Alma Mater Studiorum · Università di Bologna

Scuola di Ingegneria e Architettura
Corso di Laurea in Ingegneria e Scienze Informatiche

Tesi in Data Mining

Design and Implementation of a Data Platform for Stream Analysis: WeLASER as a Case Study

Relatore:
Prof. Matteo Golfarelli

Correlatore:
Dott. Matteo Francia

Candidato:
Francesco Montelli

Sessione: III
Anno Accademico: 2020/2021

Abstract

Precision agriculture is a management strategy of agricultural activities based on data-driven decisions. This enables smarter usage of the available resources (e.g., water and crop) and ensures higher productivity. Following the integration of precision farming with the internet of things, big data, and artificial intelligence, we are witnessing the rise of “Agriculture 5.0”. In this context, WeLASER is a European project that aims to create a system for managing weeding tasks by the adoption of robots equipped with laser technology that recognizes and burns weeds; this prevents the usage of chemical pesticides that can cause environmental damages. Such application involves the joint usage of robotic agents and data from IoT devices (e.g., weather stations) to perform effective weeding tasks. The goal of this thesis is to design, create, and test a data platform that enables the interoperability of IoT devices and robotic agents as well as data-intensive analytics on streaming data. Such data platform provides unified interfaces to collect, integrate, and analyze real-time data as well as to manage historical data.

Contents

1 Introduction

1.1 Precision Agriculture

The purpose of precision agriculture is to transform the information collected from the fields into profitable decisions through a more efficient and informed management, maximizing production [53] and profits [106], while ensuring a more sustainable production from an environmental point of view [29].

The concept of *Agriculture 4.0* arises from the integration of concepts from the world of telematics and data management to the world of precision agriculture to be able to make more informed and data-driven decisions regarding field management through better monitoring. An example of the application of said techniques is the health monitoring of the crops: the traditional approach to check the health of the crop involves the use of visual inspections of fields but this approach is not always practical due to the size of the fields and is limited by the frequency at with which the inspections can be carried out. It is also necessary to remember that visual-only analysis also makes it impossible to identify problems not visible to the naked eye [104].

As a further step in the introduction of ICT technologies into agriculture, it is possible to identify the concept of *Agriculture 5.0* which provides for the introduction of robots and AI systems [131] within normal operations, aiming to reduce the employment of personnel in tedious tasks [17, 18, 19, 107]. Some example of application include the use of data collected from drones to draw insights regarding plant health indices, plant counting and yield prediction or the monitoring of livestock, for example by identify sick animals.

Further information about the growing interest in the sector can be obtained by examining the funding obtained by startups in these sectors, [5] shows how AI and ML-based startups have registered a 450% increase in fundraising in the last years [81], while [1] states that startups involved in smart agriculture have raised over 800 million dollars of investment.

1.1.1 Internet of Things

Internet of Things (IoT) can be defined as *"an open and comprehensive network of intelligent objects that can auto-organize, share information, data, and resources, reacting and acting in face of situations and changes in the environment"* [11].

In recent years, the IoT world has increasingly become a consolidated reality made up of devices capable of interacting with the environment thanks to their ability to perform measurements or interact with it thanks to actuators [67, 122]. Such an approach makes it possible to proactively intervene in the environment while minimizing the involvement of human operators [21].

The use of IoT technologies in agriculture is booming and represents the main driver of Agriculture 4.0 [96], it is estimated that between 10% and 15% of american farmers use IoT solutions to analyze over 1,200 million hectares on approximately 250,000 farms [129], [29] also predicts that, by 2050, a productivity increase of 70% will be reached.

The adoption of IoT technologies for monitoring and data acquisition also allows saving energy and water, as shown by studies conducted by OnFarm[1], the typical farm can reduce the cost of energy from 35$ to 17$ per hectare and a reduction in water consumption of 8% [96].

IoT systems are pervasive and heterogeneous, typically in the projects in which they are inserted it is necessary to introduce Big Data technologies. Referring to the typical characteristics of Big Data projects [41], we can highlight the following properties about the collected data:

- Volume: considering the size of the areas that will be managed, it is inevitable to assume that we could have to handle, potentially, tens/hundreds of sensors also capable of generating high-dimensional data such as video streams generated by HD cameras used for visual monitoring;
- Variety: different sensors will need to be employed for proper monitoring. These heterogeneities in the devices will lead to heterogeneity in the data, both from the point of view of the meaning and of the format;
- Velocity: sensors are always active entities, capable of generating and transmitting data for their life cycle, potentially at high rates;
- Veracity: problems related to data quality can arise, for example, from damage to sensors. Damage is likely since the sensors will be placed in the external environment and will potentially have to remain there for a long time;
- Valorization: adopting these systems can be costly, both in terms of money and the time required for training. We must be able to exploit the data to have a return on the investment made. To enhance the data, these must be processed and transformed into information useful for the decision-making process.

The need to integrate technologies from the BD world was also highlighted in [59], where it is shown they are applied more and more frequently in projects in the agricultural

[1]http://www.onfarm.com/

sector, [69] also cites 34 projects that use Big Data in tasks that, for example, comprise food availability and security, handle weather and climate change, land management and animal-based research.

1.1.2 Robotics

The introduction of robots in agriculture makes it possible to significantly increase productivity and reduce operating costs [99] but, despite exponential growth in the introduction of robots [107], the costs of adoption are very high especially for smaller contexts [35].

Among the applications of robot in agriculture we can notice the introduction of autonomous vehicles to automate cropping operations or the detection of unhealthy plants.

1.2 WeLASER Project

WeLASER is a project founded on the EU societal and environmental needs noted by the European Commission in the call SFS-04-2019-2020 – “Integrated health approaches and alternatives to pesticide use” - which aims to aims to merge technologies from the ICT world like IoT, Big Data, and cloud technologies to build, assess and push into the market a precision weeding equipment based on high-power laser sources and autonomous mobile systems with the main objective of eliminating the use of herbicides while improving productivity and competitiveness; such a system would eradicate health risks and environmental adverse effects associated with the use of herbicides [125]. In Europe, the yearly production of food is guaranteed by using 130 million tonnes of synthetic herbicides every year (apart from other chemicals) which persists in the environment destroying non-target plants, beneficial insects, and produce health effects in animals and humans, including cancer, birth defects and disruption of the endocrine system.

Today it seems that the only alternative to the use of herbicide is mechanical weed control by tillage which is fuel-consuming, aggressive for soil and roots, and increases erosion. Alternative to this solutions are based on thermal effects as spot-flaming or electrical resistance heating but such approaches are not a common solution in the market.

Weeding and cropping require the introduction of technologies that will enable to have control over the environment, the degree of control depends both on technologies and the context on which we operate. For example, [120] illustrates that in some scenarios, such as indoor culture, we can exercise more control than other. As for the devices [120] we can find an interesting distinction between Monitoring Devices and Power Devices, in the first case, we have devices that will constantly monitor some parameters but not require continuous communications while the latter are devices that require a high energy consumption they produce data streams and have the ability to send alerts when certain conditions are met.

WeLASER proposal is to use a plant-individual irradiation via a laser source which could allow the use of thermal treatment with a moderate energy cost which also preserves the soil.

1.2.1 Contribution

This thesis is carried out within the initial phases of the WeLASER project, the goal is to create and test an architecture that allows the interoperability of IoT devices and robotic agents, providing a unified platform for the collection and integration of data for allow the integration of applications capable of carrying out analyzes, even in real time, on the data.

The architecture that will be developed is similar to the concept of Data Platform: an evolution [82, 108] of the concept of Data Lake. This extension is due to the need to go beyond simple data storage. To make the most of the collected data it is necessary to provide techniques and tools on several levels [114]. Needs of this type have been progressively recognized by various suppliers, such as Google and Amazon, which have begun to make available various tools to manage aspects ranging from ingestion, transformation to analysis.

The developed architecture will be used to provide a demo for the WeLASER project, to demonstrate the possibility of supporting the integration of heterogeneous and geographically distributed devices, performing data collection, real-time analyzes, and the dynamic integration of new applications.

2 Background and Related Work

This thesis is placed at the intersection between IoT and robotics systems, the aim is to provide a basis for the development of the WeLASER project, placed in the context of precision agriculture.

This chapter will serve as an introduction to the world of IoT and Robotics in the context of Precision Agriculture, we will describe the state of the art and perform an analysis of the challenges and open problems.

2.1 Internet of Things

Internet of Things (IoT) technologies enable the integration of a large number [4] of sensors and devices capable of communicating in a smart environment, thus enabling an exchange of information [75]. Among the sectors interested in these thematic, as enabling to build intelligent and interconnected spaces and environments capable of merging the physical and virtual world [25, 77], we can identify smart office, smart retail, smart agriculture, smart water, smart transportation, smart healthcare, and smart energy [85, 49, 75]. The spread of IoT technologies is also enabled by the diffusion of technologies from the telecommunications sector that allows for the embedding of wireless transmission tools such as Bluetooth, WiFi, ZigBee, and GSM into the device. These technologies allow transferring data to and from the devices allowing direct integration of the devices in the physical world [75].

Among the fastest-growing trends in IoT, we can highlight the conjunction with Big Data and Cloud environment which makes it possible to exploit more suitable technologies to manage the high amount of data generated and ensure access to greater computing power. In literature, various architectures have been proposed for managing Big Data in conjunction with IoT [75, 49, 32, 132] while in [75] a standard and reusable architecture has been proposed, here shown in Figure 2.1.

IoT Big Data analytics includes the steps involved in the analysis of data produced by sensors [42] to discover trends, unknown patterns, hidden correlations, and new information [40] to assist organizations and their business [102] giving the possibility to make informed and data-driven decisions [119]. The possibility of analyzing unstructured data that otherwise could not be handled with traditional techniques is also of fundamental importance [46].

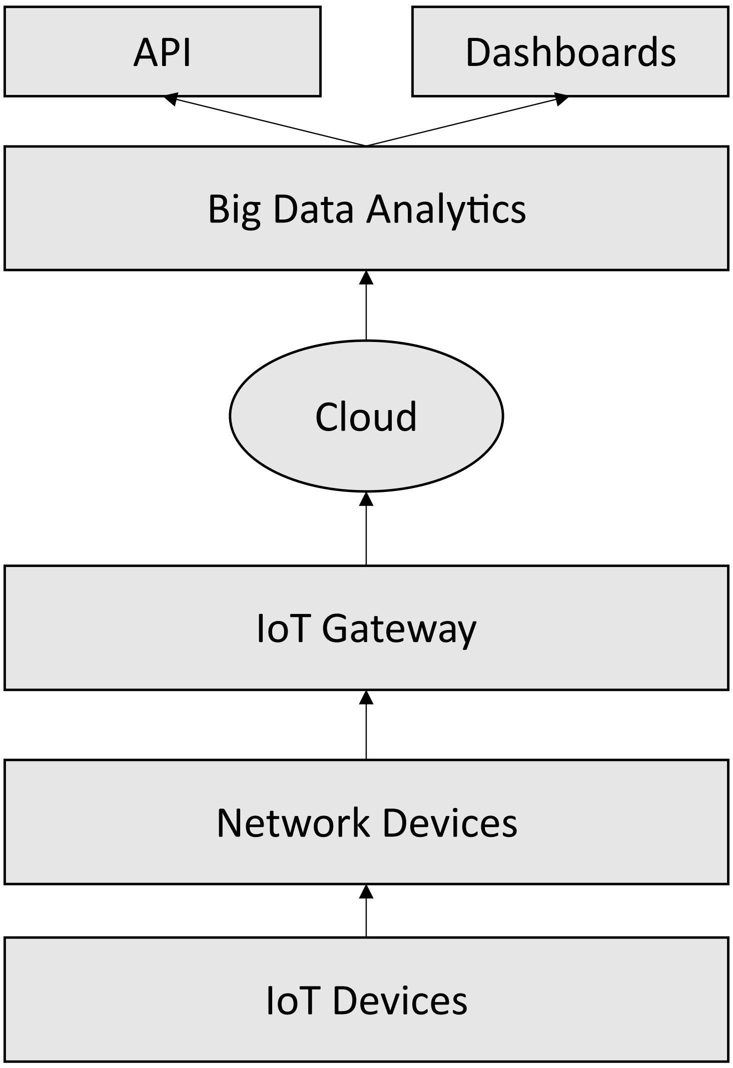


Figure 2.1: IoT and Big Data integration blueprint, adapted from [75]

In Figure 2.1 we can see a representation of the architecture of an IoT system [75]:

- Sensors: entity capable of reading environmental values;
- Actuators: entity able to interact with the surrounding environment;
- Gateway: manages a multitude of sensors and actuators. It handles the flow of data between devices and allows to perform filtering operations on data;
- Controller: manages several gateways and performs high-level processing to convert data into information;
- Middleware: the intermediary between the IoT subsystem and services. At this level, aspects related to privacy and security can be managed. It also enables interaction with cloud services;
- Applications: can interact with the middleware via API. They allow the end-user to interact with the system.

2.1.1 Challenges and Open Issues

Heterogeneity and Integration The creation of a wide IoT ecosystem entails the connection of, potentially, thousands of devices that are typically different from each other for various aspects like different protocols or data formats [16]. To ensure interoperability it is necessary to introduce layers that manage a preprocessing phase to standardize the various formats [31] and heterogeneous sources [3].

In the IoT context variety in terms of format rigidity also comes into play: we can have structured, semi-structured and unstructured data. In the case of semi and unstructured data, it is necessary to provide within the integration phase that allows for a structure to be obtained before proceeding with the integration [2]. One of the most common examples is the use of text mining techniques to extract information such as entities and relationships from a text.

Visualization Having GUI systems to analyze data allows to obtain better insights on trends [124] is paramount to have a nice user experience but creating visualization systems compatible with advanced Big Data indexing frameworks and which guarantee a low response time is a complex task [75]. In [75] we can find a summary of specific needs:

- dimensionality reduction: facilitate the identification of patterns and outliers [13, 95, 30];
- parallelization: the need to decompose a problem in simpler tasks to leverage parallel execution is emerging also for visualization algorithms [24];
- metadata management: metadata integration and visualization with dedicated tools is emerging as a need trend [110, 39].

Management Having to manage a large number of devices also causes problems of an administrative nature, just think of the costs in terms of energy and physical space. Other problems emerge when we consider the increase in possible causes of faults and the need to keep device configurations updated.

Security In 2017 there was a 600% increase in attacks against IoT devices [66] and cybercrime has become the second most reported crime globally [45]. In many cases, the intruders are not directly targeting the IoT device, but aim to use it as a weapon to attack other elements of the infrastucture [6]. IoT systems seem to be easy targets for attackers mostly because manufacturers often place great emphasis on cost, size, and usability, while security aspects tend to be neglected [72], [72] also states that some producers implement security practices mainly because the eventual exploitation of one of their IoT products will damage the company's image, thus demonstrating that safety is not one of the main aspects typically taken into consideration during the developement of new products.

Since sensors can collect a high volume of data, even of personal nature, the violation of an IoT system poses several problems regarding users' privacy. The proliferation of analytic systems is leading more and more people to be reluctant to consent to the use of systems of this type unless they are accompanied by strong service-level agreements [75].

2.2 Robotics

Robots have always been a fundamental element of industrial evolution, intending to facilitate human work, for example, by performing dangerous tasks, like cleaning nuclear sites [116], interventions in dangerous or inaccessible environments, such as maintenance of submarine cables [60], but also, to improve the working conditions of people by relieving them of the need to perform repetitive or stressful tasks. One of the peculiarities of robots is their flexibility, given the possibility to easily reprogram a robot we can use it for different tasks, without the need to redesign it.

Robots can take on varied shapes and sizes but the following components can always be identified [83]:

- Main Body: links, joint, and other structural elements of the robots;
- Actuators: act like muscles in the. Under the command of the controller they are able to activate the actuators which, in turn, allow the joints and links to move;
- Sensors: used to collect information about the outside world and the state of the robot. One of the main necessities is to be able to accurately know the position of each link of the robot;
- Controller: portion of the system delegated to control the movement by interacting with the actuators. Its role is similar to that of the cerebellum in the nervous system;
- Processor: performs calculations to determine how and where to move the robot components. In a nervous system analogy it is the brain.;
- Software:
  - Operating System: manages the processor;
  - Robotic Software: software dedicated to the calculation of joint movements. It interfaces with the controller;
  - Application-Oriented Routines: programs written to control the robot or its peripherals to perform tasks.

A major problem associated with the world of robotics is the limited resources of the single entity [61]. To overcome this limit, starting from the 90s [47, 64], the idea of having a "remote brain" to manage the robots emerged. In recent times these concepts are finding more and more traction, resulting in the conceptualization of *Cloud Robotics*: [78] the interconnection of robots over the cloud to overcome the limitation of the single robot [103].

As highlighted in [103], a survey concerning more than 100 scientific articles from the cloud robotics sector, we can highlight two main research lines:

- Cloud robotics system architecture: the study of architectures and protocols that allow communication between robotic systems;
- Cloud robotics application: application of cloud robotics to typical problems of robotics and automation systems such as computer vision systems, navigation, grasping, or manufacturing.

Several projects [10, 123, 27] aimed at the convergence between the world of robotics and the world of cloud computing to extend the capabilities of the single robot, however with out being able to define an open ecosystem on the line of that Android or Linux. In our case study, we will focus on the architectural aspects.

There are several frameworks for the management of robotic agents, but we will focus on ROS due to its wide diffusion both in academia and the industry [73, 101]. ROS is an open-source library that deals with issues related to the control and deployment of robots. It is often defined as an Operating system since it provides low-level abstractions (e.g. hardware abstraction, low-level device control, commonly used functionality, low-level device control, message-passing between processes) but it is an application that runs on Unix-based platforms.

One of the main features of ROS is its robust graph-based infrastructure that enables *Peer to Peer* (P2P) communication between nodes, each one representing the abstraction of a component. This allows developers to focus on a specific functionality by encapsulating it within the node, effectively creating a service-based distributed system.

In Figure 2.2 it is possible to see an example of an application of the architecture. The following parts can be highlighted:

- nodes: processes that perform a computation. To ensure greater modularity and reuse of components, a robot can be defined as a set of nodes, each dedicated to the management of a particular aspect;
- master: manages node registration, acts as a name server and, propagates configuration changes;
- messages: used by nodes to communicate with each other, a message consists of a data structure and data.

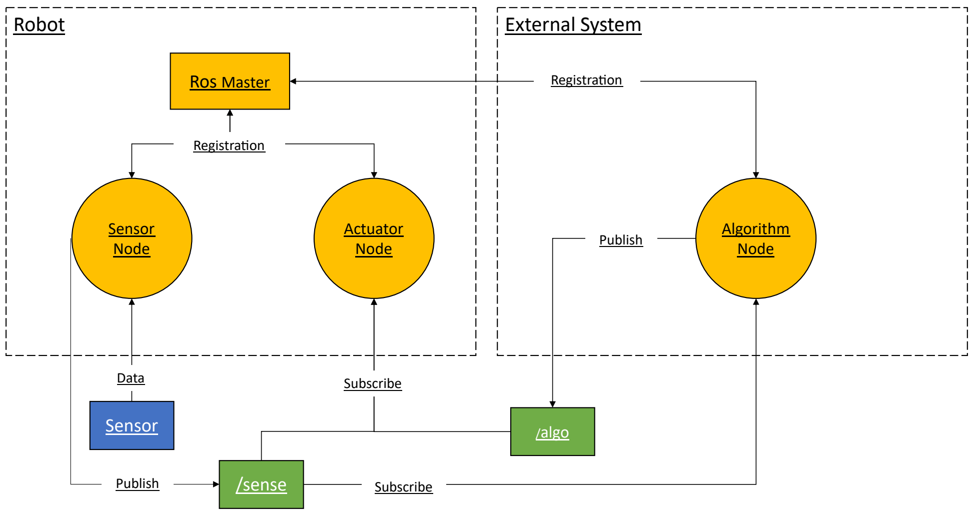


Figure 2.2: Example of a ROS graph, adapted from [118]

- topics: messages are handled with publish/subscribe semantics. The use of this methodology for the exchange of information allows for a high level of flexibility given that publishers and subscribers do not need to know each other, they only need to be aware of the existence of the topics and what information they can receive/publish on them. Even external entities can interact with robots through topics, for example, by reading the data produced and sending commands.

In the example pictured in Figure 2.2 we can see a robotic system composed by two parts: the Robot itself and a generic External System. The robot is responsible to collect data from the sensors via the *Sensor Node* while the *External System* takes care to execute an algorithm abstracted via the *Algorithm Node*. The two system can communicate by using topics: */sense* is the topic reserved to send messages about "sensed" information collected by the sensors, whilst */algo* is used by the Algorithm Node to publish commands that will be executed by the Actuator Node. An application of this example could be a robot capable of moving around the environment thanks to the data collected by the sensor which, at the same time, delegates a more complex part of computatio ton an external service capable of influencing the movement of the robot.

Further analyzing Figure 2.2 it is possible to notice a criticality: despite two very different areas of expertise are involved, namely low-level management of the robot, including the interaction with the sensor hardware, and the integration of algorithms, the same abstraction, ROS node, is employed. A solution of this kind requires the application developer to have knowledge also in the field of robotics to interact with the rest of the system as well as to create interfaces dedicated to interacting with a sub-part of the system.

2.2.1 Challenges and Open Issues

Artificial Intelligence and Deep Learning The use of techniques from the world of AI, in particular Deep Learning and Reinforcement Learning, is increasingly spreading [92], in particular as regards the possibility of having systems capable of adapting to changes in the environment in which they operate [70], however many of the solutions adopted still suffer from problems related to unpredictability and high computational cost. Another challenge to consider is the availability of training data due to the high cost associated with their generation, in addition to the technical difficulties.

