Federating IoT and cloud infrastructures to provide scalable and interoperable Smart Cities applications, by introducing novel IoT virtualization technologies

EU Funding: H2020 Research and Innovation Action GA 814918; JP Funding: Ministry of Internal Affairs and Communications (MIC)

Deliverable 5.2
Pilot Integration - First Release

Deliverable Type:	Report
Deliverable Number:	5.2
Contractual Date of Delivery to the EU:	30.06.2020
Actual Date of Delivery to the EU:	30.06.2020
Title of Deliverable:	Pilot Integration - First Release
Work package contributing to the Deliverable:	WP5
Dissemination Level:	Public
Editor:	Antonio F. Skarmeta (OdinS), Kenichi Nakamura (PAN)
Author(s):	Juan A. Martinez, Juan A. Sanchez, Antonio Skarmeta (OdinS), Kenji Kanai (WAS), Andrea Detti, Ludovico Funari (CNIT), Kenichi Nakamura (PAN), Tetsuya Yokotani (KIT), Hiroaki Mukai (KIT), Gilles Orazi, Ahmed Abid (EGM)
Internal Reviewer(s):	Giuseppe Tropea (CNIT)
Abstract:	This deliverable describes the first release of the integration of the different pilots developed within the scope of Fed4IoT.
Keyword List:	Pilot; Integration; GitHub; Kubernetes

Disclaimer

This document has been produced in the context of the EU-JP Fed4IoT project which is jointly funded by the European Commission (grant agreement n 814918) and Ministry of Internal Affairs and Communications (MIC) from Japan. The document reflects only the author's view, European Commission and MIC are not responsible for any use that may be made of the information it contains

Table of Contents

List of Figures

List of Tables

Abbreviations

Abbreviation	Definition
ADN	Application Dedicated Node
AE	Application Entity
AIMD	Additive Increase/Multiplicative Decrease
AKS	Azure Kubernetes Service
API	Application Programming Interface
ASM	Adaptive Semantic Module
ASN	Application Service Node
AWS	Amazon Web Services
CBPF	Cross-border Person Finder
CIM	Context Information Management
CSE	Common Services Entity
ETSI	European Telecommunications Standards Institute
FIB	Forwarding Information Base
GE	Generic Enabler
GDPR	General Data Protection Regulation
HTTP	HyperText Transfer Protocol
ICN	Information Centric Networks
ICT	Information and Communication Technologies
IN	Infrastructure Node
IP	Internet Protocol
ISG	Industry Specification Group
JSON	JavaScript Object Notation
MANO	MAnagement and Network Orchestration
MMG	Morphing Mediation Gateway
MN	Middle Node
MQTT	Message Queue Telemetry Transport
NGSI	Next Generation Service Interfaces Architecture
NGSI-LD	Next Generation Service Interfaces Architecture - Linked Data
NSE	Network Service Entity
OMA	Open Mobile Alliance
PIT	Pending Interest Table
PPP	Public-Private Partnership
RDF	Resource Description Framework
RPZ	Regulated Parking Zone
REST	Representational State Transfer
SDK	Software Development Kit
TCP	Transmission Control Protocol
TM	Topology Master

TN	Task Name
TV	ThingVisor
UML	Unified Modeling Language
URI	Uniform Resource Identifier
VNF	Virtual Network Functions
vSilo	Virtual Silo
vThing	Virtual thing
WLAN	Wireless Local Area Network

Table 1: Abbreviations

Fed4IoT Glossary

Table 2 lists and describes terms relevant to this deliverable.

Term	Definition
FogFlow	An IoT edge computing framework that automatically orchestrates dynamic data processing flows over cloud- and edge-based infrastructures. Used for ThingVisor development
Information Centric Networking	New networking technology based on named contents rather than IP addresses. Used for ThingVisor development
IoT Broker	Software entity responsible for the distribution of IoT information. For instance, Mobius and Orion can be considered as Brokers of the oneM2M and FIWARE IoT platforms, respectively
Neutral-Format	IoT data representation format that can be easily translated to/from the different formats used by IoT Brokers
Real IoT System	IoT system formed by real things whose data is exposed trough a Broker
System DataBase	Database for storing system information
ThingVisor	System entity that implements Virtual Things
VirIoT	Fed4IoT platform providing Virtual IoT systems, named Virtual Silos
Virtual Silo (new name for IoT slice in D2.1)	Isolated virtual IoT system formed by Virtual Things and a Broker
Virtual Silo Controller	Primary system entity working in a Virtual Silo
Virtual Silo Flavour	Virtual Silo type, e.g. "Mobius flavour" is related to a Virtual Silo that contains a Mobius broker, "MQTT flavour" refers to a Virtual Silo containing a MQTT broker, etc.
Virtual Thing	An emulation of a real thing that produces data obtained by processing/controlling data coming from real things
Tenant	User that accesses the Fed4IoT VirIoT platform to develop IoT applications through a vSilo
Root Data Domain	Set of sources providing IoT information to the VirIoT platform
Federated systems	External IoT systems that share information with VirIoT (through the System vSilo), forming a NGSI-LD global federated system
System vSilo	NGSI-LD vSilo used at system level to share information of vThings with external NGSI-LD federated systems

Table 2: Fed4IoT Dictionary

1 Introduction

1.1 Purpose of the Document

This deliverable reports the first iteration for pilot integration. It provides information about the strategy that Fed4IoT has followed in integrating the various components of the pilots into software packages that can be deployed in the respective target environments.

1.2 Executive Summary

This deliverable is the result of Task 5.2: Pilot Integration, of this project. Within the scope of this task, it deals with setting-up the deployment of test environments. The partners present the design process for each of their pilots, as well as a list of specific components for each of them, and how they integrate into the running platform.

This deliverable covers the following sections:

1) Codebase management and integration

2) Cross-border structure of the VirIoT testbed platform

3) Pilots and their integration with VirIoT platform

Regarding the codebase management and integration strategy, we define two phases. The first one focuses on how we took advantage of the GitHub portal, coupling it with Docker technology, for seamlessly integrating code of the different components. The second one focuses on Kubernetes technology, in order to demonstrate easy handling of our components.

Additionally, we show that we have already instantiated our VirIoT platform. We describe the testbed structure and how we have leveraged this technology for better development of the platform through Azure services provided by Microsoft's Cloud and the Kubernetes technology. The testbed spans EU and Japan clusters.

Finally, regarding the pilot integration, we define, for each use case, an overview description and its aim, we identify the components of the VirIoT platform to be instantiated for the pilot (ThingVisors and Virtual Silo) and how to deploy them, and we give details about pilots' data models changes, if any, from previous deliverable D5.1 of the same Work Package, where we had started designing them.

As general remark, we notice that not all pilots are currently at the same level of maturity/implementation. Indeed, this is just the first release of this deliverable.

1.3 Quality Review

The internal Reviewer for this deliverable is Giuseppe Tropea (CNIT).

Version Control Table			
V.	**Purpose/Changes**	**Authors**	**Date**
0.1	ToC	Juan Antonio Martinez, Antonio F. Skarmeta (OdinS)	25/05/2020
0.2	Initial Version	Juan A. Sanchez, Juan A. Martinez, Antonio F. Skarmeta (OdinS), Kenji Kanai (WAS), Kenichi Nakamura (PAN), Tetsuya Yokotani (KIT), Hiroaki Mukai (KIT), Gilles Orazi (EGM)	12/06/2019
1.0	Quality review	Giuseppe Tropea (CNIT)	28/06/2020
1.1	Final review	Andrea Detti (CNIT)	30/06/2020

Table 3: Version Control Table

2 Codebase Management and Integration

VirIoT, the platform we have developed within the Fed4IoT project, comprises several different components or enablers, which allow virtualising IoT information coming from different data providers. One of the goals is to have users and consumers access this information in a controlled and secure way, including mediated access to actuators.

Setting up a reliable strategy for quality control of the source code of such a platform, which comes from different partners and, given the capabilities of the platform to manage isolated containers, is implemented using several different programming languages and programming patterns, is of paramount importance.

Deliverable D2.3 on the System Architecture provides the overall design of the various interfaces. In this Section of this deliverable we focus, instead, on issues of software integration, impacting how we manage code of the Master-Controller, SystemvSilo, ThingVisors, of the various flavours of vSilos, which are implemented both in Python and in node.js programming languages, and of the security enablers.

There are different strategies that could have been applied for a seamless codebase integration of the VirIoT components, ranging from a manual approach where code is stored in different code repositories, up to more advanced ones where we can also take advantage of the latest DevOps tools for an automated deployment and continuous integration.

During the course of the project we refined management and integration of the codebases in two steps: first we took advantage of the GitHub portal, coupling it with Docker technology, for seamlessly integrating code of the different components. Secondly we focused on Kubernetes technology, in order to demonstrate easy handling and deploy of our components. We opted for using GitHub because of the familiarity that partners have with it, as well as the well-known capabilities that they provide as one of the most-known systems for managing code. Then we started using cloud providers for a coordinated deployment thanks to the Kubernetes technology. These two phases are explained in the following subsections.

2.1 GitHub Strategy and Dockerized Components

GitHub is a well-known platform that helps assuring quality when developing services and applications. Based on git repositories, GitHub is well-known world wide because of the mechanisms it provides to services and application owners to improve the quality of their development cycle, by allowing the community of developers to download, test, and even improve the quality of progressive development, in a controlled manner, thanks to the concept of *pull-request* by which they can request changes over the main branch, which must be verified by the maintainers of the code before being integrated.

This sort of tool is also a very good approach for collaborative development, as it is the case for the VirIoT platform, which has been developed in a collaborative fashion among all partners. The advantages offered by this solution make it ideal for distributing the platform to the public, and collecting feedback from the community.

Further, Docker is one of the technologies which has gained a great popularity because of its ease at configuring the running environment, defining what is going to be inside the containers, and also speeding up universal execution of the corresponding instances.

For the key components of our platform, we have written and deployed on GitHub, alongside the component's code, automated shell scripts that take care of the creation, building and updating of the corresponding Docker images, as well as pushing them to DockerHub.

This strategy has allowed us to obtain two important objectives:

- Assuring the quality of our codebase, fostering collaboration among partners and collaborative code writing.
- Provide an easy deployment cycle of our testbeds; of our open source platform.
- Give the opportunity to external developers to improve and give feedback about our open source platform, possibly supporting validation of it by the developers' community.

2.2 Kubernetes Deployment

Kubernetes allows for an orchestrated deployment in remote servers. In a second iteration, we have progressed by allowing our VirIot platform to be deployed using this technology. In the following Sections, we see how we have materialized the VirIoT platform, using this technology, in two data centres in different countries, starting to pave the ground for a system that can be exploited by the consortium.

3 VirIoT Cross-border Platform

The execution of the Fed4IoT pilots is supported by a cross-border deployment of VirIoT that is composed of Microsoft Azure Virtual Machines running Kubernetes. This section briefly describes related infrastructure-as-a-service, Kubernetes configuration and running VirIoT components (Kubernetes Pods).

3.1 Infrastructure

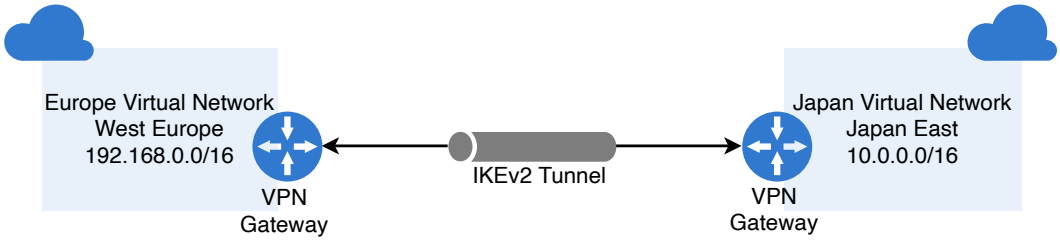

Figure 1: Site-to-Site Connection

The Azure services provided by Microsoft's Cloud were used to provide the testbed of the VirIoT platform. In this section we will describe the testbed structure and how we have leveraged this technology for a better development of the platform. In particular, we have exploited the regions West Europe and Japan East of Azure to realize our VirIoT cross-border platform. Each region's data center has some Virtual Machines that get together to realize an on-premise Kubernetes cluster.

