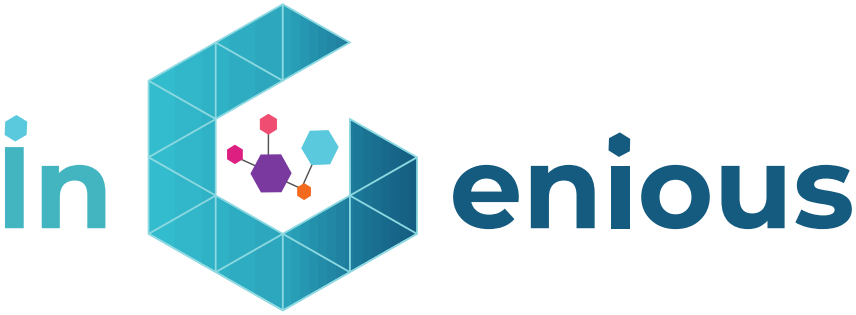
NEXT GENERATION INTERNET

Grant Agreement No.: 957216
Call: H2020-ICT-2018-2020

Topic: ICT-56-2020
Type of action: RIA



D5.2 Baseline iNGENIOUS data management framework

Revision: v1.0

Work package	WP5
Task	T5.1, T5.2, T5.3
Due date	31/03/2022
Submission date	30/03/2022
Deliverable lead	TEI
Version	1.0
Editors	Gino Ciccone (TEI), Marek Bednarczyk (PJATK), Tadeusz Puźniakowski (PJATK), Carlos Alcaide Pastrana (TIOTBD), Anssi lappalainen (AWA); Jussi Poikonen (AWA), Alexandr Tardo (CNIT), Ivo Bizon (TUD), José Luis Cárcel (FV), Christos Politis (SES), Giacomo Bernini (NXW)
Authors	Gino Ciccone (TEI), Giuseppina Carpentieri (TEI), Cosimo Zotti (TEI), Alexandr Tardo (CNIT), Marek Bednarczyk (PJATK), Tadeusz Puźniakowski (PJATK), Paweł Czapiewski (PJATK), Stefan Köpsell (BI), Kumar Sharad (BI), José Luis Cárcel (FV), Joan Meseguer (FV), Ahmad Nimr (TUD), Ivo Bizon (TUD), Pietro Piscione (NXW), Erin Seder (NXW), Giacomo Bernini (NXW), Carlos Alcaide Pastrana (TIOTBD), César Rodríguez Cerro (TIOTBD), Juan Jose Garrido Serrato (SES), Christos Politis (SES), Jussi Poikonen (AWA), Cristina Escribano (NOK), David Gómez-Barquero (UPV), Raúl Lozano (UPV)
Reviewers	Carsten Weinhold (BI), Stefan Köpsell (BI), Alexandr Tardo (CNIT), Nuria Molner (UPV), Gino Ciccone (TEI), Anton Luca Robustelli (TEI), Carlos Alcaide Pastrana (TIOTBD), Laura Gonzalez Estébanez (ASTI), Ivo Bizon (TUD), Giacomo Bernini (NXW), Erin Seder

	(NXW), Marek Bednarczyk (PJATK), Paweł Czapiewski (PJATK)
Abstract	This document describes the approach of iNGENIOUS to develop an interoperability layer, aggregating data coming from different existing and forthcoming IoT technologies. The deliverable describes the state of the current technologies and the planned innovations applied to Internet-of-Things (IoT) data management and applications.
Keywords	Smart IoT GW, IoT, DVL, Cross-DLTs, Smart application, Interoperable Layer

Document Revision History

Version	Date	Description of change	List of contributors(s)
V1.0	31/03/2022	EC Version	See Authors

Disclaimer

This iNGENIOUS D5.2 deliverable is not yet approved nor rejected, neither financially nor content-wise by the European Commission. The approval/rejection decision of work and resources will take place at the Mid-Term Review Meeting planned in June 2022, after the monitoring process involving experts has come to an end.

The information, documentation and figures available in this deliverable are written by the "Next-Generation IoT solutions for the universal supply chain" (iNGENIOUS) project's consortium under EC grant agreement 957216 and do not necessarily reflect the views of the European Commission.

The European Commission is not liable for any use that may be made of the information contained herein.

Copyright Notice

Project co-funded by the European Commission in the H2020 Programme		
Nature of the deliverable:	R*	
Dissemination Level		
PU	Public, fully open, e.g. web	✓
CL	Classified, information as referred to in Commission Decision 2001/844/EC	
CO	Confidential to iNGENIOUS project and Commission Services	

** R: Document, report (excluding the periodic and final reports)*

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

OTHER: Software, technical diagram, etc.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Executive Summary

This deliverable describes the way iNGENIOUS implements an interoperable layer, able to aggregate data coming from different existing and forthcoming IoT technologies, analysed in the first deliverable of the Work Package 5 (D5.1). It gives a detailed technical description of the new solution highlighting the addressed innovations.

Specifically, with the reference to the iNGENIOUS architecture defined in the deliverable D2.2, this document gives a technical overview of the way the Data Management is designed and gives specific implementation details of its first release. It describes how the data virtualization layer (DVL) is designed and developed in iNGENIOUS ensuring that different M2M platforms (Mobius OneM2M, PI System OSIsoft, Symphony and Eclipse OM2M) can be connected, providing the requested information to the dedicated upper cross DLT layer. The cross DLT layer secures and facilitates exchange of information across multiple DLTs solutions (Bitcoin, Ethereum, IBM, Hyperledger, IOTA). Finally, the document describes the use cases and data requirements of the IoT application layer, including AI algorithms for predicting traffic rates and congestion at ports based on heterogeneous data sources along the maritime supply chain, and a platform for remotely operating automated guided vehicles.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Table of Contents

Co-funded by the Horizon 2020 Framework Programme of the European Union

List of Figures

Co-funded by the Horizon 2020 Framework Programme of the European Union

Co-funded by the Horizon 2020 Framework Programme of the European Union

Abbreviations

AF	Asset Framework
AGV	Automatic Guided Vehicle
AI	Artificial Intelligence
AIS	Automatic Identification System
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AR	Augmented Reality
AW	Approval Weight
CORBA	Common Object Request Broker Architecture
DDL	Data Definition Language
DLT	Distributed Ledger Technology
DP	Differential Privacy
DTLS	Datagram Transport Layer Security
DVL	Data Virtualization Layer
E2E	End-to-End
ETH	Ether
FDW	Foreign Data Wrapper
FPC	Fast Probabilistic Consensus
FPE	Format Preserving Encryption
GDPR	European General Data Protection Regulation
GPS	Global Positioning System
GW	Gateway
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HMI	Human machine interface
HWK	Hash Without Key
HTTPS	HyperText Transfer Protocol over Secure Socket Layer
ICT	Information and Communications Technology
IN-CSE	Infrastructure Node Common Service Entity
IoT	Internet of Things
IP	Internet Protocol
IPC	Inter-Process Communication
JDBC	Java DataBase Connectivity
JSON	JavaScript Object Notation
KETI	Korea Electronics Technology Institute
LAN	Local Area Network
LiDAR	Light Detection And Raging
LTE	Long Term Evolution
M2M	Machine-to-Machine
ML	Machine Learning
MN-CSE	Middleware Node Common Service Entity
MPC	Multi-Party Computation
MQTT	Message Queuing Telemetry Transport
NoSQL	Non-Structured Query language
NB / NB-IOT	Narrow Band / Narrow Band - IOT
NFV	Network Function Virtualization
OCEAN	Open allianCE for IoT stANdards
ODBC	Open DataBase Connectivity
OPC	Open Platform Communications

Co-funded by the Horizon 2020 Framework Programme of the European Union

OPC-UA	Open Platform Communications United Architecture
OSI	Open Systems Interconnection
PoS	Proof of Stake
PoW	Proof of Work
REST	Representational State Transfer
RPC	Remote Procedure Call
SCADA	Supervisory Control and Data Acquisition
SLA	Service Level Agreement
SQL	Structured Query language
VDB	Virtual Database
VPN	Virtual Private Network
VR	Virtual Reality

Co-funded by the Horizon 2020 Framework Programme of the European Union

1 Introduction

1.1 Objective of the Document

This deliverable (as part of the milestone MS18) describes how iNGENIOUS Data Management Platform is designed, providing and describing technical details of its first release (baseline of the framework). The proposed solution consists of different layers (e.g., M2M layer, Data Virtualization layer, cross-DLT layer and Smart Applications layer) that are described according to the roadmap of the project for the demonstration and validation of its use cases.

The main outcome of the deliverable will be used as a baseline for the implementation of the final version of the proposed platform for data management (as part of the WP5 activities) as well as for the intermediate demonstration of iNGENIOUS innovations as foreseen by the activities of the T6.3 – Trials and Validation.

1.2 Structure of the Document

This deliverable is organized in four main sections, briefly described below:

- *Section 2 - IoT Application layer:* outlines the IoT applications implemented in the project use cases, focusing mainly on port operations. The main application areas are AI algorithms for predicting traffic rates and congestion at the port based on heterogeneous data sources along the maritime supply chain, and a platform for remotely operating automated guided vehicles in the port area using a variety of sensors and actuators in the vehicle and remote operation station.
- *Section 3 - Smart IoT Gateway*: introduces the Smart IoT Gateway (GW), a physical element with multiple interfaces that allows end-to-end M2M communication. Its main responsibility is to ensure correct and secure routing of messages between the sensors or any other deployed IoT devices, and M2M interfaces to the interoperable layer.
- *Section 4 - Data Virtualization Layer*: describes the deployment of the Data Virtualization Layer (DVL), its integration with external data sources (OneM2M platform, Tuscan Port Community System and Pseudonymization Function) and data consumers (Cross-DLT layer and AI-driven Smart Port and Shipping Platform). It is also described how to represent maritime events by implementing a set of views and/or procedures (GateIn, GateOut, Vessel Arrival and Vessel Departure). Finally, data protection is addressed by the implementation of a function for data anonymization (from the IoT solution providers).
- *Section 5* - Distributed Ledger Technologies: This section introduces the Cross-DLT layer, a software component connected to multiple Distributed Ledger Technologies (Bitcoin, Ethereum, HyperLedger Fabric and IOTA). The purpose of this layer is to record short evidence of key information coming from the DVL and to verify against any DLT network, that is part of the Cross-DLT integration, the truthfulness of recorded data. Furthermore, technical details related to a common API implementation are also described.

Co-funded by the Horizon 2020 Framework Programme of the European Union

2 IoT Application Layer

The IoT application layer focuses mainly on applications related to port operations. These include AI algorithms for predicting traffic rates and congestion at the port based on heterogeneous data sources along the maritime supply chain, and a platform for remotely operating automated guided vehicles in the port area using a variety of sensors and actuators in the vehicle and remote operation station. In the following sections we outline the functionalities and main data requirements of these applications.

2.1 AI algorithms

AI algorithms developed in Use Case 5 focus on modeling and predicting the effects of vessel port calls and related cargo operations on truck traffic rates and congestion. These are developed based on data collected from the ports of Valencia and Livorno. The developed AI system consists of multiple component models predicting different subsets of the related processes. The main targets for model development include the following:

- Vessel arrival time prediction
- Operations prediction, including e.g.
  - Operation durations (discharge, loading)
  - Cargo exchange volumes
  - Cargo exchange modalities (e.g. which containers exit the port by truck)
- Simulating and predicting cargo and hinterland traffic rates
- Simulating and predicting truck turnaround times at the port

These require multiple historical data sets for model development and Machine Learning (ML) model training, and sources of current information for online prediction service deployment. In the following subsections we outline the related historical data requirements and needs for data integrations for up-to-date data.

2.1.1 Data requirements for model development

Sufficient historical reference data sets are needed for development and testing of algorithms and ML models used for predicting future events. In the target AI system, stochastic and machine learning -based models are applied, which generally require large datasets to allow optimizing the accuracy of their output. Furthermore, distinct datasets describing events in different stages of cargo flow through the port need to contain sufficient identifiers to allow data integration (e.g. it is necessary to link vessels to port calls, container operations to port calls, and truck events to container operations). Sensitive identifiers such as truck license plate numbers can be obfuscated, but the available identifiers should still enable consistent tracking of the resources on a general level. Identified data sets for model development include the following:

- Vessel tracking data collected from the global Automatic Identification System (AIS)

Co-funded by the Horizon 2020 Framework Programme of the European Union

- Vessel port call history, including vessel particulars and cargo exchange information
- Operations data, e.g., container discharge and load events
- Truck and container entry and exit events
- Other related information such as weather measurements

2.1.2 Data requirements for service deployment

Generally, data requirements for operating the developed models with current data may differ from the data needs during model training. In the ideal case, less data is needed to perform predictions, and potentially sensitive data such as specific container or truck tracking information is not needed. Precise service data requirements depend on the outcomes of the model development, but at least the following current data is expected to be needed:

- Live global AIS positional and metadata
- Current planned port calls, including vessel details, planned arrival and departure times, berthing locations

Additional data inputs which may be useful for improving prediction accuracy or monitoring the predictions results include:

- Cargo exchange information per port call (e.g. numbers of containers to be discharged or loaded, distribution of container volumes by hinterland carrier type)
- Live gate events (truck and container entries and exits)
- Weather sensor data.

2.2 Industrial & Tactile IoT applications

As wireless communication infrastructure becomes widely available in industrial sites, the sensors and actuators within factory plants can be regarded as available manufacturing resources that can be programmed to produce specific products according to particular specifications. After the production of a determined number of pieces or after identification of possible product improvements, the resources should be easily rearranged to continue the production with the new specific requirements. In contrast, the current production lines are built to perform repetitive tasks without flexibility. For realizing such flexible production technique, a software abstraction from the actual physical devices has to be designed. This conceptual abstraction is defined as industrial & tactile application programming interface (API). In short words, the industrial & tactile API consists of a set of functions that enable the application developer to get data in and out of the system in a unified framework. Within this context, first a set of common functionalities have been identified has an essential part of the API that enables the user defined applications to exchange data easily and securely. The industrial & tactile API has to provide different levels of abstraction. Three levels can be identified as (*i*) end user application development API, which gives the end user simple and easily comprehensible graphical interface for instantiating new applications, and presents data in a format that is understandable by the end user; (*ii*) mid-level function library, which contains functions that do not

Co-funded by the Horizon 2020 Framework Programme of the European Union

need to be directly used by the end user, such as an object detection algorithm; and (*iii*) low-level API that contains functions for data packets formatting and specification of communication link parameters given the requirements given by the end user.

The following subsections describes an exemplary set of functionalists for an industrial application within the context of industry 4.0.

In AGVs UC an E2E platform is developed for remotely controlling automated guided vehicles (AGVs) in the port area. The primary motivation for enabling remote operation is to improve the driver's safety by avoiding possible hazardous situations related to operating in industrial areas. This is achieved by designing a complete IoT system which enables the vehicle operator to have continuous situational awareness of the vehicle status and surrounding environment and enables real-time communication of necessary control signals to operate the AGV safely.

2.2.1 Application example: Factory inspection

Within Factory UC, factory inspection is defined as an application where an autonomous guided vehicle (AGV) travels along a predefined track with camera and sensors integrated. The video and environmental information collected by the AGV are sent to a remote user that monitors the factory site. The quality of the video can be specified at the beginning of the application by the user. The graphical user interface from such application is illustrated in Figure 1. This example will be illustrated within the iNGENIOUS Factory UC.

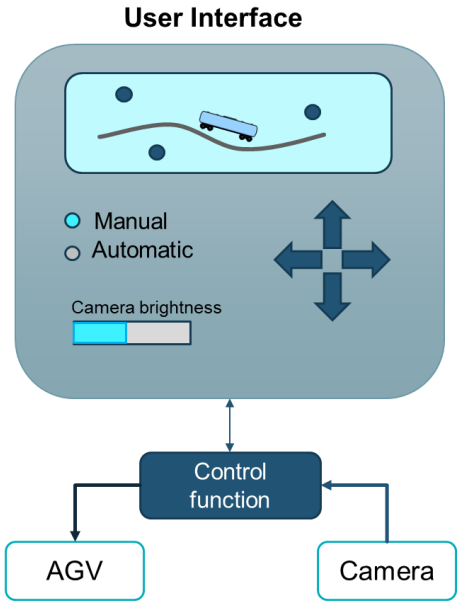


FIGURE 1: EXAMPLARY END USER GUI OF FACTORY INSPECTION APPLICATION

The identified functionalities that have to be available from the industrial & tactile API are:

- Start, stop and adjust the AGV's speed
- Transfer the measurements from the AGV to end user
- Capture current image frame and store in user's data base
- Transfer AGV's position to end user

Co-funded by the Horizon 2020 Framework Programme of the European Union

- Translation of MANO resource allocation to PHY parameters

The identified connection types of the devices are:

- AGV: UDP frames/JSON
- Camera: UDP frames

2.2.2 Data sources and related system components

The IoT system consist of IoT sensors, IoT actuators, IoT signals and an IoT interface. The components related to these subsystems are as follows:

- IoT sensors:
  - AGV:
    - Depth camera: provides a depth image to detect near obstacles
    - LiDAR: provides a scan of the vehicle surroundings
    - IMU: provides velocity and orientation of the vehicle
    - Cameras: provides a 360 real time video of the vehicle surroundings
  - Remote cockpit:
    - Wheel, pedal and gear: capture the driving movements performed by the operator
    - Glove: captures the hand movements made by the operator
- IoT actuators:
  - AGV:
    - Motor drivers: actuators to move the vehicle's wheels
  - Remote cockpit:
    - VR glasses: where the VR app is displayed
    - Glove: transmits tactile information (vibrations) to the operator
- IoT signals:
  - Obstacle avoidance signal: from LiDAR and Depth camera to VR glasses application and glove
  - Driving commands signal: from wheel, pedals and gear to AGV drivers
  - Video retransmission signal: from cameras to VR glasses
- IoT interface:

VR application: an immersive virtual reality application is designed in order to recreate an ordinary driving view. IoT real time parameters are displayed on the dashboard to keep the operator informed about velocity, latency, battery, etc.