In [93] seven challenges are identified:

- Learning complex, high-dimensional and novel dynamics: robots are called upon to manage complex dynamics which, typically, would require the intervention of human experts. The models that emerge from the training phase must be robust, therefore able to function well even in the case of new situations. For example, they must be able to interact with objects never seen before or move on surfaces never seen before;

- Learning control policies in dynamic environment: it is necessary to have control systems available to ensure safety, even of complex robots, such as multi-arm or robot swarm, or when we need to operate them in environments characterized by high uncertainty;

- Advanced manipulation: having a general and robust method to manage the grasping of deformable objects or objects characterized by complex shapes is still the subject of research;

- Advanced object recognition: It is necessary to be able to estimate changes in shape and state in deformable objects. Other tasks of this type include the ability to infer the characteristics of unknown surfaces;

- Interpreting and anticipating human actions: robots need to be able to understand human intentions when they have to interact with human operators in a collaborative context. Among the most advanced applications of this type, we can find “teaching by demonstration”: that is, the possibility of making robots learn new functions through examples;

- Sensor fusion and dimensionality reduction: ability to combine information produced by different sensors to build a unified model;

- High-level task planning: ability to generate an execution plan to carry out a higher level command.

Task Delegation and Safetiness When designing a cloud robotic system we need to choose carefully which tasks delegate to the cloud environment.

First, we need to take into account the trade-off between the energy required to perform the needed algorithm locally and the energy required to transfer all the data needed to perform the remote execution [62].

Another factor to keep in mind is safety: if a robot is entirely dependent on cloud computation a network interruption could cause an hazard on operators and the surrounding environment [62]. If the robot was completely controlled via the cloud, a network outage would result in a loss of control of the robot. Even simple slowdowns can cause damage; if a robot equipped with a proximity sensor relies only on a computation performed in the cloud environment, it may not be able to stop in time in front of an obstacle.

Abstractions ROS has made it possible to take many steps forward in the implementation of projects involving robotic agents, however the abstractions it makes available risk being insufficient in the Cloud Robotics context [73, 117]. The presence of a high number of agents and, potentially, geographically displaced, undermines the ROS model which foresees a small number of robotic agents interacting in an environment.

To work smoothly in this new environment, we need new abstractions for two types of developers: robotic system developers and cloud services developers. Robots developer needs to be able to design and manage robots easily, while continuing to exploit the possibility of reusing ready-made components as was the case with ROS. On the other hand, cloud developers need to be able to interface with robotic systems without having specific knowledge, so that they can focus on specific aspects of cloud projects.

This need also emerges in our case study: we will interface with a robotic agent that performs a task and, at the same time, analyze the data it produces, together with the data collected by a network of sensors.

3 Design

The WeLASER project aims to create a system that allows managing missions performed by a heterogeneous set of robotic agents and IoT devices, such as weather stations and cameras, to provide support for cropping and weeing operations. The ultimate aim of the project is to create a solution that allows the cooperation of the various parts that make up the system to perform the detection of the presence of weeds and then proceed with their removal using a laser beam. To meet these requirement we have designed and built a Data Platform that allows the integration of heterogeneous devices and the dynamic deployment of new applications.

To make the system as general as possible, we have adopted a modeling based on Domains and Missions. Domains are a conceptual collection of missions, that is, a set of tasks that have something in common, for example all the missions carried out by an organization. Missions are carried out by various types of devices while the devices can also be defined as aggregates. In Figure 3.1 it is possible to see the relationships between these elements.

In this chapter, we will detail the requirements and provide an architecture based on the features that must be met and the necessary components.

3.1 Requirements

3.1.1 Device Management

The WeLASER projects requires managing a heterogeneous set of devices like, for example, weather stations, cameras, and robots capable of navigating the terrain.

These devices use different protocols and technologies to communicate. From this fact emerges the necessity to provide an abstraction that masks the peculiarities of the individual devices [91, 34] and enables us to manage the entire life cycle [20] of the devices.

As introduced in Section 2.1.1, we will have to make two different types of interfaces available, designed for two different types of users: the developers of the devices and the developers of the Cloud system [52]:

- Developers of the cloud system should to be able to interact with devices as if they were "black boxes". This means that they must be able to obtain data from

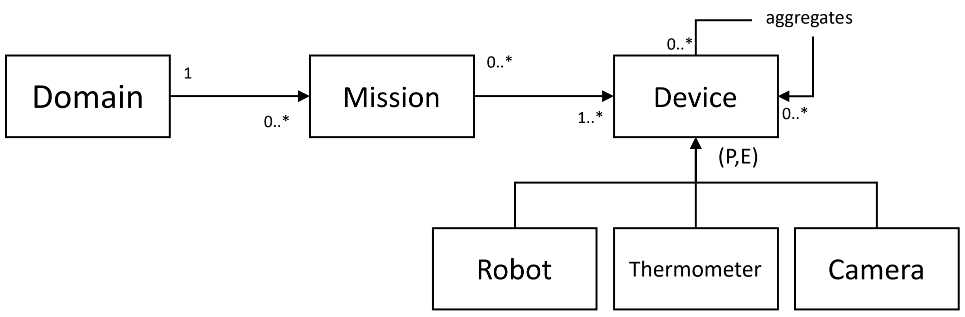


Figure 3.1: Interactions between Domain, Missions, and Devices

their sensors and submit commands without having to know about the underlying mechanism;

- Developers of the devices have very in-depth knowledge in terms of sensors and actuators but may not have in-depth knowledge of Cloud system. They should be able to work without having in-depth knowledge of the cloud systems that will manage the data produced by the devices.

We will also need to manage devices around the world, therefore we need to have a service that allows us to collect information in real-time and to intervene remotely to perform an initial debugging/investigation phase of problems without the need to go on-site, thus reducing management costs [100].

As introduced in Section 2.1.1, we must keep security aspects into consideration when building this kind of system. Devices can be the point of interest of attackers [109] or put at risk the physical safety of the operators who have to interact with them [80]. To deal this aspects we will need to introduce adequate authentication mechanism in order to ensure that only the authorized devices can communicate with the rest of the system.

Requirements regarding Device Management comprehends: device creation, device destruction, device interaction, device authentication, seamlessly robot integration, and remote administration.

3.1.2 Data Management

In the WeLASER project we will have to adjust the behavior of the robot on the basis of the readings made by a heterogeneous set of sensors, both mounted on the robot and scattered in the field. We therefore need a system capable to collect, store and use data in a safe, efficient, and economical way [23]. As stated in Section 2.2.1 this is currently an open issue in the field of Cloud Robotics.

This group of requirements can be detailed in:

- manage a diversified data level;
- archiving of data across multiple clouds and on-premises;

- store data across multiple location, both on the cloud and on-premise;
- ensure high availability;
- make data available to a growing set of applications;
- topological organization;
- archiving and deleting of data based on retention programs and compliance requirements.

3.1.3 Incremental Service Integration

WeLASER will be developed incrementally, adding new applications or services from time to time. This requirement is similar to the concept of *Application Integration*, defined by Gartner[1] as *"the process of enabling independently designed applications to work together"* [28].

By allowing applications to work together and integrating new ones as new needs arise within the project, we can guarantee a more agile evolution and, at the same time, reduce the costs for the introduction of new functions. [128]

Achieving such a goal requires meeting several requirements:

- Make applications communicate: a mean of communication that enables the interaction between applications while providing guarantees about the availability of messages is needed;
- Having a shared data model: to ensure the interoperability of the various applications, it is necessary to have a data model defined and shared by the participants;
- Have well-defined interfaces (API): providing a set of well-defined APIs for each application on the system reduces the time it takes to perform the integration.

The possibility of introduce new functionalities in a dynamic fashion will allow, in the future, the possibility of adding functionality to the robot in the form of algorithms whose computation can be delegated to cloud services. As introduced in Section 2.2.1, delegation aspects are still an open topic in research.

Requirements regarding Incremental Service Integration can be summarized in:

- Each service should be self-contained;
- Services can be added and remove dynamically;
- Services need a well defined interface.

[1]https://www.gartner.com/en

3.1.4 Near Real-Time Data Analysis

Big Data analysis is not just about batch analysis, i.e. complex computation over a large amount of data, but also analyzing *data streams*: simpler computation over smaller sets of continuously incoming data. In the WeLASER project we want to be able to analyze the collected data in near-real time to make decisions about the functioning of the robot and monitor the readings performed by the sensors.

The analysis of streaming data differs greatly from the standard analysis of data due to its specific characteristics:

- Infinite dataset: data is always generated and ingested into the system, this means that only a fraction of the dataset can be kept in memory;
- Infinite computation: the computation is always running and must be able to keep up with the data. We must remember that an overflow may occur if we can't keep up with the stream, we need a plan to prevent this to happen, e.g.: auto-scalability;
- Low-latency and approximate result: approximation may be required to accommodate the low-latency requirement. A timely approximated result, with guarantees about the correctness, is better than a late perfectly correct result.

3.2 Architecture

In this section, we will describe the high-level architecture of the system.

The architecture was derived from that of [75] (see Figure 2.1) with some changes. First of all, the concept of the cloud was not considered as a level but as a pervasive element that encompasses a large part of the architecture. Secondly, the concept of "application" has been revised: we do not consider only an analytic service but a broader context that allows the execution of a dynamic set of applications. Figure 3.2 shows the components of the architecture and how they map to the requirements.

In the following sections we will show how the previously detailed requirements are met by the introduced components.

3.2.1 Device Management

As introduced in Section 3.1.1 we will need to be able to manage the entire life cycle of a heterogeneous set of devices.

With the term "device" we refer to "smart objects" as defined in [11]: any physical(e.g.: board based on micro-controller connected to the network) or virtual(e.g.: a service that performs analysis over a data stream) entity capable of being identifiable, connected to a network, and able to measure, process and act [67, 122]. Devices, therefore, represent the essential elements through which we can monitor and control a remote environment [51].

In this project, we will use ”simulated” devices, i.e. software applications that emulate the behavior of a physical device. This decision will allow us to focus on the architectural aspects without bringing into play the complexities typical of the design and construction of physical devices.

To enable device communication with the reminder of the system we have introduced the “Device Gateway”: a component that, acting as an intermediary, allows communication with cloud services even to devices that cannot do it natively.

The “Device Adapter” layer is introduced to standardize the different protocols and technologies used by the device. Its purpose is to provide the devices with a point on which they can transmit the data they have produced and, at the same time, provide a uniform interaction model with the devices to the higher levels of the system. Various frameworks are available for managing these aspects (e.g.: Amazon IoT [12], Google IoT [48], Azure IoT [14], and Fiware [115]. A comparison between providers can be found in [50].

Common functionality can be summarized in the following points [121]:

- Device Management: in some IoT scenarios the set of devices that will be managed can be very vast, it is necessary to have specific tools to control the devices;
- Data ingestion: acquisition of data towards a repository. It is possible to act by “pull” from the source to the repository or allowing the sources to “push” the data to the repository;
- Data Storage: persistent storage of information. Different solutions can be employed depending on the context of the problem (e.g.: relational DBs, non-relation DBs, Data Lakes);
- Data Processing: knowledge extraction starting from raw data, in some cases the processing can also start during the collection phase, for example by aggregating different readings to be able to identify and correct spurious readings and reduce the volume of data transmitted.

3.2.2 Data Management

The first component we encounter at this level is the *Device Adapter*: the responsible for transmitting the data it has collected and processed to the *Integration Middleware*, a hub through which users and applications can interact with a representation of the system as a whole. Through this architecture it will be possible to access a complete and integrated representation of the status information, but also to submit commands to the devices which, thanks to the mediation of the device adapter, can be executed in a completely transparent way, thus abstracting away the different characteristics of the devices, thus provide a viable solution to the problems illustrated in Section 2.2.1.

A middleware can be defined as software that allows the interconnection of applications in a distributed system [15]. From the developer's point of view, the presence of middleware allows the development of applications without having to worry about writing a specific integration for each application, thus reducing the time and costs necessary for the development of new features. In our case, the middleware will make it possible to maintain the current state of the devices, using a coherent and homogeneous format, and publish this information on the event bus so that such data can be used by the various applications, as described in Section 3.1.2

Due to the large volume of data collected by organizations, it has become more and more complex to use traditional data-warehouse systems for integration and analysis [44], for meeting this new need the concept of *Data Lake* (DL) emerged. DL can be formally defined as *"a central repository system for storage, processing, and analysis of raw data, in which the data is kept in its original format and is processed to be queried only when needed. It can store a varied amount of formats in big data ecosystems, from unstructured, semi-structured, to structured data sources."* [26], but they can also employ mechanisms like metadata to describe and manage the data to facilitate its consumption [113].

DL differs from the more traditional Data Warehouse model by using a flat architecture, where each element has a unique identifier and a set of associate metadata [79], while addressing a set of common problems related to Big Data:

- Various types of data: different information have to use a different format, varying from structured to semi-structured and, recently, unstructured data [7];
- Frequent updates: data changes constantly, and keeping pace with such velocity poses significant challenges. Traditional solutions which require the use of a fixed schema now are posing a threat to the ability to adapt to newer representation;
- Large amounts of data: a single machine is no longer capable to handle all the required volume of data, particularly if time series came into play.

Solutions like DL provides a high degree of flexibility, required to address the characteristic veracity and variety of Big Data, but an adequate level of maintenance is needed to avoid drift of the quality, resulting in the so-called "Data Swamp" [89].

Today cloud data platforms provide several tools for operating ingestion, transformation, and analysis on data but, as described in [39], what is lacking is smart support to handle descriptive metadata and appropriate mechanism to maintain them despite various projects that have, partially, address the problem [74, 105, 97, 88]. Metadata are information about the actual data (e.g. schema, information, semantic, and provenance) which ensure that data can be easily found, trusted, and used, they can be automatically extracted during the ingestion phase and collected in a catalog and classified in different categories [98]:

- Intra dataset: describe a single dataset;
- Inter-dataset: describe the relationship between datasets;
- Global: contextual layer on the data.

The collection of metadata and its correct management brings numerous advantages like:

- Object profiling and search: Searching, selecting, and describing the objects stored in the DL; even perform a complex query based on schema description;
- Provenance and versioning: describe at different levels of detail the objects, the transformations, and relationships;
- Orchestration support: we can trigger a workflow based on some properties stored in the metadata repository.

3.2.3 Incremental Service Integration

In order to dynamically integrate new application to the system, as described in Section 3.1.3, we leverage an “Event Bus” fed with the data collected by the “Integration Middleare”.

The presence of the integration middleware ensure a common representation of the information while the presence of an event bus allows the different applications to communicate and access the data they need to operate without worrying about the presence of the underlying levels and avoiding the use of point-to-point communication between the various applications, thus simplifying interactions. We will use an event bus based on the concept of publish-subscribe in order to allow applications to logically group the messages they send and to be able to receive only the messages they are interested in.

This kind of deploy can be seen as microservice-based deploy, architectural style that structures an application through a collection of small and independent services [127] where each microservice is oriented to the resolution of a specific problem, this allows to have an excellent decomposition. These features make the implementation faster, since it is possible to build new features by relying on other components already present in the system, and the provision of these features to users, thanks to the greater ease of deployment.

The primary purpose of this organization is to ensure the independence of services and facilitate their evolution [22]; furthermore, from a cloud perspective, it is interesting to note that the independence of the various services allows being able to scale the resources assigned to each one independently, making them particularly interesting for deployment in cloud contexts [71].

The adoption of microservices architectures is also linked to the increasingly frequent adoption of AGILE design methodologies [76] and the DevOps techniques [63], typically a team can be fully responsible for the management of a service, ranging from design to deploy. Its also worth notice that the adoption of this architectural style comes with some hurdles [130], most notably is the need to define the correct decomposition granularity and management of the deployment of these solutions.

3.2.4 Neal Real-Time Data Analysis

Given the presence of an event bus, it is possible to integrate applications that allow real-time analysis by simply deploying and starting to consume from the event bus.

Thanks to a cloud deployment we can take advantage of auto-scaling mechanisms to guarantee processing with low latencies, as requested in Section 3.1.4.

At the moment three services that leverage real-time analysis have been identified:

- Dashboard: we want to provide a dashboard through which the user can visualize the current status of the devices, issue commands and interact with other services;
- Average Temperature: a mission can be made up of several devices capable of performing similar readings. The purpose of this service is to calculate the average temperature of the sensors in a given field. In this way, it will be possible to obtain a more precise reading, even in the event of the presence of defective sensors;
- Collision Detection: since several elements will have to move in the same physical space, it will be important to avoid collisions. For this purpose, a collision detection service has been provided. This service will monitor the movements of the devices by generating a notification if two devices are too close.

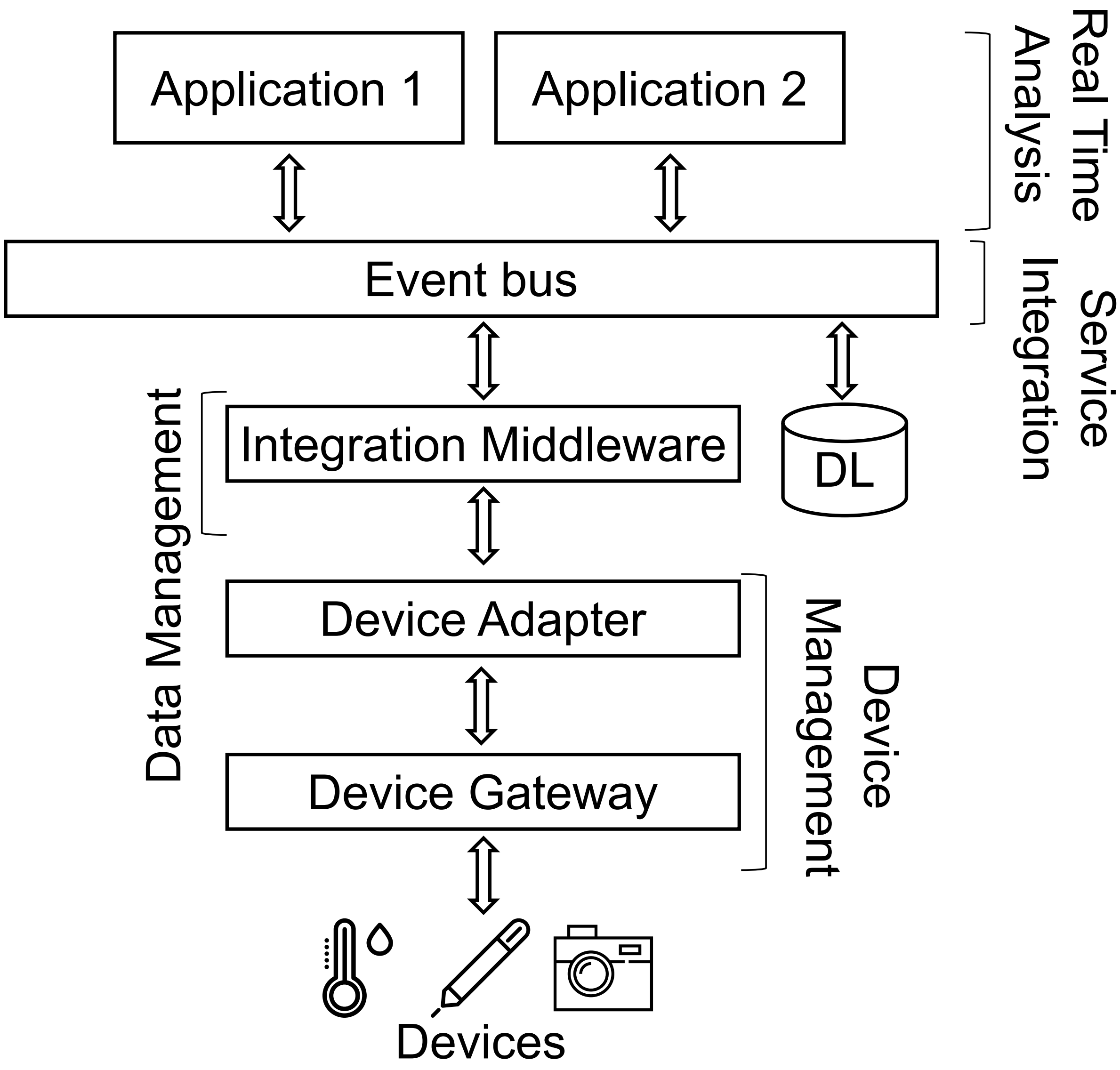


Figure 3.2: Architectural blueprint

4 FIWARE-based architecture

The purpose of this chapter is to discuss a viable implementation of the architecture presented in Section 3.2. The architecture is organized around FIWARE, since this framework is a technological requirement stated in the grant agreement of the project.

We will introduce FIWARE, one of the main technologies underlying the implemented architecture, to give an overview of remarkable aspects of this technology and the interaction between the main components used.

For each function, the components that contribute to satisfying the involved needs will be introduced. For each component, we will describe its peculiar aspects and the interactions it has with other elements of the architecture.

In the end, services related to Analytics on Stream Data will be detailed.

4.1 Introduction to FIWARE

FIWARE [115] is an open-source, cloud-based infrastructure for IoT platforms founded by the European Union and the European Commission with goal of "[...] build an open sustainable ecosystem around public, royalty-free and implementation-driven software platform standards that will ease the development of new Smart Applications in multiple sectors". The FIWARE project comprises both a set of standards based on NGSI-2 [36] and a rich catalog of components called General Enablers (GEs).

In Figure 4.1 we can see the interactions between the components that we will introduce, while in the reminder of this section we will give more details about each of them.

4.1.1 Orion Context Broker

Orion Context Broker (OCB) manages the context, a representation of the current state, and acts as a broker between context producer and context consumers by providing a rich set of HTTP API through which, for example, we can query the context and create subscriptions that will be triggered upon certain conditions. The presence of a Context Broker is the mandatory requirement to have a "Powered by FIWARE" platform. In Figure 4.2 we can see the main components of OCB, each of them will be further detailed.