The cross-border interconnection between the EU/JP data centers can be visualized in Figure 1. We have used two VPN gateways which use IPsec/IKE VPN tunneling to establish a secure, stable and continuous connection between the virtual networks of the two cloud sites.

The Virtual Machines we have deployed are of the Standard Dv3 family (2 vCPUs and 8 GiB memory) equipped with Ubuntu Server 18.04 LTS. In particular, four are operative inside the European site, whilst two in the Japan one, as Figure 2 shows.

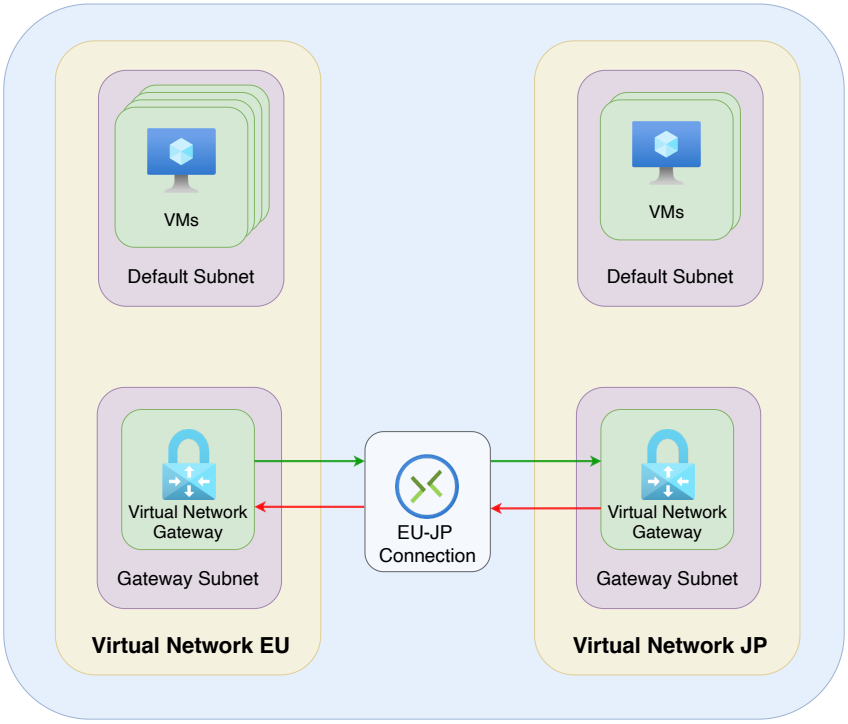

Figure 2: VirIoT testbed among EU and JP Azure regions

3.2 Kubernetes

In this section we explain the Kubernetes configuration for this testbed. Since we are leveraging Azure resources, they provide its customers with two main options for deploying a Kubernetes cluster:

- fully managed, ready-to-use Kubernetes service by Azure: Azure Kubernetes Service (AKS);

- configure your own Kubernetes cluster with resources provided by Azure (on-premise).

We have decided to go with the second choice for different reasons: first of all, using Kubernetes as AKS means the cloud provider is hiding all the complexity from the user. Running it as an on-prem bare metal deployment, means youre on your own for managing these complexities including persistent storage, load balancing, configmaps, services, availability, auto-scaling, networking, roll-back on faulty deployments, and more. This is translated into higher complexity, but indeed better understating of all the components running on the cluster, giving the ability to fine tune the requirements of the platform for all the contributors of the project.

Now we will dive into the main characteristics of the cluster. We have set up the Kubernetes cluster across all the available VMs in this manner: one machine is used as a master node and is located inside the Azure European data center, while all the remaining ones, three in Europe and two in the Japan region, are suited for being working nodes.

To support edge computing functionality we have exploited Kubernetes labels. Kubernetes labels enable users to map their own organizational structures onto system objects in a loosely coupled fashion, without requiring clients to store these mappings. By using labels on nodes, it is possible to constrain the running of a pod to a specific node that matches the exact label value (*node-afnity*) or, on the contrary, that a pod should avoid being allocated on a particular node with a different label (*pod-anti-afnity*). We are leveraging the aforementioned *node-affinity* to consider a Kubernetes distributed cluster formed by a default zone and, optionally, multiple edge zones. The default zone has no "viriot-zone" label. Edge zone nodes must be labelled with "viriot-zone" and "viriot-gw" label, as follows: the value of the "viriot-gw" key must contain a public IP address through which it is possible to access the edge cluster. We have depicted our high-level cluster view in Figure 3, as we can see, we have used the "viriot-zone" and "viriot-gw" labels to differentiate nodes not situated in the default cluster where, in this case, the master node is located. Consequently, in the testbed we have deployed nodes located in the Japan data center, labelled with *viriot-zone=japan* along with their default *viriot-gw*.

The VirIoT components are effortless deployed in Kubernetes using YAML files making the configuration fast and flexible. In order to properly work, VirIoT platform needs some necessary elements:

- the Master-Controller
- the SystemDB
- the internal MQTT cluster

The MQTT cluster must be present on each node to guarantee minimum latency between the user accessing information and the VirIoT node. The broker of choice is VerneMQ, a broker capable of clustering that can be deployed across the Virtual Machines. One replica of VerneMQ is deployed for each node and accessible among them thanks to a StatefulSet deployment and its related service.

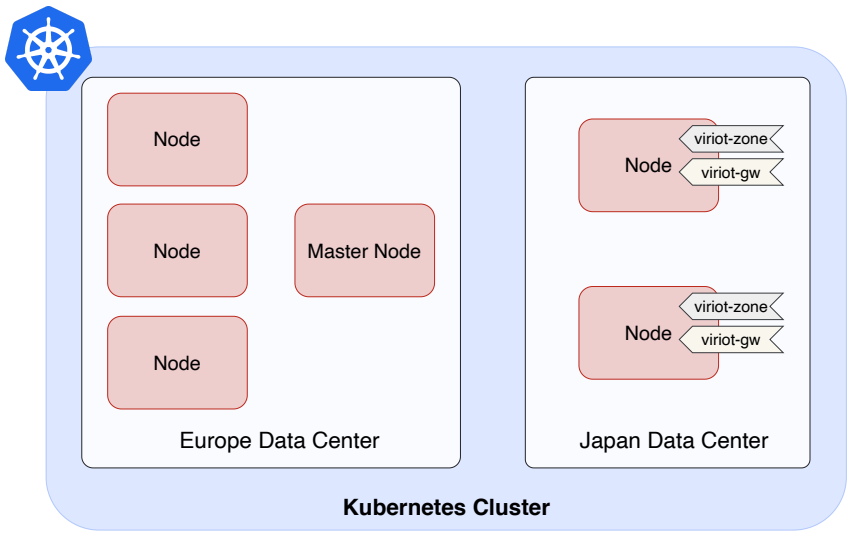

Figure 3: Kubernetes cross-border cluster

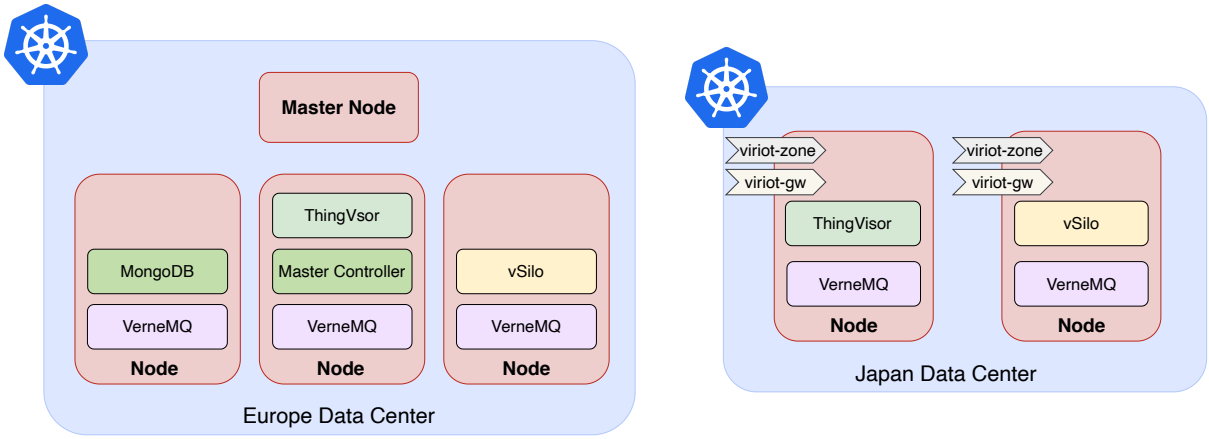

Figure 4: VirIoT main components on Kubernetes

The main element of the VirIoT control plane is the Master-Controller. It is in charge of managing the deployment of the elements of the entire platform after the requests of the both administrators and tenants. In order to correctly work, the Master-Controller needs some configurations, such as the *namespace* it is going to deployed, the IP address of the default gateway for which it can be accessed from the external and other internal IP addresses and Fully Qualified Domain Names. These configurations are deployed as ConfigMaps, that allows to decouple configuration artifacts from image content to keep containerized applications portable.

As described in deliverable D3.1, the System DB stores the run-time conguration of VirIoT. The noSQL database in use is MongoDB and it is also deployed as a StatefulSet. The data is saved as collections and it was added a further collection with respect to the time of the referenced deliverable to save the overall system setting, settingsC.

The overall description of the main elements of the VirIoT platform deployed using Kubernetes as an orchestrator can be seen in Figure 4.

4 Smart Parking Pilot

4.1 Description of the pilot

The Smart Parking pilot tries to deal with one of the most common issues of big cities, traffic congestion. One of the main causes which generate this problem is the vehicles wandering along the city, searching for a parking spot in a certain destination area. Smart Parking provides a solution to allow them to reduce the amount of time for this activity.

In this Smart Parking use case, the pilot is focused in Murcia, which is a city situated in the south-east of Spain, having a population of 450.000 residents. This city has experienced a dramatic rise of accesses to the city centre in the last years, which provoke a considerable increase in traffic congestion. Day-by-day, commuters, tourists and families traveling by car collapse the city centre with cars intending to park at commerce, financial and historical areas.

The aim of this Smart Parking use case is taking advantage of Fed4IoT framework to provide a service that tracks the state of the parking spots, to provide the drivers with this information beforehand and, as a consequence, to get more fluid traffic in the centre of the city.

4.2 Description of the components to be instantiated

4.2.1 Root Data Domain

The Smart Parking solution provided by our Fed4IoT framework integrates the information coming from the FIWARE-based Mi-Murcia platform. Figure 5 presents the most relevant components of the envisioned platform according to this concrete use case.

Our Smart Parking use case receives two kinds of context sources:

- The availability of private parking sites in terms of unoccupied parking spots.
- The probability to be able to park in the Regulated Parking Zone (RPZ). obtained via a (Machine Learning) model, trained based on the logged history of daily expended tickets.

For the case of the parking sites, the sensors, deployed in each private parking site, send the number of actual unoccupied parking spots when a vehicle enters or exits from the parking site. Usually, an IoT gateway is required to transmit this information to an IoT platform too.

For the case of RPZ, since in Murcia city we count with old-fashion parking meters equipped with highly-constrained CPU, during the day they are completely dedicated to the ticket issuance task. These devices take advantage of the night time to perform the transmission of the activity of the whole day, providing detailed information regarding the expended tickets.