In summary, the operation of the tactile internet requires defining interfaces for communication between devices, access to the network, interaction with the computation engines, and implementation service management. Therefore, the network techniques should be flexible, since the goal is to provide a similar experience for the application developer as programing on computer by abstracting the hardware and network functionalities.

Co-funded by the Horizon 2020 Framework Programme of the European Union

3 Smart IoT Gateway

In this section, we present the development activities in order to create a workflow allowing to receive (sensor) data from different sources via different means of communication and transforms, interprets and routes this data in a unified way via different routes to a central cloud M2M infrastructure. The main challenge to overcome was how to implement a system that handles network traffic via multiple routes, that can be manipulated on application layer, while still keeping it transparent to the data flow of the system and meeting the requirements of security and confidentiality. This has been achieved by using an oneM2M compliant system in combination with encrypted VPN connections and layered HTTP reverse proxies. oneM2M defines a standardized, HTTP(S) based communication framework, which is routed through configured VPN connections to reach the cloud M2M server. The VPN connections add layers of security and confidentiality on top of the oneM2M communication and abstract from complex network topology between the Smart IoT Gateway and the cloud infrastructure. Having reverse proxies at each end of the VPN connections gives the possibility to select the desired route, based on decisions taken by other components of the system (i.e., the rule engine).

3.1 Sensor data retrieval and routing to higher levels

The proliferation of small, interconnected devices has caused the appearance of multiple new technologies related to the IoT world. For this reason, it makes sense to introduce a new device to provide interoperability between different groupings of IoT devices (i.e., in the form of independent mesh networks) and logical higher-level systems. The Smart IoT Gateway (GW) is a physical element with multiple interfaces that allows end-to-end M2M communication, having as main responsibility to ensure correct and secure routing of messages from the sensors or any other deployed IoT devices and M2M interfaces to interoperable layers. To achieve this goal and other sub-goals that are not obvious at first glance, the following requirements need to be met:

- *Communication integrity*: to ensure no data is lost or modified during the message transmission or reception.
- *Message translation*: to ensure compatibility between different formats and protocols.
- *Secure link*: to ensure a confidential connection to avoid eavesdropping on messages.
- *Resilience*: to mitigate connectivity outages or any other non-nominal behaviors.
- *Flexibility*: to allow the interconnection of multiple physical interfaces.

To achieve these requirements, the Smart IoT GW has been divided into functional blocks with an implementation that isolates them as independent containers, following a philosophy similar to micro-services architecture. The blocks are presented in Figure 2.

Co-funded by the Horizon 2020 Framework Programme of the European Union

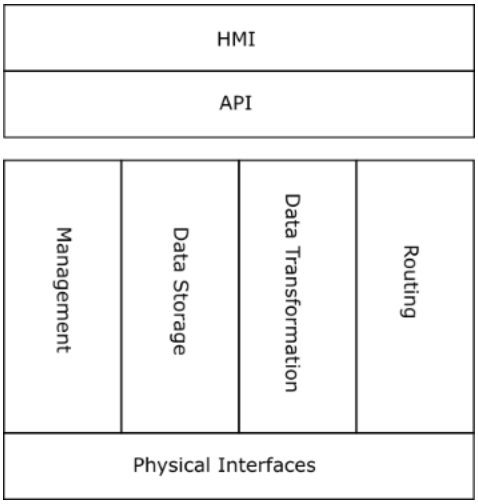


FIGURE 2: FUNCTIONAL BLOCKS OF SMART IOT GW

The following services are available in the Smart IoT GW:

- *Management*: management of the GW includes the current status of the device, internal configuration, local logs and firmware update. It will also include some limited control over the subsystems.
- *Data storage*: databases (time series and NoSQL). All data coming from the sensor space is time stamped and stored in a time series database that has the capacity of performing some operations such as data aggregation, statistical calculations, and time consolidation (i.e., decimate values and down-sample the data).
- *Data transformation*: message translation between formats and protocols. This does not include just direct or indirect mapping of fields, but also this service will encapsulate related messages coming from independent sources and perform compression when possible.
- *Routing*: incoming message routing through the gateway to a different interface, following predefined rules, respecting priorities and in a secure way.

An Application Programming Interface (API) is available for external interoperability and automation. A user-friendly Human-Machine Interface (HMI) is provided for local data monitoring.

In this document, this section will only focus on the description of the Smart IoT GW data transformation and routing, and how Smart IoT Gateway will interface with the M2M space.

3.1.1 Smart IoT GW Interfaces – M2M Space Interfaces

This section describes how the Smart IoT Gateway will interface with the M2M space, where the M2M space is the set of interfaces with all OM2M servers. The interfaces of the Smart IoT GW with the sensors have been described in D4.2 and they are not part of this document.

The Smart IoT GW interfaces with the M2M space through a local MN-CSE instance (Middleware Node Common Service Entity) implemented as Eclipse OM2M [1] in compliance with the oneM2M standard. The software instance is running on the gateway itself and – depending on rules and routes – the sensor data is pushed by the gateway's routing engine towards the local CSE.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Following the oneM2M standard, the MN-CSE local to the gateway communicates the retrieved data with the central IN-CSE (Infrastructure Node Common Service Entity), which is hosted in the SES Cloud space. This communication is performed over the respective route (Satellite, Terrestrial, Local) previously selected by the gateway's routing engine. To ensure confidentiality and integrity of the transported data, all communication between the MN-CSE in the gateway and the IN-CSE in the cloud space relies on encrypted HTTP traffic (HTTPS) and additionally, is routed through a secure VPN tunnel, allowing only external instances with a valid VPN client configuration to connect to the M2M cloud space.

In the figure below it is shown an example use case where each ship hosts a MN-CSE inside its dedicated smart IoT gateway, to which the sensor devices (here: arduinos) connect as application entities (ae). On the opposite site, the CSEs also connect to the cloud based IN_CSE to consume the published data and perform actions accordingly.

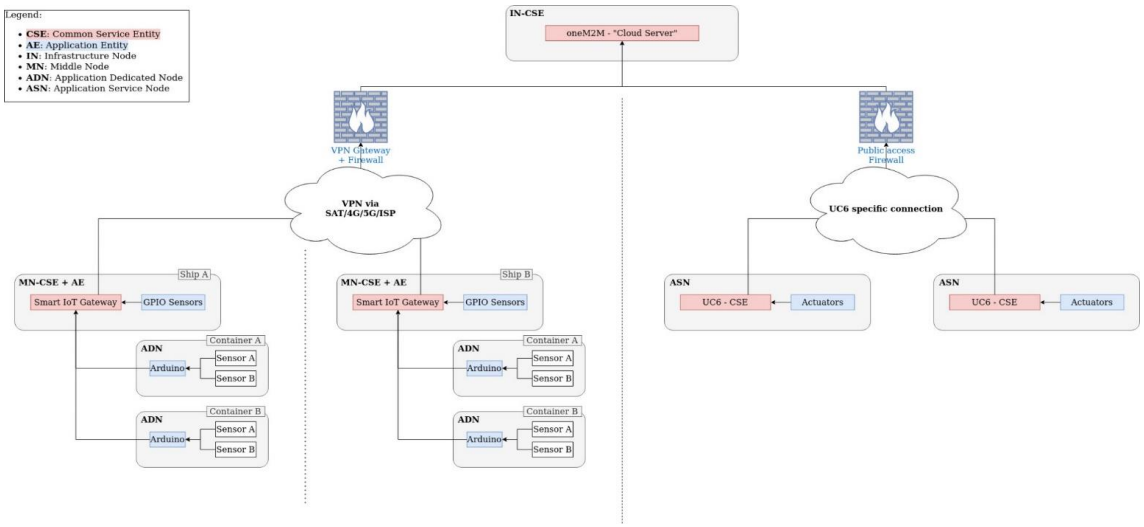


FIGURE 3: THE ABSTRACT OM2M LAYOUT.

External M2M applications which need to request data from the M2M cloud space, connect to the IN-CSE through a dedicated public IP, with a firewall managing access to specified IP sources only. The above diagram depicts the abstract layout from an OM2M perspective, where actuators as part of an external use case (here: UC6) perform actions, based on data from the M2M cloud space, published by the Smart IoT Gateway(s).

To complement the oneM2M compliant interface, which only provides a predefined number of most recent values per sensor, the sensor data will also be stored inside a separate timeseries database in the cloud infrastructure. A separate, generic RESTful HTTP API will be available, allowing to fetch the full history of sensor data on demand. This REST interface will be provided at a later stage of the project.

Since all data extracted from sensor messages also stored in a local timeseries database of each Smart IoT Gateway, it should be avoided to keep additional copies of that data, in order to reduce the amount of required physical storage capacity of each Smart IoT Gateway instance. For this reason, all data published to the M2M space is immediately pushed towards the M2M cloud IN-CSE and not stored on the local MN-CSE instance, making the IN-CSE act as a central collection point for all most recent sensor data in the system.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Additionally, this approach avoids extensive traffic load, especially regarding the satellite link, from the cloud towards the Smart IoT Gateway, which could be generated by external entities requesting data through the cloud-space directly from the gateway. This is considered an acceptable limitation of the system, since the functional disadvantages caused by this approach are outweighed by reduced networking complexity and the advantages mentioned previously. The main functional drawback being that real-time sensor data cannot be requested on-demand, which can be neglected, since the sensors are expected to publish their data only periodically in a defined time interval.

3.1.2 Smart IoT GW Data Routing

This section gives a brief description of the parameters and rules identified for the *Smart IoT Gateway* to perform adequate routing of sensor messages to the M2M cloud infrastructure.

These rules only provide a baseline and shall be used to further define the actual final routing rules to be implemented in future iterations of the project.

Route parameters

The following four parameters have been selected to determine the quality rating of the routes:

- Latency: a simple ping measurement in milliseconds to determine how long it will take for a message to reach the cloud target.
- Loss rate: using packet loss as a stability measurement in percent to determine the likelihood of a message being lost on the way to the cloud target.
- Cost: an arbitrary number {1:10}, identifying how “expensive” each message will be when sent over the respective route.
- Availability – A logical-value (0 = DOWN, 1 = UP), identifying, whether a route is available or not

Route types

The following three types are used to categorize the routes available to the gateway:

- LAN: local area network connections. This type of route is mostly very cost efficient and has low latency and loss rates
- Terrestrial: 4G/LTE/5G connections. This type of route is considered slightly more costly than LAN and has slightly higher latency
- Satellite: internet connection via a satellite terminal, using geostationary satellites, which is constantly available and should only be used when none of the other route types are available. This type of route has the highest cost as well as the highest latency

The loss rate of a route is mostly important in the transfer zone between terrestrial and satellite connections, as there the stability of the terrestrial connection gradually decreases with increasing distance from the shoreline.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Message priority and parameter weight

As not all information sent through the gateway is equally important, each message shall be categorized with one of three priority groups. For each of these priority groups, the aforementioned parameters shall receive different weights in the final calculation of the ratings of the available routes.

- Low priority: less important, repetitive messages, e.g., sensor readings, which constantly send values in the expected "nominal" range for the sensor.
  - For these messages, the cost of a route should have the highest weight and loss rate as well as latency, shall be less weighted, as a message loss or a delay is not critical to the system.
- Medium priority: messages, which transmit data outside the "nominal" range, e.g., temporary fluctuations in temperature or short outages in GPS signal
  - Here, the loss rate is more important than with low priority messages. Cost is still quite high weighted, as there might only be a temporary issue
- High priority: very important messages, communicating critical states of the monitored container, e.g., an opened door or a fire within the container
  - For high priority messages, cost is the least weighted parameter and loss rate the highest weighted one, as the system shall ensure, the message does arrive in the cloud server

The message priority shall be provided by the sensor itself and is to be extracted from the message payload by a numeric identifier (e.g., 0=low, 1=medium, 2=high).

Calculation of the route rating

Each of the configured routes is periodically monitored and the gathered information is used to update the current rating of each route for each of the three message types. This allows the gateway in the event of an arriving message to simply compare the route ratings for the identified message type and select the best rated route as the target route.

The route rating matrix is calculated, according to the following formula:

$$R(r,m) = A_r * \sum_{p \in \{1..10\}} \left(\frac{1}{V_{p,r} * SF_p} * W_{p,m} \right), where\ A_r \in \{0,1\}, V_{p,r} \in \mathbb{R}^+, SF_p \in \mathbb{R}^+, W_{p,m}$$

Explanation of the above formular, r identifying the route type and m identifying the message type

- A_r is the availability of the route r
- $V_{p,r}$ is the current value of the parameter$p$ on the route r
- SF_p is the scaling factor for the parameter p
- $W_{p,m}$ is the weight of the parameter p for the message type m

The scaling factor shall be used to equalize the effective value ranges of each individual route parameter. Latency is expected to range from 50ms up to 800ms, whereas loss rate will range from 0% to 100% and cost will be an arbitrary number from 1 to 10. Thus, reasonable scaling factors could be 0.1 for latency, 1 for loss rate and 10 for cost. Under this assumption, the weight can

Co-funded by the Horizon 2020 Framework Programme of the European Union

be used as the single factor that determines the importance of a parameter on a given route. The current value is defined as $V_{p,r} \in R^+$, which excludes 0-values for avoiding division by zero. However, for parameters which may have 0-values (for example in the case of packet loss), shall include slight corrections in their measurements to avoid 0-value assignments.

For a given message type, the route with the highest rating is to be selected as the target route. As the availability of a route is added as a 0/1 factor to the sum, an unavailable route will always have a rating of 0, ensuring that it will never be selected.

Co-funded by the Horizon 2020 Framework Programme of the European Union

4 Data Virtualization Layer

iNGENIOUS exploits the use of different M2M platforms, which are adopted by different supply chain stakeholders for collecting and storing raw data in maritime. On top of these data silos, the project envisages the implementation of a layer based on a Data Virtualization approach (Data Virtualization Layer, DVL), which will act as a federated and interoperable IoT layer for different M2M platforms as well as data sources (e.g., PCS) by providing shared access, management, and reading and writing capabilities to different entities (e.g., TrustOS, MANO Platform, Awake.AI platform) while ensuring security and privacy aspects following a role-based approach and applying pseudonymization techniques when needed.

From this perspective, Data Virtualization interoperability layer enables the federation of different IoT platforms across heterogeneous domains, overcoming the integration issues between both standard and non-standard, proprietary and custom M2M solutions widely used within the industry 4.0 verticals.

In the context of iNGENIOUS project, Data Virtualization Layer is used as a cross-UCs component by retrieving data from the underlying data sources and making them available to data consumers (pseudonymization module, TrustOS, Awake.AI platform, MANO platform, dashboard for trucks' tracking).

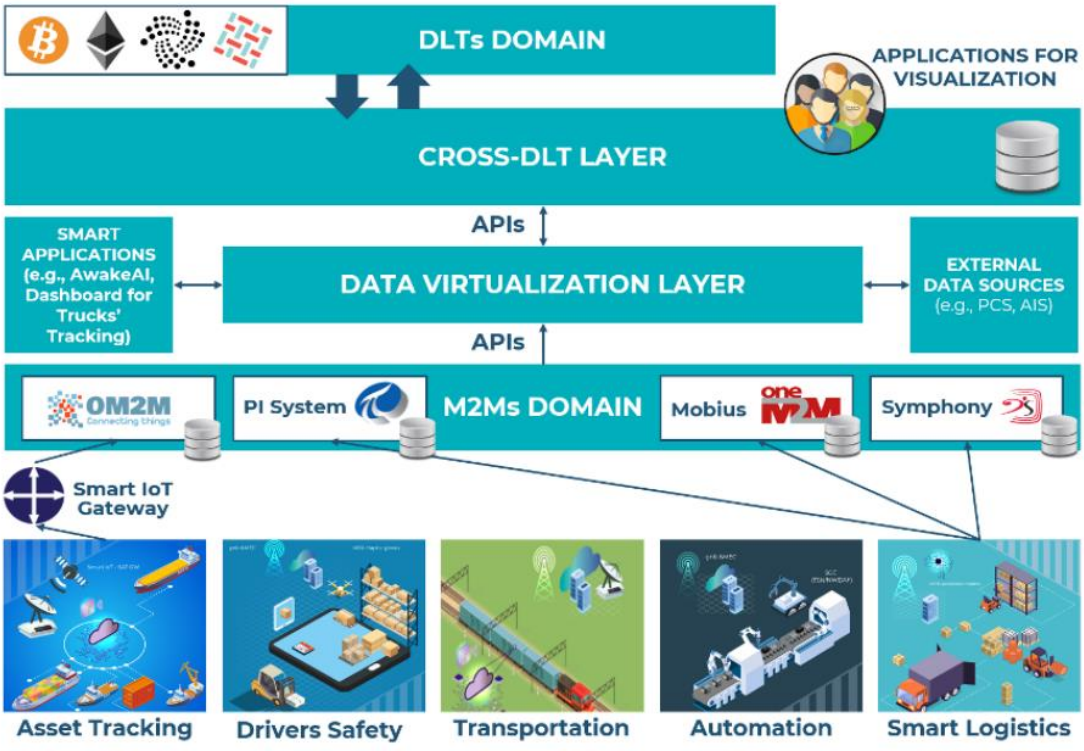


FIGURE 4: DVL AND RELATION TO INGENIOUS USE CASES.

The interoperability layer (based on DVL and Cross-DLT layer) is expected to be validated by means of a limited set of use cases within the project, namely the Situational Understanding and Predictive Models in Smart Logistics, Inter-Model Asset Tracking Via IoT and Satellite and Supply Chain Ecosystem Integration.

Co-funded by the Horizon 2020 Framework Programme of the European Union

As a matter of fact, the main aim of the UC6 implementation and validation is to test the interoperability between machine-to-machine and DLT domains throughout a centralized approach for data retrieval, access, processing and handling by means of Data Virtualization. By enabling these data flows then, DLT's capabilities are exploited to guarantee data existence and data immutability.