Context The context is a representation of the current state of the system [87]. Within the context we can find information relating to both real devices, such as a temperature

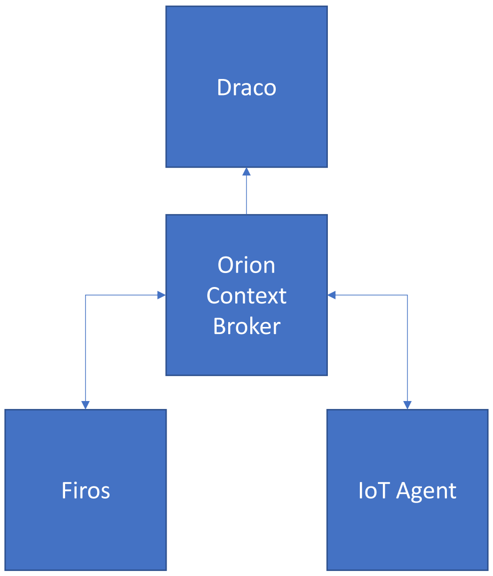


Figure 4.1: FIWARE components used and their interactions

sensor, and virtual entities, for example an aggregation of readings from multiple sensors.

For each entity that makes up the system we will have a corresponding representation within the context, thanks to this representation we can have a single point to query the system about the current state of a component or submit the execution of commands.

We can identify two main advantages in this representation:

- simplicity: everything is represented via entities composed by attributes;
- flexibility: entities not necessarily model things from the real world but can also model things in the virtual world, for example, an alarm. Real and virtual entity will share the same abstraction, i.e. they will be indistinguishable as far as representation is concerned.

FIWARE provides two abstractions to allow interaction with the context: producers, entities able to create and update information within the context like sensors, and consumers, entities that need to access context data in order to perform a task. The boundary between these two types of elements is often not well defined since an entity can act at the same time both as a producer and as a consumer. An example could be smartphones that periodically transmit information about its usage and, at the same time, receive information that will be displayed to the user.

Subscriptions Context consumers, in addition to directly querying OCB, can take advantage of the Subscription mechanism to automatically receive relevant information when certain conditions are met. Subscriptions can be created by sending a specific request, containing details about the conditions of interest and the HTTP endpoint that must receive the notification, to the OCB.

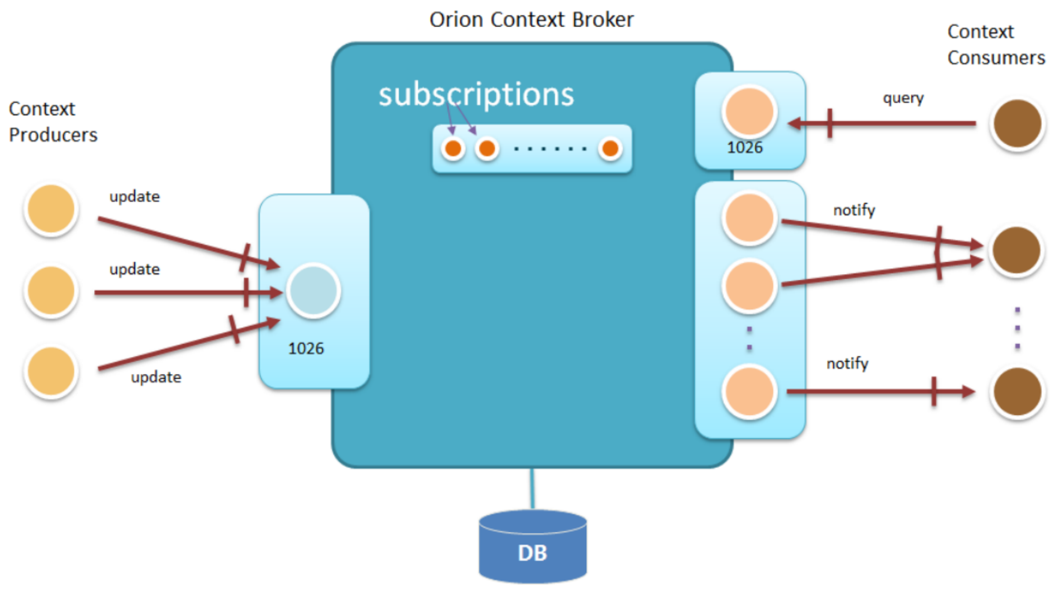


Figure 4.2: Orient Context Broker architecture

```
curl -iX POST \
  "http://{ORION_IP:ORION_PORT/v2/subscriptions" \
  -H 'Content-Type: application/json' \
  -H "fiware-service: openiot" \
  -H "fiware-servicepath: /" \
  -d '{
  "description": "Notify Draco of all device changes",
  "subject": {
    "entities": [{ "idPattern": ".*" } ]
  },
  "notification": {
    "http": { "url": "http://'${DRACO_IP}':'${DRACO_PORT_EXT}'/v2/notify" }
  }
}'
```

Listing 4.1: Subscription registration example

In Listing 4.1 we can see an example of the creation of a subscription via an HTTP request made with `curl`. The subscriptions are comprised of two main parts: subject, the trigger condition, and notification, the endpoint that will receive the notification.

Through `subject`, various parameters can be specified, for example:

- device type: it is possible to limit the validity of a condition to a specific type of device;
- idPattern: using this parameter it is possible to limit the scope of validity of a subscription to a group of specific devices. For example, if we know that a certain type of sensors produces a payload that requires particular management procedures,

we can give them a deviceID that follows a particular pattern and use the same pattern when defining the subscription;

- condition: through condition we can more precisely define the changes that must occur for a given parameter. For example, it is possible to define two subscriptions for the thermometers, one that manages the ”low temperature” case and one for the ”high temperature” case.

Service Group *Service Groups (SG)* are logical groupings of devices, they can be used to define scopes for entities and subscriptions. When defining an SG it is necessary to define the corresponding `apikey`: an authorization credential used by the devices to interface with the group to which they belong. It is also necessary to indicate the context broker to which the group will refer and the default type of entity that will be managed. The last configuration parameter is `resource`, in the example it has been set as an empty string because we are managing MQTT devices; in the case of HTTP devices, it will indicate the endpoint with which they will communicate with IoTA. In Listing 4.2 can see an example of SG creation.

```
curl -iX POST \
  'http://localhost:4041/iot/services' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
 "services": [
   {
     "apikey":      "4jggokgpepnvsb2uv4s40d59ov",
     "cbroker":     "http://orion:1026",
     "entity_type": "Thing",
     "resource":    ""
   }
 ]
}'
```

Listing 4.2: Service Group creation example

As you can see in Listing 4.2 the request is sent to the IoTA specifying also two additional headers: `service` and `servicepath`. The first header is used to identify the name of the group and, consequently, the name of the collection that will be used to store the context information of the devices belonging to the group. The second header option is used to define an hierarchical decomposition of groups. Lastly, `entity_type` provides a default type for each device which has made a request, in case if a type is not provided by the device.

4.1.2 IoT Agent

IoT Agent (IoTA) is a component that lets a group of devices interact with the OCB. Different devices use different protocols, with IoTA we can overcome those differences and connect the various devices to the context broker. IoTA, like OCB, uses the NGSI-V2 [36] specification to ensure interoperability and provide and HTTP API.

It is possible to have multiple IoTAs in the system, one for each protocol used by the devices, some examples are:

- IoTAgent-JSON [55] - a bridge between HTTP/MQTT messaging (with a JSON payload) and NGSI;
- IoTAgent-LWM2M [56] - a bridge between the Lightweight M2M protocol and NGSI;
- IoTAgent-UL [57] - a bridge between HTTP/MQTT messaging (with an Ultra-Light2.0 payload) and NGSI;
- IoTagent-LoRaWAN [58] - a bridge between the LoRaWAN protocol and NGSI.

IoTA provides two different types of communications interfaces: Southbound and Northbound. The first interface responds to the need to submit commands to devices while the second allows devices to be able to publish the information they have acquired within the context.

Southbound interface - Commands This interface enables communications that flow from the context to the devices, thus allowing to request devices to perform certain actions.

When someone requests to execute a command on the device the OCB will route that request to IoTA which will then connect to the device and request the actual execution. Upon completion, the context is updated to reflect the completion.

In Figure 4.3 we can see a visual representation of the southbound traffic:

1. The user performs a request to OCB to execute a command on a device;
2. OCB recoves the entity from the context;
3. OCB notes that the entity is managed by IoTA so sends a new requests to IoTA to invoke the command;
4. IoTA receives the request and passes it to the device; in the meantime OCB is informed that the command is pending;
5. IoT Device performs the action associated to the command then returns the result to IoTA;
6. IoTA passes the result to OCB;
7. OCB updates the context.

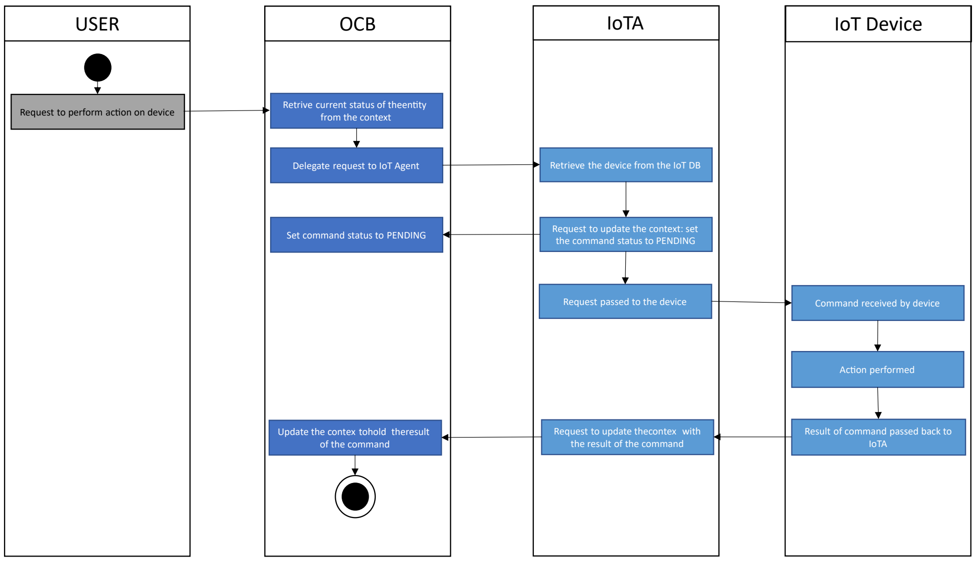

USER
OCB
IoTA
IoT Device
Request to perform action on device
Retrive current status of theentity from the context
Delegate request to IoT Agent
Retrieve the device from the IoT DB
Set command status to PENDING
Request to update the context: set the command status to PENDING
Request passed to the device
Command received by device
Action performed
Update the contex tohold theresult of the command
Request to update thecontex with the result of the command
Result of command passed back to IoTA

Figure 4.3: Southbound traffic example, adapted from [37]

Northbound interface - Measurements Devices can autonomously update the context information with the mediation of IoTA. Devices will send the newly available readings to IoTA which will, in turn, connect to OCB which will update the context, making the information available to the other components of the system.

In Figure 4.4 we can see a visual representation of the northbound traffic which can be summarized in:

1. IoT Devices send measurements and send the results to IoTA;
2. IoTA receives the update and passes the result to OCB;
3. OCB performs a context update.

4.1.3 Draco

FIWARE Draco is a project derived from Apache NiFi [9] to which it adds some operations designed to make it easier to interact with the FIWARE ecosystem. We will refer to the system with the name “Draco” but all the considerations made here are valid for both products. Draco is a data ingestion platform that can collect data from different sources, perform computations, and send the results to other systems.

Draco is based on the concepts of the Flow-Based programming [38], which is the definition of computation through a network of “Black Bock” processors that execute specific operations. The application is not seen as a single entity but as a succession of processes between which data is moved. In the context of Draco, the data is represented by FlowFile. A FowFile correlates a set of Attributes, typically a unique identifier, name, size, and other user-definable attributes, with data.

As we can see in Figure 4.5, Draco architecture is composed of a few components:

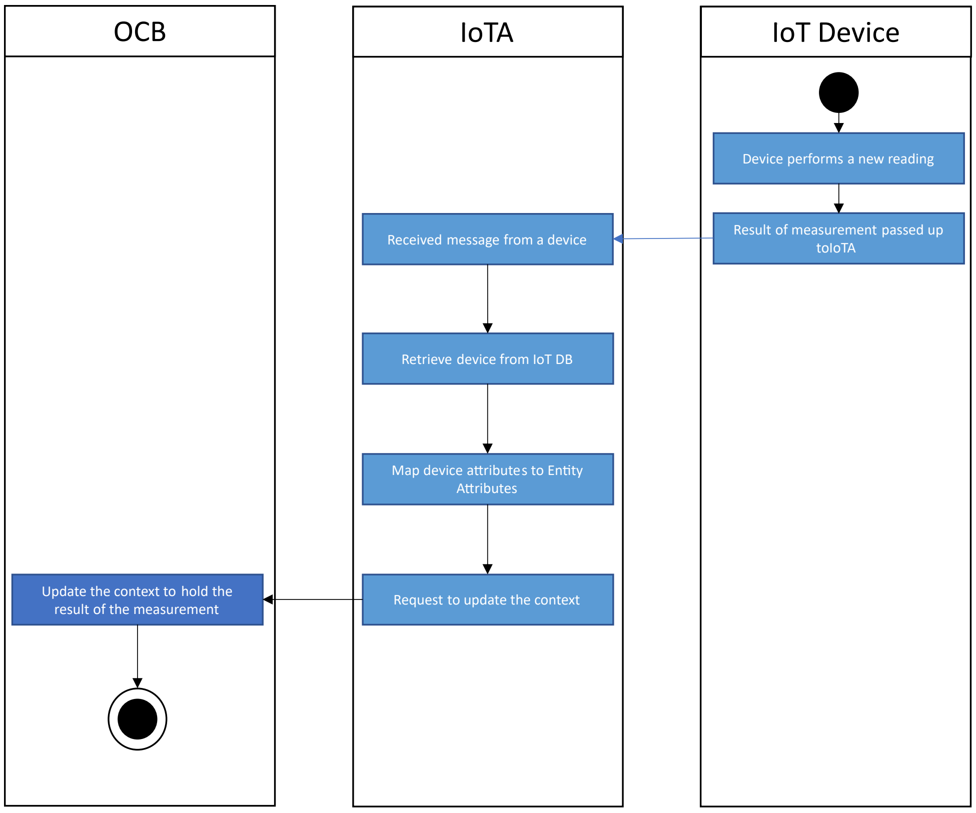

OCB
IoTA
IoT Device
Device performs a new reading
Result of measurement passed up toIoTA
Received message from a device
Retrieve device from IoT DB
Map device attributes to Entity Attributes
Request to update the context
Update the context to hold the result of the measurement

Figure 4.4: Nortbound traffic example, adapted from [37]

- FlowFile Repository: a "Write-Ahead Log" (or data record) of the metadata of each of the FlowFiles that currently exist in the system. The FlowFile metadata includes all the attributes associated with the FlowFile, a pointer to the actual content of the FlowFile (which exists in the Content Repository) and the state of the FlowFile, such as which Connection/Queue the FlowFile belongs in. This Write-Ahead Log provides Draco the resiliency it needs to handle restarts and unexpected system failures. The FlowFile Repository acts as Draco's Write-Ahead Log, so as the FlowFiles are flowing through the system, each change is logged in the FlowFile Repository before it happens as a transactional unit of work, this allows the system to know exactly what step the node is on when processing a piece of data;

- Content Repository: where the content of all FlowFiles are stored, typically is the largest of the repositories. The Content Repository holds the FlwoFile's content on disk and only read it into JVM memory when it's needed;

- Provenance Repository: where the history of each FlowFile is stored. The history is required to provide the Data Lineage of each piece of Data, every time an event occurs to a FlowFile a new provenance event is created and stored. The provenance event is handles as a snapshot of the FlowFile, after being stored, it will not change until its deletion. The Provenance Repository holds all of these provenance events for a period of time after completion;

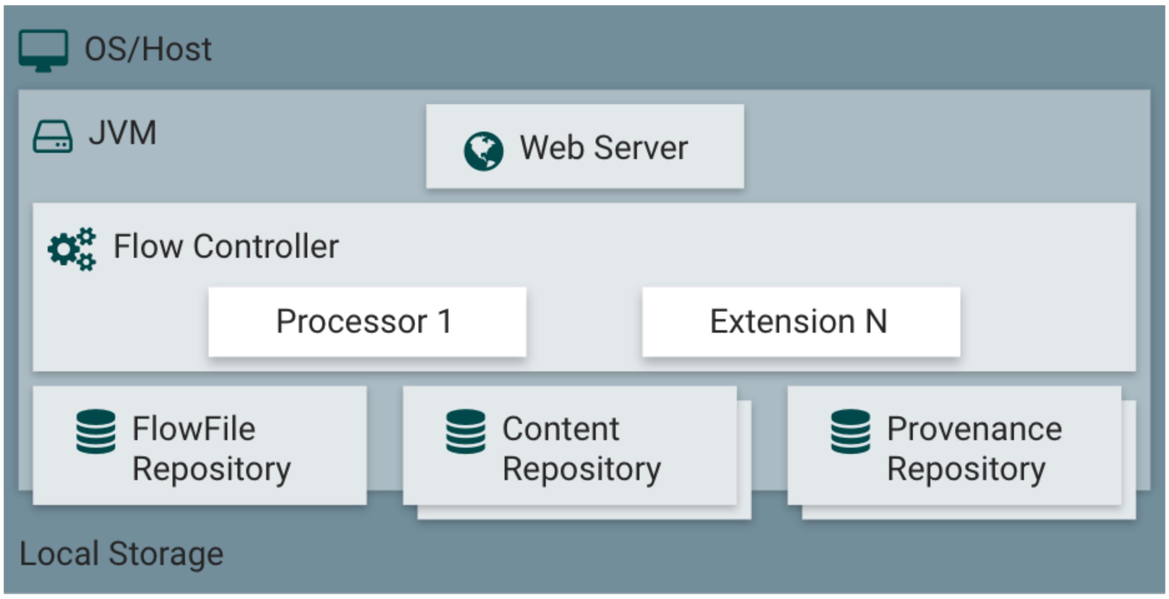


Figure 4.5: Draco architecture from NiFi official documentation

- Webserver: provides a web interface to interact with the system. Thanks to this interface we can define the flow which will handle the data and monitor the status of the applications;
- Flow Controller: runs the computations:
- Extensions: provides the execution environment for extensions.

All the repositories (FlowFile Repository, Content Repository, Provenance Repository) leverage a pluggable implementation, in this way we can transparently use different technologies for storing the data.

In Figure 4.6 we can see an example of a flow configured via the Web interface:

- NGSIV2-HTTP-Input: acts as an HTTP endpoint which receives messages from OCB through subscriptions;
- NGSITOMySQL: process incoming subscriptions and saves their content into a MySQL DB specified inside the block's configuration:
- LogAttribute: saves logs.

As shown in Figure 4.7, Draco can also be deployed inside a Cluster. In this case, every node executes the same operation but on a different subset of the data. Apache Zookeeper is used to elect the coordinator and every node reports heartbeat and status information to FIWARE. A primary node is also elected. When deployed in "cluster mode" we can interact with every instance, any modification will be replicated across all nodes.

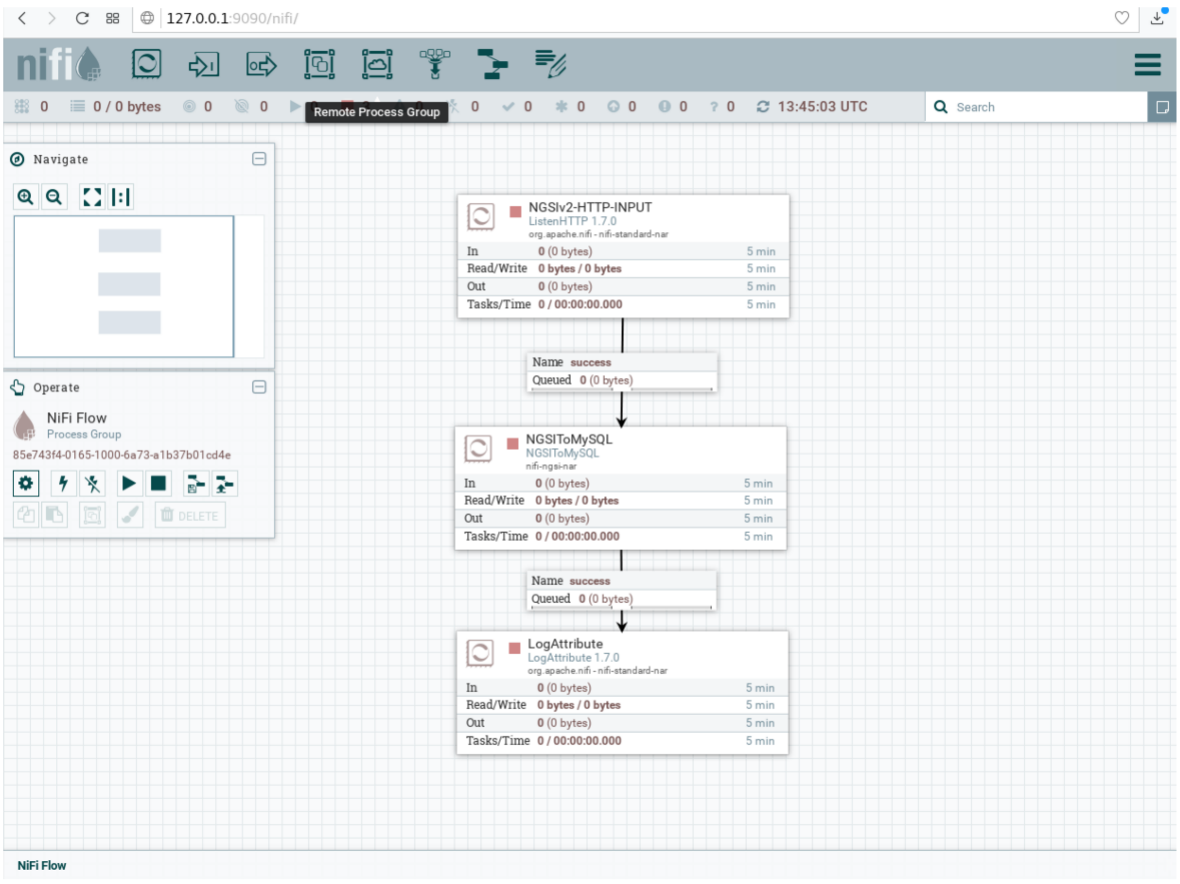


Figure 4.6: Draco flow example

4.2 Architecture

In this section we will describe the FIWARE-based architecture we have developed. First an overview will be provided to describe the overall functioning, then we will proceed with a discussion aimed at the individual components.