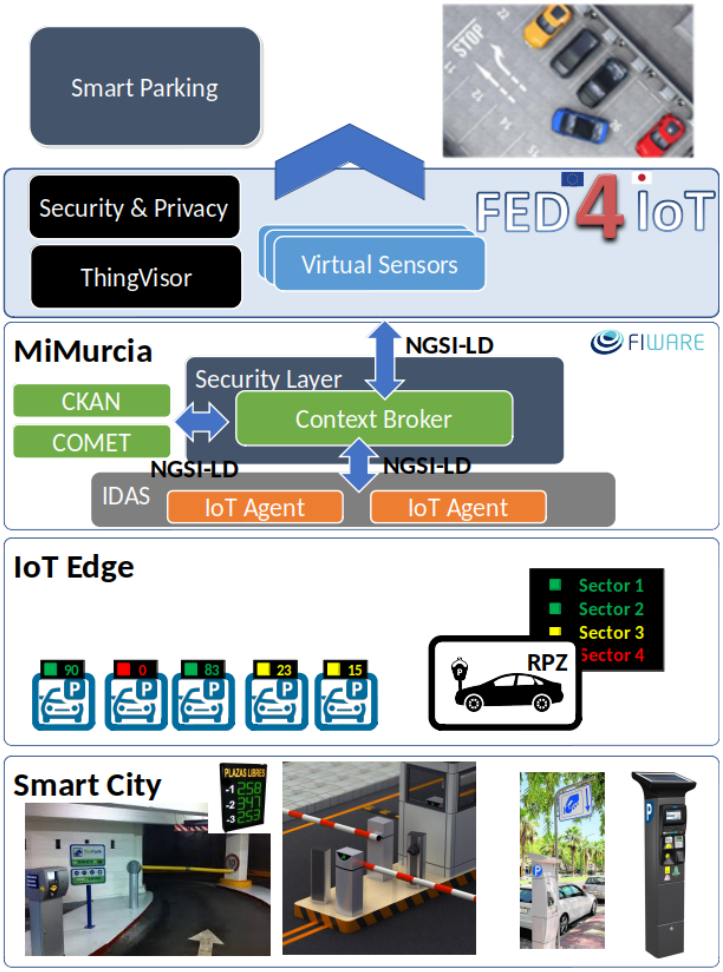

Figure 5: Smart Parking architecture

4.2.2 VirIoT Data Domain

Here we describe all components we need to instantiate in the VirIoT platform in order to have an operational Smart Parking pilot. These components are:

- `Parking Site ThingVisor`, which obtains the parking sites information coming from the FIWARE-based Mi-Murcia platform.
- `Regulated Parking Zone (RPZ) ThingVisor`, which obtains the RPZ information coming from the FIWARE-based Mi-Murcia platform.
- `Virtual Silo (orion-flavour)`, which receives the parking sites and RPZ information from the above ThingVisors and offers it to the Smart Parking pilot GUI.

Figure 6 shows the interactions between specific components of Smart Parking in VirIoT.

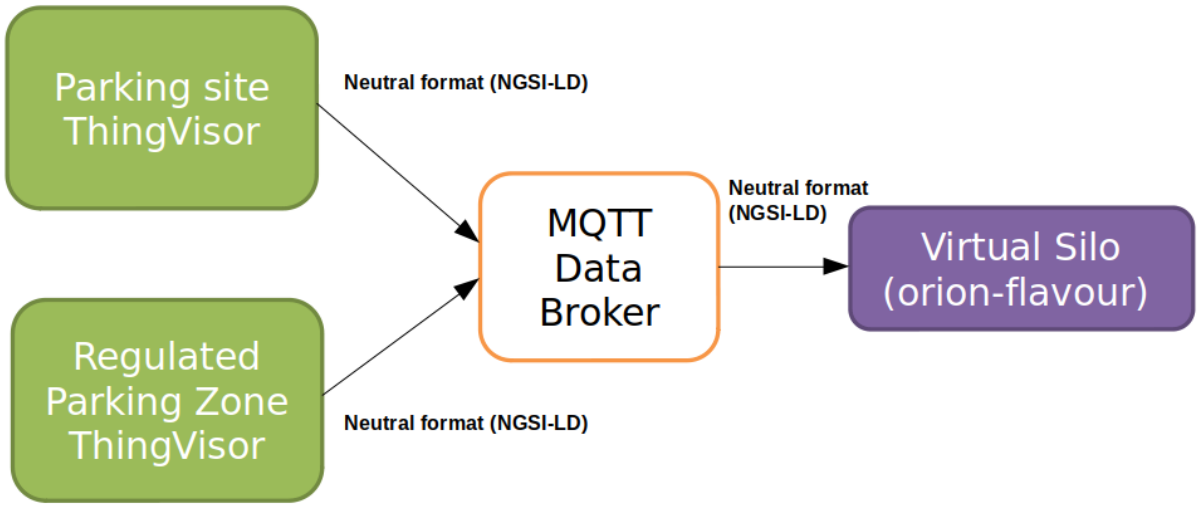

Figure 6: Smart Parking interactions VirIoT components

4.2.2.1 Parking Site ThingVisor

This ThingVisor component obtains parking sites information from the FIWARE-based Mi-Murcia platform. To do this, it subscribes to the entities of the platform which contain this specific information. When ThingVisor receives the notifications from the platform (NGSIv2 format), it processes its payload and produces a neutral format payload (NGSI-LD) which is sent to MQTT broker in a specific vThing topic. Once Thingvisor sends an NGSI-LD payload, the Virtual Silos that are subscribed to the corresponding vThings will receive the information. Figure 7 shows these functionality by describing the interactions commented above.

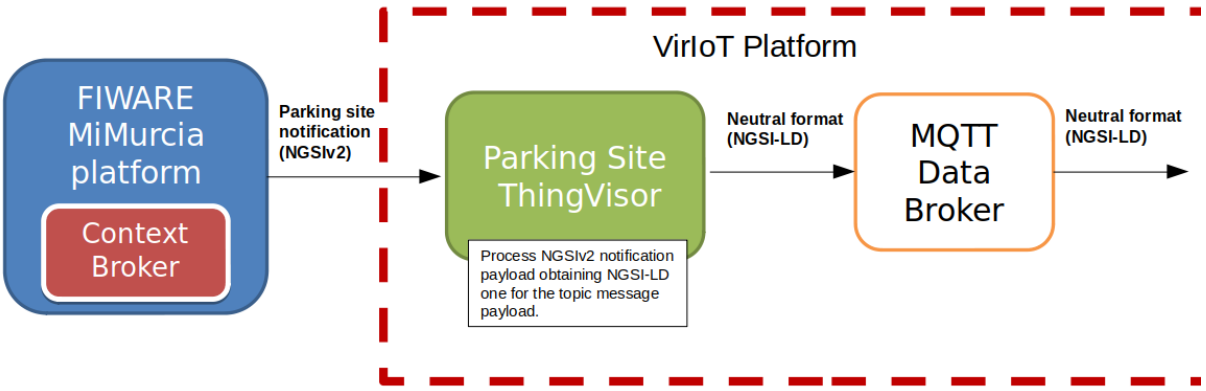

Figure 7: Parking Site functionality

4.2.2.2 Regulated Parking Zone (RPZ) ThingVisor

The functionality of this ThingVisor is the same as indicated in the `Parking Site ThingVisor`. The unique difference is that this ThingVisor subscribes to the FIWARE-based Mi-Murcia platform to obtain specific Regulated Parking Zones information. Figure 8 shows it.

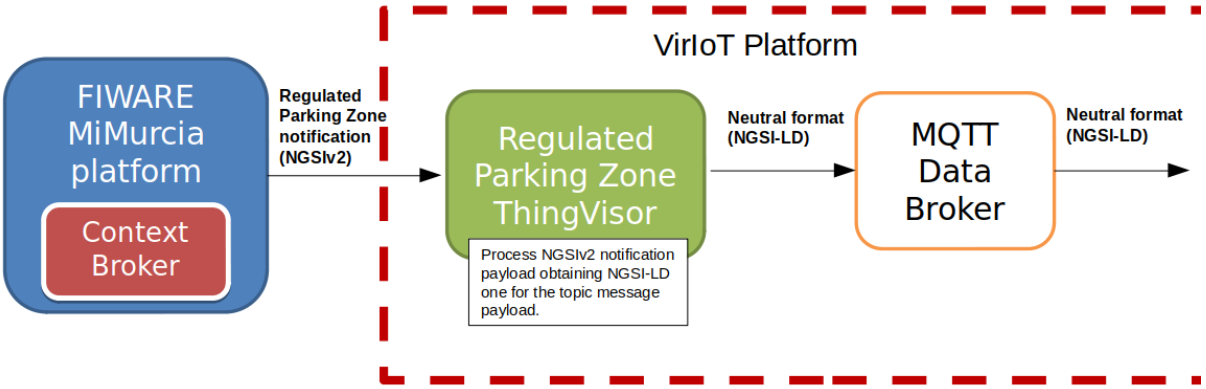

Figure 8: Regulated Parking Zone functionality

4.2.2.3 Virtual Silo

The Virtual Silo component receives Neutral-Format data (NGSI-LD) through the platform's MQTT broker it is connected to. It receives whatever NGSI-LD payload was previously published by the ThingVisors to the platform's MQTT broker.

This component has a Virtual Silo Controller which processes the Neutral-Format data and converts it to NGSIv2 data, which is stored in an Orion Context Broker, embedded in the Virtual Silo, by issuing a request to the NGSIv2 API. This way, the Virtual Silo offers to Smart Parking an NGSIv2 API to access its data, i.e., the information of parking sites and RPZ. To access data, NGSIv2 offers two options, either using the entity queries or through the subscription mechanism.

Figure 9 summarizes the functionality of this Virtual Silo.

4.2.3 Tenant Data Domain

This solution will provide a GUI, as depicted in Figure 10, allowing the user to specify both the current location and the destination, as well as parking duration, the time when she will arrive and other user preferences (maximum desired cost, maximum desired distance from parking to destination, ...) by presenting a map-based web interface/App. Once the selection is made, our Smart Parking solution makes a complex reasoning to generate an informed recommendation about the best destination area where to park the vehicle.

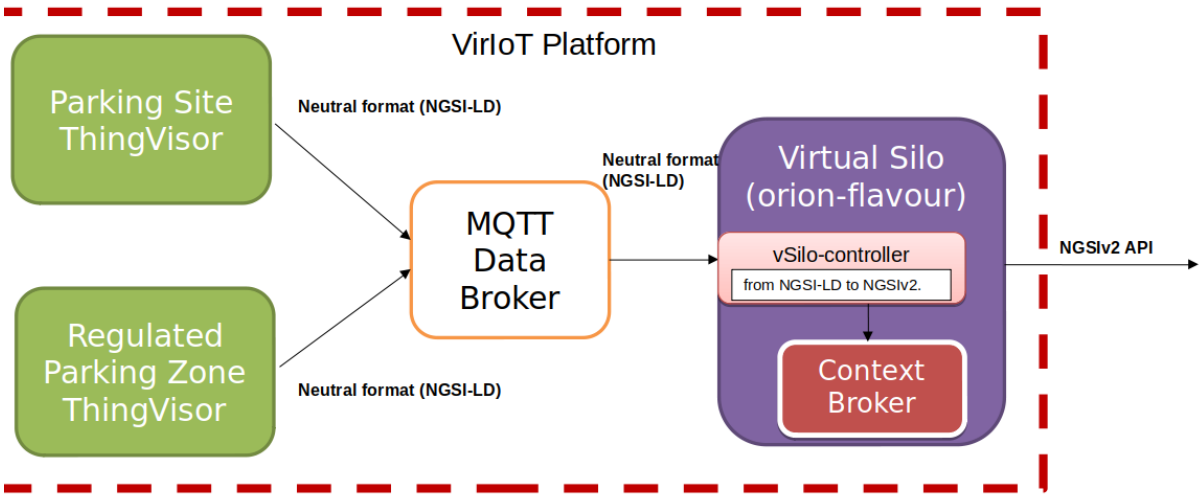

Figure 9: Virtual Silo (orion-flavour) functionality

4.3 Deployment strategy

This subsection details how the components of the Smart Parking pilot that were mentioned and detailed in previous Section 4.2 will be integrated into the VirIoT platform.

On the one hand, Smart parking ThingVisors and Virtual Silo will be deployed in the edge node of EU by the Master-Controller. There are two methods to deploy components through the Master-Controller:

- using the Command Line Interface (VirIoT/CLI)
- using Master-Controller's API

Once ThingVisors are deployed, notifications received from the FIWARE-based Mi-Murcia platform are processed and sent to the Virtual Silo, which then is able to offer the data through its NGSIv2 API of its local Orion IoT Context Broker.

On the other hand, the Smart Parking pilot application will be deployed in the cloud and can then obtain Smart Parking information by requesting the corresponding information directly to the Virtual Silo Broker via the standard NGSIv2 API. This instance of the VirIoT platform is presented in Figure 11 where the specific deployment strategy is presented.

