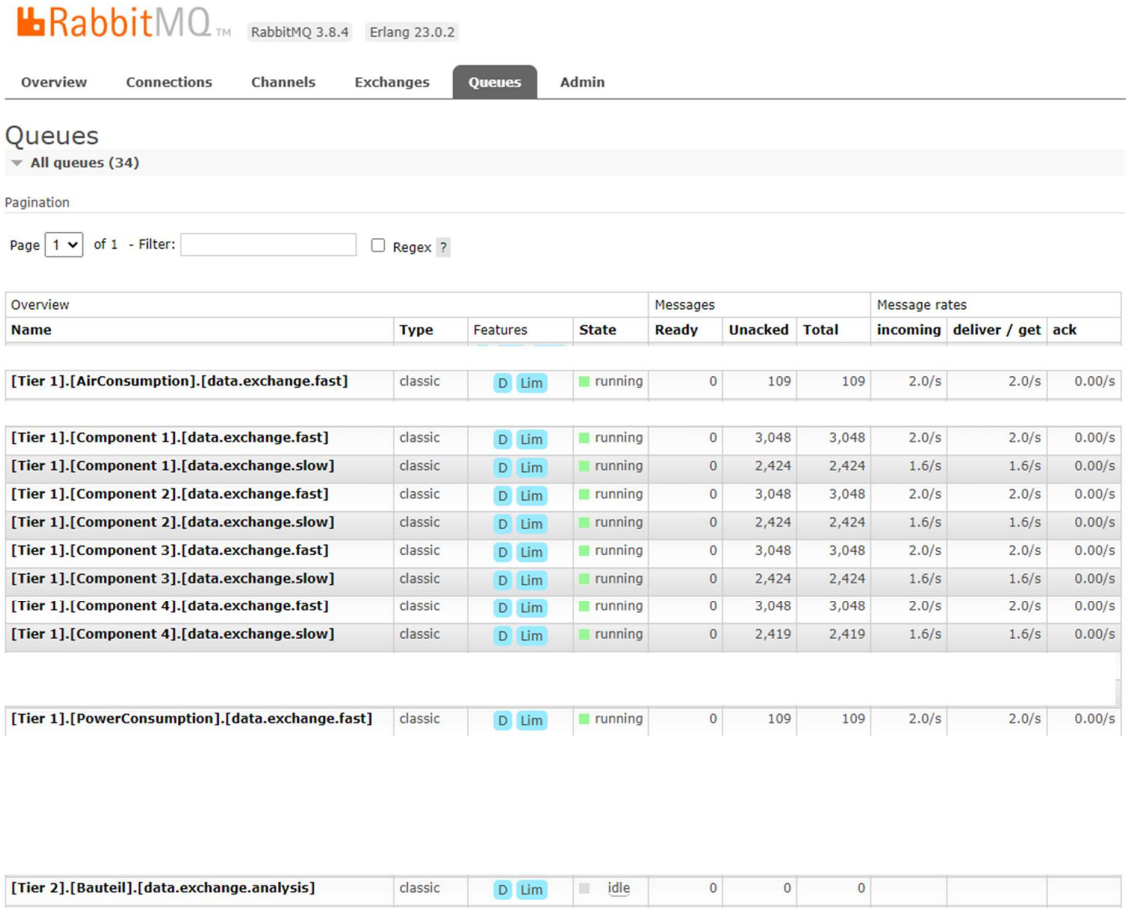

Figure 26: Status of queues on the message broker (RabbitMQ).

5.4 Development and Deployment Plan

By the time of writing this first iteration of deliverable D5.1, three modules to determine the process stability of type “Basic Analytics” have been developed and deployed on the edge device. An extensive testing and validation run were made, testing the system for functionality and robustness. Details for these tests are given in the next section.

For the second iteration the development of “Advanced Analytics” applications are planned. These applications should provide more insights and possibilities to determine the stability of the running process also considering the thermal factors of influence of the machine on the process. The development, testing and deployment of these applications will take place until the end of Task 5.1 (PM 33).

All applications are tested and deployed on the test machine in the FILL Future Zone.

5.5 Initial UC Application Validation and Testing Outcomes

First, the functionality of the different components was tested. For this purpose, a simulated production process was performed without machining any material. Various scenarios were defined to cover as many situations as possible that can occur in the shop floor. The feasibility of the values for air and energy consumption was checked by the domain experts and was found to be in order. A pre-defined part program was designed and ran, and all applications performed as expected delivering the expected results.