In Figure 4.8 we can see the proposed architecture based on FIWARE. The implementation of the architecture follows what is described in Chapter 3 but it was necessary to introduce a new component, Draco, to allow the extraction of the information collected by the integration middleware (FIWARE Context Broker) and move it to the event bus (i.e.: to perform data routing).

Each component of the architecture was deployed using Docker containers, this allowed us to maintain a high level of isolation between the parts and, at the same time, ensured us the ability to easly replicate the execution environment.

The colors used in Figure 4.8 indicate:

- blue: components developed using components from the FIWARE catalog;
- green: the MQTT Broker, acting as Device Gateway for the devices;

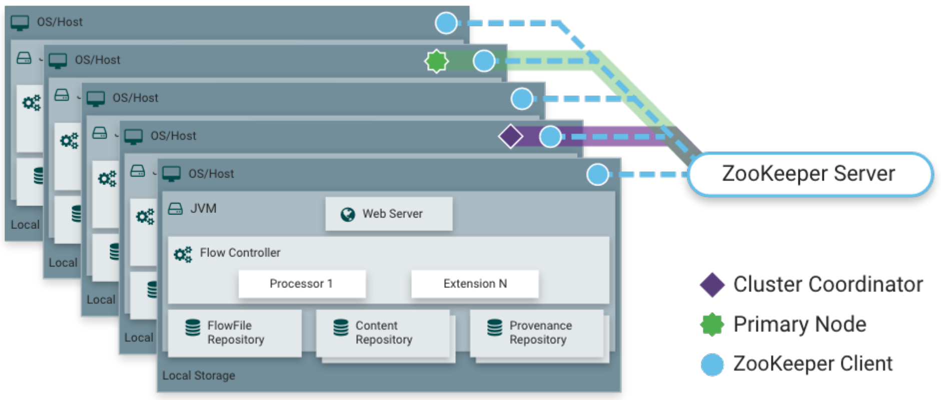


Figure 4.7: Draco Cluster architecture from NiFi official documentation [9]

	Device Management	Data Management	Service Integration	Stream Analysis
MQTT Broker	✓			
IoT Agent	✓			
FIROS	✓			
OCB		✓		
Draco		✓		
Kafka			✓	
Visual Dashboard				✓
Mission Manager	✓			
Domain Manager				
Collision Manager				✓
Replay Manager				✓

Table 4.1: Summary of components and features. The cells in which there is a checkmark indicate that the component contributes to satisfying the functionality.

- grey: storage solution, realized with MongoDB;
- red: Kafka event bus;
- yellow: developed services.

In Table 4.1 we can see a summary of the functionality and the involved components.

4.2.1 Device Management

The purpose of the components linked to the device management functions is to allow the integration of the devices with the system, abstracting the differences between technologies and protocols, ultimately allowing the interoperability of different devices and the transferring of collected data to the upper levels.

Devices

Two types of devices have been simulated: thermometers and cameras. Thermometers can transmit the readings of a temperature sensor whereas the cameras are in charge

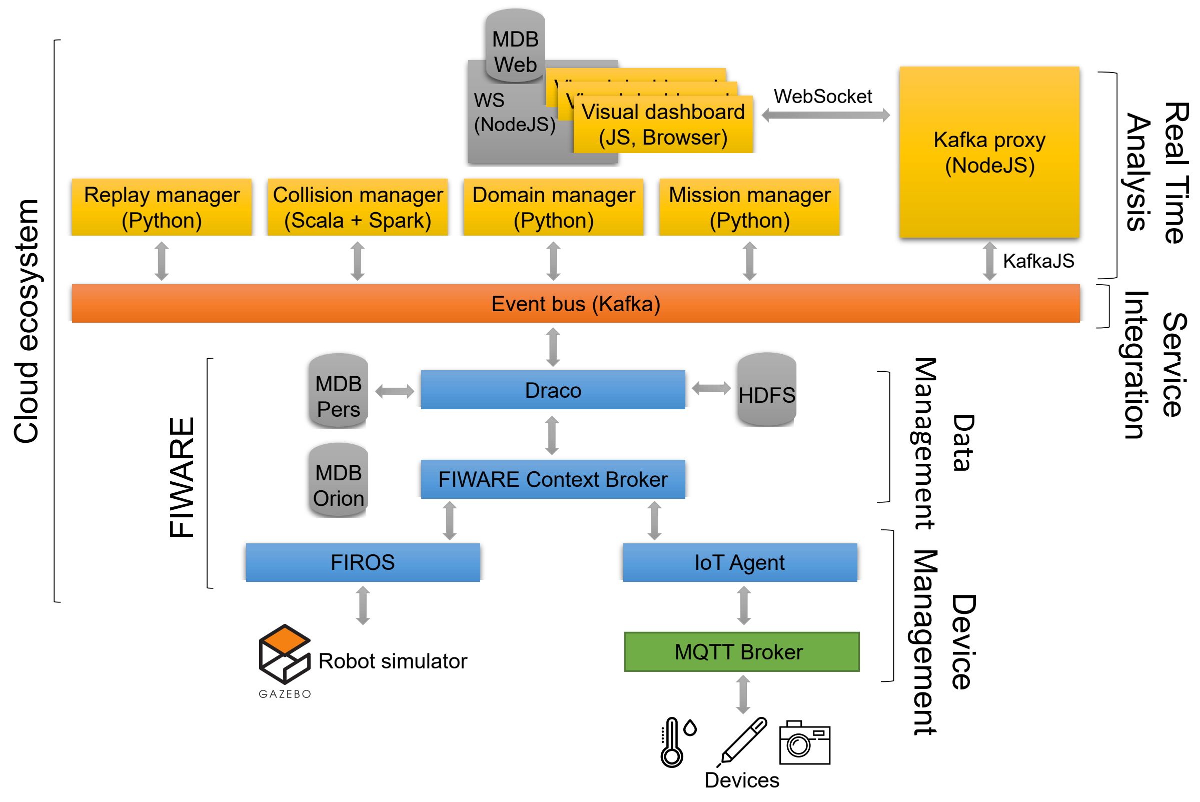


Figure 4.8: Final architecture based on Fiware

of periodically transmitting an image. In addition to these peculiarities, both types of devices can transmit information about their position (latitude and longitude) and execute basic commands like on and off. In both cases, devices have been realized using JavaScript for the implementation and the MQTT protocol to handle communications.

To allow a device to communicate with the rest of the infrastructure, first of all, a registration request must be sent to IoTA, in Listing 4.3 an example.

```
curl \
  -iX POST \
  "http:/IoTA_IP:IoT_PORT/iot/devices" \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "devices": [{
    "device_id":   "camera1",
    "entity_name": "urn:ngsi-ld:camera:1",
    "entity_type": "Camera",
    "transport": "MQTT",
    "commands": [
      { "name": "on", "type": "command" },
      { "name": "off", "type": "command" }
    ],
    "attributes": [
      { "object_id": "img", "name": "Image", "type": "String" },
      { "object_id": "stat", "name": "Status", "type": "Boolean" },
      { "object_id": "time", "name": "Time", "type": "Integer" }
    ]
  }]
}'
```

Listing 4.3: Device registration example

Looking at the requests we can highlight some interesting aspects.

First of all, `fiwre-service` and `fiware-servicepath` are sent as headers, as described in Section 4.1.1, to specify which grouping the devices belong to.

Another interesting point is the configuration of the attributes. For each attribute two identifier,`object_id` and `name` are specified, this solutions allows to map the names of the variables used by the device to transmit the data in the names of the changes on the integration middleware side (OCB). This allows to use compact names within the devices to reduce bandwidth consumption and more meaningful names within the context to allow users to interpret the information represented more easily.

Finally, we can notice that two identifiers, `device_id` and `entity_name`, are supplied for the device. This need arises from the fact that it is necessary to be able to identify the device both from the point of view of OCB, with `entity_name`, and IoTA, with `device_id`.

To be able to receive commands, devices must register as a subscriber on the `/<API-KEY>/<DEVICE-ID>/cmd` MQTT topic. `API-KEY` is the api key of the service group they belong, while `DEVICE-ID` is the identifier of the device (`device_id` in Listing 4.3). The received messages will contains a simple JSON that maps the name of the command into an object containing parameters for the execution.

After executing the command, a message containing the execution result must be sent on the `/<API-KEY>/<DEVICE-ID>/cmdexe` topic. In Section 4.1.2 it is possible to see the complete flow of communication.

Measurements are sent by posting a message on the topic `/<API-KEY>/<DEVICE-ID>/attrs`. The message will contain a key-value pair for each attribute we want to update. In Figure 4.3 it is possible to see the complete flow of information.

MQTT Broker

To put in communication devices and IoTA we use the MQTT protocol, a publish-subscribe-based messaging protocol frequently used in IoT projects. As MQTT broker we used Apache Mosquitto [33] with a custom configuration file which enforces the use of a password for user authentication. Said password is distributed to devices via a .env file and mounted inside the broker's container.

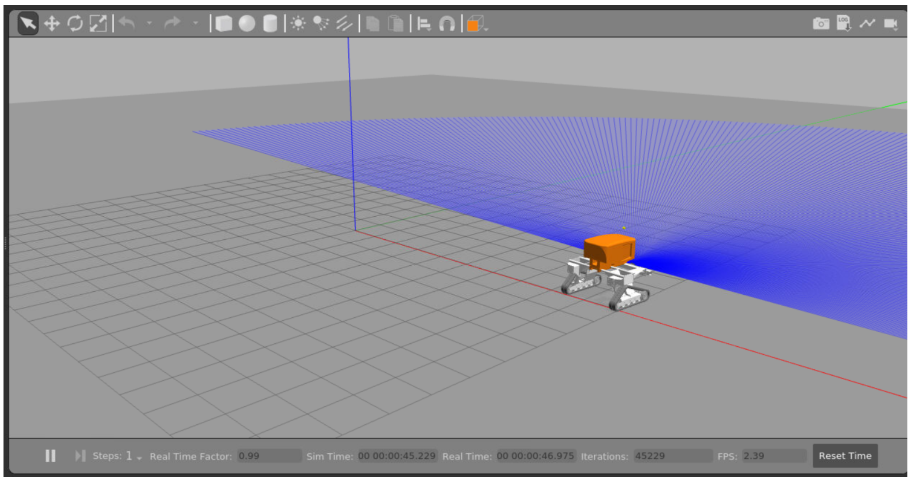


Figure 4.9: Robot simulation environment

FIROS and Robot Simulator

FIROS is a tool developed within the context of the FIWARE project to facilitate communication between ROS-based robots and cloud services through the mediation of FIWARE's OCB.

The main advantage of FIROS is the transparent integration of robots within the ecosystem: once the robot has been correctly registered we can interact with it like any other entity within the system by accessing the collected data and submitting commands. These interactions can take place by hiding the presence of ROS at the robot level as FIROS is in charge of acting as an intermediary. The simulated robotic agents were developed by Centre for Automation and Robotics of the UPM-CSIC [54] and are executed inside a dedicate docker container which also handles ROS and FIROS.

In Figure 4.9 we can see the robot running in its simulation environment.

IoT Agent

IoT agent, already introduced in Section 4.1.2, is used to allow the integration of devices, thus enabling the interaction between devices and the system by providing an abstraction of the presence of different technologies and protocols in the underlying levels.

4.2.2 Service Integration

The purpose of the Service Integration Levels is to provide a platform for integrating new functionality. Thanks to the presence of the event bus, new applications can be inserted dynamically to consume the data produced by the devices.

To develop the event bus we decided to use Kafka, an open-source stream processing platform developed by the Apache Software Foundation [8], this decision was made because of the possibility of easily scaling the solution and to ensure interoperability with

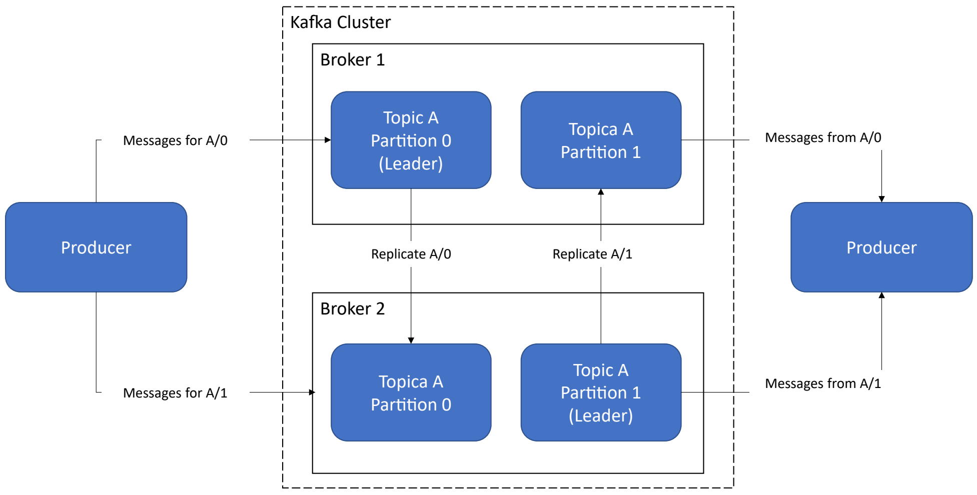


Figure 4.10: Kafka's Architecture

an existing architecture that uses this technology.

In Figure 4.10 we can see the main components of Kafka's architecture:

- Broker: maintains several specialized queues and handles the replication, a mechanism that ensures data availability when a broker goes down. For each topic, a partition is elected a "master" while the others will act as "slaves", when a partition master fails a fail-over mechanism handles the election of a new master to allow the resume of normal operation. Producers and Consumers will interact with the broker which contains the leader partition;

- Producer: can publish messages on topics;

- Consumer: can consume the messages present in the topics they are subscribed to. Each consumer is in charge of maintaining an offset for each partition to keep track of the position of the last consumed message. The offset is periodically transmitted to the broker so that the consumer, in the event of a restart due to a failure, can request it again and resume consuming from where it was interrupted;

- Topic and partitions: the broker can manage several queues called topics, in this way it is possible to group messages concerning the same topic. Kafka allows you to split a queue into multiple partitions to be able to divide it among multiple brokers, this is particularly usefull In Big Data contexts where it is possible that a queue cannot be contained entirely within a single broker.

Each message can be seen as a line of a record containing two elements: a Key (optional) and the Content. The presence of the key ensure that messages with the same key are placed in the same partition.

To ensure better performance, messages are transmitted in groups called batches. It is possible to configure the batch size to adjust the tradeoff between latency and throughput:

as the size of the batch increases we will have a greater throughput but at the expense of a greater latency due to having to wait for a sufficient number of messages before starting the transmission. Compression techniques are also applied to the batch to reduce their footprint during transmissions.

Kafka provides three different semantics for message delivery:

- Exactly once: each message is sent only once and no messages will be lost;
- At least once: default behavior. It is preferable to send a message several times than to lose it. If the producer does not receive a confirmation of receipt from the broker then the message is sent again;
- At most once: each message is delivered only once, to be used when a message should be lost than consumed twice. It is necessary to intervene on some configuration parameters:
  - Producer side: disable send retry;
  - Consumer side: it is necessary to communicate the new offset to the broker immediately after obtaining the message but before consuming it. This way if the consumer fails; during processing it is not retried after the reboot.

Since we have used kafka both to allow the interconnection of services and to guarantee the possibility of accessing data in real time, we have defined two groups of topics, service and data.

- `service.*`: used for communications regarding services. Each service will listen on a specific topics to receive command and send response;
- `data.*`: used to handle streams of data:
  - `data.raw`: raw data captured from devices;
  - `data.collision`: stream of the detected collisions;
  - `data.domainID.realtime`: real-time devices update for the specified domain. Contains updates for all the missions associated to the specified topic;
  - `data.domainID.realtime.missionID`: real-time updates for the specified mission;
  - `data.missionID.replay.replayID`: used to send the replied data of missionID.

For `data.*` topics, we decided to use a finer granularity to give the possibility to applications to focus on specific aspects, to reduce the workload associated with filtering operations. This choice offers particular advantages when the applications must be executed on machines that are not particularly performing.

[!htb]

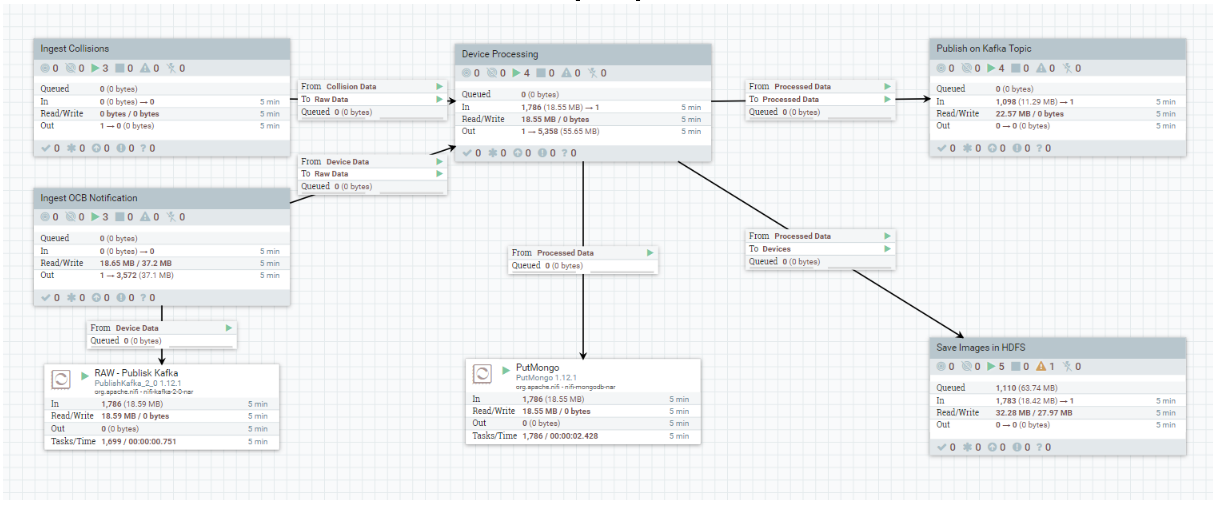


Figure 4.11: Draco Flow

4.2.3 Data Management

The purpose of data management services is to take care of data management. These requirements include the ingestion of data coming from devices, processing, routing towards the layers that will handle the data integration.

This section will describe the components that will take care of these features and their use.

FIWARE Context Broker

OCB allows to provide, with the help of a MongoDB instance, a unified representation of the current state of the devices. This representation, thanks to the mediation of the underlying levels, abstracts the differences between the devices. OCB provides users with a series of APIs that allow access to information and submit commands to the devices, thus acting as a hub for interactions. The possibility of defining subscriptions described in Section 4.1.1 has been exploited to ensure that every modification is directed towards Draco, the component that will take care of the processing and routing part towards the event bus.

Draco

In the context of Data Management, Draco takes care to handle the processing pipeline, showed in Figure 4.11, Through which we can manage all the necessary aspects, ranging from ingestion, preprocessing to data distribution. The main aspects involved in the processing are:

- Ingest OCB Notifications: Draco acts as a sink of all the notifications generated by OCB. In this way, we collect all the events, both those concerning the devices and those concerning the robotic agents;

- Data Preprocessing: after the ingestion phase, the data is brought to the portion of the pipeline dedicated to preprocessing. During these operations, attributes, derived from the content, are added to the FlowFile to facilitate subsequent operations;
- Save Images in HDFS: images obtained from camera will be saved on HDFS. The FlowFiles coming from the cameras are routed to a pipeline dedicated to image management that performs a simple ETL before saving the extracted images inside HDFS;
- Save Mission Data: this part of the Draco flow takes care of storing the preprocessed data in a MongoDB instance dedicated to mission persistence. Stored events are grouped by missions within specific collections.

4.3 Analytics on Streaming Data

This section discusses the services that perform analysis on data streams or provide support services. The services have been grouped into three macro-categories: near real-time, i.e. services that analyze in near real-time the data produced by the system, debugging, i.e. support services that allow you to analyze past events of the system, and data management, i.e. support services that deal with initiating analyzes.

4.3.1 Near-Real Time Analysis

Dashboard

The dashboard service consists of a NodeJS [84] application that provides interfaces to interact with other applications and monitor the current state of the system. The solutions adopted in the creation of the dashboard allow for the transparent management of both IoT devices and robots, thus respecting the detailed requirements in Section 3.1.1 and providing a possible solution to the problems analyzed in Section 2.2.1. The web server listens to Kafka topics dedicated to other services to keep the dashboards updated, for example by adding a new mission to the list of those available. This service uses the Proxy Service to allow pages running on clients to receive events published on kafka topics

The visual dashboard is composed by two main parts, the map (Figure 4.12) and the cards (Figure 4.13). The map, developed with OpenLayers [86], displays the real-time position of the devices whereas the card displays the current status of the device and the available commands. In both cases, we used a color-based approach to differentiate based on the type of device.