4.4 Data Model

This subsection details the updates regarding the data model of the Smart Parking use case. In this sense, FIWARE-based Mi-Murcia platform has the following entity types that will be exposed to Smart Parking use case.

If we compare to the previous version of the implemented data model, we notice that the ticket entity has been now removed.

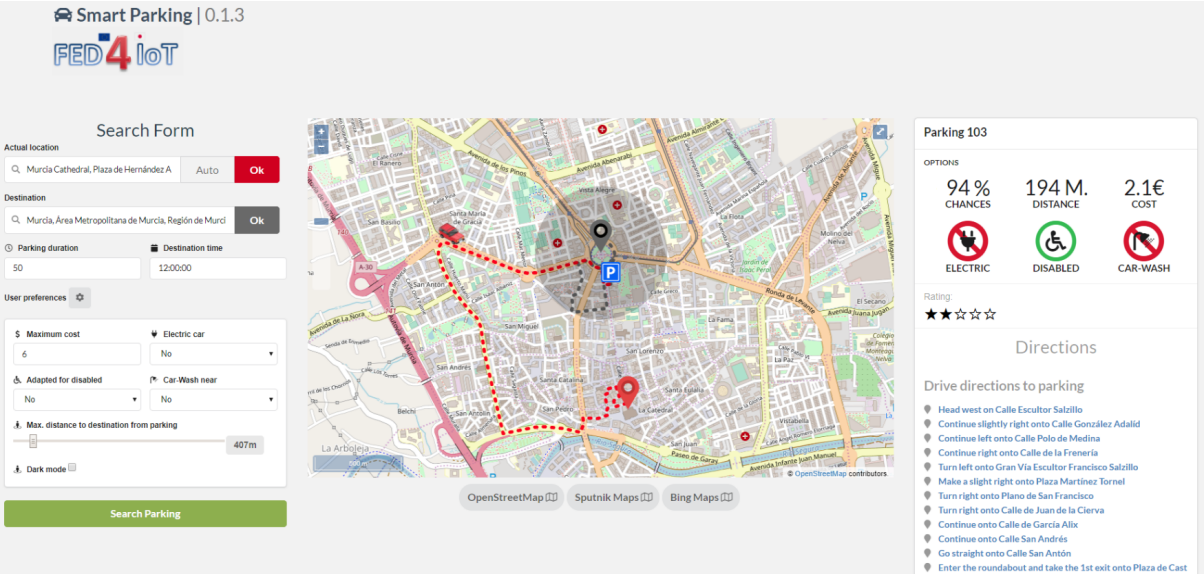

Figure 10: Smart Parking map-based GUI

Type	Description
parkingsite	This entity contains a harmonised description of a parking site.
policy	This entity contains a harmonised description of a parking site policy.
sector	This entity contains a harmonised description of a Regulated Parking Zone sector.
parkingmeter	This entity contains a harmonised description of a Regulated Parking Zone parking meter.

Table 4: Smart parking entity types

Here, we update the overview of entity types which have changed. All the details regarding the following entities have been placed into the Annex.

Sector - This entity contains a harmonized description of a Regulated Parking Zone sector. This entity has attributes to define:

- Policies, considering the public holidays.
- The geolocation polygon of the sector.

Parking meter - This entity contains a harmonized description of a Regulated Parking Zone parking meter. This entity has attributes to define:

- Parking probability.
- Related sector.
- The geolocation point of the parking meter.

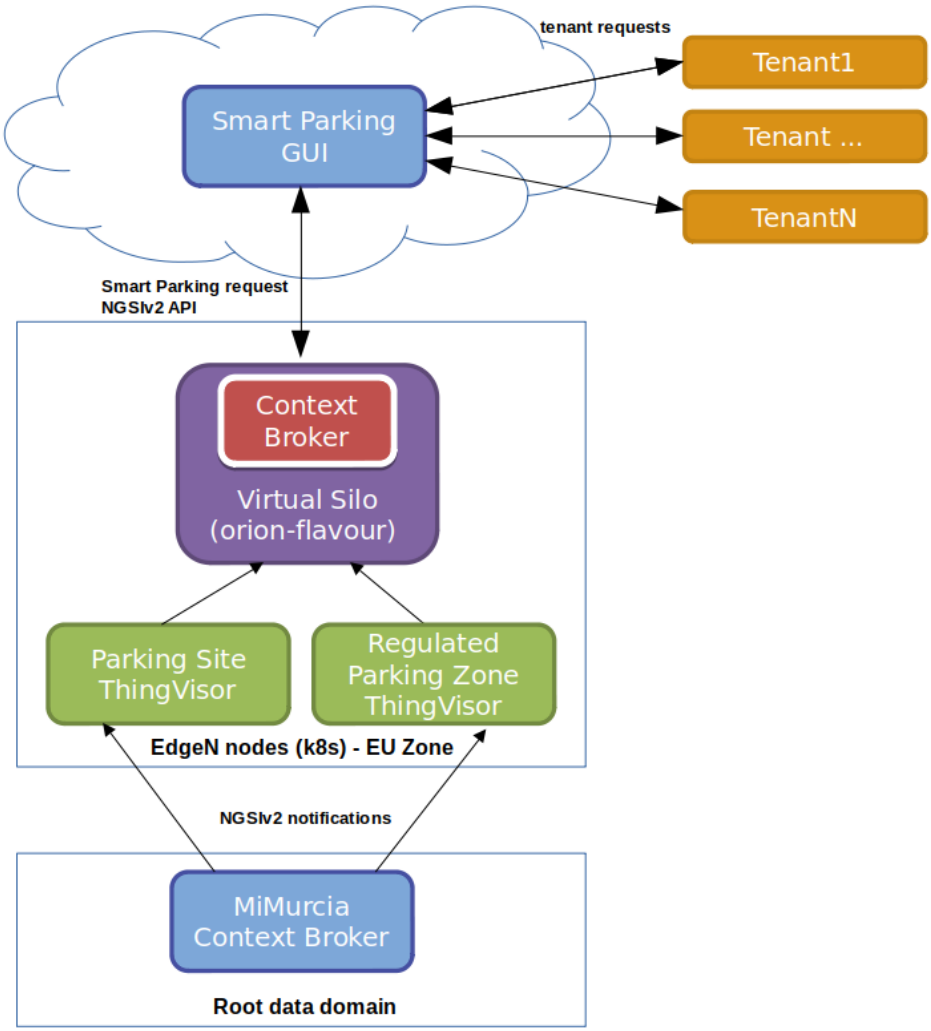

Figure 11: Deployment strategy for Smart Parking pilot

5 Illegal Waste Deposit Management Pilot

5.1 Description of the pilot

The idea behind this pilot is to use a camera associated with some deep learning machine vision algorithm to perform the monitoring of a given site, and to send events as they occur on the field. Our aim is to apply this scheme to the detection of incivilities, such as illegal waste deposits.

Figure 12: Example of illegal deposit

We are in contact with the waste management service of the city of Grasse that helps us to understand where and how these illegal deposits happen. It turns out that there are some spots where this occurs more frequently. They already track these incivilities using trail cameras. They hide them for a couple of days, permanently recording photos of the site. After having taken back the camera, an operator looks through the thousands recorded images to track incivilities and send plate numbers of vehicles to the local police. It is important that the camera is hidden, otherwise it would be subject to vandalism.

The idea is thus to deploy a small and discrete system to monitor such a spot with a smart camera and send alerts when an incivility is detected by the system. To achieve this goal, the following has to be done inside the project:

- machine vision algorithm to perform the detection of events;
- hardware setup with camera, computer and connectivity to be deployed on the field;
- thingVisor implementation and data modeling for integration into the Fed4IoT platform.

In order to train a deep learning model, we need to collect some real world examples of illegal waste deposits. Our contact with the waste management service of the city of Grasse helps us with this task. We are frequently visiting them to get some new images they took using their trail cameras. An example of such images is given in Figure 12. We are in the process of collecting this dataset and training our detection model. However, this is a long term task and we need to work in parallel on the other tasks. This is why

we decided, as an intermediate step, to work on a deployment of a site monitoring camera that will be able to perform some simple detections.

It turns out that our local authorities have another issue we could try to solve with a smart camera. They need to know how the carpooling parkings they build are used by people. How many cars park in them each day? What is the average parking time? Which kind of cars are using it? Could we infer from "in and out" events how much CO_2 emissions are avoided by the carpoolers using this parking? Technically we are very close to the illegal waste deposit detection: we need the same hardware deployed on the field, and the interface to the Fed4IoT platform is quite similar. The detection algorithm is much simpler since we only need to detect license plates for which existing algorithms are already usable.

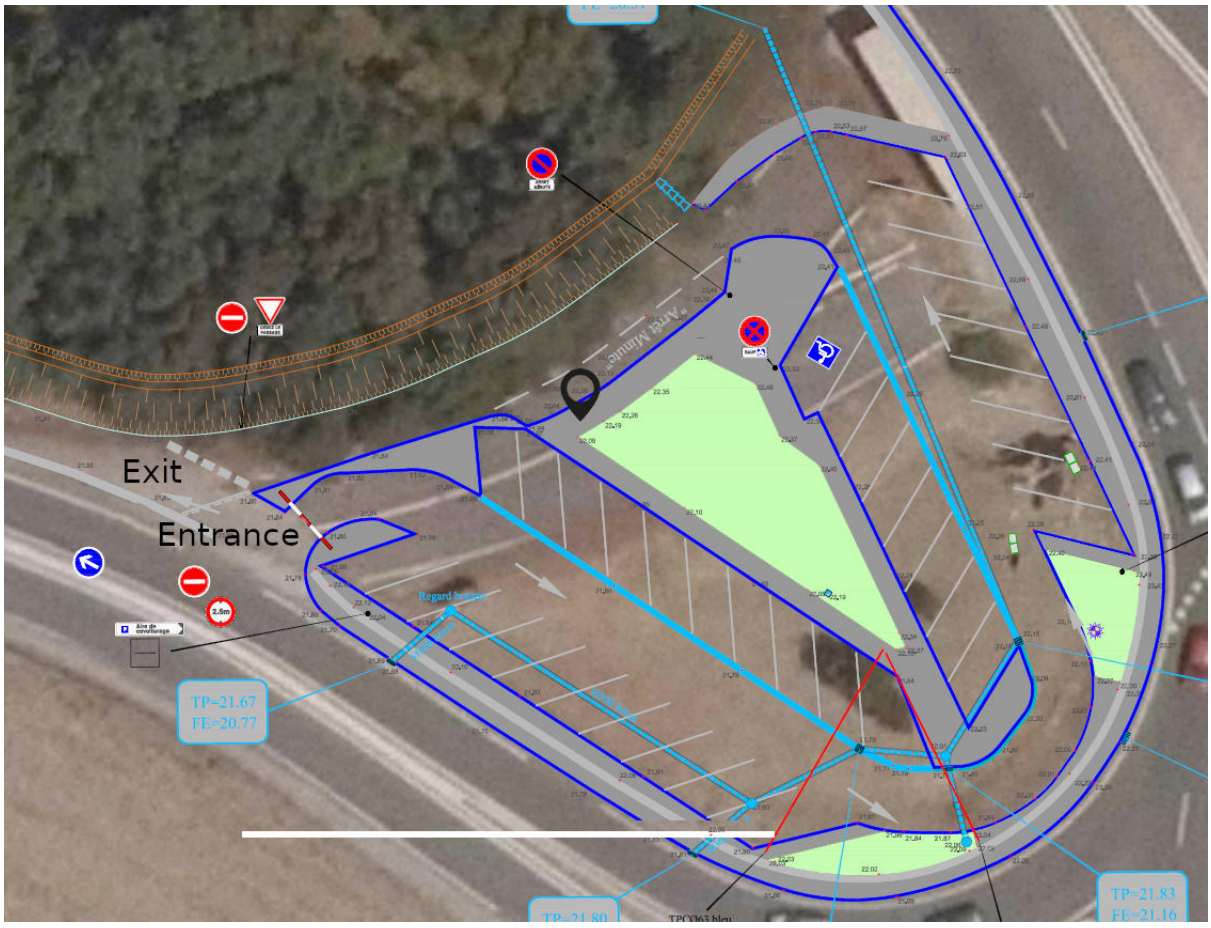

Figure 13: Site for the deployment of the car counting use case. The camera will be located on a mat at 4.5m height on the black spot. From this point of view it is possible to see incoming and outgoing vehicles with a single camera.