However, its potential goes beyond the selected application fields. From an architecture perspective, the interoperability layer can be also used in cross use-case scenarios, providing new data management capabilities for end users. The DVL suits fine for data lake management, regardless of the underlying physical layer, including physical devices and connectivity resources. For this reason, the DVL can act as a collector for data coming from different data sources and related to different application domains such as transportation, asset tracking, smart factories, and logistics. Once data have been collected, processed, and aggregated by DVL according to supported data, the Cross-DLT layer ensures secure events management by means of different available DLTs.

In this chapter, the main development and integration approaches involving Data Virtualization Layer component are presented and discussed by describing the underlying machine-to-machine layer including different M2M platforms brought by the project in order to clarify how such platforms are used in the scope of different use cases and what kind of data are managed.

4.1 DVL Introduction

Data Virtualization is an advanced approach to data integration. Data Virtualization can be considered as an easier way to integrate, federate and transform data coming from multiple data sources into a single and unified environment in real-time. It is not about collecting data from different data sources but connecting them and leveraging data warehouses, data lakes or different data infrastructures already in place. In the scope of the iNGENIOUS project, we relied on Teiid v16.0 platform [2] an open-source implementation of the data virtualization concept. Teiid is a flexible Java-based tool that provides integrated access to multiple data sources through a single and uniform API, so that applications can access information using standard interfaces such as JDBC, ODBC, OData or REST. This approach is relevant in those cases, like iNGENIOUS, when the information resides in multiple heterogenous data sources, including sources that do not support standard query languages and formats. It is not a database management system: it does not store any data and it acts as a data gateway for accessing data in an optimal manner.

4.1.1 Teiid deployment in a staging environment

Teiid is foremost an extensible Java project. There are several options to utilize it. In the scope of iNGENIOUS project, we relied on WildFly-based implementation (integration of Teiid into WildFly Java Application Server [3]). This configuration provides robust and well documented options for transaction management, connection pooling, security configuration,

Co-funded by the Horizon 2020 Framework Programme of the European Union

resource management, clustered deployment and data access policies management.

First of all, we deployed a local instance of WildFly server as well as Teiid platform in a staging environment in the Port of Livorno, hosted in a dedicated virtual machine with the following technical specifications:

- CPU host AMD EPYC 7281 with 4 core.
- 8GB of RAM.
- 100GB of storage space.
- Ubuntu 20.04 Server as the main operating system.

The DVL instance (Teiid v.16.0.0) has been then properly configured (on top of JBoss WildFly Application Server v.19.1.0) using a docker image. The deployment has been tested using JBoss console and the relative virtual databases by means of SQL client (SQuirreL).

The considered node will be accessible, up and running until the end of the project so that involved partners will be provided with access capabilities to relevant data.

4.1.2 Teiid integration with available data sources and external applications

In the context of iNGENIOUS project, Teiid is expected to be used as an intermediate layer for data access according to a given set of data sources as well as applications, brought by the partners from the consortium.

On one side we identified the following data sources to be used to fed DVL with relevant data:

- Mobius OneM2M Standard Platform: a M2M platform, based on oneM2M standard implementation, for meteorological data collection, processing and storage currently used in Livorno seaport. The historical data are expected to be consumed by AI-driven Smart Port and Shipping Platform by means of an interface.
- Eclipse OM2M Platform: a M2M platform which is expected to be used for collecting data coming from distributed IoT devices installed on the container to be transported by COSCO ship line (data are expected to be collected by means of the Smart IoT Gateway). In this case, data coming from the sensor monitoring the status of the container door will be used to feed the DVL component so that Seal Removal event can be considered at TrustOS level.
- Symphony M2M Platform: a M2M platform that is expected to be used for data collection, processing and storage in trucks' tracking scenario. DVL will be then used to retrieve such data allowing an external user dashboard to visualize them in a user-friendly manner.
- PISoft M2M Platform: a M2M platform used in Valencia seaport for collecting and storing gates access data. By integrating this platform with DVL, we will be able to define GateIn, GateOut, Vessel Arrival and Vessel Departure events to be stored on TrustOS for the case of Valencia seaport.

Co-funded by the Horizon 2020 Framework Programme of the European Union

- Tuscan Port Community System: the main Port Community System used in Livorno seaport for GateIn, GateOut, Vessel Arrival and Vessel Departure data provisioning.
- Valencia Port Community System: the main Port Community System used in Valencia seaport for Vessel Arrival and Vessel Departure events definition as well as implementation.

On the other side, we identified the following set of applications (consumers) that are expected to be used to consume aggregated data at DVL level.

- TrustOS: a Cross-DLT solution which extracts relevant data from DVL (e.g., vessel arrival, vessel departure, gate in, gate out, seal removal, etc.), converts them in a digital asset and distributes Trust Points across available DLTs: Bitcoin, Ethereum, IOTA and HyperLedger Fabric. Trust Points can be then used to guarantee the proof-of-integrity.
- AI-driven Smart Port and Shipping Platform: a smart IoT application that consumes data related to vessels' and trucks' traffic, port calls and meteorological conditions at Port of Valencia and Livorno. The main purpose of this integration is to guarantee a predictive model development for trucks' flows in both seaports by consuming historical data from DVL.
- Pseudonymization Module: a function allowing to detect personal data and pseudonymize them according to available pseudonymization techniques such as Format Preserving Encryption (FPE), Hash Without Key (HWK), Hashing with static key or BLURRING. In the scope of the iNGENIOUS project, the trucks' plate number is considered as the personal data, so a pseudonymization method is applied accordingly. Once personal data are detected, a pseudonym is created and the main correspondence between "plain" and "pseudonymized" data is stored within a local database management system (MongoDB) so that AI-driven Smart Port and Shipping Platform can consume historical data throughout the DVL for data analysis activities.
- Cross-layer MANO Platform: the iNGENIOUS network slice management and orchestration software stack that is expected to be integrated with DVL to collect relevant data for the runtime optimization of network services and slices. The main aim of this integration is to enable reactive and proactive adaptation of network slice and service instances based on both application-level (and possibly) network related data collected from the DVL

4.1.3 Teiid and maritime events definition

In order to come up with a commercially standardized way to define maritime events such as GateIn, GateOut, vessel arrival, vessel departure and container's seal removal, we relied on event data model adopted by Tradelens platform [4]. Tradelens is a blockchain-based platform allowing end users to securely store and exchange data between them (e.g., carriers, ports, terminals, customs, etc.).

We could have relied on a custom data model for the definition of such events but the current approach allows us to extend interoperability capabilities of the iNGENIOUS cross-DLT layer by making it (in principle) compliant with commercial blockchain-based platforms such as Tradelens. According to this

Co-funded by the Horizon 2020 Framework Programme of the European Union

data model, the maritime events have the following data structures that have been implemented at DVL level:


```
{
   "originatorName": "A Container Moving Enterprise",
   "originatorId": "ACME",
   "eventSubmissionTime8601": "2018-03-10T11:30:00.000-05:00",
   "transportEquipmentId": "d6ab2fe6-8331-623s-2173-kjshf76ehd",
   "transportEquipmentRef": null,
   "equipmentNumber": null,
   "carrierBookingNumber": null,
   "billOfLadingNumber": null,
   "eventOccurrenceTime8601": "2018-03-13T11:30:00.000-05:00",
   "transportationPhase": "Import",
   "location": {
      "unlocode": "NLRTM"
   },
   "vehicleId": "VEH625-259",
   "vehicleName": "Vehicle Name",
   "fullStatus": "Full",
   "terminal": null
}
```

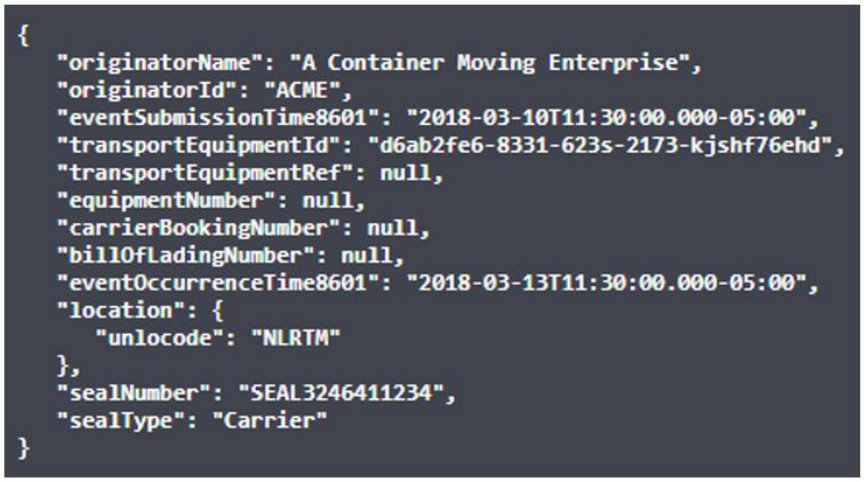

FIGURE 5: GATEIN/GATEOUT EVENTS DATA MODEL (TRADELENS PLATFORM [4]).


```
{
   "originatorName": "A Container Moving Enterprise",
   "originatorId": "ACME",
   "eventSubmissionTime8601": "2018-03-10T11:30:00.000-05:00",
   "transportEquipmentId": "d6ab2fe6-8331-623s-2173-kjshf76ehd",
   "transportEquipmentRef": null,
   "equipmentNumber": null,
   "carrierBookingNumber": null,
   "billOfLadingNumber": null,
   "eventOccurrenceTime8601": "2018-03-13T11:30:00.000-05:00",
   "transportationPhase": "Import",
   "location": {
      "unlocode": "NLRTM"
   },
   "vehicleId": "VEH625-259",
   "vehicleName": "Vehicle Name",
   "voyageId": "VOY838632",
   "terminal": null
}
```


FIGURE 6: VESSEL ARRIVAL AND VESSEL DEPARTURE EVENTS DATA MODEL (TRADELENS PLATFORM [4]).

GateIn, GateOut, Vessel Arrival and Vessel Departure events are part of the demonstration of both UC5 and UC6, while seal removal event is under assessment (the data model is defined but further investigation is required to understand what kind of data would be available from a smart IoT device installed on the container) as part of the UC4 and its relative model can be seen in the figure below (sealRemoved event from Tradelens platform [4]):

Co-funded by the Horizon 2020 Framework Programme of the European Union



```
{
   "originatorName": "A Container Moving Enterprise",
   "originatorId": "ACME",
   "eventSubmissionTime8601": "2018-03-10T11:30:00.000-05:00",
   "transportEquipmentId": "d6ab2fe6-8331-623s-2173-kjshf76ehd",
   "transportEquipmentRef": null,
   "equipmentNumber": null,
   "carrierBookingNumber": null,
   "billOfLadingNumber": null,
   "eventOccurrenceTime8601": "2018-03-13T11:30:00.000-05:00",
   "location": {
      "unlocode": "NLRTM"
   },
   "sealNumber": "SEAL3246411234",
   "sealType": "Carrier"
}
```


FIGURE 7: SEAL REMOVED EVENT DATA MODEL (TRADELENS PLATFORM [4]).

According to this data model, four different procedures have been implemented on DVL after the identification of the main data sources to be used for data retrieval:

- **LatestGateInEvent**: when invoked, allows to retrieve the latest occurred gate-in event. This procedure will be used by the TrustOS-based Cross-DLT layer.
- **LatestGateOutEvent**: when invoked, allows to retrieve the latest occurred gate-out event. This procedure will be used by the TrustOS-based Cross-DLT layer.
- **HistoricalGateInEvent**: when invoked, allows to retrieve historical gate-in events. This procedure will be used by the pseudonymization module.
- **HistoricalGateOutEvent**: when invoked, allows to retrieve historical gate-out events. This procedure will be used by the pseudonymization module.
- **LatestVesselArrivalEvent**: when invoked, allows to retrieve the latest occurred Vessel Arrival event, according to the eventSubmissionTime8601 attribute.
- **LatestVesselDepartureEvent**: when invoked, allows to retrieve the latest occurred Vessel Departure event, according to the eventSubmissionTime8601 attribute.
- **HistoricalVesselArrivalEvent**: when invoked, allows to retrieve the whole set of historical data for the Vessel Arrival event.
- **HistoricalVesselDepartureEvent**: when invoked, allows to retrieve the whole set of historical data for the Vessel Departure event.

The above-mentioned procedures allow to retrieve gate-in and gate-out data for the case of Livorno seaport: the same procedures will be also implemented for the case of Valencia seaport according to available data sources (e.g., the port community system and machine-to-machine platform). During the second half of the project, we expect to implement also procedures for the vessel arrival and vessel departure events for both seaports.

In order to interact with DVL by invoking the available set of procedures, a REST interface from OData (an open protocol to allow the creation and

Co-funded by the Horizon 2020 Framework Programme of the European Union

consumption of interoperable RESTful APIs) has been implemented. Moreover, access details have been already shared with involved partners in order to perform preliminary communication tests.

In the following picture, as an example, the main result of the http REST query request performed to DVL, is depicted (LatestGateInEvent procedure):


```
{
"originatorName": "Port of Livorno",
"originatorId": "ITLIV",
"eventSubmissionTime8601": "2021-07-31T06:38:00.2463483+01:00",
"transportEquipmentId": NULL,
"transportEquipmentRef": NULL,
"equipmentNumber": "CRLU1146243",
"carrierBookingNumber": NULL,
"billOfLadingNumber": "MEDUCJ362325",
"eventOccurrenceTime8601": "2021-07-31T06:36:52.4230000+01:00",
"transportationPhase": "Import",
"location": "ITLIV",
"vehicleId": "FJ871ED",
"vehicleName": "Truck",
"fullStatus": "F",
"terminal": "LRN",
"voyageId": "19436"

}
```


FIGURE 8: LATESTGATEINEVENT RESPONSE EXAMPLE (LIVORNO SEAPORT).

4.2 Mobius OneM2M System, integration & data aggregation

Mobius [8] is an open-source server platform implementing oneM2M standards available in the OCEAN (Open allianCE for IoT stANdards), an open source-based global partnership project for IoT (CNIT is a developer partner). One advantage of using the public IoT server (from KETI - Korea Electronics Technology Institute) is that it is possible to use a web-based oneM2M resource monitoring application that makes it easy to monitor sensing values and actuation commands for the IoT devices in real-time.

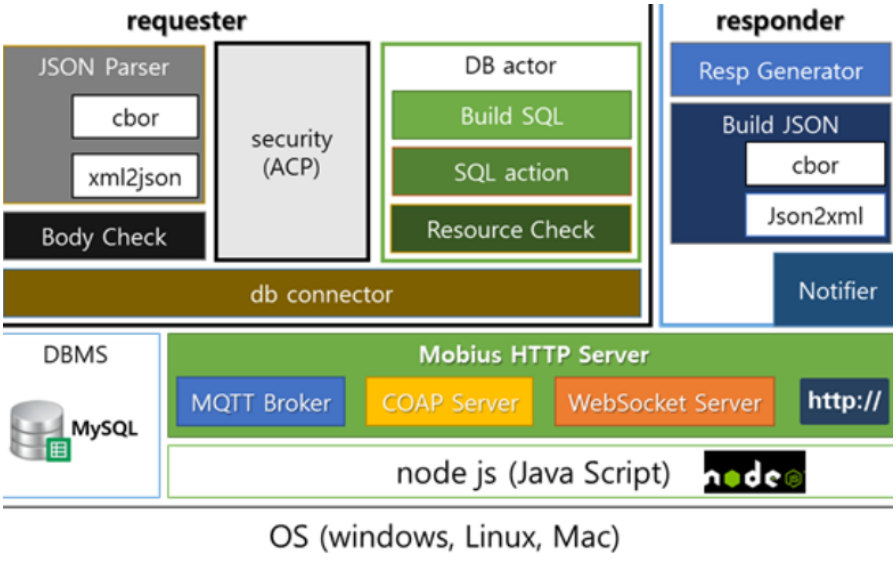


FIGURE 9: MOBIUS SOFTWARE ARCHITECTURE.

Co-funded by the Horizon 2020 Framework Programme of the European Union

The Port Authority of Livorno in collaboration with CNIT, has employed the public Mobius-based IoT server provided by KETI in order to perform smart-objects and IoT devices management at seaport. Currently, the Mobius platform allows to manage and interact with meteorological stations.

The following section describes the main configuration and deployment that we rely on for the development activities foreseen by the iNGENIOUS project.

4.2.1 Mobius OneM2M Platform deployment and integration

The access to data sources in Teiid takes two components: a translator and a platform-dependent access mechanism to provide source access. A translator provides an abstraction layer between Teiid Query Engine and physical data source, that knows how to convert Teiid issued query commands into source specific commands and execute them. Translators also have some logic to convert the result data that came from the physical source into a form that Teiid Query Engine is expecting.

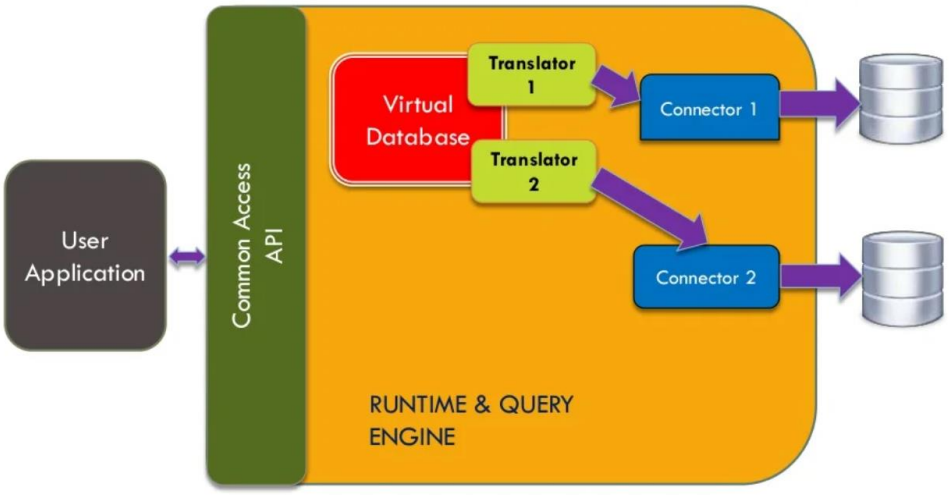


FIGURE 10: TEIID QUERY ENGINE.