A machine in production is subject to many external influences that cannot be influenced by FILL as machine builders and depend heavily on the process over the entire production line. This includes for example unexpected downtimes, shortage of parts and unauthorized entry into the security zone resulting in shutdown, etc. As the data required for the analytics applications is gathered and analyzed batch-wise, these issues have to be considered when testing to prevent failures and abort of the analytics processes. This is especially important for the module "Analysis of parts produced", as the system has to tackle the situation if the process gets stuck during the machining of a component or several subprocesses are replied after error handling. One of these situations might be an unexpected downtime during the machining, where the application started collecting the relevant data, but cannot finish since the part hasn't been unloaded yet and gets stuck due to some failure. The RabbitMQ is capable of caching only a specific number of messages in the queues, afterwards messages get discarded resulting in loss of information.

To prevent this situation, the module was tested under several conditions (machine failure, machine shutdown, manual intervention of the operator, repeated subprocesses, etc.) to evaluate it for its robustness. This was an iterative process, and the module was adapted adequately in case irregularities occurred. The final test was performed with complete satisfaction.

An example is given in Figure 27. A trial was run with a simulated error and the parts were not processed completely, therefore they are marked with a diamond symbol as "Incomplete". Right afterwards two parts with reduced velocity were produced resulting in longer cycle times compared to the others.

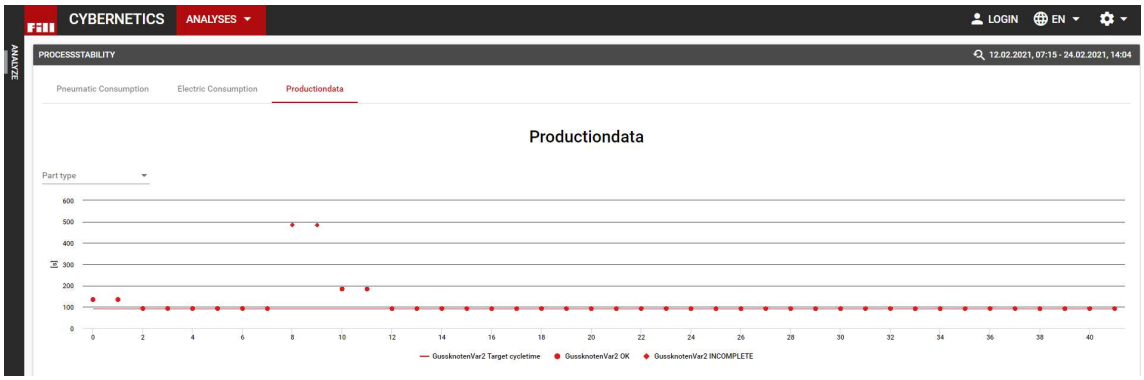

Figure 27: Testing the functionality of the Module "Parts produced".

5.6 Demonstration and Trial plan

As of PM 20, two demonstrations of this UC are planned, the first will be at the intermediate review and the second one for the final trial.

In the intermediate review, the functionality of the applications will be shown, showing the process stability under running conditions with the different results the modules provide. Different parts will be produced with varying cycle times to show the influence on the process stability.

The integration with the Pledger core platform is still work in progress at the point of writing this deliverable. However, the applications need to be capable to communicate with RabbitMQ on edge as well as with the StreamHandler Platform to support the placement considerations described in 5.2.1.3 and 5.2.2.3. In the first demonstration, the containers will be deployed on the edge, as well as in the cloud to show this functionality.

This demonstration will be extended with demonstrating the next developed applications for assessing the process stability in more detail, also taking the thermal behavior of the machine into account.

6 Conclusions

This deliverable reports the status of the application development for the three use cases, an effort executed in Task 5.1. The documentation covers the progress of development up to the point of writing the deliverable in M20. The key concepts relevant to Pledger to the application development, such as the modularization, the virtualization, resources requirements, etc., and a common terminology used to describe these concepts is defined. These concepts have been determined so that all UCs use the same language when describing their application design, in a way that all partners understand it. This is especially important for understanding how the applications can be deployed and what their requirements and dependencies are.

The details provided for each UC application show that their respective design is already finished: The overall UC application in every UC has been broken down into subservices, as planned in WP2. Based on the information provided in this document, the development status and design choices made for each UC application module, both on a functional level, as well as on the level of integration with the rest of the platform become clear. Since every UC has a very different scope on a functional level, and each deployment is unique, the three applications are considerably different from each other when it comes to their design, requirements (both software and hardware), and development plan. The differences also affect how each application relies on Pledger core components, such as the placement and scaling tools, with every application allowing for different submodules to be scaled and placed, each with its own decision criteria and recommendations. Depending on the UC and how much information can be shared publicly about application details, more or less in-depth information could be provided.