5.1.1 Deployment Site

The deployment site has been identified (see Figure 13). Its entrance and exit can be monitored using a single camera.

The chosen hardware, to be deployed on site, is built around a Jetson Nano computer. This is a low power, small form factor computer for embedded edge computing built by Nvidia. It runs a quad-core CPU associated with a 472 GFlops GPU that is able to run deep learning algorithm, its maximum power consumption running at full GPU capacity is around 10W.

The camera has a 4MP sensor with IR illumination for night vision up to 50m. It also provide an optical zoom and pan/tilt abilities to tune the framing of the image. We also rely on integrated motion detection and ftp capabilities to send only candidate images to the processing unit.

We considered to power the system using solar panels. We determined that 150W solar panels with a 90Ah battery should power the system for a near 100% uptime. But for some practical considerations (avoid the theft of the solar panels) we were allowed to plug our system on a lighting system about 100m away. The solar panels system will thus not be deployed for this part of the experiment.

5.2 Description of the components to be instantiated

5.2.1 Root Data Domain

On the edge side, the software architecture is as depicted in Figure 14. The motion detection capabilities of the camera are activated such that each time the scene is changing, an image is uploaded into a fixed folder of the edge computer. A first process (the feeder) is responsible to detect that a new image has arrived and triggers a license plate detection by the ALPR component. It then reports the result in a message queue, this message contains essentially the location of each license plate on the image (which helps to determine if it is an entrance or exit event) and the detected license number. It is then sent by another process, responsible for the delivery of the message to the Fed4IoT VirIoT platform over an LTE connection.

At this moment, the system is ready for deployment on the field, we are waiting for the execution of public works to install the mat and bring the power cable. It is planned to be done on the 1st of July. We will install our setup the same day. It is currently installed for tests near a road and detects correctly the vehicles passing on the road.

5.2.2 VirIoT Data Domain

Once the Edge processing is done, the message is received on a virtual computer in the cloud that does the following pre-processings: query a license plate public database to get some data of interest concerning the car (as vehicle type, engine type, CO_2 emissions) and then immediately pseudonymize the license plate number using a hash algorithm.

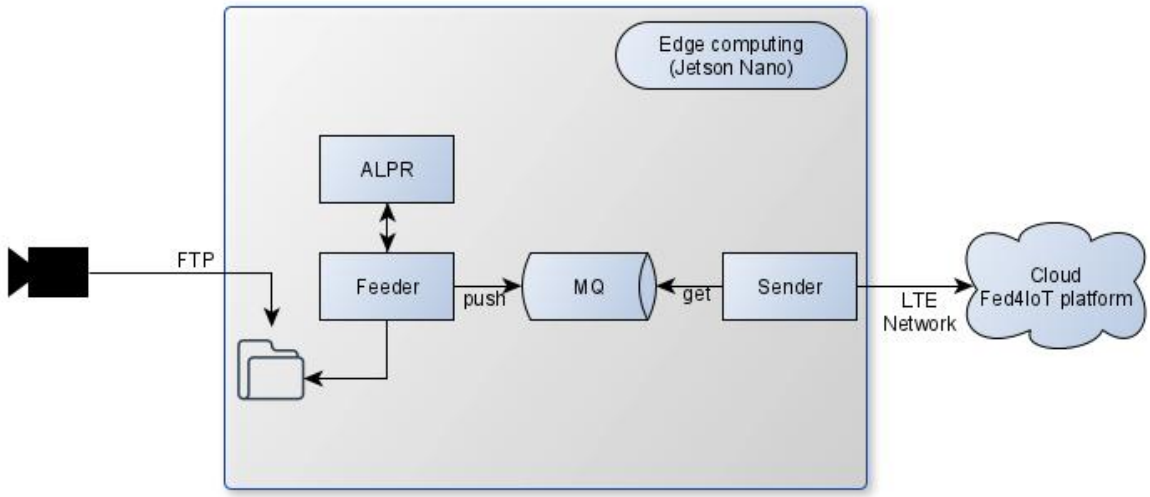

Figure 14: Edge software architecture

The license plate is no more used nor stored in the following processes, ensuring users privacy.

We first thought that the event data will be transmitted using a LoRaWAN network, we thus implemented a LoRaWAN thingVisor, connected to the MQTT broker of the LoRa Chirpstack network server. This ThingVisor helps to connect incoming small pieces of data to their full NGSI-LD context. But it turns out that the LoRa network is not well suited to our needs, in particular it does not have enough bandwidth to send all the license plate detection events. We then switched to LTE connectivity, pushing data to the same MQTT broker as the one used by the LoRa server, using a compatible message format. This allows our LoRaWAN ThingVisor to be used for the integration of this smart camera setup into the Fed4IoT VirIoT platform.

The entry/exit event detection for a given car park will be provided as a vThing in the virtual data domain that will provide the NGSI-LD context described by the use case data model (see the update of this model in annex).

At the moment of this writing, this ThingVisor implementation is at a beta stage: it has all the needed functionalities but is not yet production ready. Some work has still to be done in the testing and error management areas.

5.2.3 Tenant Data Domain

The tenant data domain will be handled by an instance of the Stellio NGSI-LD broker, connected by a specialized vSilo. Some dashboards for the use case monitoring will then be created using Grafana.

5.3 Deployment strategy

All the software components described for this use case are represented in the figure 15. On the monitored sites the cameras and the edge processing capabilities will be installed. They will be connected with LTE (but also possibly using LoRa or WiFi) to a root data domain MQTT broker on which some micro services are listening and provide:

- data format translation (essentially to put into the same format data coming from the different networks)
- query the license plate database API and add the related data to the output message
- an anonymizer that computes a hash for the license plate

The MQTT server is the one used by the LoRaWAN installation already deployed. The microservices will be installed on a dedicated virtual machine.

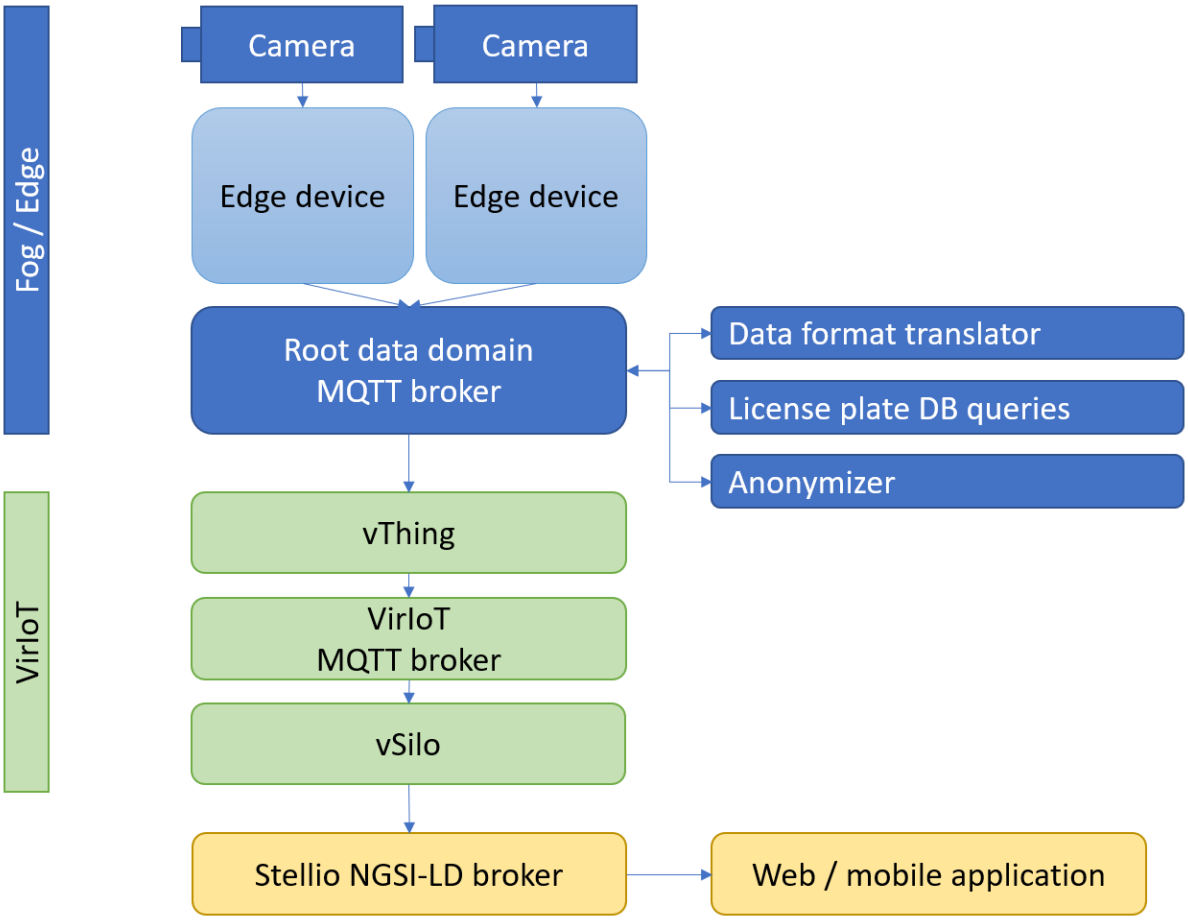

Figure 15: Deployment strategy of the smart camera

There will be one vThing instance for each smart camera installed on the field, deployed on the VirIoT cross-border platform, EU site. It will be connected to the Root Data Domain MQTT broker, listening to messages transformed by the micro services.

Eventually, a vSilo making the link to a Stellio NGSI-LD broker will be instantiated in the VirIoT cross-border platform, EU site. The cloud-based application will provide dashboards for the monitored sites, will query the needed data to this broker.

5.4 Data Model

Despite we have already provided a first version of the data model in D5.1, we have updated it as we are developing this pilot. So, we have taken the opportunity to bring all the new updates to this document. They can be found in the Annex at the end of this document.

6 Cross-border Person Finder Pilot

6.1 Description of the pilot

The Cross-border Person Finder (CBPF hereinafter) pilot is one possible application to demonstrate inter-operable capability of Fed4IoT's VirIoT platform. The pilot aims at virtually sharing the core functionality of CBPF application among surveillance cameras installed in multiple cities, and then at performing complex image processing at the edge, by complying to the requirements of data protection regulations, such as GDPR, avoiding to move personal data to the centralized cloud if not necessary.

Currently, according to the white paper on tourism in Japan [1], the number of international tourists is rapidly increasing, and tourism industry is grown remarkably. According to the white paper [1], in Japan, the number of international visitors to Japan by air or sea is approximately 28.69 million in 2018, and this number is the second rank in Asia. As well as increasing of inbound tourists, outbound tourists (i.e., Japanese overseas travelers) are also increasing, and the numbers of them approximately 19 million in 2018. Those who inbound and outbound tourists tend to be elderly people or youth and not always able to speak local languages fluently. Therefore, their families are often worried about their safety, and demand for smart applications that can notify the tourist safety and monitor the tourist traces becomes increase.

Based on the above background, the CBPF pilot provides an application able to notify the geographic location entered by a person, when authorized users, such as police officers, city hall staff and families, issue a request to try to locate the person. A conceptual representation of the CBPF pilot is shown in Figure 16. The CBPF application attempts to find the requested person from video feeds coming from surveillance cameras installed in multiple sties (e.g., EU and JP smart cities). In the pilot, in order to comply to the requirement of GDPR, the images captured by the surveillance cameras do not travel among cross-border countries (EU and JP). Thus, image processing is performed in edge devices (possibly in-camera) at EU or JP sites, and the geographic location information is extracted from the processed data.

In order to promote CBPF services, there are two security aspects we should consider. The first aspect is user consent when the CBPF application in initiates service for a specific person, in order to ensure that the person agrees to provide information which is later necessary for the CBPF to operate (e.g., portrait of the person to be found). The second aspect is verification of essential workers providing CBPF services. It can be achieved with identity verification and attribute verification in case anonymity of workers is required. It should be noted that the verification of essential workers can be used for other applications (e.g., COVID-19) to verify essential workers such as doctors and essential service employees (e.g., grocery store, and pharmacy). Based on the user consent and the identity/attribute verification, CBPF application authorizes user to provide or receive information necessary for carrying-out the CBPF service.