A translator is represented in DDL as a foreign data wrapper (FDW). Together this access is referred to as a Teiid connector. According to this nomenclature, a REST connector (including .xml VDB definition file) has been implemented in order to allow the integration between DVL and Mobius OneM2M IoT Server Platform deployed as a docker image within a dedicated virtual machine with the following specifications (Livorno staging environment):

- CPU host AMD EPYC 7281 with 4 cores.
- 8GB of RAM.
- 100GB of storage space.
- Ubuntu 20.04 server as the main operating system.

In the following picture, the virtual-database definition file (.xml) for the communication between DVL component and Mobius M2M standard platform is depicted:

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<vdb name="mobius_onem2m" version="1">

    <description>Connecting to a Mobius onem2m database</description>

    <!--property name="{http://teiid.org/rest}auto-generate" value="true"/-->

    <model name="ONEM2M" type="VIRTUAL">
      <metadata type="DDL"><![CDATA[

        CREATE VIEW SensorRegistryView (
            pi string(4000),
            ty string(4000),
            ct string(4000),
            ri string(4000),
            rn string(4000),
            lt string(4000),
            et string(4000),
            st integer,
            cs integer,
            cr string(4000),
                con string(4000)
        )
            AS
        SELECT
            A.pi, A.ty, A.ct, A.ri, A.rn, A.lt, A.et, A.st, A.cs, A.cr, A.con FROM
            (EXEC Onem2mSource.invokeHttp(
            action=>'GET',
            endpoint=>'http://172.25.1.11:7579/Mobius/SENSORS_REGISTRY/sensors_registries/la',
            headers=>jsonObject('application/json' as "Content-Type", NOW() as "X-M2M-RI"
            ))) AS f,
            JSONTABLE(
            TO_CHARS(f.result, 'utf-8', true),
            '$.m2m:cin' COLUMNS "pi" string, "ty" string, "ct" string, "ri" string, "rn" string, "lt" string,
            ) AS A;
```

FIGURE 11: MOBIUS ONE M2M CONNECTOR IMPLEMENTATION - VDB DEFINITION FILE (.XML).

4.3 PI System OSIsoft, integration

PI System is a M2M platform developed by OSIsoft used in heavy industrial environments for covering data processing and data streaming functionalities as part of industrial IoT ecosystems [14]. As described in D5.1, PI System is composed of several software modules to cover data collection, time-series historization, finding, analysis, delivery and visualization functionalities for managing real-time data and events. Data can be automatically collected from different sources such as control systems (e.g., SCADA), industrial computers, dedicated HW, laboratory equipment or other external systems.

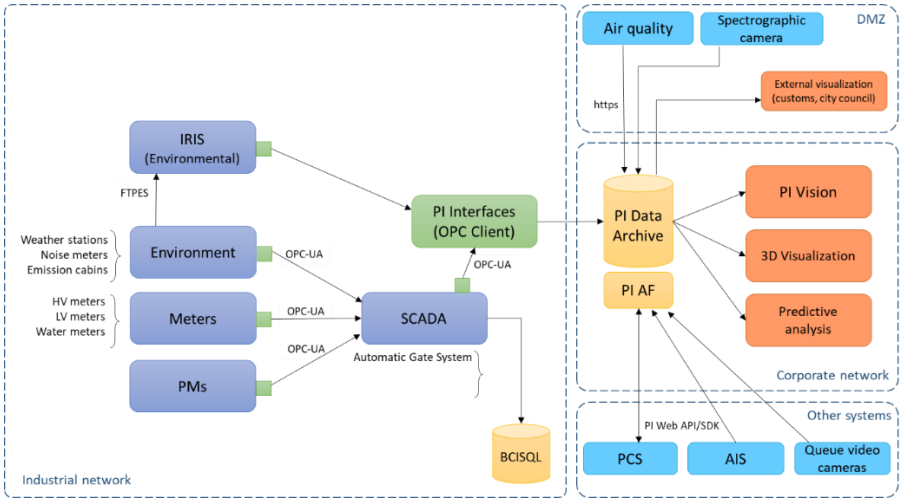


FIGURE 12: VALENCIA PORT PI DEPLOYMENT.

Co-funded by the Horizon 2020 Framework Programme of the European Union

In iNGENIOUS, PI System OSIsoft will be used to store and exchange the data related to GateIn and GateOut events performed at the terrestrial accesses of the Port of Valencia in the context UC5 (Situational Understanding and Predictive Models in Smart Logistics Scenarios) and UC6 (Supply Chain Ecosystem Integration). To meet the needs of these use cases and integrate the data obtained at terrestrial accesses with the DVL, FV will exploit a pre-production deployment of PI system available at the Port of Valencia ICT infrastructure.

4.3.1 Integration Approach

The integration of the data gathered by PI System and the DVL will be supported by involving three main PI modules: PI Data Archive, PI Asset Framework (AF) and PI Web API.

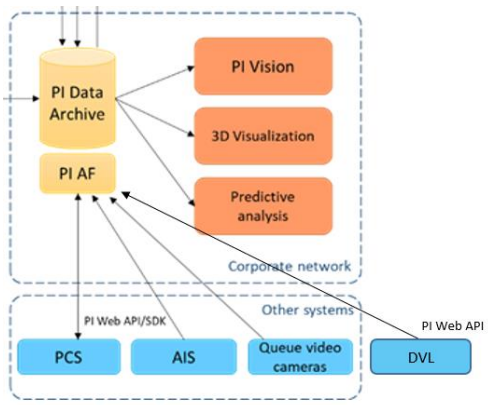


FIGURE 13: PI SYSTEM AND DVL INTEGRATION

The PI Data Archive is the time-series database of PI System, where data obtained from the different data sources is stored and structured. Based on the data stored in the Data Archive, PI AF contextualises raw data and organizes assets and signals following hierarchical structures in order to provide structured data sets. After that, PI Web API provides the access layer through a RESTful interface allowing external client applications to read and write over the PI Data Archive and the PI AF over HTTPS programmatically.

PI Web API defines a set of methods for retrieving information stored in PI System from external platforms. To enable all reading and writing operations an Authorization Header with hash encryption is configured to authorise the API use. The URL base of the API will be:

/api/stream/[method]

and all responses will have JSON format. The four different methods defined for retrieving information (GET) from PI Web API are:

1. */signalinfo/{signalId}:* Returns the value of the request signal in a specific time period, i.e., from '*starttime*' to '*endtime*'. If no date is specified, the last value of the signal is return. The response has the following format:

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
{
    "name": "SINUSOID",
    "startdate": null,
    "enddate": null,
    "values": [
        {
         "value": "Calc Failed"
         "timestamp": "2022-01-13T14:04:00"
        }
    ]
}
```


FIGURE 14: SIGNALINFO API RESPONSE

2. */signalstream/{signalId}*: Returns the value of a signal following a socket subscription approach. The JSON response has the following format:


```
{
  "Name": "SINUSOID",
  "Value": "505,555542"
  "Timestamp": "2022-01-13T14:35:40"
}
```


FIGURE 15: SIGNALSTREAM API RESPONSE

3. */assetinfo/{assetName}*: Returns the value of the requested asset. The response has the following format:


```
{
  "assetName": "Mixing Tank 1",
  "attributes": [
  {
    "Name": "Diameter",
    "piPoint": null,
    "Value": {
      "Value": "42"
      "Timestamp": "27-oct-2021 14:28:23"
    }
  },
  {
    "Name": "Height",
    "piPoint": null,
    "Value": {
      "Value": "42"
      "Timestamp": "27-oct-2021 14:28:23"
    }
  }
  ...
 ]
}
```


FIGURE 16: ASSETINFO API RESPONSE

4. */eventinfo/{eventId}}*: Returns the value of a specific event, which occurs when specific conditions are fulfilled. To specify the event related parameters like *serverName, databaseName, starttime and endtime* need to be indicated. If *endtime* parameter is not specified, a list of

Co-funded by the Horizon 2020 Framework Programme of the European Union

values registered since the event started are given. Other optional parameters like *name*, *template*, *finished*, *duration* or *durationConditional* can also be indicated. The JSON response has the following format:


```
{
  "items": [
    {
      "finished": true,
      "duration": "00:36:00",
      "elements": [
        {
          "name": "Pump04",
          "attributes": [
            {
              "name": "Bearing Temperature",
              "piPoint": "OSIDemo_Pump Station _Pump04.Bearing Temperature.0bbd043e-8ef9-599f-3838-b2cff22e1d06",
              "value": {
                "value": "180,380127",
                "timestamp": "2022-01-13T14:10:00"
              }
            }
            ...
          ]
        }
      ],
      "name": "OSIDemo - Pump Downtime Event 2019-11-07 22:48:00.000 - Pump04",
      "description": "",
      "template": "Pump Downtime Event",
      "startTime": "2019-11-07T21:48:00Z",
      "endTime": "2019-11-07T22:24:00Z",

    }
    ...
  ]
}
```


FIGURE 17: EVENTINFO API RESPONSE

5. */dbstructure/{serverName/{databaseName}:}:* Returns a tree with the data structure of a specific database. The JSON response has the following format:


```
{
  "name": "Demos OSI",
  "children": [
    {
      "name": "Assets",
      "children": [
        {
          "name": "TX211",
          "children": null
        }
        ...
      ]
    },
```


FIGURE 18: DBSTRUCTURE API RESPONSE

4.4 Symphony platform, integration

Symphony is a service-oriented generalized IoT platform developed and commercialized by Nextworks, capable of integrating thousands of interconnected devices in support of multiple vertical needs and services [12]. It embeds several functionalities (interfacing with field bus protocols, data acquisition and actuators control, data storage and processing, rule-based engines, application logic and GUIs) into a unified fully decomposed and distributed IP-based platform.

As depicted in Figure 19Figure 19, Symphony is a complete IoT platform characterized by a modular architecture which allows to interact with a variety of hardware devices, IoT sensors or actuators in a seamless and unified

Co-funded by the Horizon 2020 Framework Programme of the European Union

manner. Internally, Symphony integrates a number of services for notifications, event management, analytics and automated reactions which can be applied to a variety of applications, like access control, technical systems monitoring, industrial automation, energy management, etc.

The Symphony core component is the Hardware Abstraction Layer (HAL), which provides protocol-specific southbound plugins to enable the interconnection with heterogeneous protocol gateways and devices deployed in the IoT field. The data collected from the field is associated to "plain objects", which are generalized items expressed in a simple unified format built by tuple of *identifier, timestamp, value*. The data is then distributed towards the upper layer Symphony components through a variety of northbound plugins, to enable data processing and elaboration and generate "objects with semantics", that constitute "meaningful" inputs for the upper layers in the platform. Symphony currently supports CORBA (Common Object Request Broker Architecture) and REST northbound plugins.

In terms of relevance for iNGENIOUS, the HAL provide key features for the control and managements of IoT devices and related data. First it supports on-demand runtime configuration, and it allows to create and instantiate southbound plugins for new field buses at runtime, using the predefined set of plugins already available in Symphony. However, the support of new protocols and IoT devices requires the development of dedicated southbound plugins. Second, the HAL gives the possibility to configure the information models supported by new IoT devices or data sources to be plugged into Symphony, enabling a flexible adaptation to different kinds of sensors and providing data model interoperability. Finally, the HAL enables the configuration of sensors' digital twins, as it possible to create multiple plain objects controlled and managed through the same fieldbus southbound plugin, and thus store data related to the status of the object and possibly run some commands on the physical device through the specific plugin.

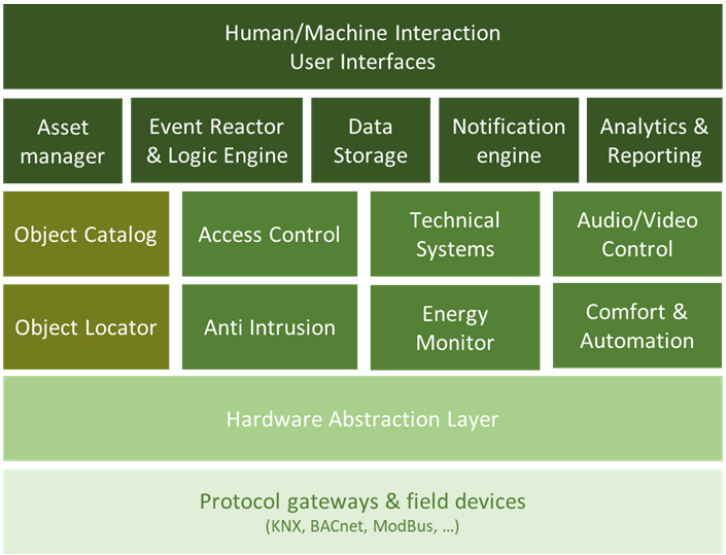


FIGURE 19: SYMPHONY HIGH-LEVEL ARCHITECTURE

Co-funded by the Horizon 2020 Framework Programme of the European Union

4.4.1 Integration approach

Symphony can be used in the Port of Livorno, and specifically in the context of UC5 (Situational Understanding and Predictive Models in Smart Logistics Scenarios) and UC6 (Supply Chain Ecosystem Integration), as M2M platform to integrate the IoT tracking sensors mounted onboard of vehicles within the port area. In practice, an instance of the Symphony platform is expected to be deployed and run in the Port of Livorno staging environment, and thus collect positioning data and events from the IoT tracking sensor devices and expose them to the DVL.

In particular, the IoT tracking sensor device type used in the Port of Livorno is a Micktrack MT821, a waterproof asset GPS tracker that use CAT M1 and NB-IoT technologies to provide low power consumption and optimized data transmission. These devices implement a custom communication protocol [5] to publish the asset positioning data towards an external server.

For the purpose of the interconnection with the DVL, Symphony requires two different types of integrations, as depicted in Figure 20Figure 20. First, an integration at the level of the HAL southbound plugins has to be implemented, to interface with the IoT tracking sensors communication protocol and thus be able to collect the related data. Second, an integration at the level of the HAL northbound is required, to expose the IoT tracking sensor internal objects directly towards the DVL as plain objects. To do that, a specific HAL northbound plugin is required, e.g., based on the AMQP or MQTT protocols (as described in the next sub-section).

Moreover, as the Symphony HAL do not store itself any data (indeed it keeps in memory the last value collected from a specific object to distribute it towards the other platform components), an additional Symphony component is required as further northbound integration. Indeed, the Symphony Data Storage component is used as well to store all the historical IoT tracking sensor objects data, and expose it to the DVL (which in turn do not store any data as well).

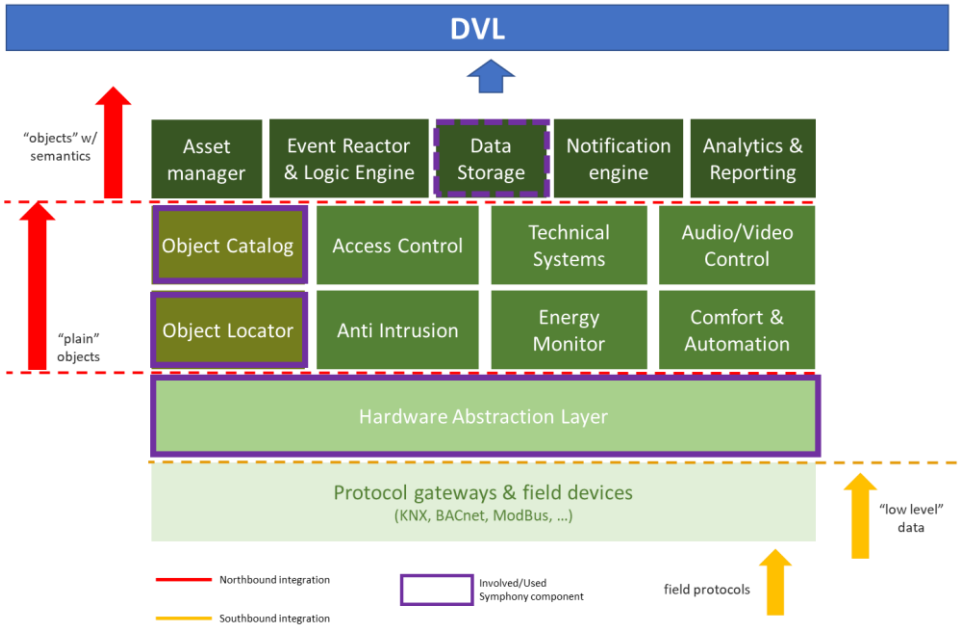


FIGURE 20: SYMPHONY INTEGRATION APPROACH

Co-funded by the Horizon 2020 Framework Programme of the European Union

4.4.2 Symphony enhancements

The implementation of the Symphony integration approach described above requires the development of few enhancements in the legacy Symphony platform components. In particular, as depicted in Figure 21, most of these enhancements refer to new capabilities and functionalities within the HAL and Data Storage, assuming to expose towards the DVL the IoT tracking sensor object data directly.