Cards are built dynamically by generating a table following the JSON object representation of the device and inserted under the map, each one represents the current state

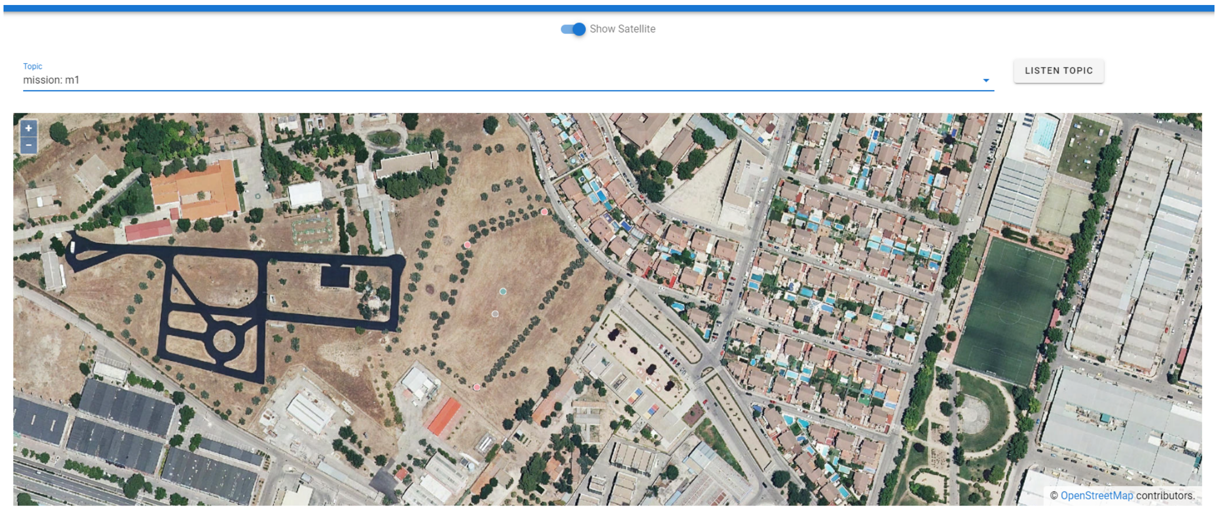
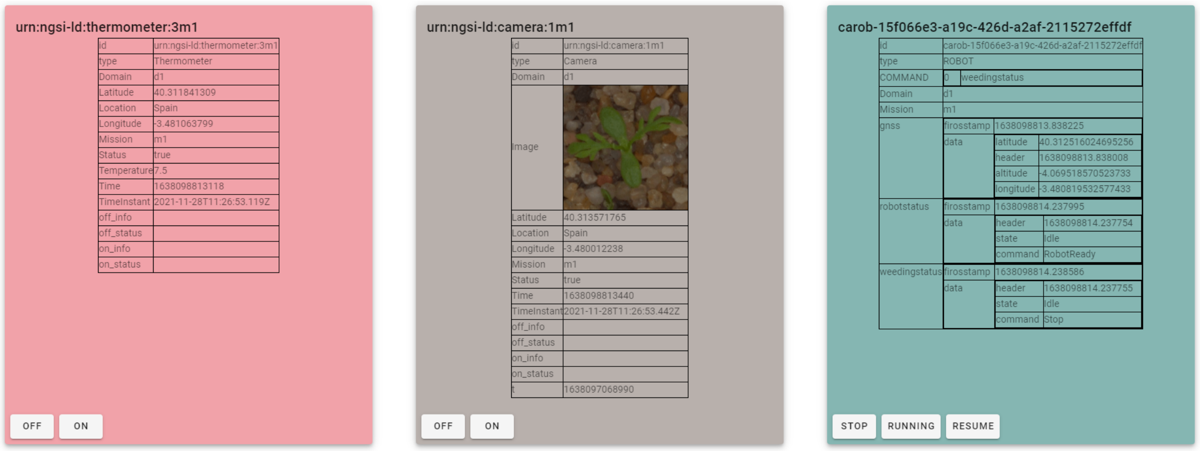


Figure 4.12: Visual Dashboard, topic selector and map



urn:ngsi-ld:thermometer:3m1

id	urn:ngsi-ld:thermometer:3m1
type	Thermometer
Domain	d1
Latitude	40.311841309
Location	Spain
Longitude	-3.481063799
Mission	m1
Status	true
Temperature	7.5
Time	1638098813118
TimeInstant	2021-11-28T11:26:53.119Z
off_info	
off_status	
on_info	
on_status	

urn:ngsi-ld:camera:1m1

id	urn:ngsi-ld:camera:1m1
type	Camera
Domain	d1
Image	
Latitude	40.313571765
Location	Spain
Longitude	-3.480012238
Mission	m1
Status	true
Time	1638098813440
TimeInstant	2021-11-28T11:26:53.442Z
off_info	
off_status	
on_info	
on_status	
t	1638097068990

carob-15f066e3-a19c-426d-a2af-2115272effdf

id	carob-15f066e3-a19c-426d-a2af-2115272effdf		
type	ROBOT		
COMMAND	0	weedingstatus	
Domain	d1		
Mission	m1		
gnss	firosstamp	1638098813.838225	
	data	latitude	40.312516024695256
		header	1638098813.838008
		altitude	-4.069518570523733
		longitude	-3.480819532577433
robotstatus	firosstamp	1638098814.237995	
	data	header	1638098814.237754
		state	Idle
		command	RobotReady
weedingstatus	firosstamp	1638098814.238586	
	data	header	1638098814.237755
		state	Idle
		command	Stop

Figure 4.13: Visual Dashboard, devices card

and the available commands of a device. It's worth noticing that devices and robots are seamlessly represented and differences between devices are smoothly handled, we can see different types of devices shown side-by-side and managed with the same tools.

Figure 4.14 shows a detail of the topic selection display. There are two default topics, collision and canary, which allow, respectively, to view the collisions and some demonstration devices that are deployed during system setup. As previously mentioned, by listening to the Domain Manager and the Mission Manager, topics dedicated to the visualization of domain and missions are automatically created. When a domain is visualized, the data of all its missions are displayed whilst a mission is visualized only the data of the devices that take part in said mission are displayed.

The data of the devices shown on the interface are obtained by consuming the appropriate Kafka topics. The acquisition of data from the topics takes place directly through the web browser with the mediation of the Proxy Service, a service which consists of an HTTP API implemented through a simple application developed with NodeJS. Applica-

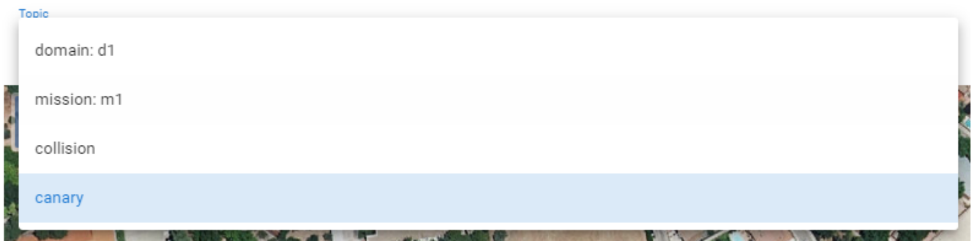


Figure 4.14: Visual Dashboard, topic selection example

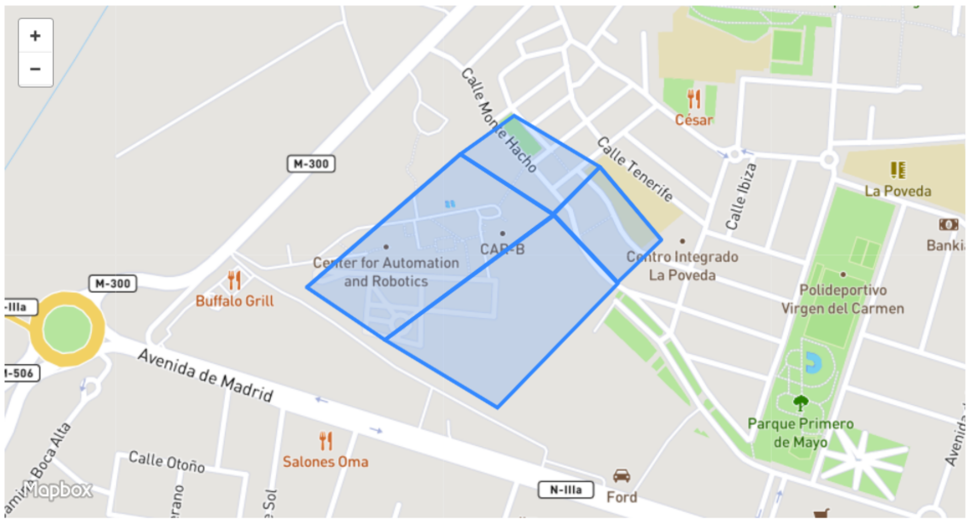


Figure 4.15: GeoJSON file containing information about the boundaries of the fields

tions can send a GET to `/api/register/:topic`, the response will contain the address of the WebSocket that can be used to consume the desired topic. Applications that consume through the socket are also able to use it to publish messages on the corresponding Kafka topic by sending a message containing a JSON object to the previously instantiated socket, proxy will take care of what is received on the correct Kafka topic.

Geo-Referenced Analytics

The purpose of this service is to serve as an example to verify the possibility of executing geo-referenced queries on the entities that make up the system. As a case study, we choose to compute the average temperature registered by thermometers in a field since, as illustrated in Section 1.1.1, the devices may have data quality issues, and performing an aggregation could be a viable solution to provide a more reliable result.

To determine belonging to a field, a SpatialJoin was performed between the position of the thermometer and the content of the file that contains the fields to a polygon that describes their boundaries. Figure 4.15 shows the GeoJSON file which contains information about the boundaries of the filed whereas Figure 4.16 shows the execution steps involved.

This service was created using the SparkStreaming [111] library in its Structured [112] version, the implementation took place using Scala, while the SpatialJoin operation was

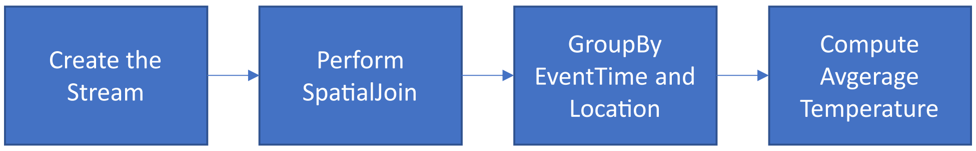


Figure 4.16: Average Temperature, execution steps

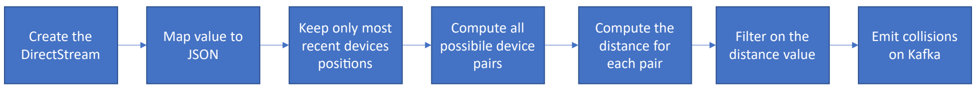

Create the DirectStream
Map value to JSON
Keep only most recent devices positions
Compute all possibile device pairs
Compute the distance for each pair
Filter on the distance value
Emit collisions on Kafka

Figure 4.17: Collision Detection, execution steps

performed using the GeoSpark [43] library.

An Spatial Join is a join-like operation where matches are not found based on the value of a key but on geometric properties, in this case, the presence of a point (device position) within a polygon (description of the boundaries of a field) was verified. This operation made it possible to extend the device information by adding a reference to the field, used later during the grouping operation.

As you can see in Figure 4.16, a grouping based on the event time was used. This means that the grouping was performed based on the time at which the event was generated and not on its arrival inside the system.

Alerting

The purpose of this service is to calculate the Haversine Distance [94] between the devices to determine which devices are too ”close”, therefore with a distance below a certain threshold value, and generate a collision on a dedicated Kafka topic to make the information available also to other services. In our demo the dashboard will show a representation of the collisions, in fig. 4.18 we can see an example of said visualization.

In Figure 4.17 it is possible to see the steps that make up the collision detection service. First of all, only the information of interest is extracted, that is the information about the GPS position and the event time (time in which the device generated the measurement at which the GPS value refers). For each device, only the last known position is then kept. At this point, the possible pairs of devices are generated and then proceed to a filter operation which allows limiting the selection only to the pairs that have a distance lower than the predetermined threshold value. At this point, the collisions are broadcast on a dedicated Kafka topic, `data.collision`, so that other services can use this information.

It is interesting to highlight a choice made about the format adopted for collisions. Collisions were represented with a format compatible with that of the Fiware entities to make their management more homogeneous, this solution was adopted to better respect the requirement stated in Section 3.1.2.Figure 4.19 shows a comparison between collisions

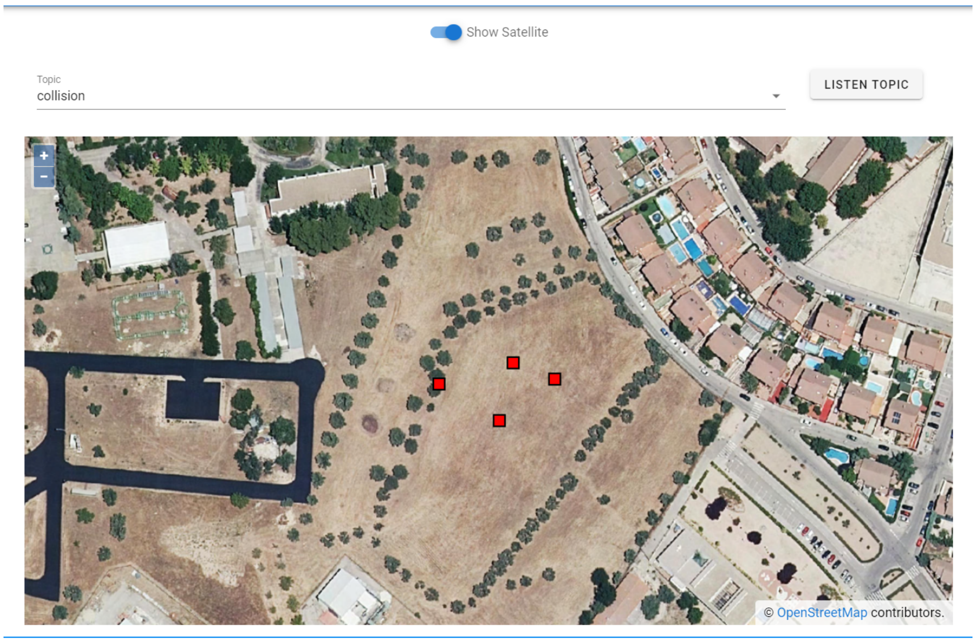


Figure 4.18: Collision Detection, example of collisions

and devices[1].

For each collision we store:

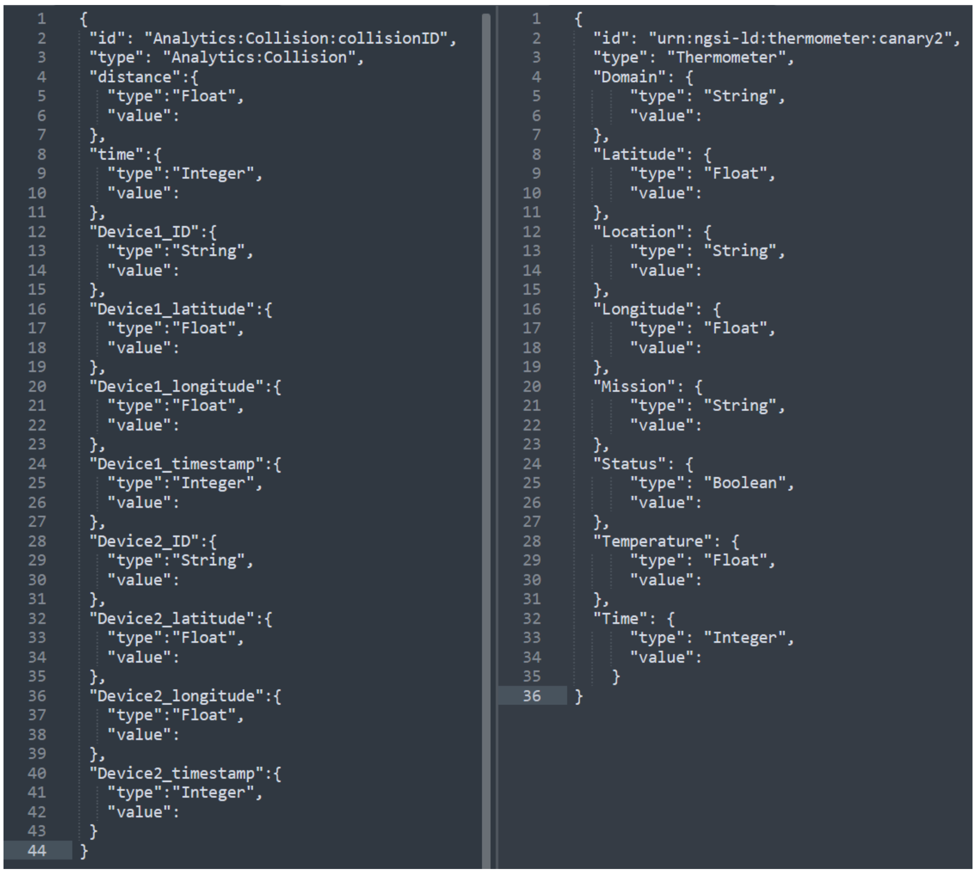
- ID: id of the collision;
- Type: collision type;
- Distance: the computed distance between the involved devices;
- Time: when the collision occurred;
- Device information: for every device we store id, timestamp, latitude and longitude.

4.3.2 Debugging - Replay

To make the functioning of a mission debuggable, the Replay Management service has been added.

This service takes advantage of the Data Management services provided by Draco described in Section 4.2.3. During the data management a copy of the events that occurred in the system is saved in a dedicate MongoDB instance. The purpose of this service is to access the stored copy and retransmit past events, respecting the original timing. Having a shared format for Robot and Device gave us the possibility to manage these two entities homogeneously.

[1]values and metadata omitted for the sake of brevity



```
{
  "id": "Analytics:Collision:collisionID",
  "type": "Analytics:Collision",
  "distance":{
    "type":"Float",
    "value":
  },
  "time":{
    "type":"Integer",
    "value":
  },
  "Device1_ID":{
    "type":"String",
    "value":
  },
  "Device1_latitude":{
    "type":"Float",
    "value":
  },
  "Device1_longitude":{
    "type":"Float",
    "value":
  },
  "Device1_timestamp":{
    "type":"Integer",
    "value":
  },
  "Device2_ID":{
    "type":"String",
    "value":
  },
  "Device2_latitude":{
    "type":"Float",
    "value":
  },
  "Device2_longitude":{
    "type":"Float",
    "value":
  },
  "Device2_timestamp":{
    "type":"Integer",
    "value":
  }
}
```

```
{
  "id": "urn:ngsi-ld:thermometer:canary2",
  "type": "Thermometer",
  "Domain": {
      "type": "String",
      "value":
  },
  "Latitude": {
      "type": "Float",
      "value":
  },
  "Location": {
      "type": "String",
      "value":
  },
  "Longitude": {
      "type": "Float",
      "value":
  },
  "Mission": {
      "type": "String",
      "value":
  },
  "Status": {
      "type": "Boolean",
      "value":
  },
  "Temperature": {
      "type": "Float",
      "value":
  },
  "Time": {
      "type": "Integer",
      "value":
    }
}
```

Figure 4.19: Collision Detection, comparison between collisions and device format

This service allows you to perform a replay of the missions and consists of two sub-services: *Replay Manager* (RM) and *Replay Execution* (RE). The first takes care of receiving replay requests from users and then starts a replay executor dedicated to managing the specific replay while the latter takes care of managing a specific replay. The replay takes place by acquiring data from the MongoDB instance dedicated to the persistence of mission data and the subsequent re-transmission on a dedicated topic.

Have to separate components provides several advantages:

- simpler data handling: since each RE will manage a specific replay, its implementation will be simpler than an implementation that provides for the management of multiple replays for multiple missions;
- easier to handle multiple replays: since every RE will handle a specific replay having multiple replay running means to have multiple RE. This also enables to have multiple replays of the same mission;
- responsiveness: having an entity dedicated to the execution of the replay, disconnected from the entity to which users turn to request an execution, allows you to keep the system constantly responsive, even in the event of any malfunctions during a replay;

4.3.3 Data Managment - Domain and Mission Management

To meet the need introduced in Section 3.1.1 we have developed two services: Domain Manager (DM) and Mission Manager (MM).

DM enables the creation of “domains”, i.e. collections of missions, whilst MM handles the starting and stopping of “missions”, i.e. group of devices that works together.

Through the MM it is possible to specify the name of the mission to be started or stopped. In the case of startup, the MM will take care of starting the docker containers needed to run the devices, while in the case of stopping they will proceed with their stop. Both services are implemented using Python [126] and the `kafka-python` [68] library to interface with Kafka. Interaction with docker containers takes place through a bash script that the container runs on the physical machine via an SSH connection.

5 Empirical Evaluation of Performance

In this chapter, we will describe the tests that were performed to evaluate the overall performance of the architecture.

To evaluate performance we designed a scenario consisting of a set of MQTT devices that periodically transmit information about their status and a final consumer, thus providing an end-to-end system. Data produced by the devices, thanks to the mediation of the MQTT broker, are received by IoTA which is in charge of informing OCB of the change, at this point, OCB will generate a notification for Draco who will manage the routing. Data, in the end, will be published on a Kafka topic to be consumed by a Kafka Consumer, represented by a Python script, which will consume the messages.