In the pilot, we give a typical use case of CBPF application as a tourist monitoring tool for personal safety. However, the core functionalities of CBPF are to detect (and track)

requested persons and ensure security. Thus, we expect that the CBPF application, including individual components of the CBPF, can be adopted to various application demands, such as tracking traces of criminal persons and tracking virus infected persons (e.g., COVID-19).

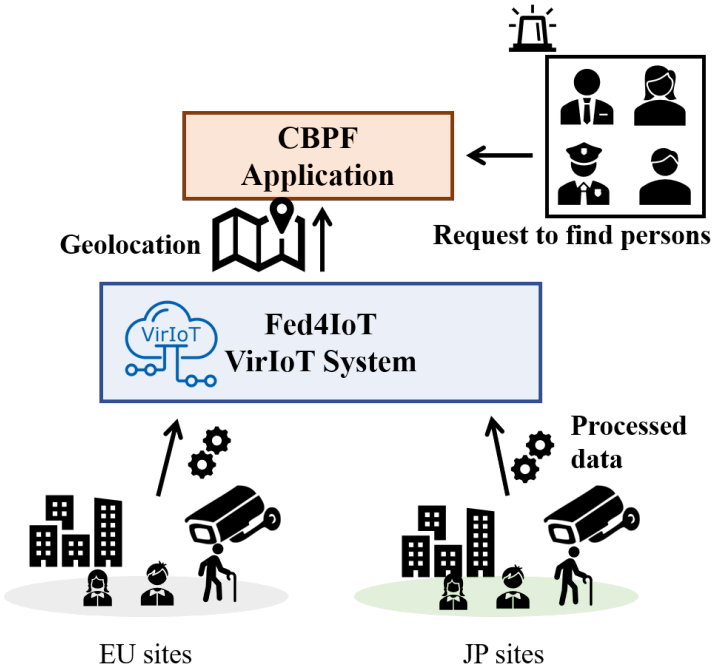

Figure 16: Concept image of Cross Border Person Finder pilot

6.2 Description of the components to be instantiated

In this Section we describe the components, with a focus on a solution for the attribute-based authentication system.

6.2.1 Root Data Domain

Camera Virtualization is a set of a method to use a surveillance camera as sensors. This function will be integrated into surveillance camera, but it is implemented to the board computer in this pilot.

Figure 17 illustrate how sensor functions are extracted from the camera. Surveillance cameras are connected to the data repository. Sensor functions can be set via REST API (Camera Virtualization API) from Thing Visor. The following two functions are implemented for the pilot.

- Human counter
- Face feature extraction

If the face feature extraction function is activated, the processing result can be sent to the face matching service. The face matching service will be integrated to compare the face feature of detected person and a missing person.

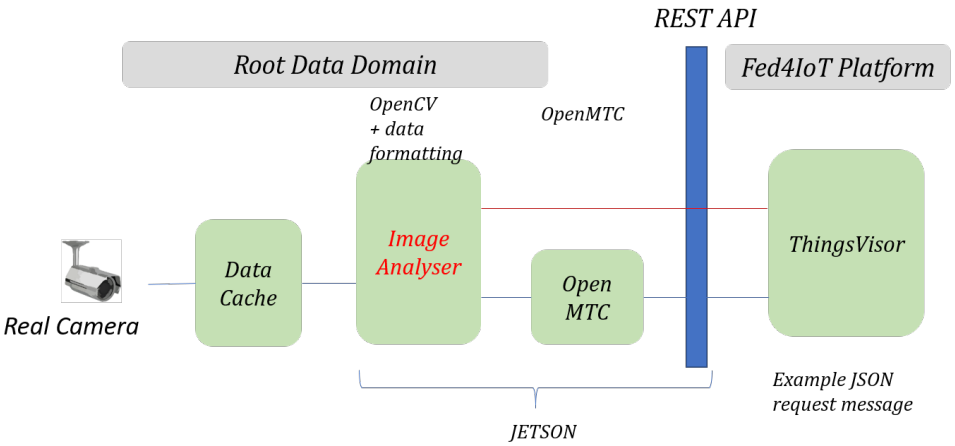

Figure 17: Camera Virtualization API

6.2.2 VirIoT Data Domain

In the pilot, we aim at having a privacy-preserving ThingVisor for detecting a specific persons face and some attributes corresponding to the person. This specific ThingVisor produces such context information as a Virtual Thing by converting the information to the Neutral-Format. We assume that the required image processing, such as detecting humans face is provided as Camera Virtualization APIs, and the APIs will be called from ThingVisor. We assume that ThingVisor is developed by using ThingVisor Factory described in deliverable D2.3. In addition, because CBPF application is operated as the application receives a users request, such as image or attribute of target person, the request message is forwarded from vSilo to ThingVisor. Therefore, vSilo and ThingVisor require a capability of handling downstream traffic like sensor actuation. After ThingVisor receives the request, ThingVisor attempts to detect the requested person by calling Camera Virtualization APIs with specifying the users specific request. Once ThingVisor succeed the detection, ThingVisor produces such context information, including camera meta data, as Virtual Thing by neutral data format such as NGSI-LD. The Virtual Thing is transmitted to vSilo and vSilo finally publishes Virtual Thing to CBPF application.

6.2.2.1 Attribute-based authentication

To ensure the security of CBPF pilot, an authorized access to Virtual Things shall be protected to prevent unexpected use of information. This can be done by service authorization to ThingVisor.

There are three points to verify the access right to ThingVisor.

- ThingVisor itself has a function to verify the request message has an access permission to ThingVisor.
- vSilo has a function to verify that the request from CBPF application has an access permission to ThingVisor.
- CBPF application has a function to verify that the (end) user of CBPF application has an access right to ThingVisor

Figure 18 illustrates how an unauthorized access to ThingVisor is protected.

Token is used to identify user and/or user attribute. The user select his/her identity and/or attribute to be authenticated and issue the token signed by holder device. The user send a service request message to CBPF application attaching the signed token. CBPF application send an request to authenticate his/her identity and/or attribute to Attribute-bases authentication system and receive a response and verify the use requesting service is a right person.

Then the CBPF application send a service authorization request to Attribute-based authentication system and then CBPF application receives an authorization code from attribute-based authentication system.

Finally CBPF application will send a request to vSilo attaching an authorization code. vSilo can request an certificate (and public key, if vSilo does not have effective public key) corresponding to an authorization code. vsilo can verify an authorization code and it will allow the service if the verification of authorization is successful.

This verification function can be also implemented to ThingVisor.

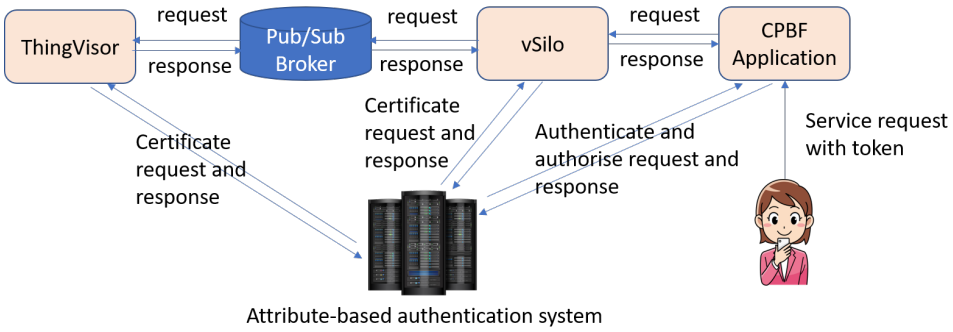

Figure 18: Protection of unauthorized access to ThingVisor

6.3 Deployment strategy

Because the CBPF pilot will try to demonstrate interoperability of Fed4IoT VirIoT system, the pilot will be deployed at multiple domains. Candidate sites are Murcia and Grasse as EU sites and Kumamoto and Hakusan as JP sites. A plan for deployment of the pilot is shown in Figure 19. As described in Section 6.1, the pilot needs to comply the requirement of GDPR, in other words, the surveillance camera images are prohibited

to be exchanged among EU and JP. Therefore, as shown in the figure, ThingVisor for CBPF will be deployed in the edge nodes of EU and JP sites, respectively. The deployment will be performed by Fed4IoT master controller. Inside ThingVisor, the images will be processed by using Camera Virtualization APIs, and at this moment, the processed data will be transformed not to be identified the individual personal information.

Unlike ThingVisor, vSilo and CBPF application will be deployed in the cloud. This is because the CBPF application will be accessed by various users, such as police officers, city hall staffs and families, and vSilo needs to broker Virtual Things produced at multiple sites.As shown in Figure 18, as well as vSilo and CBPF application, attribute-based authentication system is also deployed in the cloud in order to provide secure operations among ThingVisor, vSilo and CBPF application.

In the demonstration, we will deploy ThingVisor to multiple sites at EU and JP. The ThingVisor will be run on small compact PCs, such as Jetson Nano and/or barebone PCs, and these nodes will be operated as Kubernetes worker nodes (e.g., edge nodes). In addition, in the cloud side, we will deploy vSilo, CBPF application and attribute-based authentication system on the public cloud servers, such as Microsoft Azure, and Sakura Cloud. Through the demonstration, we will confirm data acquisition and interoperability by using VirIoT system.

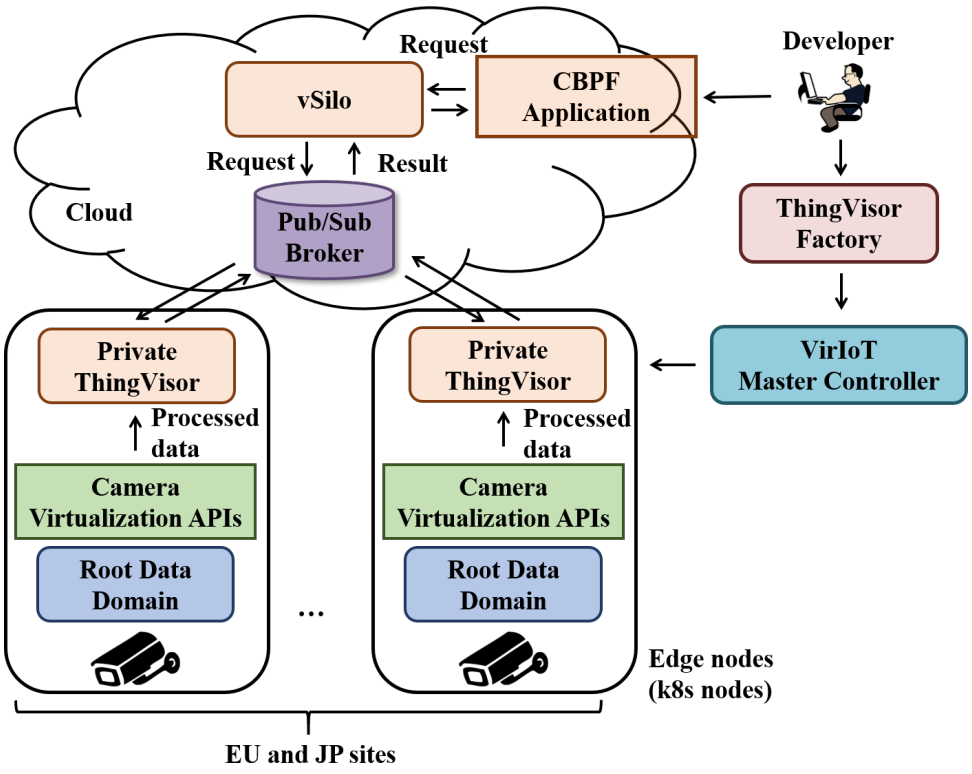

Figure 19: Deployment plan of Cross Border Person Finder pilot

6.4 Data model

Data model used for Cross-border Person Finder is not updated from deliverable D5.1.