In spite of these differences and the considerable complexity of the use case applications, by the time of writing this deliverable, each UC has developed a prototype that allows to showcase the UCs core functionality. These prototype versions have been tested and validated and they serve as the base for the three demos that are presented in the interim review. Beyond the prototype description, each UC also describes the work that is still to be done in each application, as well as future deployment and testing plans.

There are several next steps to be addressed by all UC applications: First of all, each UC application still has ongoing development processes related to the implementation of the functionality. While the prototypes already implemented the core functionality, the full functionality is to be reached within the next months, so that the pilots can be executed. In UC1 and UC2 this will require mainly modifications or additions to the existing modules, whereas in UC3 additional modules will be developed to extend the UC's functionality. Second, the integration with Pledger core components is still in an early stage, and while some initial integration, along with its testing, have already been performed, an important part of the remaining work will focus on finalizing the integration. A task that forms part of this integration process and that needs to be executed by all UC leaders is to refine the SLA metrics and thresholds that will be used to trigger placement and scaling operations. Also, the integration with the DLT is a point that will be covered by all UCs. As a third and last major next step is the pilot phase and execution of the UC applications in the final scenarios, which requires the two previous steps to be finished or very advanced stage. This work will likely incur in optimizations or adaptions of the code of the UC applications to assure the optimal performance.

The final application development status with a report about the final versions of the UC applications and covering the remaining work done will be provided in the second iteration of this deliverable in PM 33.

7 References

[1] PLEDGER, D2.1: PLEDGER's Detailed Use Cases, Ochoa-Aday, Leonardo, Betzler, August. 2020.

[2] PLEDGER. D2.2 - Pledger Requirements Analysis, Iadanza, Francesco. 2020

[3] PLEDGER. D2.3 - Pledger Overall Architecture. Voutyras, Orfefs. 2020

[4] Hetzner home page, https://www.hetzner.com, Retrieved on 27/07/2021.

[5] Amazon Web Services home page, https://aws.amazon.com, Retrieved on 27/07/2021./

[6] Open Source MANO home page, https://osm.etsi.org/ , Retrieved on 27/07/2021.

[7] Docker home page, https://www.docker.com/, Retrieved on 27/07/2021.

[8] Kubernetes home page, https://kubernetes.io/, Retrieved on 27/07/2021.

[9] OpenStack home page, https://www.openstack.org/, Retrieved on 27/07/2021.

[10] Gitlab home page, https://about.gitlab.com/, Retrieved on 27/07/2021.

[11] Jenkins home page, https://www.jenkins.io/, Retrieved on 27/07/2021.

[12] Hololens homepage, https://www.microsoft.com/en-us/hololens, Retrieved on 27/07/2021.

[13] Unity home page, https://unity.com/, Retrieved on 27/07/2021.

[14] Whisper protocol, https://eth.wiki/concepts/whisper/whisper, Retrieved on 27/07/2021.

[15] Vanetza home page, https://www.vanetza.org/, Retrieved on 27/07/2021.

[16] DSRC Committee. "Dedicated short range communications (DSRC) message set dictionary" Soc. Automotive Eng., Warrendale, PA, USA, Tech. Rep. J2735_202007 (2020).

[17] ETSI. "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service." ETSI EN 302 637-3 V1.2.1 (2014).

[18] Barcelona Super Computing (BSC) home page, https://www.bsc.es/, Retrieved on 27/07/2021.

[19] OpenDaylight home page, https://www.opendaylight.org/, Retrieved on 27/07/2021.

[20] Open vSwitch home page, https://www.openvswitch.org/, Retrieved on 27/07/2021.

[21] Enns, R., Ed., et al., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011

[22] Open Networking Foundation (ONF) home page, https://opennetworking.org/, Retrieved on 27/07/2021.

[23] APU single board, https://www.pcengines.ch/apu2.htm , Retrieved on 27/07/2021.

[24] APU2.E4 model, https://www.pcengines.ch/apu2e4.htm, Retrieved on 27/07/2021.

[25] Gateworks home page, https://www.gateworks.com/, Retrieved on 27/07/2021.

[26] Odroid-H2 device, https://www.hardkernel.com/shop/odroid-h2plus/, Retrieved on 27/07/2021.

[27] RabbitMQ home page, https://www.rabbitmq.com/, Retrieved on 27/07/2021.

[28] Pika libraries, https://pika.readthedocs.io/en/stable/index.html, Retrieved on 27/07/2021.

[29] Panda libraries, https://pandas.pydata.org/, Retrieved on 27/07/2021.