Different types of tests have been planned. Each type aims to verify how a certain configuration parameter (e.g.: number of devices, transmission period, number of subscriptions, payload size) affects performance. For each test, the aim is to evaluate the variation in performance with respect to the reference parameter, while keeping the others fixed. The test can be summarized in:

- period: starting from 5 messages per second, we reduce the period until we reach 625 messages per second. The remaining variables remain unchanged between the various experiments. See Table 5.2

- device: starting from 1 device, we increase the device number until we reach 625 messages per second. The remaining variables remain unchanged between the various experiments. See Table 5.3

- payload: starting from 1KB we progressively increase the payload to 100KB, the remaining variables remain unchanged between the various experiments. See Table 5.3

- subscription: starting from 1 subscription we progressively add new subscription until we reach 6 active subscriptions, the remaining variables remain unchanged between the various experiments. See Table 5.4

Period (ms)	Devices (#)	Subscriptions (#)	payload (byte)	Messages/Second
1000	5	1	1000	5
100	5	1	1000	50
40	5	1	1000	125
20	5	1	1000	250
10	5	1	1000	500
8	5	1	1000	625

Table 5.1: Tests performed on periods

Period (ms)	Devices (#)	Subscriptions (#)	payload (byte)	Messages/Second
200	1	1	1000	5
200	10	1	1000	50
200	25	1	1000	125
200	50	1	1000	250
200	100	1	1000	500
200	125	1	1000	625

Table 5.2: Tests performed on devices

Period (ms)	Devices (#)	Subscriptions (#)	payload (byte)	Messages/Second
200	50	1	1000	250
200	50	1	10000	250
200	50	1	50000	250
200	50	1	65000	250
200	50	1	80000	250
200	50	1	100000	250

Table 5.3: Tests performed on payload

Period (ms)	Devices (#)	Subscriptions (#)	payload (byte)	Messages/Second
200	25	1	1000	125
200	25	2	1000	250
200	25	3	1000	375
200	25	4	1000	500
200	25	5	1000	625
200	25	6	1000	1250

Table 5.4: Tests performed on subscriptions

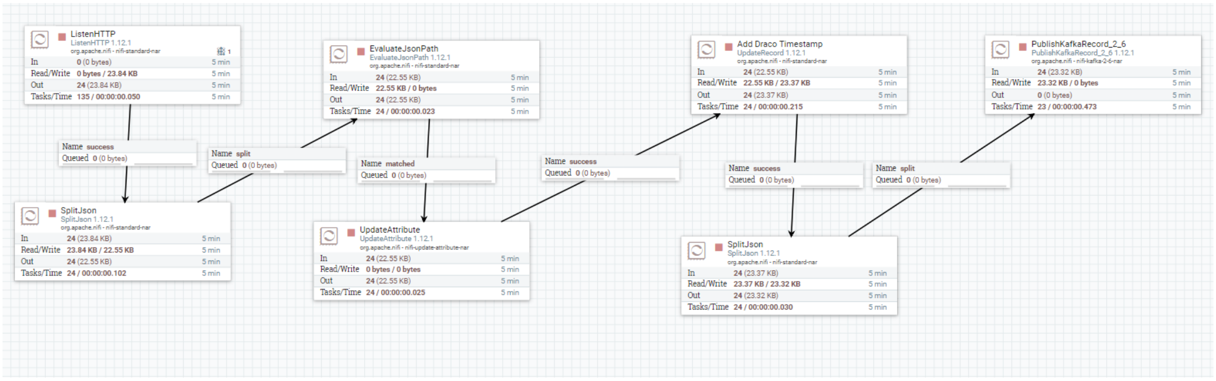

ListenHTTP
ListenHTTP 1.12.1
org.apache.nifi - nifi-standard-nar
1
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 23.84 KB 5 min
Out 24 (23.84 KB) 5 min
Tasks/Time 135 / 00:00:00.050 5 min
Name success
Queued 0 (0 bytes)
SplitJson
SplitJson 1.12.1
org.apache.nifi - nifi-standard-nar
In 24 (23.84 KB) 5 min
Read/Write 23.84 KB / 22.55 KB 5 min
Out 24 (22.55 KB) 5 min
Tasks/Time 24 / 00:00:00.102 5 min
Name split
Queued 0 (0 bytes)
EvaluateJsonPath
EvaluateJsonPath 1.12.1
org.apache.nifi - nifi-standard-nar
In 24 (22.55 KB) 5 min
Read/Write 22.55 KB / 0 bytes 5 min
Out 24 (22.55 KB) 5 min
Tasks/Time 24 / 00:00:00.023 5 min
Name matched
Queued 0 (0 bytes)
UpdateAttribute
UpdateAttribute 1.12.1
org.apache.nifi - nifi-update-attribute-nar
In 24 (22.55 KB) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 24 (22.55 KB) 5 min
Tasks/Time 24 / 00:00:00.025 5 min
Name success
Queued 0 (0 bytes)
Add Draco Timestamp
UpdateRecord 1.12.1
org.apache.nifi - nifi-standard-nar
In 24 (22.55 KB) 5 min
Read/Write 22.55 KB / 23.37 KB 5 min
Out 24 (23.37 KB) 5 min
Tasks/Time 24 / 00:00:00.215 5 min
Name success
Queued 0 (0 bytes)
SplitJson
SplitJson 1.12.1
org.apache.nifi - nifi-standard-nar
In 24 (23.37 KB) 5 min
Read/Write 23.37 KB / 23.32 KB 5 min
Out 24 (23.32 KB) 5 min
Tasks/Time 24 / 00:00:00.030 5 min
Name split
Queued 0 (0 bytes)
PublishKafkaRecord_2_6
PublishKafkaRecord_2_6 1.12.1
org.apache.nifi - nifi-kafka-2-6-nar
In 24 (23.32 KB) 5 min
Read/Write 23.32 KB / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 23 / 00:00:00.473 5 min

Figure 5.1: Draco architecture

5.1 Setup

Three machines equipped with 64Gb of RAM and AMD Ryzen 5 3600 6-Core CPU where used in order to distribute the workload:

- Machine 1: Devices;
- Machine 2: OCB, IoTA, MQTT Broker, and Kafka Broker;
- Machine 3: Draco and Kafka Consumer.

In the remainder of the section, we will describe the main components involved and some salient parts of the configurations adopted.

Devices Devices involved in these experiments consists of a simple “logger” implemented as a simple JavaScript program capable of transmitting the current time, a payload, and its status (running/not running). Like seen in Section 4.2.1, devices communicate through MQTT with the Fiware ecosystem, during their operation they produce a log file storing the device identifier, status, and current time.

Draco The Draco flow in figure Figure 5.1 shows the steps involved in the ingestion of the incoming data, the processing, and the production of data on the Kafka topic.

We can summarize the processing steps in:

1. `ListenHTTP` acts as a webserver and receives all the context updates from OCB, using `SplitJson` we then isolate every entity contained in every subscription
2. `EvaluateJsonPath` and `UpdateAttribute` are used to extract some fields from the messages, namely the device id and the event time
3. `AddDracoTimestamp` adds a timestamp that represents the moment when Draco has finished processing the data, thus symbolizing the time needed by a webserver to process the data

4. `PublishKafkaRecord` will publish the collected topic on Kafka, thus enabling the consumer to collect the desired data

Kafka Consumer The consumers consists in a Python scripts executed inside a Docker container.

This service receives every message produced by Draco (i.e.: every status update from the devices) and logs them to a file containing:

- deviceID: id of the device that generated the data
- deviceStatus: status of the device that generated the data
- kafkaTimestamp: timestamp associated to the consumed message
- dracoTimestamp: when the message was process by Draco
- deviceTimestamp: when the message was created by the Device
- consumerTimestamp: when the message was processed by the Consumer

5.1.1 Optimization

To ensure better performance than those obtainable with the basic configurations, we intervened on various system components. Noteworthy configurations will be outlined below.

MongoDB index OCB doesn't create any index or database collection by default. In this scenario, like suggested in the performance tuning guide [90], we have create an index for the *entities* collection before running the tests.

***ulimit* Configuration for OCB and IOTA** As suggested in the official docker image of IOTA[65] we have increased the resources assigned to the user inside the docker container via the *ulimit* directive of docker-compose. We have set the values for *core* and *memlock* to "60000000" to roughly match the available RAM (64GB) on the physical machine which is running the container.

***loglevel* Configuration for OCB and IOTA** The presence of the console log could negatively affect the performance of the Fiware components. To avoid such issues we have set the log-level to "ERROR". By doing so we are still able to debug the most serious error but without the greater overhead generated by the default "INFO" level. As reported in the performance-tuning guide there are situations in which the switch between log level WARN and log-level info could lead to an increase of performance up to 50% [90]

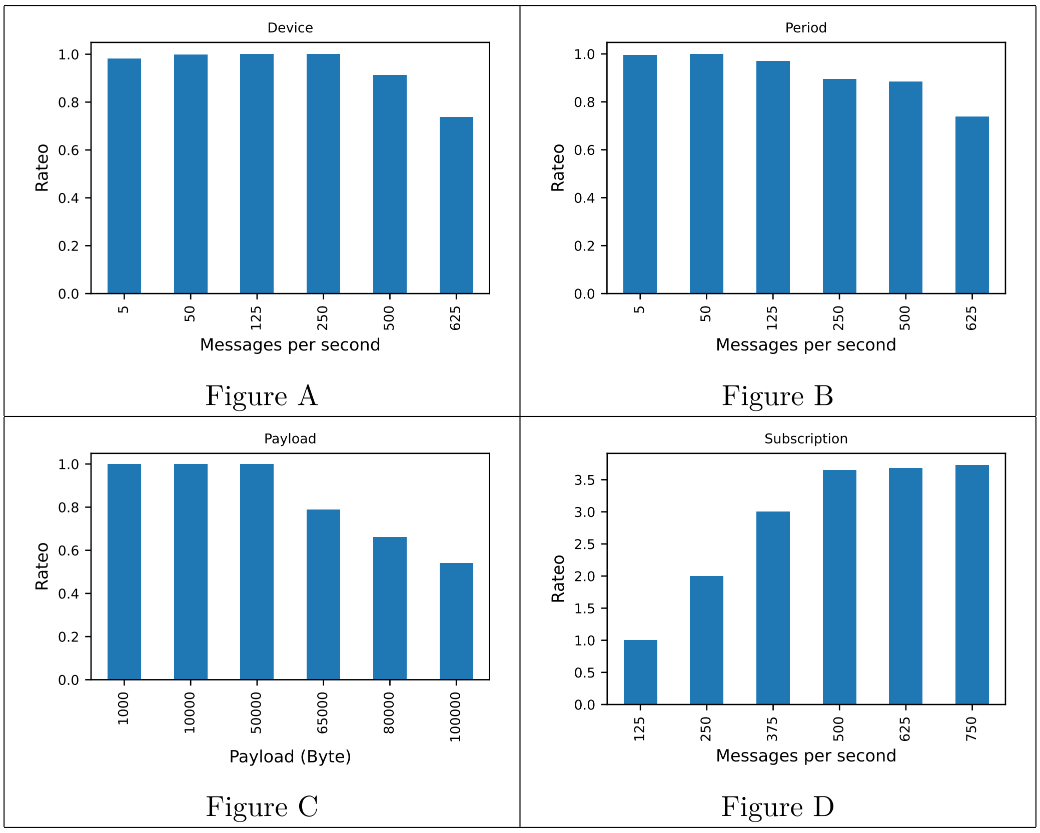


Figure 5.2: Cumulative reception rateo

5.2 Results

The purpose of this section and to present the collected results. All the configurations described at the beginning of this chapter have been analyzed, both as regards the number of messages exchanged and the delay.

First, to highlight the general behavior of the architecture, the overall results will be presented in the form of aggregated graphs. Some detailed graphs about the most interesting situations will be presented below.

5.2.1 Cumulative

The average results of the various experiments were collected in this group of graphs. The purpose is to make it possible to quickly view the trend of the metrics of interest, i.e. the rate of receipt of messages and average end-to-end delay, as the number of messages handled per second or the payload increases.

Reception Rateo The graphs in Figure 5.2 represent the relationship between messages sent by devices and received by the consumer as the number of messages handled per second or the payload changes.

We can see that:

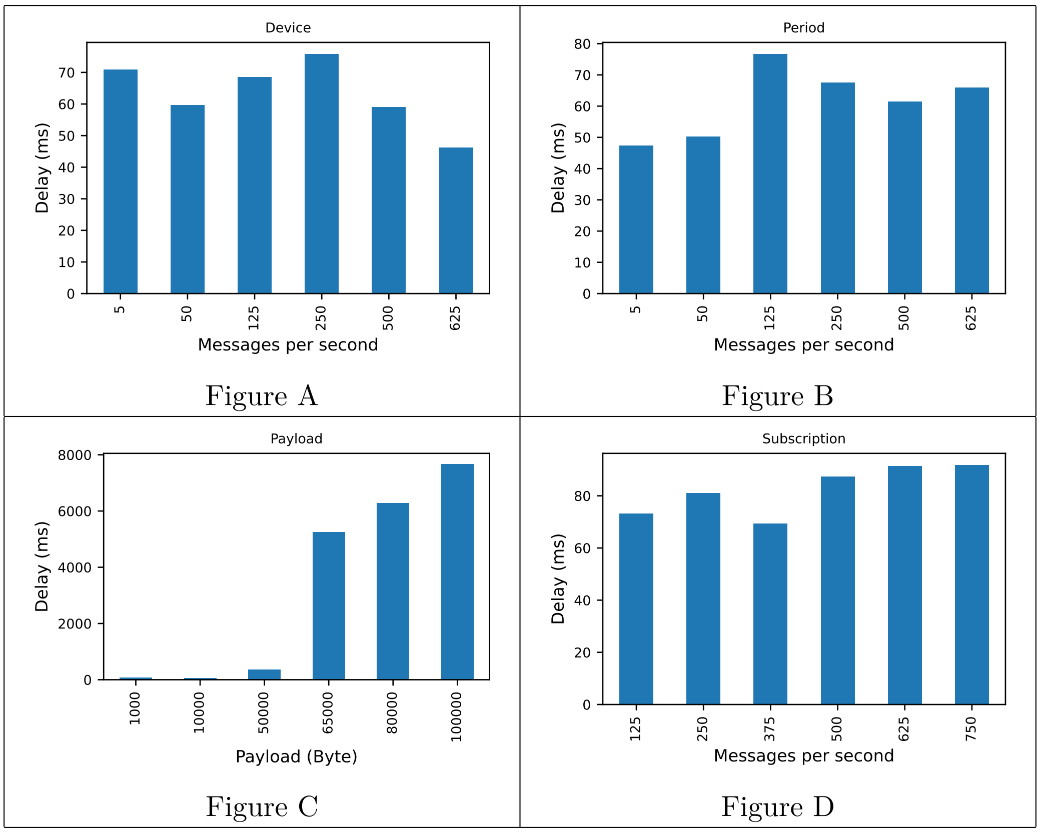


Figure 5.3: Average delay

- in A and B there is a marked drop in the reception of messages after exceeding 500msg/sec;
- in C we can notice that after 50KB we have a marked drop in reception of messages;
- in D we expected to see an increase in reception due the multiple subscriptions registered. This is observed for upto three subscription but after this value this assumption no longer holds.

Delay The graphs in Figure 5.3 have been computed realizing the average end-to-end delay. The aim is to visualize the progress of the experiment as the metric of interest varies while eliminating the presence of the peaks highlighted in Section 5.2.3. We can notice that:

- in A, B, and D we keep the same end-to-end delay while the number of messages handled per second varies;
- in C we see a rapid rise in the delay after 65KB. This is compatible with what was observed in Section 5.2.2.

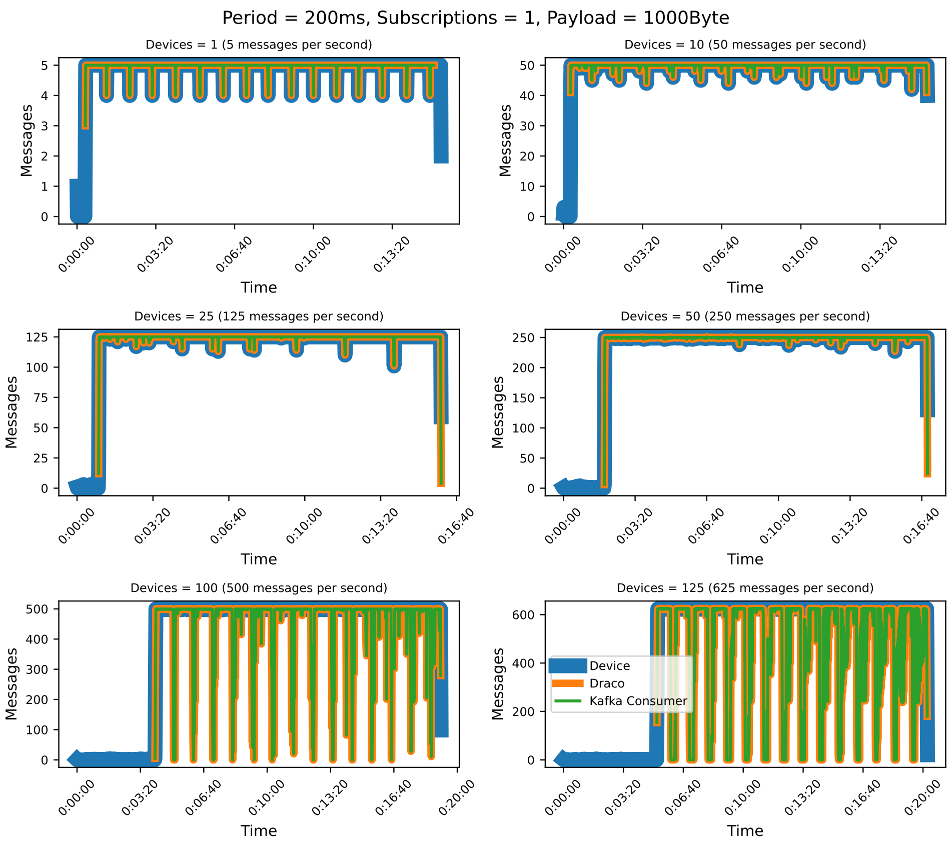


Figure 5.4: Message count distribution in time, "devices" testing

5.2.2 Number of messages

In this set of experiments we can see the trend of the count of the record messages. Every graph is composed by a sub-graph for each experiment configuration. On the X axis we can see the time of the experiment while on the Y axis we can see the massage count for every component:

- Device: messages sent by the devices
- Draco: messages received by Draco
- Kafka Consumer: messages received by the kafka consumer

Device It is possible to notice how up to 250msg/sec the behavior of the system is stable, with the exception of some small oscillations, while beyond this value the system stops being reliable given the presence of moments in which no messages are recorded. See Figure 5.4

Period In this case the system is reliable up to 500msg/sec despite a little misalignment. Over this value the system becomes unreliable. See Figure 5.5

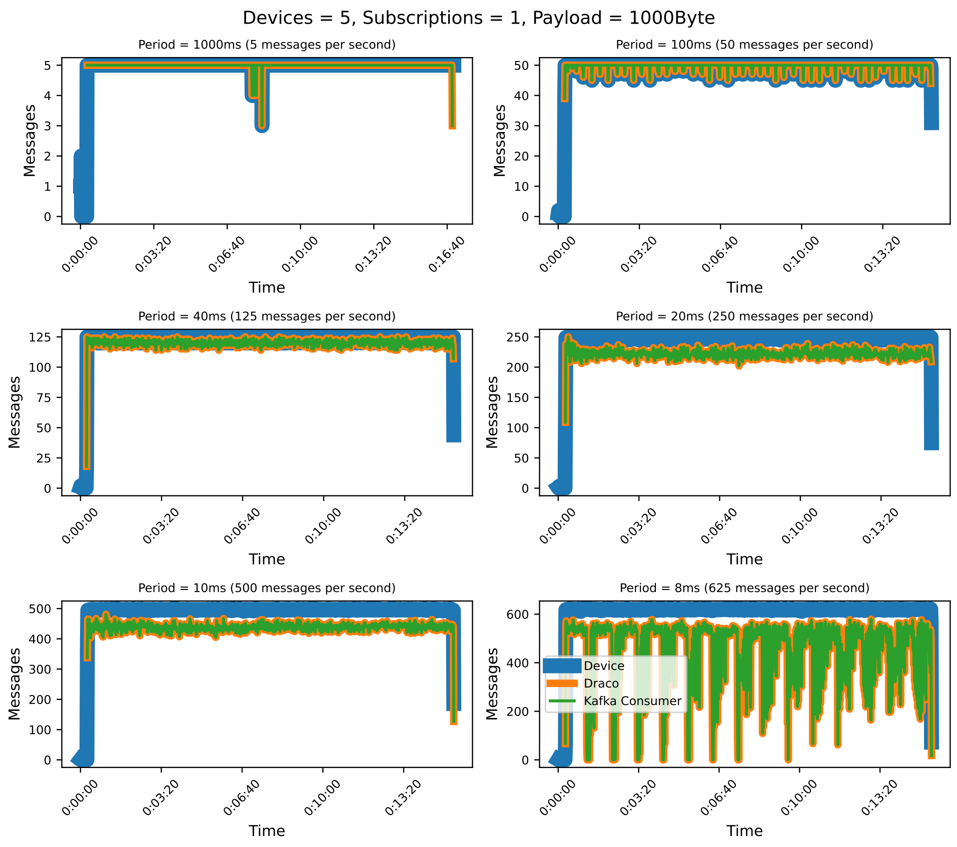


Figure 5.5: Message count distribution in time, "period" testing

Payload The system remains reliable with payloads up to 50KB, over this value we can see some misalignment like those showed before. See Figure 5.6

Subscriptions The system managed to behave correctly with up to 3 subscriptions, meaning a total of 375msg/sec with results in line with those seen previously. See Figure 5.7

5.2.3 Delay

The aim of this group of graphs is to visualize the delay introduced by each step the computation, namely:

- Draco: time elapsed between sending by the device and the end of the computation of Draco;
- Kafka/Draco: delay introduced by Draco to publish the messages on the Kafka Broker;
- Kafka Consumer: delay introduced by the consumer.