7 Wildlife Monitoring Pilot

7.1 Description of the pilot

In rural areas, damage to agricultural products by wildlife is serious. It is effective to use IoT technology to collect information, such as captured and approaching animals, for countermeasures against wildlife damage, but the cost required for sensor device installation and application development is a problem. In local cities, budgets and manpower are not enough, those factors prevent the introduction of IoT technology. To solve this problem, sensor device installers and application software developers can be connected to the Fed4IoT virtual IoT environment (the VirIoT platform), which may enable reuse/repurposing of sensors and reduces the time and cost required for IoT system development. For verification purposes, a wildlife monitoring pilot system will be deployed in Hakusan City, Ishikawa Prefecture, Japan, where wildlife damages are serious. In addition, a sensor device will be installed for purposes other than animal damage control. By making these exist as Virtual Thing in the VirIoT platform, it will be possible to develop applications other than animal damage control applications, such as environment monitoring.

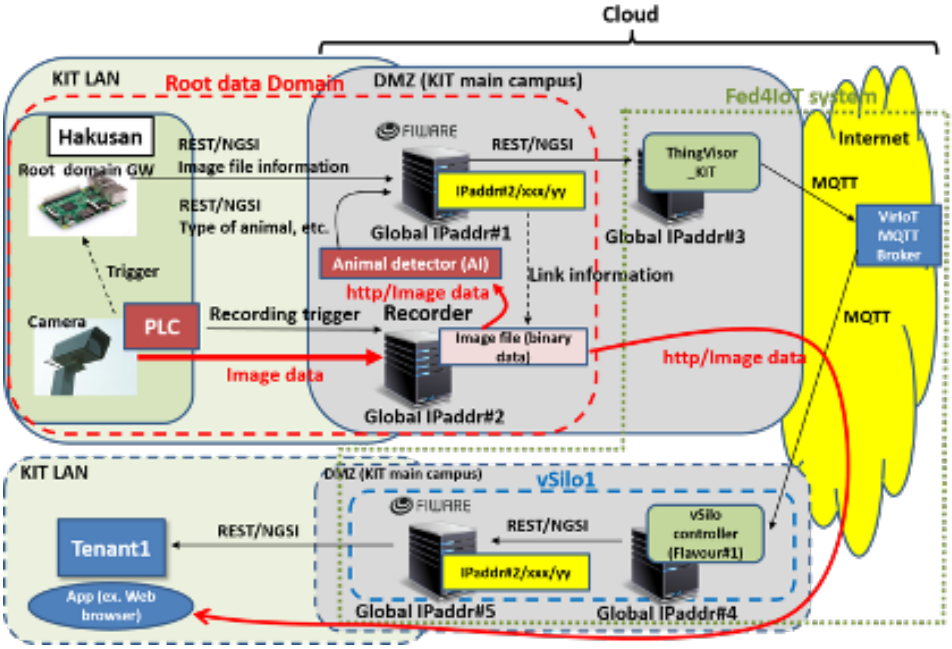

Figure 20: Pilot system at Kanazawa Institute of Technology

7.2 Description of the components to be instantiated

Figure 20 shows the configuration of the test environment for the pilot system at Kanazawa Institute of Technology. Cameras and sensors are installed in the field to collect information necessary for animal damage control, such as animal types and images, and envi-

ronmental information such as temperature and humidity. The application displays the detected animal position, environmental information, and the like. FIWARE is used as a data utilization platform.

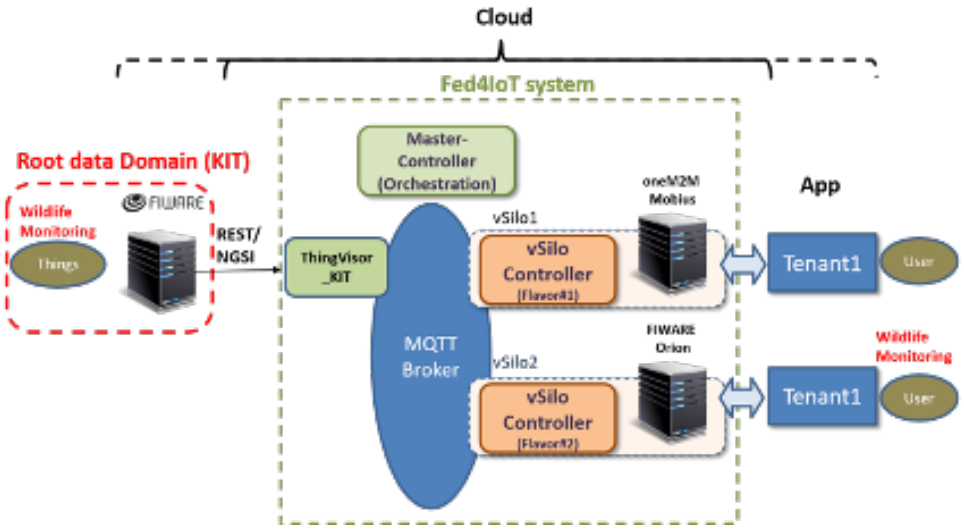

Figure 21: Data exchanges via Fed4IoT system

As shown in Figure 21, collected data from actual devices exists as Virtual Things in VirIoT system, and they can be used by various application software developers as needed. VirIoT system creates vSilo to collect the data needed by application software developers from various Virtual Things. Since the data format is interchangeable (e.g. oneM2M, NGSI, etc.) in VirIoT, assets can be used all over the world, depending on the end-applications data formats.

7.3 Deployment strategy

The pilot system will be deployed around Kanazawa Institute of Technology Hakusan campus at Hakusan City, Ishikawa Prefecture, Japan. Physical devices such as cameras, sensors and the root domain gateway will be installed outside field. The recorder of the cameras, some servers for related software running and animal detector (Jetson Nano) will be installed in a KIT office. The recorder and the servers are accessible from the Internet. Necessary ThingVisor and vSilos will be deployed in the JP site of the VirIoT cross-border platform, exploiting the local MQTT Broker of the cluster for achieving low latency.

7.4 Data model

Table 5 lists the Virtual Things in VirIoT, based on the Wildlife Monitoring use case and its Real Things, as we are going to virtualize them to be flexibly reused in different vSilos. The Table also reports the structure of the attributes that will be internally exposed as

Virtual Things	Attributes	Description
Thermometer	Location	Location of the device
	Temperature	Sensed data
Hygrometer	Location	Location of the device
	Humidity	Sensed data
Rain gauge	Location	Location of the device
	Rainfall	Sensed data
Illuminometer	Location	Location of the device
	Illuminance	Sensed data
Animal detector	Location	Location of the device
	Animal is present	Sensed data
	Animal type	Results of judgement from photo by Jetson-nano
Camera	Location	Location of the device
	Photo data of animal	URL of the image file
	Camera type	Type of camera
	Resolution	Provisioned value

Table 5: Virtual Things for Wildlife Monitoring

NGSI-LD properties of the corresponding Entities, conveyed from the ThingVisors to the vSilos.

Figure 22 shows an example of the NGSI-LD Entity associated with the Thermometer Virtual Thing. Other Virtual Things have a similar structure.

```
{
  "id": "urn:ngsi-ld:KIT:Thermometer01",
  "type": "Thermometer",
  "location":{
    "type": "GeoProperty",
    "value": {
      "type": "Point",
      "coordinates": [36.5313, 136.6285]
    }
  },
  "temperature": {
    "type": "Property",
    "observedAt": "2020-05-12 16:02:56.343000",
    "value": "1",
    "unitCode": "CEL"
  }
}
```

Figure 22: An example of NGSI-LD entity published by the thermometer Virtual Thing

8 Conclusion

This deliverable reflects the work carried out during Task 5.2: "Pilot Integration". Although this is only the first iteration, in this document we can already provide various kinds of information regarding the integration process we designed for each of the pilots. This integration process covered a description of the pilot, the specific instances of the VirIoT platform they require, including the specific Thing Visors and Virtual Silos, and also the place where such instance is going to be deployed (in terms of EU or Japan data centres).

Along with this information, in this document we have also included the codebase management and integration strategy that we have followed so that our VirIoT platform is easy to be developed and deployed. We have highlighted the use of dockerized components, GitHub repositories, as well as the use of Kubernetes.

We have described the details of our current testbed deployment, how it is based on cloud resources from Microsoft Azure Clouds, and how these clusters of virtual machines span EU and Japan regions.

This information will be completed in the second version of the document, where more specific information will be provided by all the project's pilots.

9 Annex: Data model updates

9.1 Smart Parking Pilot

The updates in the data model of Smart Parking use case are:

- `Sector` removes `numSpaces` and `numSpacesCapacity` attributes
- `Parkingmeter` adds`parkingProbability` attribute
- `Ticket` is removed from Smart Parking data model.

The updated data model for sector and parkingmeter entities currently available in Murcia is NGSIv2.

The updated data model for `sector` and `parkingmeter` entities is detailed below (NGSIv2).

9.1.1 Sector

Sector attributes			
Attribute Name	**Attribute Type**	**Description**	**Constraint**
id	@id	Provides a unique identifier for an instance of the entity either in the form of a URI (i.e. either a publicly accessible URL or a URN).	Mandatory
type	@type	Defines the type of the entity. In this case sector.	Mandatory
timestamp	DateTime	Indicates the date/ time when the entity was last observed in ISO 8601 format. The value of this will be set by the server when the entity was observed, if the entity has not been observed it may have a null value.	Optional
name	Text	Indicates the name of sector.	Recommended
policy	RelationShip	Indicates the parking site policy. This attribute value must contain an identifier of an existing policy entity.	Recommended
policyPHolidays	RelationShip	Indicates the parking site policy (public holidays). This attribute value must contain an identifier of an existing policy entity.	Optional
location	geo:json	The location point of the parking meter. See GeoJSON Specification (RFC 7946).	Recommended

Table 6: Sector attributes

9.1.1.1 NGSIv2 Context Definition

The following NGSIv2 context definition applies to the sector entity.

Listing 1: smartparking:sector:context

```
"@context": {
    "name": "http://purl.org/goodrelations/v1#name",
    "policy": "http://ontology.eil.utoronto.ca/icity/Parking/
       ParkingPolicy",
    "policyPHolidays": "http://ontology.eil.utoronto.ca/icity/
       Parking/ParkingPolicy",
    "location": "https://schema.org/location"
}
```

9.1.1.2 Example of sector entity

The following is an example instance of the sector entity (NGSIv2 format).

Listing 2: Example of smartparking:sector

```
{
    "id": "urn:ngsi-ld:sector:Sector:1",
    "type": "sector",
    "@context": {
        "type": "StructuredValue",
        "value": [
            "http://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.
               jsonld",
            "https://odins.org/smartParkingOntology/sector-
               context.jsonld"
        ],
        "metadata": {}
    },
    "timestamp": {
        "type": "DateTime",
        "value": "2019-04-29T12:30:00Z",
        "metadata": {}
    },
    "name": {
        "type": "Text",
        "value": "Sector 1",
        "metadata": {}
    },
    "policy": {
        "type": "Relationship", "value": "urn:ngsi-ld:policy:
           Sector:1",
        "metadata": {
```

```
            "entityType": {
                "type": "Text",
                "value": "policy"
            }
        }
    },
    "policyPHolidays": {
        "type": "Relationship",
        "value":"urn:ngsi-ld:policy:Sector:1:PublicHoliday",
        "metadata": {
            "entityType": {
                "type": "Text",
                "value": "policy"
            }
        }
    },
    "location": {
        "type": "geo:json",
        "value": {
            "type": "Polygon",
            "coordinates": [
                    [  [-1.134584, 37.996461], [-1.127803, 37.997983], [-1.126236, 37.995582],
                    [-1.124928, 37.994753], [-1.124348, 37.992825], [-1.125099, 37.991743],
                    [-1.124992, 37.989257], [-1.127395, 37.989561], [-1.130271, 37.989341],
                    [-1.131193, 37.990711], [-1.131386, 37.992098], [-1.134584, 37.996461]  ]
            ]
        },
        "metadata": {}
    }
}
```


9.1.2 Parking meter

Parking meter attributes			
Attribute Name	**Attribute Type**	**Description**	**Constraint**
id	@id	Provides a unique identifier for an instance of the entity either in the form of a URI (i.e. either a publicly accessible URL or a URN).	Mandatory

Attribute Name	Attribute Type	Description	Constraint
type	@type	Defines the type of the entity. In this case parkingmeter.	Mandatory
timestamp	DateTime	Indicates the date/ time when the entity was last observed in ISO 8601 format. The value of this will be set by the server when the entity was observed, if the entity has not been observed it may have a null value.	Optional
name	Text	Indicates the name of parking meter.	Recommended
parkingProbability	Number	Parking probability, between 0 and 1.	Mandatory
sector	RelationShip	Indicates the sector of parking meter. This attribute value must contain an identifier of an existing sector entity.	Mandatory
location	geo:json	The location point of the parking meter. See GeoJSON Specification (RFC 7946).	Recommended

Table 7: Parking meter attributes

9.1.2.1 NGSIv2 Context Definition

The following NGSIv2 context definition applies to the parking meter entity.