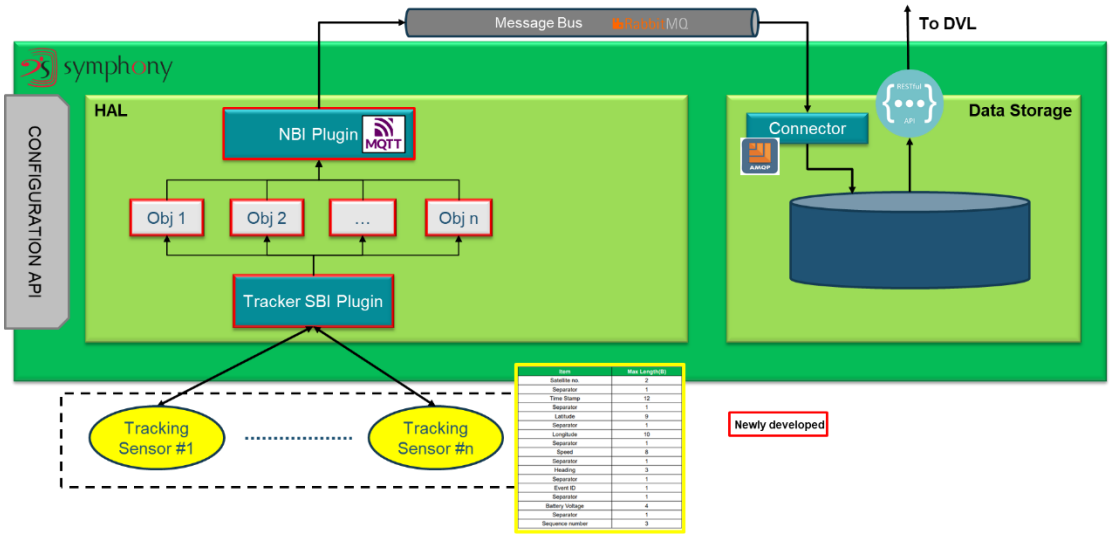


FIGURE 21: OVERVIEW OF SYMPHONY HAL ENHANCEMENTS

First, a dedicated new HAL southbound plugin is required to collect the data from the IoT tracking sensor devices and map it to specific and dedicated plain objects maintained by the HAL. The Micktrack MT821 communication protocol [5] relies on a server, to be offered and implemented in this new Tracker HAL southbound plugin, where to publish the vehicle GPS data. The structure of a generic message sent by the IoT device is depicted in Figure 22, while the specific report message for the GPS information of the vehicle is shown in Figure 23. The Tracker HAL southbound plugin is responsible to collect and parse these periodic messages and make it available to the HAL.

Content	Length(Byte)
Head	2
Separator	1
Mode	1
Separator	1
IMEI	15
Separator	1
Data Type	2
Separator	1
Report Data	90

<Head> Fixed characters, which is "MT"
<Separator>Fixed character, which is ";"
<Mode >the device working mode
<IMEI>Device IMEI number (15 digital)
<Data Type> check the follow table

FIGURE 22: IOT TRACKING SENSOR MESSAGE FORMAT

Co-funded by the Horizon 2020 Framework Programme of the European Union

Item	Max Length(B)
Satellite no.	2
Separator	1
Time Stamp	12
Separator	1
Latitude	9
Separator	1
Longitude	10
Separator	1
Speed	8
Separator	1
Heading	3
Separator	1
Event ID	1
Separator	1
Battery Voltage	4
Separator	1
Sequence number	3

FIGURE 23: IOT TRACKING SENSOR GPS MESSAGE

Indeed, as part of this new Tracker HAL southbound plugin, a translation feature is also required to filter, map and transform structured data received from the IoT tracking sensor devices into the internal format expected by the HAL. In addition, as the plugin is an implementation of a base Symphony HAL plugin, it can be dynamically instantiated and (re-) configured through dedicated REST API calls (as described in the next subsection), enabling its on-demand creation and configuration.

Moreover, at its northbound, the HAL natively gives the possibility to define and configure the specific protocol and mechanism to expose the processed data fetched from the fieldbus. The default HAL northbound plugin is based on the CORBA framework, and it is used for the communication between the Symphony HAL and the rest of the platform components shown in Figure 19Figure 12. In iNGENIOUS, the requirement to expose the data collected from the IoT tracking sensor devices towards the DVL imposes the need of a dedicated new northbound plugin. Following the same Symphony approach adopted at the southbound, the HAL northbound is a plugin-based interface where it is possible to develop and integrate new specific plugins to export data towards external entities following different protocols and technologies. Moreover, at runtime it is also possible to instantiate new plugin instances, the same way it is done with the southbound ones (as described in the next subsection).

For the integration with the DVL, the selected approach for the HAL northbound plugin is based on a MQTT based interface. In particular, this northbound plugin allows to integrate the Symphony Data Storage and store there the data collected from the HAL through the use of a message bus based on RabbitMQ [6]. Indeed, as anticipated above the DVL does not store itself any data, therefore the iNGENIOUS Symphony deployment has to include the Data Storage component as well, that is integrated with the message bus through an AMQP connector, and on top of which a dedicated REST interface exposes the data towards the DVL. This approach offers to the DVL the possibility to retrieve historical data with filtering options to retrieve

Co-funded by the Horizon 2020 Framework Programme of the European Union

data related to specific time windows and IoT Tracking sensor devices. Details on this REST interface are provided in the next subsection.

4.4.3 APIs and data models

The iNGENIOUS Symphony deployment described above offers several REST APIs for different purposes: platform configuration and data retrieval.

The platform configuration APIs refer to specific REST APIs through which northbound and southbound plugins, as well as HAL plain digital objects can be dynamically created and configured according to the specific requirements of the IoT data to be managed.

For what concerns the HAL southbound and northbound plugins configuration, all request and response messages are formatted as JSON objects. The following REST operations are supported:

- GET /plugin
  - To retrieve the configurations of all the available HAL plugins
- POST /plugin:
  - To create a new southbound or northbound plugin, by issuing the JSON reported below, where the plugin specific configuration in the case of the IoT tracking sensors includes information of the UDP server (IP, port) to which the devices connect to send the GPS data messages reported in Figure 23

```
{
        "name": <Plugin Name>,
        "class": <Plugin Class>,
        "enabled": true,
        "config": {
                <Plugin Specific Configuration>
        }
}
```

- GET /plugin/{plugin_name}
  - To retrieve the configurations of the specific southbound or northbound plugin <*plugin name*>
- DELETE /plugin/{plugin_name}
  - To delete the southbound or northbound plugin <*plugin name*>
- PUT /plugin/{plugin_name}
  - To update the following attributes of the configuration of the specific southbound or northbound plugin <*plugin name*>

```
{
        "enabled": true,
        "config": {
                <Plugin Specific Configuration>
```


Co-funded by the Horizon 2020 Framework Programme of the European Union

```
        }
}
```

For what concerns the HAL digital objects configuration, all request and response messages are again formatted as JSON objects. The following REST operations are supported:

- GET /datapoint
  - To retrieve the configurations of all the HAL digital objects
- POST /datapoint:
  - To create a new HAL digital object, by issuing the JSON reported below, where the plugin field identifies the southbound plugin associated to this digital object, and the datapoint specific configuration in the case of the IoT tracking sensors includes information on which attributes to filter and keep from the GPS data messages collected by the southbound plugin

```
{
  "id":"<object_id>",
  "type":"<object_type>",
  "plugin":"<SBI_PLUGIN_NAME>",
  "config": {
     <Datapoint Specific Configuration>
  }
}
```

- GET /datapoint/{object_id}
  - To retrieve the configurations of the specific HAL digital object *<object_id>*
- DELETE /datapoint/{object_id}
  - To delete the HAL digital object *<object_id>*
- PUT /datapoint/{object_id}
  - To update the following attributes of the configuration of the specific HAL digital object *<object_id>*

```
{
        "config": {
                <Datapoint Specific Configuration>
        }
}
```

On the other hand, in terms of actual integration with the DVL, the Symphony Data Storage exposes specific REST APIs to retrieve the IoT tracking sensors data on-demand. In particular the following query operation is supported, with response messages formatted as JSON objects:

- GET / cmd/retrieve_config?section=timeseries

Co-funded by the Horizon 2020 Framework Programme of the European Union

- To retrieve the full list of available HAL digital object timeseries within the Symphony Data Storage. In particular, this operation returns information on the total number of timeseries available, i.e., how many HAL digital objects have data stored (*records* filed), and as shown below, for each digital object (identified with its HAL *object_id*) some additional information concerning the type of associated IoT object and southbound plugin

```
{
        "records": <num_of_timeseries>,
        "timeseries": {
                "<object_id>": {.....}
        }
}
```

- GET /object/{object_id}
  - To retrieve the timeseries data for the HAL digital object identified with *object_id*. In particular, the returned JSON message includes the full list of collected object data collected by Symphony. Each entry has its own timestamp (*ts*). For the case of the IoT tracking sensor (shown below), the GPS position, speed and heading attributes are available (in the *v* field)

```
{
        "data": [{
                        "ts": "< unix_timestamp >",
                        "v": {
                                "lat": "< latitude >",
                                "long": "< longitude >",
                                "speed": "< speed >",
                                "heading": "< heading >"
                        }
                },
                {
                        "ts": "< unix_timestamp >",
                        "v": {
                                "lat": "< latitude >",
                                "long": "< longitude >",
                                "speed": "< speed >",
                                "heading": "< heading >"
                        }
                },
                ........
        ]
}
```


Co-funded by the Horizon 2020 Framework Programme of the European Union

- GET /object/{object_id}?start=<start_timestamp>,&end=<end_timestamp>
    - To retrieve the timeseries data for the HAL digital object identified with *object_id*, limited to a time window delimited by <start_timestamp> and <end_timestamp>. The returned JSON message is structured as the one in the previous GET operation

4.5 Eclipse OM2M, integration

The Eclipse OM2M project [1], initiated by LAAS-CNRS, is an open-source implementation of oneM2M and SmartM2M standard. It provides a horizontal M2M service platform for developing services independently of the underlying network, with the aim to facilitate the deployment of vertical applications and heterogeneous devices. The Smart IoT GW relies on the Eclipse OM2M implementation of the oneM2M standards. Essentially, OM2M implements components and applications for each of the standards described in oneM2M. Among its main features are:

- Not only compliant with oneM2M but also with SmartM2M [13];
- Open Service Gateway initiative (OSGi)-based architecture extensible via plugins;
- Implements a restful API with a generic set of service capabilities.

Provides oneM2M services such as machine registration, application deployment, container management, resource discovery, access right, authorization, subscription/notification, group management and non-blocking requests.

As described in Section 3.1.1, the Smart IoT Gateway interfaces with the M2M space through a local MN-CSE instance (Middleware Node Common Service Entity) implemented as *Eclipse OM2M* in compliance with the oneM2M standard. Following the oneM2M standard, the MN-CSE local to the gateway communicates the retrieved data with the central IN-CSE (Infrastructure Node Common Service Entity), which is hosted in the *SES Cloud space*. To ensure privacy and security of the transported data, all communication between the MN-CSE in the gateway and the IN-CSE in the cloud space relies on encrypted HTTP traffic (HTTPS) and additionally, is routed through a secure VPN tunnel, allowing only external instances with a valid VPN client configuration to connect to the M2M cloud space.

External M2M applications which need to request data from the M2M cloud space (e.g., DVL), connect to the IN-CSE through a dedicated public IP, with a firewall managing access to specified IP sources only (see Figure 3 in Chapter 3.1.1).

According to this, the M2M space (based on Eclipse OM2M solution) is expected to be used for the storage of data coming from installed sensors (the one which acts as the container's seal). Once data are stored, the DVL can easily retrieve them by means of a specific view/procedure so that SealOpening event can be available at Cross-DLT layer. Data format and structure is still under investigation and further details will be available in D5.3 – Final iNGENIOUS Data Management Platform.

Co-funded by the Horizon 2020 Framework Programme of the European Union

4.6 Pseudonymization Function

The DVL integrates in its architecture a pseudonymisation module used for the obfuscation of personal data. In this way, applications that need to use data from M2M platforms will not have access to sensitive data.

Pseudonymisation is the processing of personal data used in INGENIOUS in such a manner that the personal data can no longer be attributed to a specific data subject entering the port, without the use of additional information.

The only personal data that DVL handles in INGENIOUS are the license plate numbers of trucks entering the port (vehicleId).

The Pseudonymisation function (PF), denoted P, is a function that substitutes the vehicleId by a pseudonym. The pseudonymization function is integrated in the Pseudonymization module composed by microservice architecture based on 3 levels (see Figure 24Figure 24):

- Front-end
- Back-end
- Database

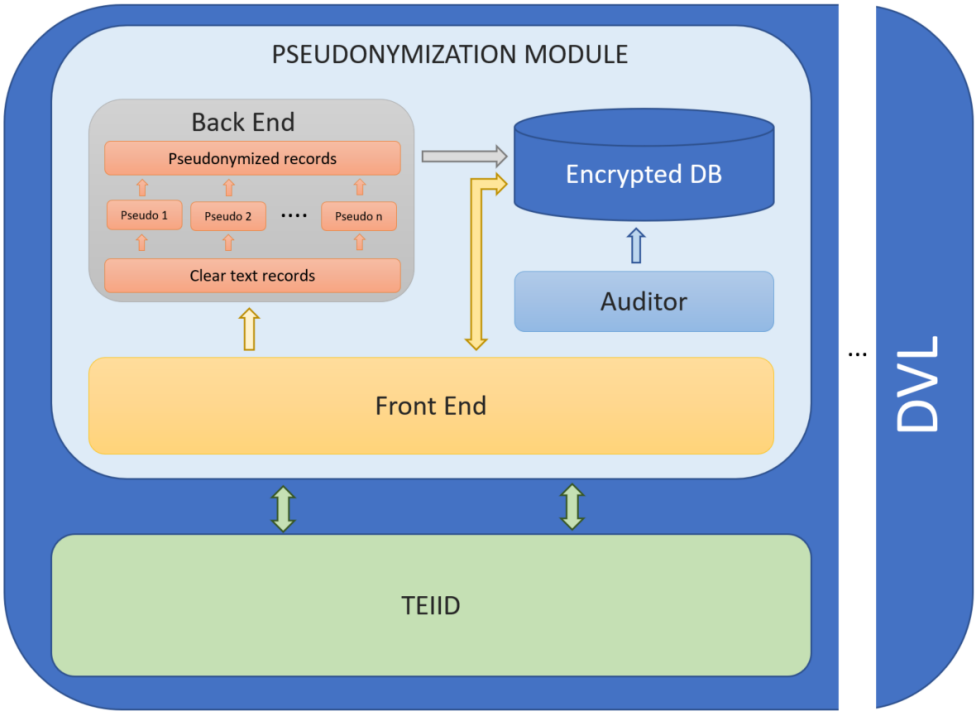


FIGURE 24: PSEUDONYMIZATION MODULE

THE FRONT-END USES/EXPOSES INTERFACES VIA HTTPS TO RETRIEVE EVENTS IN CLEAR FORMAT FROM TEIID, SEE [4.1 DVL INTRODUCTION], AND TO DEPLOY THE EVENTS IN PSEUDONYMIZED FORM WHEN REQUESTED, SEE WORKFLOW IN FIGURE 25FIGURE 25

Figure 25. The back-end elaborates data and store them into the encrypted database.

Co-funded by the Horizon 2020 Framework Programme of the European Union

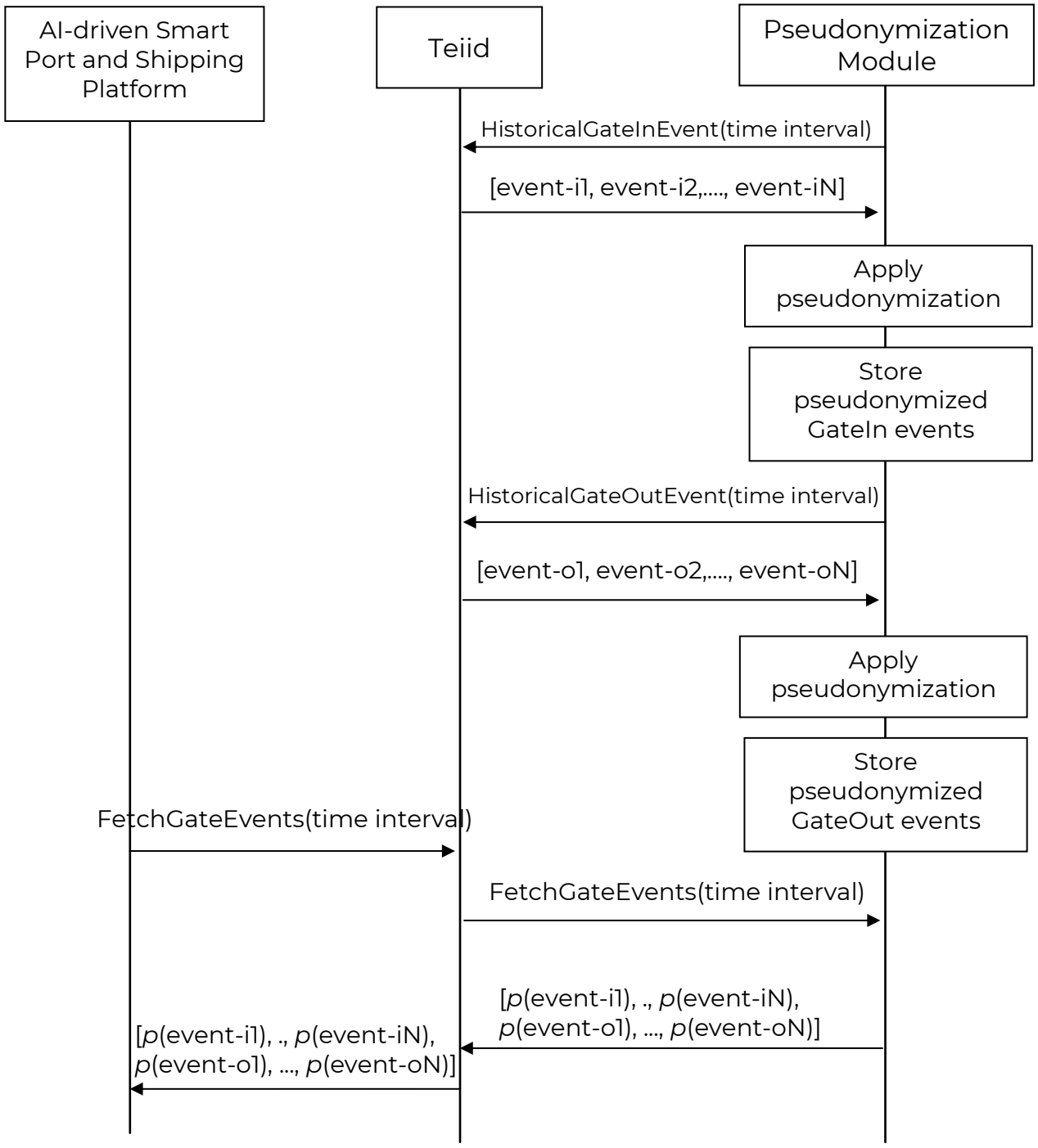


FIGURE 25: PSEUDONYMIZATION WORKFLOW

Using dedicated interfaces developed in the DVL layer (Teiid) the module reads once a day all historical *GateIn* and *GateOut* port entrance events in the specified interval time (set to 24h for the project scope).
The Pseudonymisation function applies the selected pseudonymization algorithm on the fetched personal data in GateIn/GateOut events, before storing them into the encrypted database.