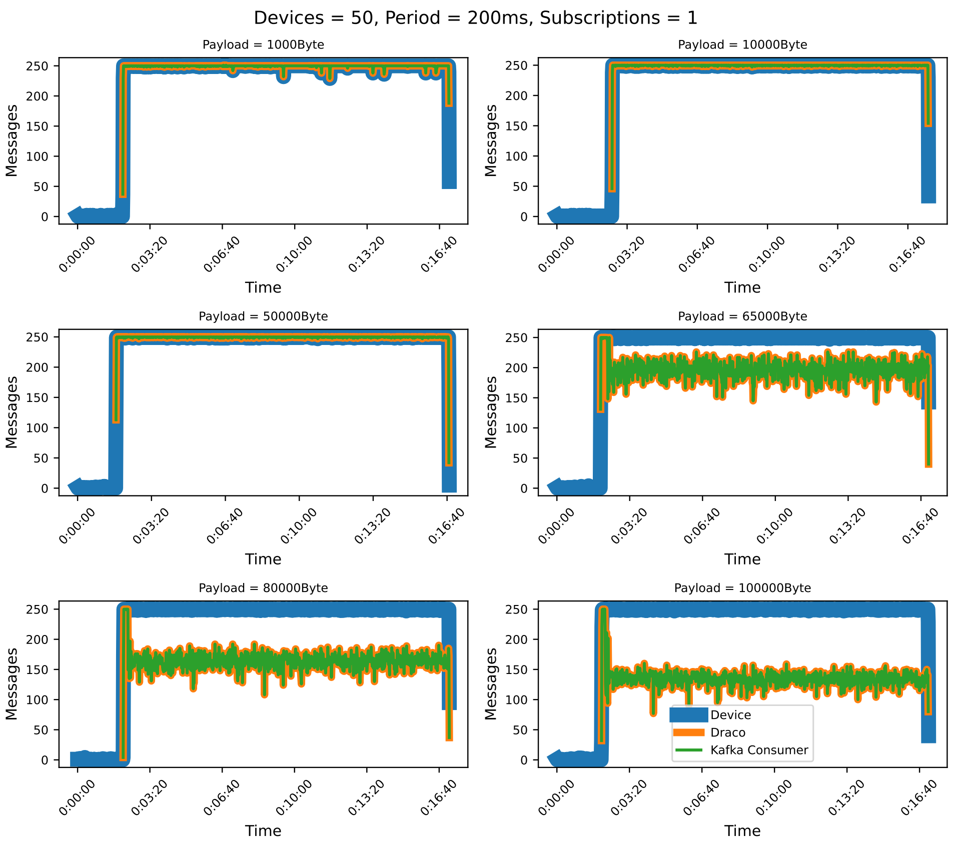


Figure 5.6: Payload: Message count distribution in time

Device In Figure 5.8 we can notice that in every scenario we had some peak delays but the peak value increased progressively with the number of messages, reaching peaks of 1500ms with more than 500msg/sec, thus in correspondence of what discussed in Section 5.2.2. It is also possible to notice how the greater delay is introduced by the reception of messages by Draco.

Period We can draw conclusions similar to those seen previously in Section 5.2.3: progressive increase of the delay peaks that exceed 1000ms introduced by the reception of messages by Draco with 500msg/sec, like in Section 5.2.2.

Payload At 65KB we can see a uniform delay of 6000ms which rises to more than 8000ms with 100KB. These results, as shown in Figure 5.9, are compatible with those discussed in Section 5.2.2 where we noticed a misalignment at 6500KB.

Subscriptions In these experiments there was no progressive increase in the delay as the number of messages handled increases, this observation is in contrast to what was observed in Section 5.2.2 where with 500msg/sec we noticed a misalignment.

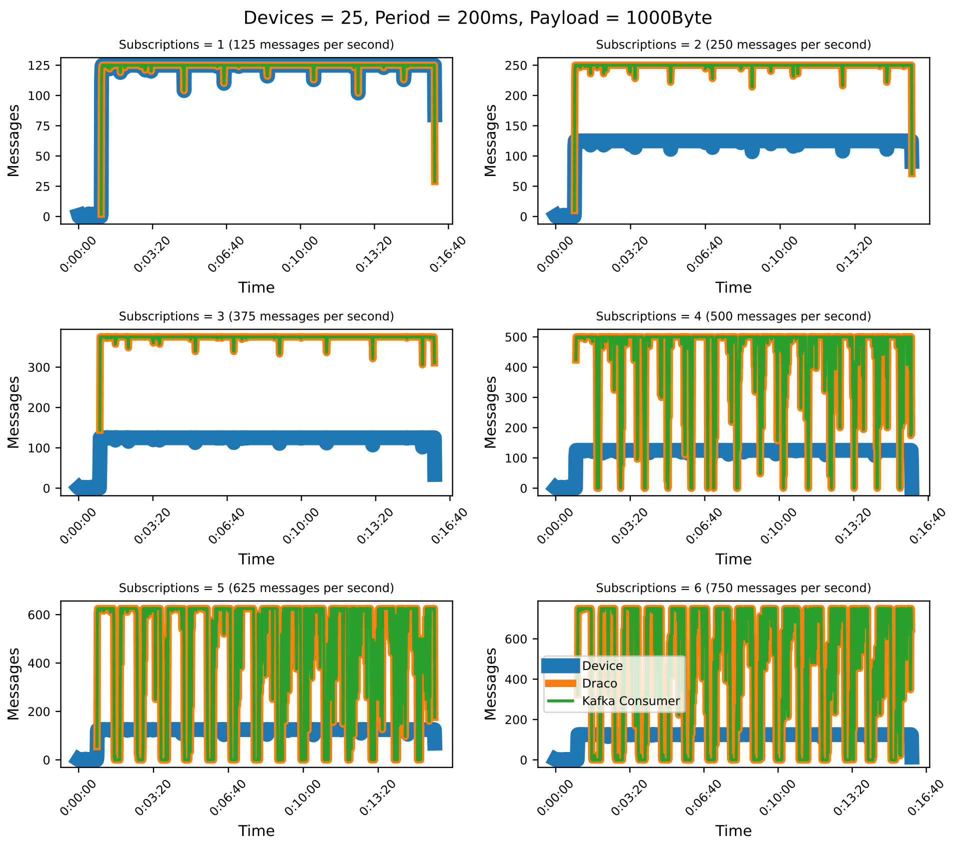


Figure 5.7: Message count distribution in time, “subscription” testing

5.2.4 Conclusions

- Payload: the maximum payload handled by the system without message loss is 50Kbyte with 250msg/sec, see Figure Figure 5.3 subplot C;
- Subscriptions: as highlighted in Figure 5.3 sub-figure D, during the tests concerning the subscriptions, the system behaved correctly only up to 3 subscriptions, thus recording a triplication of messages. However, it should be noted that in this case, each Subscription concerns each entity in the system, thus causing a multiplication of the sent messages, in a real scenario such a situation is difficult to find;
- We have noticed a cap of 500msg/sec handled correctly in almost every situation;
- In all tested circumstances an IoTA crash was detected, leading to the loss of messages.

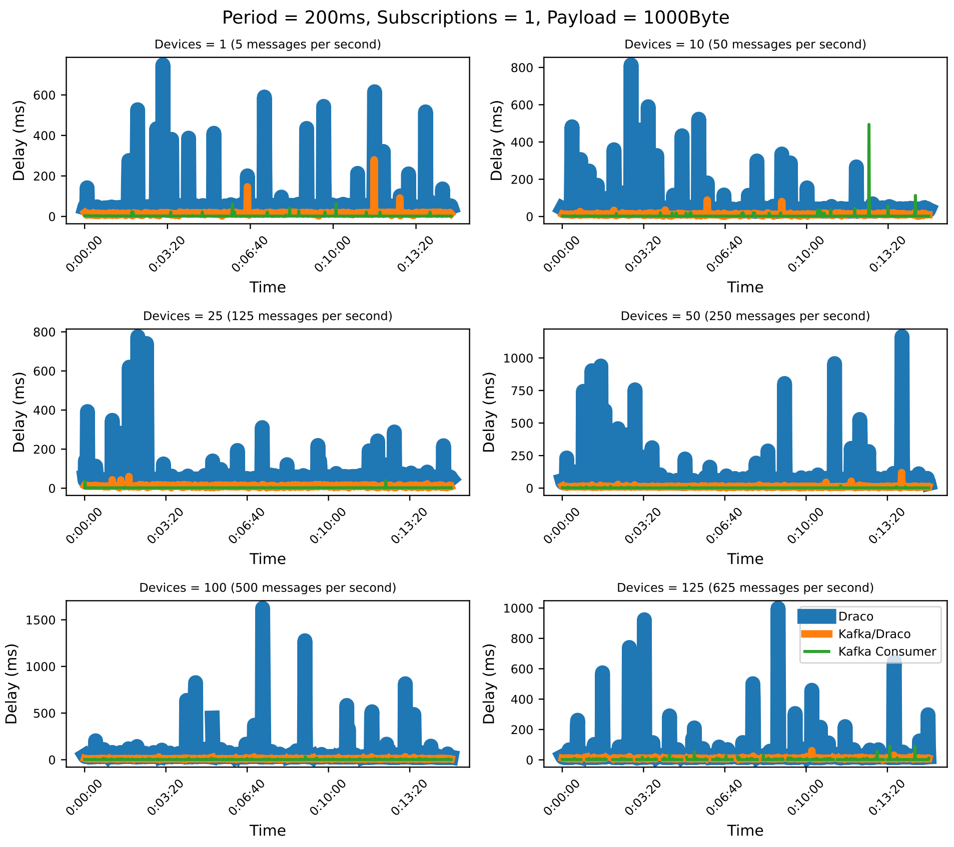


Figure 5.8: Delay distribution in time, “devices” testing

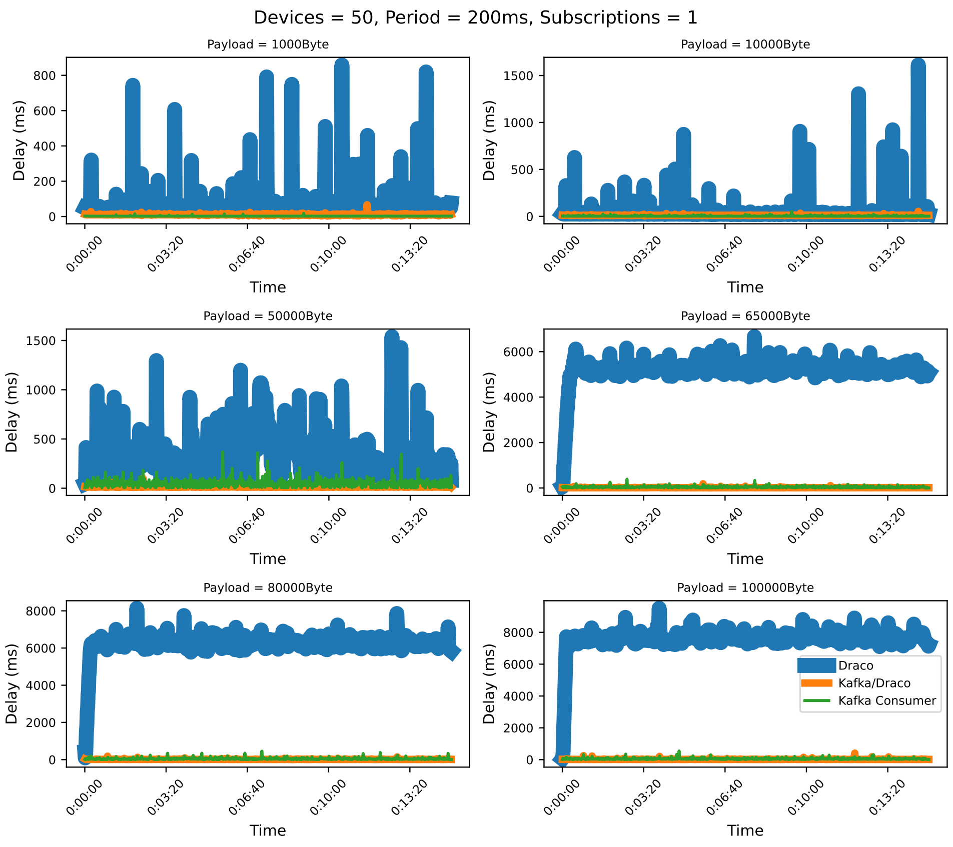


Figure 5.9: Delay distribution in time, “payload” testing

6 Conclusion and Future Work

This thesis introduces the basis of the Data Platform for WeLASER, a project funded by the European Union in the context of precision agriculture. WeLASER stands at an intersection between IoT and Robotics, it is, therefore, necessary to put these two worlds in communication by providing mechanisms that allow the interoperability of technologies, including with external modules made available thanks to services Cloud. The purpose of the thesis is the creation of a Data Platform that allows an end-to-end interaction between devices and applications: we want to be able to collect, store and process, even in real-time, the data produced by the device, creating a complete ecosystem also capable to regulated the functioning of the devices by submitting commands.

The problems of the sectors involved were analyzed to identify the most interesting issues and the main challenges that must be faced. After the study of the literature, an architecture was designed to guarantee the possibility of acquiring data from devices, whether they are IoT devices or robotic agents, managing their integration, and, finally, sharing them with various applications through an event bus. The proposed architecture was finally implemented using Fiware: an ecosystem of open-source components designed to provide an infrastructure for cloud-based IoT project based on open standards.

To verify the possibility of extending the architecture with new applications, we have also designed and built a set of services capable of interacting with data, even in real time:

- Dashboard: the dashboard allows to visualize the status of the system in real-time, also adapting to types of unknown devices, and acts as a control panel for other applications;
- Collision Detection: this services executes real-time geo-referenced analytics to monitor the movement of devices, issuing an event when two devices are too close;
- Device administration: allows the start and stop of missions, in this way it is possible to insert and remove devices from the system dynamically;
- Replay management: service which, by exploiting the information stored by the component of the system responsible for data processing, allows the retransmission of the stored data. In this way, it is possible to carry out analyzes on these data again by exploiting the applications already developed but also to perform debugging operations.

To verify the performance offered by the architecture, tests were carried out to verify the performance trend as the workload changes. These tests took into consideration parameters such as number of devices, update frequency, and size of the payload.

Future developments concern two fronts: the WeLASER project as a whole, as further years of development are expected starting from what has been produced up to this point, and extensions relating to the proposed architecture.

- *Define a unified data-model for the entities*: it is necessary to define a uniform model for the representation of entity data. The architecture made available allows the integration of homogeneous devices but to make the most of this possibility it is necessary to define more accurate modeling of the devices. The devices could be produced by different organizations, having a defined and well-documented standard would facilitate interoperability. Also in our example which saw the collaboration of two organizations, us and the Centre for Automation and Robotics of the UPM-CSIC [54], it is possible to see a discrepancy in the representations of the entities;

- *Integration with new analysis services*: to ensure compliance with the requirements of the WeLASER project, it will be necessary to integrate new analysis services, in particular Artificial Intelligence systems aimed at performing analysis on the images collected by the cameras;

- *Integration with real devices*: for now simulated devices have been used, integrating real devices will entail the introduction of new problems related to their management. Progress in this direction is already being studied/developed in other realities connected to the WeLASER project;

- *Improve domain and mission modeling*: domains and missions have been modeled in an extremely simplified way for the sole purpose of creating a working demo of the system, it is necessary to proceed with a more detailed modeling phase to define more accurately what these entities effectively are. A possible solution could include their representation as a Fiware entity. Such a solution could allow uniform modeling with the rest of the system and also act as a container for the aggregate data produced, such as the average temperature collected by the weather stations present within the mission, or on the status of the mission such as, for example, the progress. Missions are currently hard-coded within the mission manager. It would be useful to provide a configuration interface for missions through which specify device type, number, and parameters. Saving mission templates could also be an interesting feature;

- *Metadata integration*: at present, only the data relating to the sensor are collected but, to ensure quality management with future extensions, it will also be necessary to collect and manage metadata.Among the main advantages of the introduction of metadata we can identify the management of provenance and profiling. Data provenance is metadata pertaining to the history of a data item, it essentially describe a transformation pipeline, including the origin and the performed operations. Object profiling includes all the functionalities aimed at searching, selecting and describing the stored entities;

- *Further performance testing*: we have currently verified the performance of the architecture using end-to-end tests aimed at measuring the delay and the total number of messages. Further tests could be performed verifying the variation of the performances with the variation of the architecture used. Extensions can also concern the evaluation of individual components and the definition of a set of metrics aimed at measuring the architecture as a whole, independently of the elements that compose it;

Bibliography

[1] *Ag Tech Deal Activity More Than Triples.* https://web.archive.org/web/20211113153418/https://www.cbinsights.com/research/agriculture-farm-tech-startup-funding-trends/. (Accessed on 11/13/2021).

[2] Divyakant Agrawal et al. "Challenges and Opportunities with Big Data. A community white paper developed by leading researchers across the United States". In: *Computing Research Association, Washington* (2012).

[3] Bagrudeen Bazeer Ahamed, Thirunavukarasu Ramkumar, and Shanmugasundaram Hariharan. "Data integration progression in large data source using mapping affinity". In: *2014 7th International Conference on Advanced Software Engineering and Its Applications.* IEEE. 2014, pp. 16–21.

[4] Ejaz Ahmed et al. "Internet-of-things-based smart environments: state of the art, taxonomy, and open research challenges". In: *IEEE Wireless Communications* 23.5 (2016), pp. 10–16. DOI: 10.1109/MWC.2016.7721736.

[5] *AI, Robotics, And The Future Of Precision Agriculture - CB Insights Research.* https://web.archive.org/web/20211113153730/https://www.cbinsights.com/research/ai-robotics-agriculture-tech-startups-future/. (Accessed on 11/13/2021).

[6] Saad Alabdulsalam et al. "Internet of things forensics–challenges and a case study". In: *IFIP International Conference on Digital Forensics.* Springer. 2018, pp. 35–48.

[7] Adanma Cecilia Eberendu and. "Unstructured Data: an overview of the data of Big Data". In: 38.1 (Aug. 2016), pp. 46–50. DOI: 10.14445/22312803/ijctt-v38p109. URL: https://doi.org/10.14445/22312803/ijctt-v38p109.

[8] *Apache Kafka.* https://web.archive.org/web/20211104160641/https://kafka.apache.org/intro. (Accessed on 11/04/2021).

[9] *Apache NiFi.* https://nifi.apache.org/. (Accessed on 11/24/2021).

[10] Rajesh Arumugam et al. "DAvinCi: A cloud computing framework for service robots". In: *2010 IEEE international conference on robotics and automation.* IEEE. 2010, pp. 3084–3089.

[11] Kevin Ashton. "That "Internet of Thing" Thing". In: 2009.

[12] *AWS IoT | Settore industriale, dei consumi, commerciale, automobilistico | Amazon Web Services*. https://aws.amazon.com/it/iot/. (Accessed on 11/03/2021).

[13] Ahmad Taher Azar and Aboul Ella Hassanien. “Dimensionality reduction of medical big data using neural-fuzzy classifier”. In: *Soft computing* 19.4 (2015), pp. 1115–1127.

[14] *Azure IoT – Internet of Things Platform | Microsoft Azure*. https://azure.microsoft.com/en-us/overview/iot/. (Accessed on 11/03/2021).

[15] David Bakken. “Middleware”. In: (Mar. 2001).

[16] Sharu Bansal and Dilip Kumar. “IoT Ecosystem: A Survey on Devices, Gateways, Operating Systems, Middleware and Communication”. In: 27.3 (Feb. 2020), pp. 340–364. DOI: 10.1007/s10776-020-00483-7. URL: https://doi.org/10.1007/s10776-020-00483-7.

[17] Avital Bechar and Clément Vigneault. “Agricultural robots for field operations: Concepts and components”. In: *Biosystems Engineering* 149 (2016), pp. 94–111. ISSN: 1537-5110. DOI: https://doi.org/10.1016/j.biosystemseng.2016.06.014. URL: https://www.sciencedirect.com/science/article/pii/S1537511015301914.

[18] Avital Bechar and Clément Vigneault. “Agricultural robots for field operations. Part 2: Operations and systems”. In: *Biosystems Engineering* 153 (2017), pp. 110–128. ISSN: 1537-5110. DOI: https://doi.org/10.1016/j.biosystemseng.2016.11.004. URL: https://www.sciencedirect.com/science/article/pii/S1537511015301926.

[19] M. Bergerman et al. “Robotics in agriculture and forestry”. English. In: *Springer handbook of robotics*. Ed. by Bruno Siciliano and Oussama Khatib. 2nd ed. Springer Verlag, 2016, pp. 1463–1492. ISBN: 9783319325507. DOI: 10.1007/978-3-319-32552-1_56.

[20] Umar Biliyaminu et al. “Evaluation of IoT Device Management Tools”. In: 2018.

[21] Luca Calderoni, Antonio Magnani, and Dario Maio. “IoT Manager: An open-source IoT framework for smart cities”. In: *Journal of Systems Architecture* 98 (2019), pp. 413–423. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2019.04.003. URL: https://www.sciencedirect.com/science/article/pii/S1383762118306520.

[22] Tomas Cerny, Michael J. Donahoo, and Jiri Pechanec. “Disambiguation and Comparison of SOA, Microservices and Self-Contained Systems”. In: *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*. RACS ’17. Krakow, Poland: Association for Computing Machinery, 2017, pp. 228–

235. ISBN: 9781450350273. DOI: 10.1145/3129676.3129682. URL: https://doi.org/10.1145/3129676.3129682.

[23] *Che cos'è il data management?* URL: https://web.archive.org/web/20211027100130/https://www.oracle.com/it/database/what-is-data-management/.