Listing 3: smartparking:parkingmeter:context

```
"@context": {
    "name": "http://purl.org/goodrelations/v1#name",
    "parkingProbability": "https://odins.org/smartParkingOntology
       /parkingProbability",
    "sector": "https://odins.org/smartParkingOntology/sector",
    "location": "https://schema.org/location"
}
```

9.1.2.2 Example of parking meter entity

The following is an example instance of the parking meter entity (NGSIv2 format).

Listing 4: Example of smartparking:parkingmeter

```
{
    "id": "urn:ngsi-ld:parkingmeter:Parquimetro:166",
    "type": "parkingmeter",
    "@context": {
        "type": "StructuredValue",
        "value": [
            "http://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.
               jsonld",
```

```
            "https://odins.org/smartParkingOntology/parkingmeter-context.jsonld"
        ],
        "metadata": {}
    },
    "timestamp": {
        "type": "DateTime",
        "value": "2019-04-29T12:30:00Z",
        "metadata": {}
    },
    "name": {
        "type": "Text",
        "value": "BUENOS LIBROS",
        "metadata": {}
    },
    "parkingProbability": {
        "type": "Number",
        "value": 0.310721062618596,
        "metadata": {}
    },
    "sector": {
        "type": "Relationship",
        "value": "urn:ngsi-ld:sector:Sector:1",
        "metadata": {
            "entityType": {
                "type": "Text",
                "value": "sector"
            }
        }
    },
    "location": {
        "type": "geo:json",
        "value": {
            "type": "Point",
            "coordinates": [ -1.130981829, 37.99491059 ]
        },
        "metadata": {}
    }
}
```


9.2 Illegal Waste Deposit

Figure 23 presents main entities and attributes of the use case.

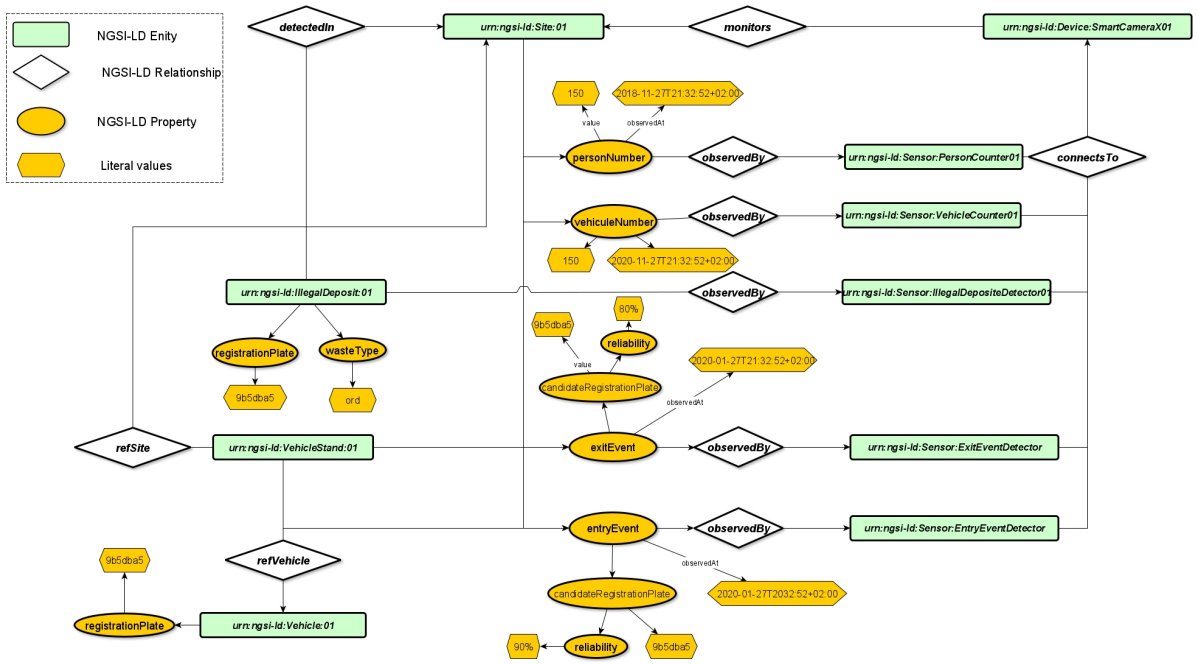

Figure 23: Illegal Waste Deposit Management Data Model

9.2.1 Entities Overview

Entities type and their basic attributes have been defined in Deliverable 4.2. In this section we will focus more on entities and attributes related to the use case pilot.

Illegal Waste Deposit Management	
Entity Name	**Entity Type**
Site	Site
SmartCamera	Device
Vehicle	Vehicle
IllegalDeposit	IllegalDeposit
VehicleStand	VehicleStand
EntryEventDetector	Sensor
ExitEventDetector	Sensor
PersonCounter	Sensor
VehicleCounter	Sensor
IllegalDepositDetector	Sensor

Table 8: Illegal Waste Deposit Management use case entities

9.2.2 Site Entity Use Case Attributes

Attribute Name	Attribute Type	Description
personNumber	Property	Indicates the number of persons detected by the PersonCounter Sensor installed in the SmartCamera.
vehicleNumber	Property	Indicates the number of vehicle detected by the VehicleCounter Sensor installed in the SmartCamera.
entryEvent	Property	This property is created when a vehicle entry event is detected on the site. The entry event is composed of the candidateRegistrationPlate Property and the reliability. The candidateRegistrationPlate is a set of possible registration plate of an incoming vehicle.
exitEvent	Property	This property is created when a vehicle exit event is detected on the site. The exit event is composed of the candidateRegistrationPlate Property and the reliability. The candidateRegistrationPlate is a set of possible registration plate of an outgoing vehicle.
observedAt	TemporalProperty	Indicates the date/ time that the instance of the value of the property was captured in ISO 8601 format.
observedBy	Relationship	Relates he observed properties to their sources.

Table 9: Site Use Case Attributes

In the following listing an example of a Site Entity with the previous properties is presented.

Listing 5: Example of the Site Entity

```
{
    "id": "urn:ngsi-ld:Site:01",
    "type": "Site",
    "createdAt": "2020-01-01T01:20:00Z",
    "modifiedAt": "2020-05-04T12:30:00Z",
    "personNumber": {
        "type": "Property",
        "value": "150",
        "observedAt": "2020-01-01T01:20:00Z",
        "observedBy": {
            "type": "Relationship",
            "object": "urn:ngsi-ld:Sensor:PersonCounter01"
```

```
        }
    },
    "vehicleNumber": {
        "type": "Property",
        "value": "150",
        "observedAt": "2020-01-01T01:20:00Z",
        "observedBy": {
            "type": "Relationship",
            "object": "urn:ngsi-ld:Sensor:VehicleCounter01"
        }
    },
    "entryEvent": {
        "type": "Property",
        "observedAt": "2020-01-01T01:20:00Z",
        "candidateRegistrationPlate": {
            "type": "Property",
            "value": "9b5dba5",
            "reliability": {
                "type": "Property",
                "value": "80",
                "unitCode":"P1"
                }
            }
        "observedBy": {
            "type": "Relationship",
            "object": "urn:ngsi-ld:Sensor:EntryEventDetector"
        }
    },
        "exitEvent": {
        "type": "Property",
        "observedAt": "2020-01-01T01:20:00Z",
        "candidateRegistrationPlate": {
            "type": "Property",
            "value": "9b5dba5",
            "reliability": {
                "type": "Property",
                "value": "80",
                "unitCode":"P1"
                }
        }
        "observedBy": {
                "type": "Relationship",
                "object": "urn:ngsi-ld:Sensor:ExitEventDetector"
        }
    },

```

```
    "@context": [
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld",
        "https://raw.githubusercontent.com/easy-global-market/ngsild-api-data-models/master/smartCameraUseCase/jsonld-contexts/smartCamera-context.jsonld"
    ]

}
```

9.2.3 Illegal Deposit Entity Use Case Attributes

Attribute Name	Attribute Type	Description
registrationPlate	Property	Provides This property the registration plate of the vehicle that causes the Illegal Deposit.
wasteType	Property	Provides the type of the detected deposit.
detectedIn	Relationship	Relates the detected Illegal Deposit to the correspondent Site.

Table 10: Illegal Deposit Use Case Attributes

In the following listing an example of an Illegal Deposit is presented.

Listing 6: Example of the Illegal Deposit Entity

```
{
    "id": "urn:ngsi-ld:IllegalDeposit:01
    "type": "IllegalDeposit",
    "createdAt": "2020-05-01T01:20:00Z",
    "modifiedAt": "2020-05-04T12:30:00Z",
    "registrationPlate": {
        "type": "Property",
        "value": "9b5dba5",
        "observedAt": "2020-01-01T01:20:00Z"
    },
    "wasteType": {
        "type": "Property",
        "value": "Wood",
        "observedAt": "2020-01-01T01:20:00Z"
    },
    "detectedIn": {
        "type": "Relationship",
        "object": "urn:ngsi-ld:Site:01"
```

```
    },

     "@context": [
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.
           jsonld",
        "https://raw.githubusercontent.com/easy-global-market/
           ngsild-api-data-models/master/smartCameraUseCase/
           jsonld-contexts/smartCamera-context.jsonld"
    ]

}
```

9.2.4 Vehicle Stand Use Case Attributes

The Vehicle Stand is an entity created when an Entry and an Exit Event are detected in a site with the same registration Plate in a time interval. In a site we have a list of Entry and Exit properties with a list of Registration plate and a reliability values. When an Entry and an Exit events have the same or a similar registration plate (according to the reliability values), both properties are deleted from the Site entity and then added to a new Vehicle Stand Entity. The main Vehicle Stand attribute are depicted in Table 11.

Attribute Name	Attribute Type	Description
entryEvent	Property	Same as defined in the Site Entity.
exitEvent	Property	Same as defined in the Site Entity.
refSite	Relationship	Relates the Vehicle Stand to the correspondent Site.
refVehicle	Relationship	Relates the Vehicle Stand to the correspondent Vehicle.

Table 11: VehicleStand Use Case Attributes

In the following listing an example of a Vehicle Stand is presented.

Listing 7: Example of the Vehicle Stand Entity

```
{
    "id": "urn:ngsi-ld:VehicleStand:01
    "type": "VehicleStand",
    "createdAt": "2020-05-01T01:20:00Z",
    "modifiedAt": "2020-05-04T12:30:00Z",
    "entryEvent": {
        "type": "Property",
        "registrationPlate": {
            "type": "Property",
            "value": "9b5dba5"
        },
```

```
        "observedAt": "2020-01-01T01:20:00Z"
    },
    "exitEvent": {
        "type": "Property",
        "registrationPlate": {
            "type": "Property",
            "value": "9b5dba5"
        },
        "observedAt": "2020-01-02T01:20:00Z"
    },
    "refSite": {
        "type": "Relationship",
        "object": "urn:ngsi-ld:Site:01"
    },
    "refVehicle": {
        "type": "Relationship",
        "object": "urn:ngsi-ld:Vehicle:01"
    },
     "@context": [
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.
           jsonld",
        "https://raw.githubusercontent.com/easy-global-market/
           ngsild-api-data-models/master/smartCameraUseCase/
           jsonld-contexts/smartCamera-context.jsonld"
    ]

}
```

9.2.5 Vehicle Entity

Using on the Registration Plate number of a Vehicle, it is possible to query national data bases for more details about the Vehicle. The Vehicle Entity follows the model of Vehicle in https://schema.org/Vehicle.

References

[1] Japan Tourism Agency: White Paper on Tourism in Japan, 2019. [Online]. Available: https://www.mlit.go.jp/kankocho/en/siryou/content/001312296.pdf