The new pseudonymized events will be retrieved by DVL, when a new request comes from AI-driven Smart Port and Shipping Platform.

Co-funded by the Horizon 2020 Framework Programme of the European Union

4.6.1 Front-End

Front-end exposes the following Rest APIs:

- configTemplate: this interface is used to set the pseudonymization technique to apply.
- HistoricalGateInEvent: when invoked, allows the PF to retrieve historical GateIn according to a specified interval time (see chapter 4.1.3)
- HistoricalGateOutEvent: when invoked, allows the PF to retrieve historical GateOut according to a specified interval time (see chapter 4.1.3)
- FetchGateEvents: when invoked, allows the DVL to retrieve historical pseudonymized GateIn/GateOut events according to a specified interval time
- ClearData: this interface is used to delete a specified event

4.6.2 Back-End

The back-end is the core of the pseudonymization function. It receives the events GateIn/GateOut, elaborates the personal data (vehicleId) substituting them with the proper pseudonyms (applying the configured pseudonymization technique) and saves the events in the Passing_Pseudo table, ready to be fetched by DVL via FetchGateEvents.

The admitted pseudonymization techniques are listed below:

- Format Preserving Encryption (FPE)
- Blurring
- Hashing without key (HWK)
- Hashing with static key

Hashing without key is the default technique, so if it is the desired one it is not necessary to configure anything. Instead, if the preferred technique is different from Hashing it is necessary to set it by configTemplate API.

Hashing with static key technique is scope of next deliverable D5.3.

Note, AI-driven Smart Port and Shipping Platform uses vehicleId data for statistic scope, it can use pseudonym instead of clear format value but to the same vehicleId must always correspond the same pseudonym. This requirement limits the choice of possible techniques, in fact more sophisticated pseudonymisation techniques (such as probabilistic encryption) cannot be used.

4.6.3 Database

The encrypted database hosts different tables:

- *Configuration table*, that retains the pseudonymization function configurations, such as encrypted method and data retention period.
- *Conversion table*, used when it is necessary to maintain a map between the personal data in clear text and its pseudonym (Blurring/Hashing without key).

Co-funded by the Horizon 2020 Framework Programme of the European Union

- *Passing_Pseudo table*: it contains all the GateIn/GateOut events with pseudonymized personal data
- *Retention Data table*, it contains the references to all the pseudonymized events with proper expiration date

4.6.4 Retrieve pseudonymization data from PF

When an application wants to retrieve data (in pseudonym format) from the PF (in iNGENIOUS AI-driven Smart Port and Shipping Platform), it sends a request to the DVL, which fetches data from the PF using the interface *FetchGateEvents*, as shown in workflow in Figure 26Figure 25:

/FetchGateEvents: Returns all the pseudonymized events in a specific time period, i.e. from 'startdate' to 'enddate'. Besides it is possible to fetch GateIn events only or GateOut events only if the optional parameter 'gate' is used, see Figure 26 and Figure 27Figure 27Figure 27. In case 'gate' parameter is not specified both GateIn and GateOut events in specified time period are returned, see Figure 28Figure 28 and Figure 29Figure 29.

```
https://172.26.1.115:8081/FetchGateEvents/
{
        'startdate' : '2022-05-16T00:00:00.0000000+01:00',
        'enddate' : '2022-05-20T00:00:00.0000000+01:00',
        'gate' : IN
}
```

FIGURE 26: FETCHRECORD USING INTERVAL TIME AND SELECTED EVENTS

the result is:


```
[
 {
        "GATE": "IN",
        "users":
                {
                        "originatorName": "Port of Livorno",
                        "originatorId": "ITLIV",
                        "eventSubmissionTime8601": "2021-05-17T17:10:00.5774856+01:00",
                        "transportEquipmentId": "",
                        "transportEquipmentRef": "",
                        "equipmentNumber": "MSKU2871249",
                        "carrierBookingNumber": "",
                        "billOfLadingNumber": "AK000601",
                        "eventOccurrenceTime8601": "2021-05-17T17:08:02.3620000+01:00",
                        "transportationPhase": "Import",
                        "location": "ITLIV",
                        "vehicleId": "7aa6c046029f0c68d20825bdb138ae62045faa1d",
                        "vehicleName": "Truck",
                        "fullStatus": "F",
                        "terminal": "LRN",
                        "voyageId": "19466"
                }
 }
]
```


FIGURE 27: EXAMPLE: RETRIEVED INFO USING FETCHRECORDS WITH SELECTED EVENT (GATEIN)

If the function is invoked without any preferred event:

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
https://172.26.1.115:8081/FetchGateEvents/
{
        'startdate' : '2022-05-16T00:00:00.0000000+01:00',
        'enddate' : '2022-05-20T00:00:00.0000000+01:00'
}
```

FIGURE 28: FETCHRECORD USING INTERVAL TIME ONLY

Both the GateIn and GateOut events are retrieved:


```
[
 {
        "GATE": "IN",
        "users":
                {
                        "originatorName": "Port of Livorno",
                        "originatorId": "ITLIV",
                        "eventSubmissionTime8601": "2021-05-17T17:10:00.5774856+01:00",
                        "transportEquipmentId": "",
                        "transportEquipmentRef": "",
                        "equipmentNumber": "MSKU2871249",
                        "carrierBookingNumber": "",
                        "billOfLadingNumber": "AK000601",
                        "eventOccurrenceTime8601": "2021-05-17T17:08:02.3620000+01:00",
                        "transportationPhase": "Import",
                        "location": "ITLIV",
                        "vehicleId": "7aa6c046029f0c68d20825bdb138ae62045faa1d",
                        "vehicleName": "Truck",
                        "fullStatus": "F",
                        "terminal": "LRN",
                        "voyageId": "19466"
                }

 },
  {
        "GATE": "OUT",
        "users":
                {
                        "originatorName": "Port of Livorno",
                        "originatorId": "ITLIV",
                        "eventSubmissionTime8601": "2021-05-18T17:23:01.123451+01:00",
                        "transportEquipmentId": "",
                        "transportEquipmentRef": "",
                        "equipmentNumber": "MSKU3271229",
                        "carrierBookingNumber": "",
                        "billOfLadingNumber": "AK000601",
                        "eventOccurrenceTime8601": "2021-05-18T17:10:07.3411200+01:00",
                        "transportationPhase": "Import",
                        "location": "ITLIV",
                        "vehicleId": "5ba4a0d20825bdb138ae62045f22321cbaaa1d356",
                        "vehicleName": "Truck",
                        "fullStatus": "F",
                        "terminal": "LRN",
                        "voyageId": "15432"
                }

 }
]
```


FIGURE 29: EXAMPLE: RETRIEVED INFO USING FETCHRECORDS WITHOUT SELECTED GATE EVENT

4.6.5 Auditor

The auditor deals with the elimination of data that have an expired retention period. According to the GDPR, it is important to keep the data for a limited

Co-funded by the Horizon 2020 Framework Programme of the European Union

period of time (retention period), depending on the use that determine its duration. Every night the auditor removes (clearing field) all the pseudonyms in each event which retention period has expired.

The default retention period is set to 5 years, but it can be modified at instantiation. The Auditor module is scope of next deliverable D5.3.

4.7 DVL-Cross-DLT layer integration

The integration between DVL and TrustOS will be addressed by creating an intermediate software component that will act as a "bridge" between the two of them, see Figure 30. As DVL and TrustOS are software components that exposes their functionalities through a REST API, both are passive components, and a "bridge" application is necessary.

This "bridge" will extract the data from the DVL making the HTTP request in a long polling way, making consecutive requests after a period of time looking for new data in the DVL. This period of time is yet to be defined.

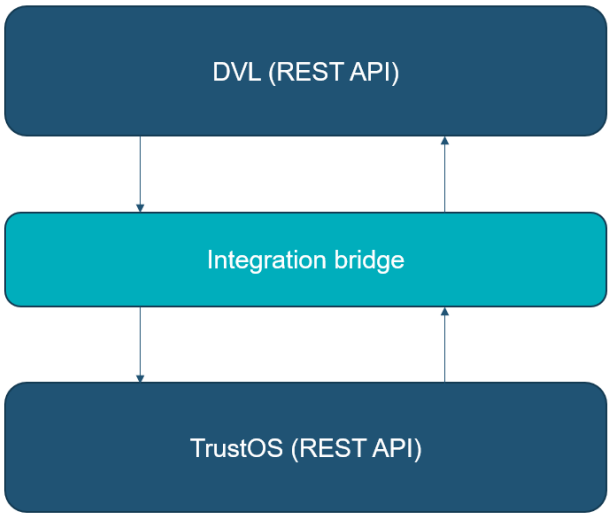


FIGURE 30: INTERMEDIATE COMPONENT (BRIDGE) FUNCTIONAL DIAGRAM.

Once the "bridge" retrieves new data from DVL it will transform this data from the DVL structure to the TrustOS platform required structure to create new digital assets. These digital assets have three main fields:

- AssetID. This field identifies the digital asset uniquely.
- Data. This field contains any data that does not vary during the lifetime of the asset.
- Metadata. This field contains data that may vary or not during the lifetime of the asset. This file can be modified by calling the update method and providing only the new data for this field. New information is not overwritten but added to the asset history to have full traceability at the end of the process.

After transforming the data, a new digital asset will be created or updated if it already exists.

The creation of the TrustPoints of these data is still pending of decide who or when they are created.

Co-funded by the Horizon 2020 Framework Programme of the European Union

5 Cross-DLT Layer

In this chapter, APIs' development and integration activities between different sets of available DLTs and the cross-DLT layer are described and discussed. After an introduction to Distributed Ledger Technologies (DLTs), the common API to be implemented and supported by all the DLT connectors is detailed. Finally, for each available DLT in iNGENIOUS project, the integration approach as well as the development activities status is reported.

5.1 DLT Introduction

Distributed Ledger Technology is a digital system for storing information in an immutable, distributed, and decentralized form. Its infrastructure, protocols and management are performed by multiple entities, and nobody controls them unilaterally. There is a difference between distributed and decentralized approaches and Vitalik Buterin explained it in his post "The Meaning of Decentralization": "distributed means not all the processing of the transactions is done in the same place" [10], whereas "decentralized" means that not one single entity has control over all the processing", Figure 31Figure 31Figure 31. DLT meets both of these characteristics.

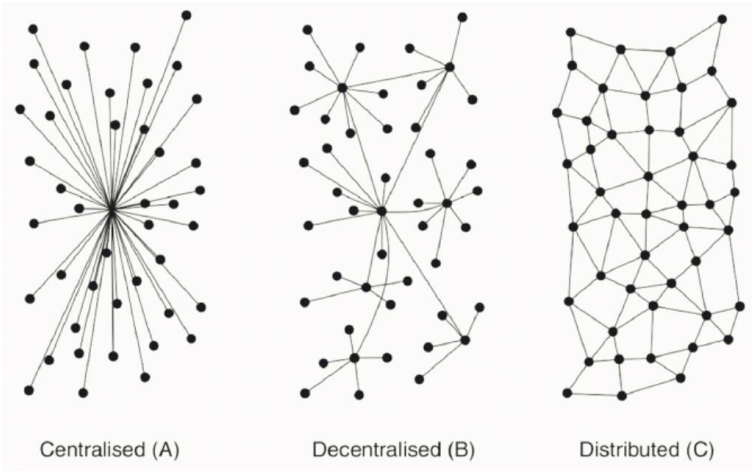


FIGURE 31: (A) CENTRALIZED, (B) DECENTRALIZED , (C) DISTRIBUTED NETWORKS, [21]

Thanks to these aspects immutability appears in the sense that altering information that is replicated in different places is computationally very complex and it is even more difficult if each of these copies belongs to a different entity. Another strength of the technology is that it uses cryptography to store information securely. Blockchain is a type of DLT but with its own characteristics. Blockchain uses hashes to group a set of transactions in a block which also contains a reference to the previous one thus forming a block chain. Blockchain is a type of DLT but not all DLTs are Blockchains.

From the DLT structure perspective in the context of iNGENIOUS project, we can distinguish two approaches:

1. Blockchain – a chain of blocks linked with each other using hash, each block containing multiple transaction and maintaining the DLT specific structure

Co-funded by the Horizon 2020 Framework Programme of the European Union

2. DAG – a direct acyclic graph structure (Tangle) where each node represents a micro-transaction. There is no blockchain and the integrity of the tangle is maintained using combination of coordinator server and the small PoW in each transaction (node). Moreover, the Tangle is intended to be highly scalable, without fees and allowing near-instant transfers. The interconnectedness of Tangle's architecture doesn't require total verification across the ledger. Instead, all parties are verifying simultaneously and, as a result, the energy and time required to complete transactions are shortened. In addition, Tangle's verification process purports to ensure that there are no duplicate transactions that would lead to double spending issue.

There are different types of DTLs systems from the access rights perspective, Privacy, performance and security apply to permissioned networks. Whereas security and transparency apply to public networks.:

- Permissionless/public distributed ledger: Anyone can access and validate the information without the need for authorization from a central entity. Its nature is usually open source.
- Permissioned distributed ledger: The reading and validation of information must be authorized by certain entities or by the network.
- Hybrid distributed ledger: Leverage the advantages of public and permissioned ledgers by combining privacy, performance, security, and transparency.

Every public DLT system is governed by a consensus mechanism which is the set of rules to manage the validation of the world state of the blockchain ledger. There are many different consensus mechanisms. We will focus on the following ones.

- The first one to appear was Proof of Work (PoW) with Bitcoin publication [7]. The immutability of the ledger in the PoW is achieved using probabilistic proof that the amount of energy was consumed in order to validate the block of transactions. The blockchain version with the greatest work done is the one considered “correct”. This mechanism uses the external resource to influence internal state of the system. Currently the amount of energy consumed by the largest PoW solution is comparable to the energy production of the typical developed nation state [11].
- Proof of Stake - In this case, validators – the nodes that can generate blocks – needs to lock some amount of native token in given PoS solution in order to participate in validation. The feature of this solution is that there is no additional energy expenditure other than just signing and verifying transactions. There is no external influence on the system other than users decisions about signing transactions or performing validation. This kind of system requires validators to obtain the amount of native tokens from the existing owners.
- Proof of Authority - The last consensus mechanism is typical of the permissioned distributed systems. Here a set of nodes are selected to be the validators of the information. The consortium relies on the state of the DLT signing the blocks.
- The IOTA consensus mechanism combines a binary voting protocol (Fast Probabilistic Consensus - FPC) and a virtual voting protocol or approval

Co-funded by the Horizon 2020 Framework Programme of the European Union

(Approval Weight – AW) which represents the weight of branches (and messages), similar to the longest chain rule in Nakamoto consensus. In FPC, a node simply asks other nodes about their opinion related to a transaction. However, instead of selecting a leader based on a puzzle (PoW) or stake (PoS), it allows every node to express its opinion by simply issuing any message and attaching it in a part of the Tangle (based on its initial opinion on messages and possibly utilizing the like switch to express its opinion on branches).

Most consensus mechanisms try to achieve the immutability of the ledger, however it is always some spectrum that is based on probability or human decisions. The DLT design tires to maximize effort that must be done in order to rewrite transaction history.

5.2 DLTs API definition

In order to have a common way of calling to APIs regardless of the DLT technology, a common API have been defined to fulfil all the necessary requirements to store the TrustPoints in any Distributed Ledger.

The different connectors implemented for the CrossDLT layer implements this API, then the TrustOS platform will act as a bridge and distribute the information of the TrustPoints among the different ledgers making a unique request to the DLT connectors compliant with the common API.

To achieve this, three methods have been defined that all the connectors must implement and be compatible with (minimum requirement).

The first method is a read method of the information stored. This method needs some kind of identifier that matches with some transaction or a tuple of the information asset identifier with a timestamp. It is implemented as HTTP GET request that need the following parameters:

- Transaction ID: This information will be stored in the TrustOS platform and comes from the response of a storing request. If this parameter is used, no other is needed as the information is stored in a unique transaction of the specific ledger.
- Asset ID and timestamp: This tuple identifies the evidence information related to some digital asset stored in the TrustOS platform and that it has been stored previously on a specific ledger. If these parameters are used, there is no need to use the transaction identifier.

The response of this request may be observed in the following Figure 32Figure 32. It is a JSON data that contains the information about the evidence (TrustPoint) stored in the ledger.