[24] Hank Childs et al. "Research challenges for visualization software". In: *Computer* 46.5 (2013), pp. 34–42.

[25] Alem Čolaković and Mesud Hadžialić. "Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues". In: *Computer Networks* 144 (2018), pp. 17–39. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2018.07.017. URL: https://www.sciencedirect.com/science/article/pii/S1389128618305243.

[26] Julia Couto et al. "A Mapping Study about Data Lakes: An Improved Definition and Possible Architectures". In: KSI Research Inc. and Knowledge Systems Institute Graduate School, July 2019. DOI: 10.18293/seke2019-129. URL: https://doi.org/10.18293/seke2019-129.

[27] Christopher Crick et al. "Rosbridge: Ros for non-ros users". In: *Robotics Research*. Springer, 2017, pp. 493–504.

[28] *Definition of Application Integration - Gartner Information Technology Glossary*. https://web.archive.org/web/20210129013031/https://www.gartner.com/en/information-technology/glossary/application-integration. (Accessed on 11/03/2021).

[29] C Díez. "Hacia una agricultura inteligente (Towards and intelligent Agriculture)". In: (2017).

[30] Ciro Donalek et al. "Immersive and collaborative data visualization using virtual reality platforms". In: *2014 IEEE International Conference on Big Data (Big Data)*. IEEE. 2014, pp. 609–614.

[31] Ali Dorri et al. "Blockchain for IoT security and privacy: The case study of a smart home". In: *2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops)*. IEEE. 2017, pp. 618–623.

[32] Ren Duan, Xiaojiang Chen, and Tianzhang Xing. "A QoS architecture for IOT". In: *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*. IEEE. 2011, pp. 717–720.

[33] *Eclipse Mosquitto*. https://mosquitto.org/. (Accessed on 11/24/2021).

[34] Mahmoud Elkhodr, Seyed Shahrestani, and Hon Cheung. "The Internet of Things : New Interoperability, Management and Security Challenges". In: 8.2 (Mar. 2016), pp. 85–102. DOI: 10.5121/ijnsa.2016.8206. URL: https://doi.org/10.5121/ijnsa.2016.8206.

[35] *Farming 4.0: The future of agriculture? – EURACTIV.com.* https://web.archive.org/web/20211113152726/https://www.euractiv.com/section/agriculture-food/infographic/farming-4-0-the-future-of-agriculture/. (Accessed on 11/13/2021).

[36] *fiware-ngsiv2-2.0-2018_09_15.* https://fiware.github.io/specifications/ngsiv2/stable/. (Accessed on 11/27/2021).

[37] *FIWARE/tutorials.IoT-Agent: FIWARE 202: Provisioning the Ultralight IoT Agent.* https://github.com/FIWARE/tutorials.IoT-Agent. (Accessed on 11/04/2021).

[38] *Flow Based Programming.* URL: https://web.archive.org/web/20211021101653/http://www.jpaulmorrison.com/fbp/fbp2.htm (visited on 10/21/2021).

[39] Matteo Francia et al. "Making data platforms smarter with MOSES". In: *Future Generation Computer Systems* 125 (2021), pp. 299–313. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2021.06.031. URL: https://www.sciencedirect.com/science/article/pii/S0167739X21002260.

[40] J. Gantz and E. Reinsel. "Extracting Value from Chaos". In: (2011).

[41] *Gartner Glossary - Big Data Definiton.* Oct. 2021. URL: https://web.archive.org/web/20211022095923/https://www.gartner.com/en/information-technology/glossary/big-data (visited on 10/22/2021).

[42] *Gartner Says Solving 'Big Data' Challenge Involves More Than Just Managing Volumes of Data | Business Wire.* https://web.archive.org/web/20211114002950/https://www.businesswire.com/news/home/20110627005655/en/Gartner-Says-Solving-Big-Data-Challenge-Involves-More-Than-Just-Managing-Volumes-of-Data. (Accessed on 11/14/2021).

[43] *Geospark Analytics.* https://www.geospark.io/. (Accessed on 11/24/2021).

[44] Corinna Giebler et al. "Leveraging the data lake: current state and challenges". In: *International Conference on Big Data Analytics and Knowledge Discovery.* Springer. 2019, pp. 179–188.

[45] *Global Economic Crime and Fraud Survey 2018.* https://www.pwc.com/it/it/publications/docs/pwc-global-economic-crime-fraud-survey-2018.pdf. (Accessed on 11/14/2021).

[46] Navneet Golchha. "Big data-the information revolution". In: *Int. J. Adv. Res* 1.12 (2015), pp. 791–794.

[47] Ken Goldberg and Roland Siegwart. *Beyond Webcams: an introduction to online robots.* MIT press, 2002.

[48] *Google Cloud IoT - Servizi IoT completamente gestiti.* https://cloud.google.com/solutions/iot. (Accessed on 11/03/2021).

[49] Jayavardhana Gubbi et al. “Internet of Things (IoT): A vision, architectural elements, and future directions”. In: *Future Generation Computer Systems* 29.7 (2013). Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond, pp. 1645–1660. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2013.01.010. URL: https://www.sciencedirect.com/science/article/pii/S0167739X13000241.

[50] Jasmin Guth et al. “Comparison of IoT platform architectures: A field study based on a reference architecture”. In: *2016 Cloudification of the Internet of Things (CIoT)* (2016), pp. 1–6.

[51] Soumil Heble et al. “A low power IoT network for smart agriculture”. In: IEEE, Feb. 2018. DOI: 10.1109/wf-iot.2018.8355152. URL: https://doi.org/10.1109/wf-iot.2018.8355152.

[52] F. Herranz et al. “Cloud robotics in FIWARE: A proof of concept”. In: *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)* 9121 (June 2015), pp. 580–591. DOI: 10.1007/978-3-319-19644-2-48.

[53] S. Himesh. “Digital revolution and Big Data: a new revolution in agriculture.” In: 13.021 (Apr. 2018). DOI: 10.1079/pavsnnr201813021. URL: https://doi.org/10.1079/pavsnnr201813021.

[54] *Home - Centre for Automation and Robotics*. https://www.car.upm-csic.es/. (Accessed on 11/29/2021).

[55] *Home - Fiware-iotagent-json*. https://fiware-iotagent-json.readthedocs.io/en/latest/. (Accessed on 11/20/2021).

[56] *Home - Fiware-iotagent-lwm2m*. https://fiware-iotagent-lwm2m.readthedocs.io/en/latest/. (Accessed on 11/20/2021).

[57] *Home - Fiware-iotagent-ul*. https://fiware-iotagent-ul.readthedocs.io/en/latest/. (Accessed on 11/20/2021).

[58] *Home - LoRaWAN IoT Agent*. https://fiware-lorawan.readthedocs.io/en/latest/. (Accessed on 11/20/2021).

[59] *How big data will change agriculture - Proagrica*. https://web.archive.org/web/20210224171347/https://proagrica.com/news/how-big-data-will-change-agriculture/. (Accessed on 11/13/2021).

[60] Bo HU et al. “Design and Implementation of Amphibious Robot for Subsea Cable Maintenance”. In: *DEStech Transactions on Engineering and Technology Research* (Dec. 2017). DOI: 10.12783/dtetr/ameme2017/16187.

[61] Guoqiang Hu, Wee Peng Tay, and Yonggang Wen. “Cloud robotics: architecture, challenges and applications”. In: *IEEE network* 26.3 (2012), pp. 21–28.

[62] Guoqiang Hu, Wee Peng Tay, and Yonggang Wen. "Cloud robotics: architecture, challenges and applications". In: *IEEE network* 26.3 (2012), pp. 21–28.

[63] Michael Hüttermann. *DevOps for developers*. Apress, 2012.

[64] Masayuki Inaba et al. "A platform for robotics research based on the remote-brained robot approach". In: *The International Journal of Robotics Research* 19.10 (2000), pp. 933–954.

[65] *IOTAgent-JSON DockerHub*. URL: https://web.archive.org/web/20190506012643/https://hub.docker.com/r/fiware/iotagent-json/ (visited on 10/20/2021).

[66] *istr-23-2018-en*. https://docs.broadcom.com/doc/istr-23-2018-en. (Accessed on 11/14/2021).

[67] TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU. *Overview of the Internet of things*. Tech. rep. 2012.

[68] *kafka-python · PyPI*. https://pypi.org/project/kafka-python/. (Accessed on 11/24/2021).

[69] Andreas Kamilaris, Andreas Kartakoullis, and Francesc X. Prenafeta-Boldú. "A review on the practice of big data analysis in agriculture". In: *Computers and Electronics in Agriculture* 143 (2017), pp. 23–37. ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2017.09.037. URL: https://www.sciencedirect.com/science/article/pii/S0168169917301230.

[70] Artúr István Károly et al. "Deep learning in robotics: Survey on model structures and training strategies". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51.1 (2020), pp. 266–279.

[71] Nane Kratzke and Peter-Christian Quint. "Understanding Cloud-native Applications after 10 Years of Cloud Computing - A Systematic Mapping Study". In: *Journal of Systems and Software* 126 (Jan. 2017), pp. 1–16. DOI: 10.1016/j.jss.2017.01.001.

[72] Gurjan Lally and Daniele Sgandurra. "Towards a framework for testing the security of IoT devices consistently". In: *International workshop on emerging technologies for authorization and authentication*. Springer. 2018, pp. 88–102.

[73] Raffaele Limosani et al. "Connecting ROS and FIWARE: Concepts and Tutorial". In: *Robot Operating System (ROS): The Complete Reference (Volume 3)*. Ed. by Anis Koubaa. Cham: Springer International Publishing, 2019, pp. 449–475. ISBN: 978-3-319-91590-6. DOI: 10.1007/978-3-319-91590-6_13. URL: https://doi.org/10.1007/978-3-319-91590-6_13.

[74] Antonio Maccioni and Riccardo Torlone. "KAYAK: a framework for just-in-time data preparation in a data lake". In: *International Conference on Advanced Information Systems Engineering*. Springer. 2018, pp. 474–489.

[75] Mohsen Marjani et al. “Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges”. In: *IEEE Access* 5 (Mar. 2017). DOI: 10.1109/access.2017.2689040.

[76] Robert Cecil Martin. *Agile Software Development: Principles, Patterns, and Practices*. USA: Prentice Hall PTR, 2003. ISBN: 0135974445.

[77] Ibrahim Mashal et al. “Choices for Interaction with Things on Internet and Underlying Issues”. In: *Ad Hoc Netw.* 28.C (May 2015), pp. 68–90. ISSN: 1570-8705. DOI: 10.1016/j.adhoc.2014.12.006. URL: https://doi.org/10.1016/j.adhoc.2014.12.006.

[78] Gerard T McKee and Paul S Schenker. “Networked robotics”. In: *Sensor Fusion and Decentralized Control in Robotic Systems III*. Vol. 4196. International Society for Optics and Photonics. 2000, pp. 197–209.

[79] Natalia Miloslavskaya and Alexander Tolstoy. “Big data, fast data and data lake concepts”. In: *Procedia Computer Science* 88 (2016), pp. 300–305.

[80] Răzvan MUREȘAN. *Gartner: IoT breaches may result in physical damage. People safety becomes the primary goal.* URL: https://web.archive.org/web/20211026154733/https://www.bitdefender.com/blog/hotforsecurity/gartner-iot-breaches-may-result-in-physical-damage-people-safety-becomes-the-primary-goal (visited on 10/25/2021).

[81] Rajkumar Murugesan et al. “Artificial Intelligence and Agriculture 5 . 0”. In: 8 (July 2019), pp. 1870–1877.

[82] Sergi Nadal et al. “An integration-oriented ontology to govern evolution in Big Data ecosystems”. In: *Information Systems* 79 (2019). Special issue on DOLAP 2017: Design, Optimization, Languages and Analytical Processing of Big Data, pp. 3–19. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2018.01.006. URL: https://www.sciencedirect.com/science/article/pii/S0306437917304660.

[83] Saeed Niku. *Introduction to robotics : analysis, control, applications*. Hoboken, N.J: Wiley, 2011. ISBN: 978-0-470-60446-5.

[84] *Node.js*. https://nodejs.org/en/. (Accessed on 11/24/2021).

[85] Eiman Al Nuaimi et al. “Applications of big data to smart cities”. In: 6.1 (Aug. 2015). DOI: 10.1186/s13174-015-0041-5. URL: https://doi.org/10.1186/s13174-015-0041-5.

[86] *OpenLayers - Welcome*. https://openlayers.org/. (Accessed on 11/24/2021).

[87] *Orion Context Broker: introduction to context management (I) - FIWARE*. https://www.fiware.org/2015/02/19/orion-context-broker-introduction-to-context-management-i/. (Accessed on 11/04/2021).

[88] Thorsten Papenbrock et al. “Data profiling with metanome”. In: *Proceedings of the VLDB Endowment* 8.12 (2015), pp. 1860–1863.

[89] Charikleia Paschalidi. *Data Governance: A conceptual framework in order to prevent your Data Lake from becoming a Data Swamp*. 2015.

[90] *Performance tuning - Fiware-Orion*. URL: https://web.archive.org/web/20211020100450/https://fiware-orion.readthedocs.io/en/master/admin/perf_tuning/index.html (visited on 10/20/2021).

[91] Thinagaran Perumal, Soumya Kanti Datta, and Christian Bonnet. “IoT device management framework for smart home scenarios”. In: *2015 IEEE 4th Global Conference on Consumer Electronics (GCCE)*. 2015, pp. 54–55. DOI: 10.1109/GCCE.2015.7398711.

[92] Harry A Pierson and Michael S Gashler. “Deep learning in robotics: a review of recent research”. In: *Advanced Robotics* 31.16 (2017), pp. 821–835.

[93] Harry A. Pierson and Michael S. Gashler. “Deep Learning in Robotics: A Review of Recent Research”. In: *CoRR* abs/1707.07217 (2017). arXiv: 1707.07217. URL: http://arxiv.org/abs/1707.07217.

[94] *Plane Trig p 01*. https://web.archive.org/web/19991010004728/http://www.geocities.com/ResearchTriangle/2363/trig02.html. (Accessed on 11/04/2021).

[95] Valentin L Popov and Markus Heß. *Method of dimensionality reduction in contact mechanics and friction*. Springer, 2015.

[96] *Precision agriculture yields higher profits, lower risks | HPE*. https://web.archive.org/web/20201109031303/https://www.hpe.com/us/en/insights/articles/precision-agriculture-yields-higher-profits-lower-risks-1806.html. (Accessed on 11/13/2021).

[97] Christoph Quix, Rihan Hai, and Ivan Vatov. “Metadata extraction and management in data lakes with GEMMS”. In: *Complex Systems Informatics and Modeling Quarterly* 9 (2016), pp. 67–83.

[98] Franck Ravat and Yan Zhao. “Data lakes: Trends and perspectives”. In: *International Conference on Database and Expert Systems Applications*. Springer. 2019, pp. 304–313.

[99] N. Reddy et al. “A critical review on agricultural robots”. In: *International Association of Engineering and Management Education* 7 (July 2016), pp. 183–188.

[100] *Remote Monitoring and Mainteneance: Mission-Critical Operationsat the Competitive Edge*. URL: https://www.oracle.com/us/solutions/internetofthings/iot-remote-monitoring-brief-2881653.pdf.

[101] *ROS Community Metrics Report*. 2017. URL: http://download.ros.org/downloads/metrics/metrics-report-2017-07.pdf.

[102] Philip Russom et al. “Big data analytics”. In: *TDWI best practices report, fourth quarter* 19.4 (2011), pp. 1–34.

[103] Olimpiya Saha and Prithviraj Dasgupta. “A comprehensive survey of recent trends in cloud robotics architectures and applications”. In: *Robotics* 7.3 (2018), p. 47.

[104] Verónica Saiz-Rubio and Francisco Rovira-Más. “From smart farming towards agriculture 5.0: A review on crop data management”. In: *Agronomy* 10.2 (2020), p. 207.

[105] Pegdwendé N Sawadogo et al. “Metadata systems for data lakes: models and features”. In: *European conference on advances in databases and information systems*. Springer. 2019, pp. 440–451.

[106] David Schimmelpfennig. “Farm Profits and Adoption of Precision Agriculture”. In: Economic Research Report 1477-2016-121190 (2016), p. 46. DOI: 10.22004/ag.econ.249773. URL: http://ageconsearch.umn.edu/record/249773.

[107] Redmond Ramin Shamshiri et al. “Research and development in agricultural robotics: A perspective of digital farming”. In: 11.4 (2018), pp. 1–11. DOI: 10.25165/j.ijabe.20181104.4278. URL: https://doi.org/10.25165/j.ijabe.20181104.4278.

[108] Sugam Sharma. “Expanded cloud plumes hiding Big Data ecosystem”. In: *Future Generation Computer Systems* 59 (2016), pp. 63–92. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2016.01.003. URL: https://www.sciencedirect.com/science/article/pii/S0167739X16000054.

[109] Keith Shea and Wind River. *DEVICE MANAGEMENT IN THE INTERNET OF THINGS – Why It Matters and How to Achieve It*. URL: /web/20211026154658/https://www.new-techeurope.com/2017/06/07/device-management-internet-things-matters-achieve/ (visited on 10/25/2021).

[110] Phil Simon. *The visual organization: Data visualization, big data, and the quest for better decisions*. John Wiley & Sons, 2014.

[111] *Spark Streaming | Apache Spark*. https://spark.apache.org/streaming/. (Accessed on 11/24/2021).

[112] *Structured Streaming Programming Guide - Spark 3.2.0 Documentation*. https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html. (Accessed on 11/24/2021).

[113] Ignacio G Terrizzano et al. “Data Wrangling: The Challenging Yourney from the Wild to the Lake.” In: *CIDR*. 2015.

[114] *The 4 Trends That Prevail on the Gartner Hype Cycle for AI, 2021.* https://web.archive.org/web/20211113155019/https://www.gartner.com/en/articles/the-4-trends-that-prevail-on-the-gartner-hype-cycle-for-ai-2021. (Accessed on 11/13/2021).

[115] *The Open Source platform for our smart digital future - FIWARE.* https://www.fiware.org/. (Accessed on 11/03/2021).

[116] *The robots are coming to clean up our nuclear sites.* https://web.archive.org/web/20211122141655/https://cordis.europa.eu/article/id/358596-the-robots-are-coming-to-clean-up-our-nuclear-sites. (Accessed on 11/22/2021).

[117] Giovanni Toffetti and Thomas Michael Bohnert. "Cloud Robotics with ROS". In: *Robot Operating System (ROS): The Complete Reference (Volume 4).* Ed. by Anis Koubaa. Cham: Springer International Publishing, 2020, pp. 119–146. ISBN: 978-3-030-20190-6. DOI: 10.1007/978-3-030-20190-6_5. URL: https://doi.org/10.1007/978-3-030-20190-6_5.

[118] Giovanni Toffetti and Thomas Michael Bohnert. "Cloud robotics with ROS". In: *Robot operating system (ROS).* Springer, 2020, pp. 119–146.

[119] Chun-Wei Tsai et al. "Big data analytics: a survey". In: *Journal of Big data* 2.1 (2015), pp. 1–32.

[120] Giuliano Vitali et al. "Crop Management with the IoT: An Interdisciplinary Survey". In: *Agronomy* 11.1 (2021). ISSN: 2073-4395. DOI: 10.3390/agronomy11010181. URL: https://www.mdpi.com/2073-4395/11/1/181.

[121] Giuliano Vitali et al. "Crop Management with the IoT: An Interdisciplinary Survey". In: *Agronomy* 11.1 (2021). ISSN: 2073-4395. DOI: 10.3390/agronomy11010181. URL: https://www.mdpi.com/2073-4395/11/1/181.

[122] Jeffrey M Voas. *Networks of 'things'.* Tech. rep. 2016. DOI: 10.6028/nist.sp.800-183. URL: https://doi.org/10.6028/nist.sp.800-183.

[123] Markus Waibel et al. "RoboEarth". In: 18.2 (June 2011), pp. 69–82. DOI: 10.1109/mra.2011.941632. URL: https://doi.org/10.1109/mra.2011.941632.

[124] Lidong Wang, Guanghui Wang, and Cheryl Ann Alexander. "Big data and visualization: methods, challenges and technology progress". In: *Digital Technologies* 1.1 (2015), pp. 33–38.

[125] *We Laser Project: Eco-Innovative weeding with laser.* https://welaser-project.eu/. (Accessed on 11/05/2021).

[126] *Welcome to Python.org.* https://www.python.org/. (Accessed on 11/24/2021).

[127] *What are microservices?* https://web.archive.org/web/20211103203108/https://microservices.io/. (Accessed on 11/03/2021).

[128] *What is Application Integration? | IBM.* https://web.archive.org/web/20210308134533/https://www.ibm.com/cloud/learn/application-integration. (Accessed on 11/03/2021).

[129] *What is IoT in Agriculture? Farmers Aren't Quite Sure Despite $4bn US Opportunity - report - AgFunderNews.* https://web.archive.org/web/20210507032040/https://agfundernews.com/iot-agriculture-farmers-arent-quite-sure-despite-4bn-us-opportunity.html. (Accessed on 11/13/2021).

[130] Benjamin Wootton. *Microservices - Not a free lunch! - High Scalability -.* https://web.archive.org/web/20211022104722/http://highscalability.com/blog/2014/4/8/microservices-not-a-free-lunch.html. (Accessed on 11/03/2021). Apr. 2014.

[131] Ilaria Zambon et al. "Revolution 4.0: Industry vs. Agriculture in a Future Development for SMEs". In: *Processes* 7.1 (2019). ISSN: 2227-9717. DOI: 10.3390/pr7010036. URL: https://www.mdpi.com/2227-9717/7/1/36.

[132] Yanyong Zhang et al. "ICN based Architecture for IoT". In: *IRTF contribution, October* (2013).