```
{
    "assetID": "98989981137",
    "timestamp": "1567594895",
    "merkleRootHash": "NGz693K+cqYasFNbcfhEr+Ziy/Y/jsfOt0FNKcqYa5E=",
    "prevTrustPointHash": "Vz3ZFr+wTOt0FNbcfhEr+Ziy/Y/jsfNGz693KcqYa5E="
}
```


FIGURE 32: JSON RESPONSE INCLUDING THE STORAGE EVIDENCE.

Co-funded by the Horizon 2020 Framework Programme of the European Union

The second method is a writing method that makes it possible to store the TrustPoint information in the ledger of the connector that is implementing this API. This method receives in the body a JSON containing the TrustPoint information, which is the same as the response of the previous GET request. This method is implemented as HTTP POST request that accepts JSON data in the body. Its response is also a JSON containing the following data:

- Transaction ID: this is the identifier of the transaction in which the information has been stored. This is unique for each DLT technology.
- Smart Contract address: this field is optional, as not every DLT used in this project allows the usage of smart contract. If smart contract can be used, the response may contain the address of the smart contract used to store the TrustPoint information:


```
{
    "txID": "0x6ae1fc4bab1f2cdddd9d785513ec9b51fc8d24258d9bfaf49be4571d519c448d",
    "smartContractAddress": "0x6ae1fc4bab1f2cdddd9d785513ec9b51fc8d24258d9bfaf49be4571d519c448d"
}
```


FIGURE 33: RESPONSE INCLUDING THE ADDRESS OF THE SMART CONTRACT.

The third and last method is a method that verifies that the information stored in the TrustOS platform matches the information stored in another ledger like Ethereum or Bitcoin. For that, it is implemented as a POST HTTP request that needs a body containing in JSON format the following information:


```
{
    "txID": "0x6ae1fc4bab1f2cdddd9d785513ec9b51fc8d24258d9bfaf49be4571d519c448d",
    "assetID": "98989981137",
    "timestamp": "1567594895",
    "merkleRootHash": "NGz693K+cqYasFNbcfhEr+Ziy/Y/jsfOt0FNKcqYa5E=",
    "prevTrustPointHash":"Vz3ZFr+wTOt0FNbcfhEr+Ziy/Y/jsfNGz693KcqYa5E="
}
```


FIGURE 34: ATTRIBUTES OF THE RESPONSE FOR THE VERIFY METHOD.

What every connector does when it receives this information is to retrieve the TrustPoint indicated in this response from the DLT and compare both of them. If it matches, it can be considered verified, and the response would be something like the following:


```
{
    "txID": "0x6ae1fc4bab1f2cdddd9d785513ec9b51fc8d24258d9bfaf49be4571d519c448d",
    "assetID": "98989981137",
    "timestamp": "1567594895",
    "merkleRootHash": "NGz693K+cqYasFNbcfhEr+Ziy/Y/jsfOt0FNKcqYa5E=",
    "prevTrustPointHash":"Vz3ZFr+wTOt0FNbcfhEr+Ziy/Y/jsfNGz693KcqYa5E=",
    "verified": true
}
```


FIGURE 35: POSITIVE RESPONSE USING VERIFY METHOD.

A new field ("verified") that says that the information has been checked against the specific DLT is added.

Co-funded by the Horizon 2020 Framework Programme of the European Union

5.3 IOTA integration

In this chapter the setup and configuration of the IOTA private tangle, as well as the implementation of the main interface to be used for the interaction between TrustOS component and IOTA IRI node, are described.

5.3.1 IOTA Tangle deployment and setup

In order to deploy a private IOTA node for iNGENIOUS development and integration activities, two separate virtual machines have been instantiated, hosting a private IOTA tangle as well as a message broker for the management of the main requests coming from TrustOS (namely InteroperaChain), with the following technical specifications:

VM#1 (Private Tangle):

- CPU Intel XEON E5-2680 with 2 cores.
- 4GB of RAM.
- 30GB of storage space.
- Ubuntu 20.04 Server as the main operating system.

VM#2 (InteroperaChain):

- CPU Intel XEON E5-2680 with 2 cores.
- 4GB of RAM.
- 50GB of storage space.
- Ubuntu 20.04 Server as the main operating system.

Despite of the legacy version of the current v.1.5 IOTA protocol (Chrysalis) it is expected to move to v.2.0 (Coordicide) during the lifetime of the project. In order to perform IOTA Private Tangle deployment, we relied on the following standard specifications [17]:

- IOTA Hornet node software (providing IOTA full node capabilities).
- Wallet built for Chrysalis.

As far as the technical specifications of the Tangle are concerned, we used a default configuration available on GitHub [18]. Such of settings are fully compatible with a DevNet and no relevant issues are expected to arise in case of the MainNet (this will be further assessed during the project lifetime). By means of this configuration, we achieved a permissioned IOTA network (DevNet) orchestrated by the coordinator (managed by CNIT) for the transaction's validation.

5.3.2 IOTA-TrustOS API implementation

According to the technical specifications of a common interface for accessing a generic DLT, described in Section 5.2 (namely Cross-DLT Common Interface), we relied on OpenAPI standard for the IOTA API definition and implementation (using Python as a target language for compiling). As a matter of fact, the OpenAPI Specification (OAS) [19] defines a standard, language-agnostic interface to RESTful APIs which allows to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. An OpenAPI definition can

Co-funded by the Horizon 2020 Framework Programme of the European Union

then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases.

According to such advantages, the API implementation for interacting with IOTA IRI Node is based on the following supported requests:

- .../api/v1/trustpoint: **gets** the Trustpoint for an assetID from IOTA DLT;
- .../api/v1/trustpoint: **posts** the Trustpoint information into the IOTA DLT;
- .../api/v1/trustpoint/verify: **posts** and verifies the received Trustpoint information against the one stored in the IOTA DLT.

The sequence diagrams, describing a high-level interaction between involved components (TrustOS, InteroperaChain and IOTA IRI Node) are already documented in D5.1 - Key technologies for IoT data management benchmark.

In order to provide a description of the interface as well as a way to perform tests, a Swagger-compliant interface (an Interface Description Language for describing RESTful APIs expressed using JSON [20]) has been released, as depicted in the figure below (available at [16]):

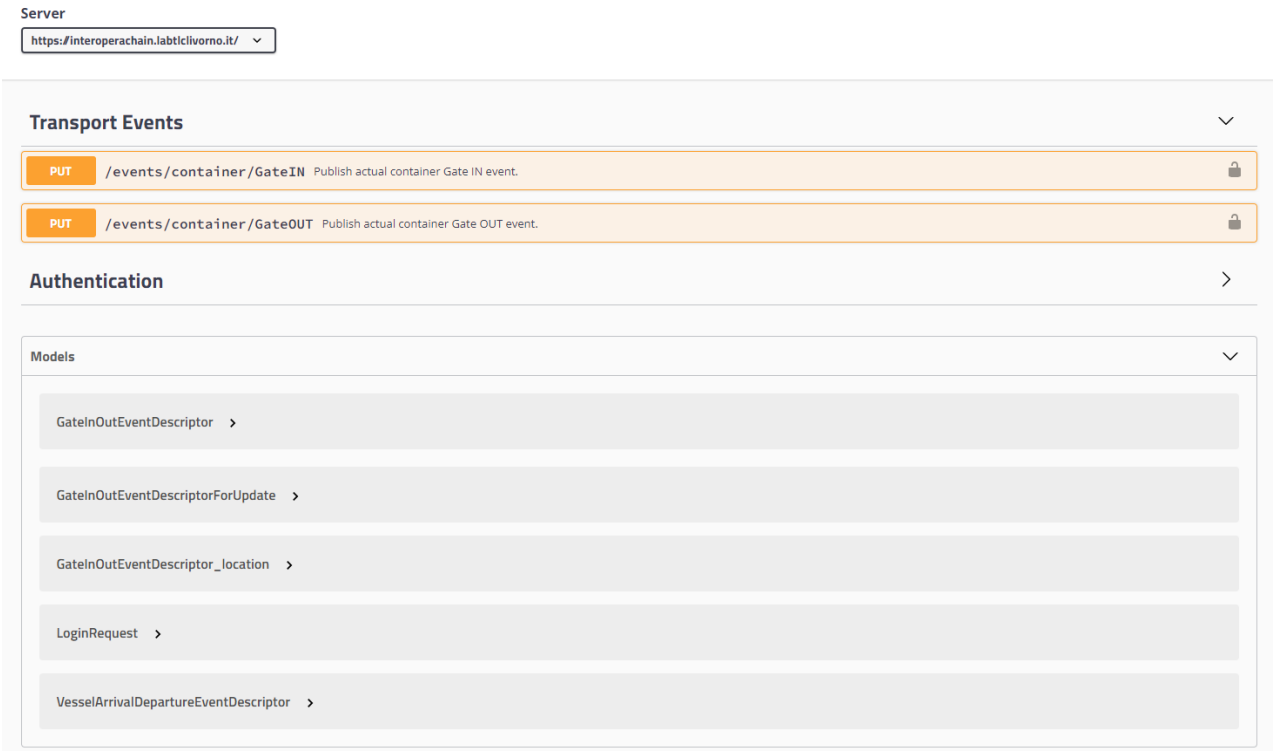


FIGURE 36: SWAGGER-COMPLIANT INTERFACE TO INTERACT WITH IOTA NODE

The following figure shows a set of supported schemas and models:

Co-funded by the Horizon 2020 Framework Programme of the European Union

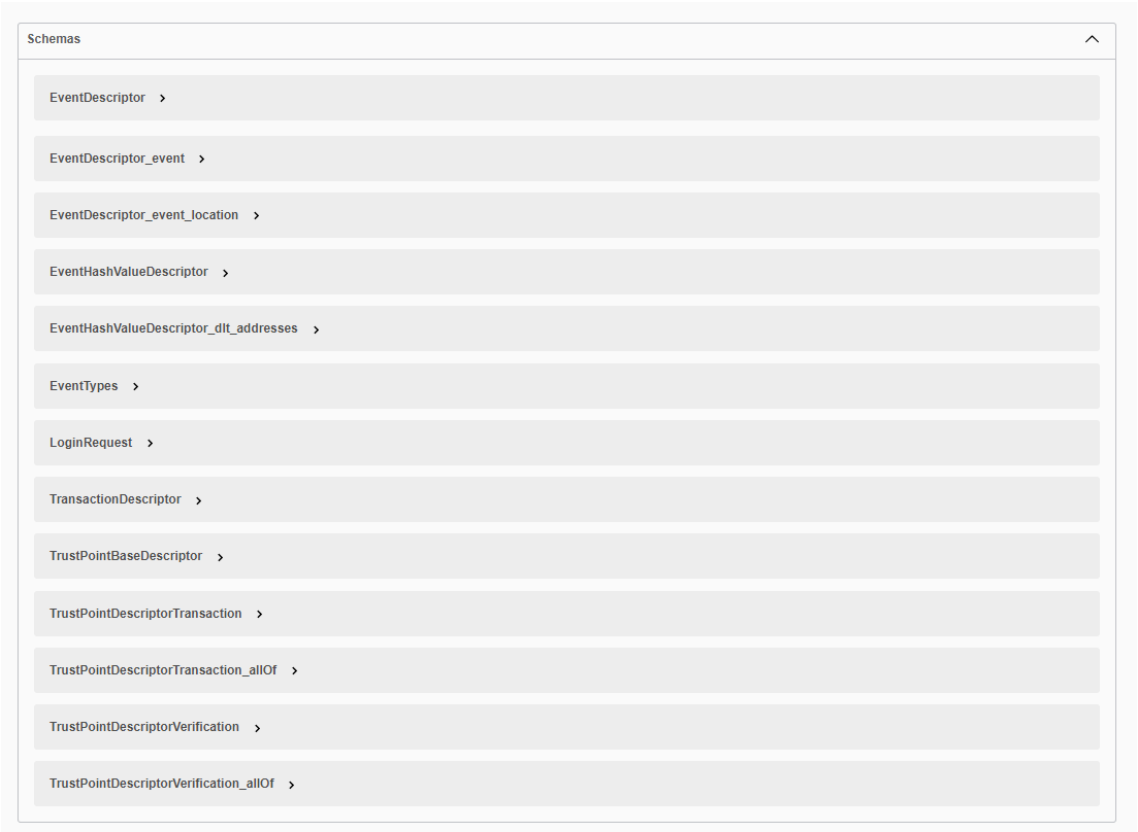


FIGURE 37: SUPPORTED SCHEMAS/MODELS.

As far as the authentication is concerned, we adopted JSON Web Token internet standard (JWT) [15], allowing data consumers (TrustOS) to perform logging operation to service APIs, see Figure 38Figure 38.

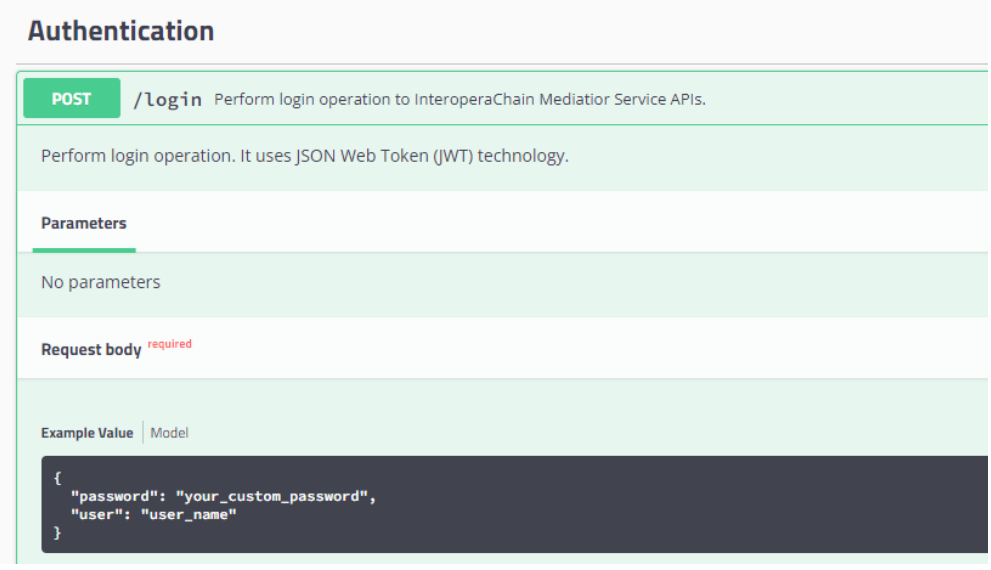


FIGURE 38: JWT-BASED AUTHENTICATION METHOD FOR LOGIN

Moreover, in order to interact with the private Tangle and explore the history of ingested transactions a client with a graphic user interface is adopted, see Figure 39. The client is based on Hornet - Community Driven IOTA Node and currently it is used for offline testing purposes to make sure the implemented interfaces are properly working.

Co-funded by the Horizon 2020 Framework Programme of the European Union

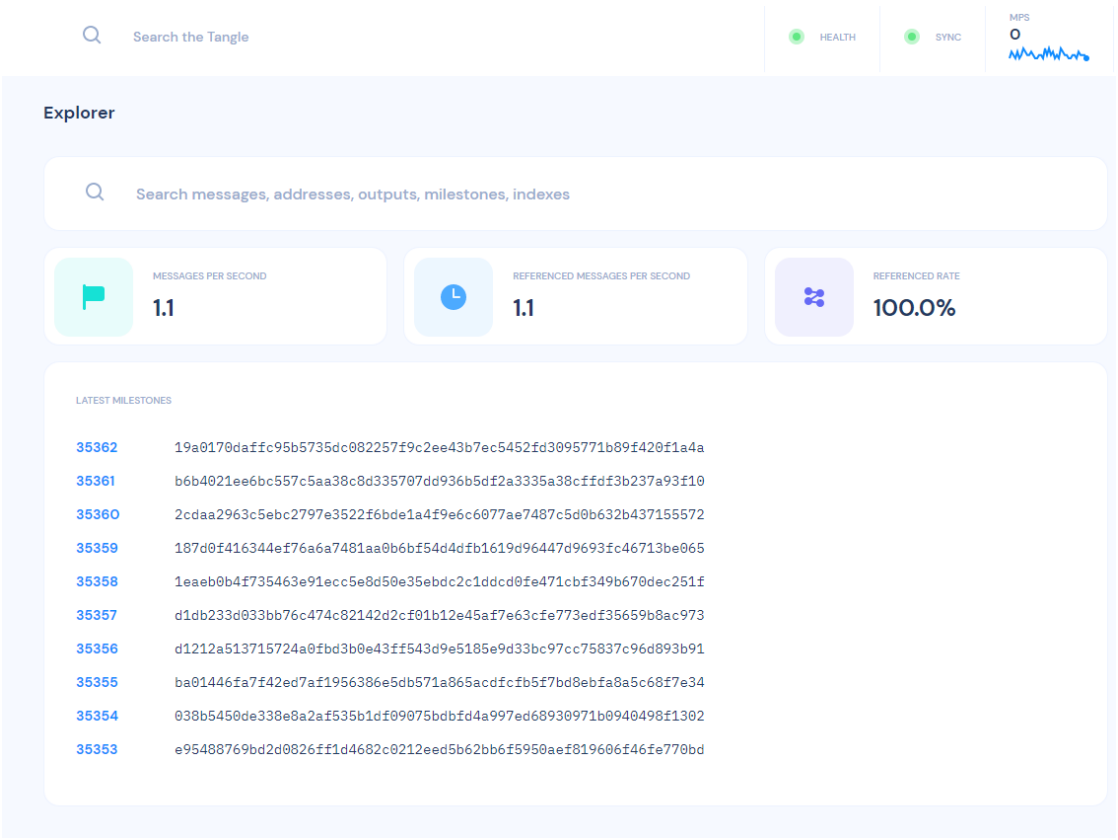


FIGURE 39: TANGLE GRAPHIC USER INTERFACE

The interface is under testing and it is expected to be completed by the end of M18 of iNGENIOUS project. Further technical implementation details will be included in D5.3 - Final iNGENIOUS data management platform (M28).

5.4 Hyperledger Fabric integration

For enabling the integration of Hyperledger Fabric into iNGENIOUS architecture, FV is developing a dedicated web application designed to store and expose data related to GateIn and GateOut events at the Port of Valencia.

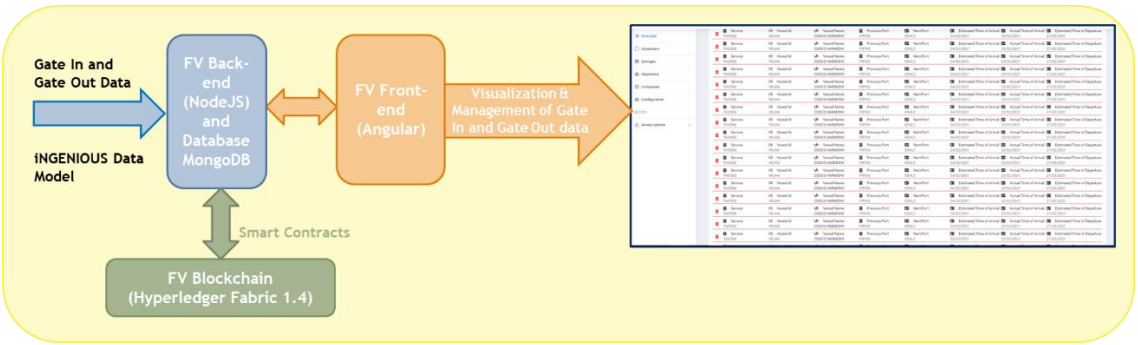


FIGURE 40: FV APPLICATION ARCHITECTURE

FV application is composed of two different modules: the back-end and the front-end. iNGENIOUS back-end is developed in Node.js programming language and follows a REST API architecture where data can be read and written thanks to the use of GET, POST, PUT and DELETE methods. The back-end application has a connection with a MongoDB database for storing the data model and other metadata (i.e. user and configuration info) needed for

Co-funded by the Horizon 2020 Framework Programme of the European Union

running the application. On the other hand, iNGENIOUS front-end application is developed by combining Typescript and Angular, and allows the user to visualize and perform all the operations previously defined at the back-end level. The data model used to store data in iNGENIOUS application has been reused from the Tradelens-based data model defined by CNIT (See Section 4).

A smart contract (running on Hyperledger Fabric network) enables the reading, writing, updating and deleting of GateIn and GateOut data. The functions and methods defined in the Smart Contract are similar to the ones defined in the back-end application. To store and read the data in the blockchain, the front-end module allows the user to decide if GateIn and GateOut data is stored in blockchain or not.

To enable the integration of Hyperledger Fabric network with the rest of DLTs, FV will implement an API following the DLT API definition described in Section 5.2, where the connection with other DLTs is based on the exchange of information related to TrustPoints. The three methods described in the API definition will be implemented as part of the API defined in the FV back-end project. Additionally, since TrustOS is based on a Hyperledger Fabric network, a more simplified way of interaction will also be further explored. An example of the block diagram for enabling the interaction with the rest of DLTs is shown below:

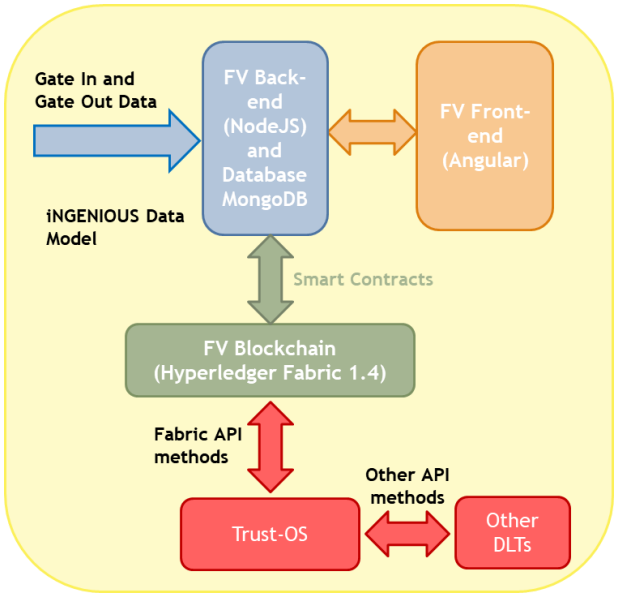


FIGURE 41: FV APPLICATION ARCHITECTURE INTEGRATED WITH TRUST-OS

5.5 Bitcoin integration

To integrate Bitcoin in the iNGENIOUS ecosystem, a dedicated web application and web API was created. The application is divided into two layers. The top layer, developed with Python and Django framework, provides the API for basic operations related to storing and verifying data on the Bitcoin node. The application uses typical architecture with model-view-controller design pattern. Models provide the object-relational mapping to entities in the SQLite database. The views are responsible for rendering the appropriate HTML code. The controller handles the requests, responses, and sets database connections. The lower layers of the application, created with C++, provide the wrapper for the functionality provided in the bitcoin command-line interface,

Co-funded by the Horizon 2020 Framework Programme of the European Union

also functionalities for generating raw transactions that contains data. Summary of the architecture is shown in the following figure:

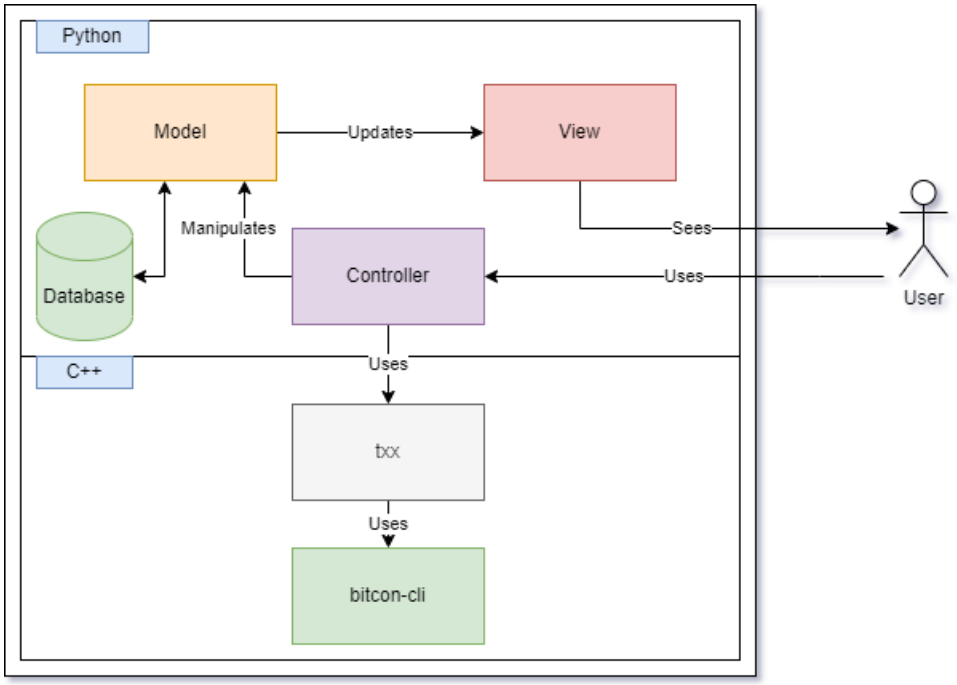


FIGURE 42: ARCHITECTURE OF THE APPLICATION INTEGRATING BITCOIN

5.5.1 Hardware

The application is hosted on the server provided by PJATK. Besides the application, the server is also hosting a full Bitcoin node. The server has the following specifications:

- CPU host Intel Xeon with 12 cores.
- 24GB of RAM.
- 3TB of storage space.
- Ubuntu 20.04 server as the main operating system.

5.5.2 Software Endpoints

The following four different endpoints are to be provided by the API:

1. /api/v1/trustpoint: POST request for storing the data in the Bitcoin node. The user needs to provide the assetId, timestamp, merkleRootHash, prevTrustPointHash, and optionally the txfee in JSON format. An example of the JSON is provided in the following code snippet:

```
{
"assetID": 98989981137,
"timestamp": 1567594895,
"merkleRootHash":"NGz693K+cqYasFNbcfhEr+Ziy/Y/jsfOt0FNKcqYa5E=",
"prevTrustPointHash":"Vz3ZFr+wTOt0FNbcfhEr+Ziy/Y/jsfNGz693KcqYa5E=",
}
```

FIGURE 43: ARCHITECTURE OF THE APPLICATION INTEGRATING BITCOIN

Co-funded by the Horizon 2020 Framework Programme of the European Union

If no fee is specified, the adaptive fee calculation will be utilized. The response has the JSON format specifying the status of the operation, transaction id, transaction fee, and how many bitcoins are left after the transition. An example of the JSON response is provided in the following snippet:

```
{
"status": "success",
 "txID": "f69ed4288ce50fe1f071a7af755f13cf8079a519fc37fbcc26dbe8784adcddb3",
"txfee": 1e-05,
"bitcoin_left": 0.00428
}
```

FIGURE 44: ARCHITECTURE OF THE APPLICATION INTEGRATING BITCOIN

2. */api/v1/trustpoint?txID=<tiD>?assetID=<assetID>?timestamp=<timestamp>?* get request which checks if the transaction with specified txID has been registered in the application. The assetID and timestamp as optional query params are planned - the implementation is in progress.
3. /api/v1/trustpoint/verify POST according to the common API provided in chapter 5.2, returns verified if the transaction is present in the database./api/v1/doc/: returns documentation of the API. Documentation is in preparations.
4. /admin/: Provides admin panel for reviewing and searching the transaction. Example view of the admin panel is provided in the figure below.

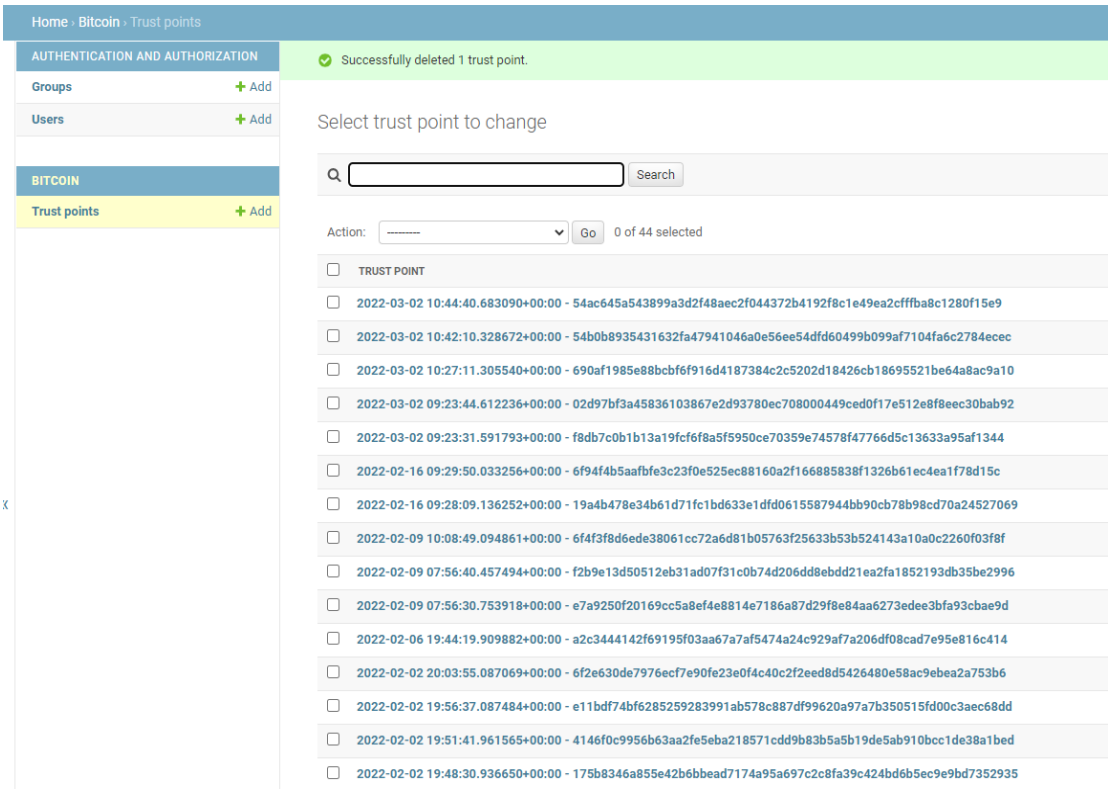


FIGURE 45: ADMIN PANEL

Co-funded by the Horizon 2020 Framework Programme of the European Union

At the moment, the underlying methods for verification, storing and reading Trustpoints from the Bitcoin DLT are implemented, however only one TrustPoint endpoint is functional - storage. The other endpoints are in active development and testing. The admin panel allows for the verification of the database state that contains all of the Trustpoints handled by the API. The API will follow the specification described in the chapter 5.2.

All information provided to the Trustpoint endpoint are also stored in the internal SQL database. The admin panel is intended to ease querying and reviewing the stored data. It is also provides user management functionality. Currently the API works on the Bitcoin Testnet, however it is easy to set it up to work on the Bitcoin Mainnet. Given a transaction ID that is returned by the /trustpoint/ endpoint of the API, one can independently verify the details of the transaction using a full Bitcoin node or any publicly available block explorer.

5.6 Ethereum integration

Ethereum is an open source public blockchain platform that allows applications to be developed and executed in a decentralized manner. The main objective of Ethereum is to provide a decentralized environment that allows to execute business logic or rules. Thus, a user could send a certain amount of his or her balance to another user when a series of conditions are met, for example on the first day of each month.

The potential of Ethereum is enormous, as it is possible to carry out automatic operations without the need for third parties based on any programmable condition. This is possible thanks to the fact that the programming language it offers is Turing complete, that is, it has the capacity to solve any type of computational problem. The business logic that the Ethereum network can execute is implemented within computer programs called smart contracts.

The way to participate in the network is through a node or client. A node is a client software implementation that participates in the maintenance of the chain and the network by verifying the transactions of each block.

Ethereum clients implement the remote procedure call protocol (JSON-RPC). It is a lightweight, stateless protocol that defines a set of structures and rules for making requests to servers (in this case Ethereum clients) implementing the same JSON-RPC protocol. Through the integration of Ethereum it is possible in the future to use not only this platform but also other Ethereum-compatible blockchains such as Hyperledger Besu, Polygon, GoChain, etc.

The current scope is to register public evidence of the information system in order to leverage the transparency and the decentralization of the public blockchains. Thus, any final user has the ability to check the validity of the information. The concept that represents this type of evidence is what is called “trustpoint” (already explained in deliverable 5.1).

Ethereum integration involves the following developments:

- Infrastructure: There are different ways to interact with an Ethereum blockchain, you can deploy your own node, or you can access through a third-party node. Infura offers this third-party node as a service to avoid the complexity of administration and management of a self-hosted node. The used Ethereum Network is Kovan Testnet [9]. The goal using testnets is to

Co-funded by the Horizon 2020 Framework Programme of the European Union

avoid the costs associated with registering information in Ethereum due to high fees.

- Web3 service: a microservice to facilitate the compilation, deployment and interaction with smart contracts.
- Smart Contract: to store trust points evidences in Ethereum an smart contract has been implemented using Solidity as a programming language. Main methods are described below:
  - setEvidence: Method to store a trust point evidence in the network. It stores the trust point identificator, the transactions merkle root hash and the previous trust point, all linked to a timestamp.
  - getEvidence: Method to get a trust point evidence from the network.

An instance of this smart contract has been deployed in Kovan Testnet in order to be used in testing phase to validate the registration of public evidence through all the DLT connectors.

Co-funded by the Horizon 2020 Framework Programme of the European Union

6 Conclusions

This deliverable has described all technical components of Data Management framework with focus on Data Virtualization Layer and Cross-DLT layer as the main components to address both the cross-domain and cross-platform interoperability. The data aggregation approach between different M2M platforms and available data sources has been presented and the first implementation has been detailed with respect to iNGENIOUS use cases.

On top of this interoperable layer, we described the IoT application layer which focuses mainly on applications related to port operations. These included AI algorithms for predicting traffic rates and congestion at the port based on heterogeneous data sources along the maritime supply chain, and a platform for remotely operating automated guided vehicles in the port area.

Subsequently, the underlying layer of Data Management framework has been described in terms of available set of M2M platforms and data sources to be integrated (e.g., Mobius OneM2M, Eclipse OM2M, PISystem and Symphony), as well as in terms of hardware components to be used in a limited set of use cases (e.g., Smart IoT Gateway).

Finally, we addressed the cross-DLT interoperability by describing a common interface which is expected to allow interacting with available DLTs (e.g., Bitcoin, IOTA, Ethereum and Hyperledger Fabric) based on a centralized approach for the data orchestration.

The implemented solution represents the first release of the interoperable layer and will be used for platforms integration and validation by means of use cases described in deliverable D5.1. The final version of the Data Management framework will be released as deliverable D5.3 in M28, including all the improvements required to fulfil the requirements and the KPIs defined in D2.1.

Co-funded by the Horizon 2020 Framework Programme of the European Union

7 References

[1] Eclipse Foundation, Eclipse OM2M: https://www.eclipse.org/om2m/
[2] https://github.com/teiid/
[3] https://www.wildfly.org/
[4] TradeLens | Digitizing Global Supply Chains: https://www.tradelens.com/
[5] Micktrack Communication Protocol, https://www.mictrack.com/downloads/protocols/Mictrack_Communication_Protocol_For_MT825_V2.0.pdf
[6] RabbitMQ, https://www.rabbitmq.com
[7] Bitcoin Whitepaper by Satoshi Nakamoto: bitcoin.pdf
[8] https://github.com/IoTKETI/Mobius
[9] TESTNET Kovan (KETH) Blockchain Explorer (etherscan.io)
[10] The Meaning of Decentralization. "Decentralization" is one of the words... | by Vitalik Buterin | Medium
[11] https://ccaf.io/cbeci/index
[12] Symphony, https://www.nextworks.it/en/products/symphony
[13] https://www.etsi.org/committee/smartm2m
[14] https://www.osisoft.com/pi-system
[15] https://jwt.io
[16] https://interoperachain.labtlclivorno.it/ui/
[17] https://wiki.iota.org/hornet/getting_started/private_tangle
[18] https://github.com/gohornet/hornet/blob/develop/private_tangle/config_private_tangle.json
[19] https://www.openapis.org
[20] https://swagger.io
[21] https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274

Co-funded by the Horizon 2020 Framework Programme of the European Union