This project has received funding from the European's Union Horizon 2020 research innovation program under grant agreement No. 769355

PIXEL Information System architecture and design - Version 2

Deliverable No.	D.6.2	**Due Date**	31/10/2019
Type	Report	**Dissemination Level**	Public
Version	1.0	**Status**	Release 2
Description	This document is the second version of the final architecture. It reports about the PIXEL Reference Architecture (analysis and design of the different modules).		
Work Package	WP6		

Authors

Name	Partner	e-mail
Carlos E. Palau	P01 UPV	cpalau@dcom.upv.es
Benjamin Molina	P01 UPV	benmomo@upvnet.upv.es
Ignacio Lacalle	P01 UPV	igluab@upv.es
Miguel A. Llorente	P02 PRO	mllorente@prodevelop.es
José A. Clemente	P02 PRO	jclemente@prodevelop.es
Flavio Fuart	P03 XLAB	flavio.fuart@xlab.si
Gašper Vrhovšek	P03 XLAB	gasper.vrhovsek@xlab.si
Dejan Štepec	P03 XLAB	dejan.stepec@xlab.si
Vito Čuček	P03 XLAB	vito.cucek@xlab.si
Marko Kuder	P03 XLAB	marko.kuder@xlab.si
Marjan Šterk	P03 XLAB	marjan.sterk@xlab.si
Marc Despland	P06 ORANGE	marc.despland@orange.com
Charles Garnier	P05 CATIE	c.garnier@catie.fr
Erwan Simon	P05 CATIE	e.simon@catie.fr
Eirini Tserga	P10 ThPA SA	etserga@thpa.gr
Gilda De Marco	P04 INSIEL	gilda.demarco@insiel.it
Thanassis Chaldeakis	P11 PPA SA	ahaldek@gmail.com

History

Date	Version	Change
10-June-2019	0.1	ToC and task assignments
22-October-2019	0.2	First draft of the document
20-Nov-2019	0.3	Internal Review

Key Data

Keywords	Reference Architecture, IoT, Data Acquisition Layer, Information Hub, Operational Tools, Dashboard, Notifications, Security, Deployment, Agile Development.
Lead Editor	Jose A. Clemente P02, PRO
Internal Reviewer(s)	GPMB, CREOCEAN

Abstract

This deliverable contains the second version of the PIXEL architecture for the information systems and its initial design. It reports all the analysis and design activities performed throughout the project until month 18 (M18).

The document describes the process of defining an IoT platform to perform the ICT-supported activities of PIXEL: gathering IoT and structured data from ports to improve efficiency of processes and operations, by means of the application of simulations and prediction based on models and algorithms, finally leading to a better environmental performance of the port.

The document firstly describes the concept of reference architecture focusing on those that serve as a basis for PIXEL. These are **RAMI** and **IIRA** for their European and global relevance and **IoT-A** for its orientation towards interoperability, a main requirement of PIXEL.

After this, a chapter is presented about the relation between Agile Development and the chosen Architecture or how the features of Agile Development (**Containers**, **APIs**) will intervene in the development of PIXEL.

The next section (section 4) presents the global architecture of the solution and the functional blocks with their components. In first instance the global architecture is described to give the full picture of the software solution proposed. After this, the different functional blocks are presented.

The **Data Acquisition Layer** (DAL) has the mission of gathering data from heterogeneous data sources and persist them in a single storing point. The solution proposed for this is the creation of IoT Agents adapted to the different data sources generic formats (some examples are provided in section **5.3 Data Examples**), giving balance between flexibility and standardization. Other additional components of this block are the data hub, the short term historic or the proxies for agents. The central broker receives and pushes the data to the next instance layer.

The **Information Hub** (IH) is a functional block in charge of centralising all the data retrieved from DAL, homogenising and storing in a database capable to support big queries and scale horizontally. Unlike the DAL, the Information Hub is designed to be high performant and scalable, and the data is stored to support long-term queries. This is considered the central storage point of the IoT solution in PIXEL and is the block that replies the queries from other functional blocks (such as the Operational Tools or Dashboards). The IH's main components are a high-performance data broker and a NoSQL database, although it contains accessory components that support its correct functioning.

The **Operational Tools** (OT) are devoted to enable the analysis and reasoning over the data gathered by the platform both in real time and in batch processes. These tools are built to support models and algorithms developed in the activities of WP4. OT includes a publication interface, model engine, a predictive algorithms engine, an event processor and an internal database.

The **Dashboard and Notifications** (DN) module has the capability of representing the data registered in the IH in meaningful combined visualizations in real time. Also it provides the capability to send notifications based on the status of the data of the IH. Finally, this module provides (aggregates and homogenises) all the user interfaces for the different functional blocks. This block is composed by the notifications engine, chart and dashboards agent and the different UI agents for the modules.

Finally, the Security block is a cross-layer module that will perform the security of the other blocks, including authentication and authorization control. This architecture is based in standard architectural approaches, following best practices in the field of internet security.

In section 5 the **Information model** is described. This chapter will also contain examples of already defined models such as the data that are received and what are the transformations that take place in the DAL to normalize the data.

The document also describes how PIXEL will lead with deployment and scalability (multi-instance, multi-tenant). Environment and testing strategies are also discussed in these sections, detailing the principles of multi-environment deployment and testing activities. A section has been included to detail how the deployment will take place in the different ports, building a link with WP7.

The final chapter summarises the conclusions of the document and establishes the links to depending activities in related packages (WP4, WP6 and WP7).

Statement of originality

Table of contents

List of tables

List of figures

List of acronyms

Acronym	Explanation
ACID	Atomicity, Consistency, Isolation, Durability
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
BI	Business Intelligence
CD	Continuous Delivery
CEP	Complex Event Processing
CI	Continuous Integration
CLI	Command Line Interface
CN	Core Network
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CQL	Cassandra Query Language
CCSA	China Communications Standards Association
CSV	Comma Separated Values
DB	Database
ESB	Enterprise Service Bus
ETSI	European Telecommunications Standards Institute
FP7	Seventh Framework Programme
GIS	Geographical Information System
GPMB	Grand Port Maritime de Bordeaux - Port of Bordeaux
HA	Highly-available
HTTP	HyperText Transfer Protocol
IDM	Identity Manager
IIOT	Industrial IoT
IMO	International Maritime Organization
IoT	Internet of Things
IoT-A	Internet of Things Architecture
IP	Internet Protocol
IIRA	Industrial Internet Reference Architecture
ITU	International Telecommunication Union
JSON	JavaScript Object Notation
KMZ	Keyhole Markup Language
LDAP	Lightweight Directory Access Protocol
LAN	Local Area Network
LTS	Long-Term Storage
LXC	Linux Containers
MQTT	Message Queuing Telemetry Transport
M2M	machine-to-machine
NGSI	Next Generation Service Interfaces
OT	Operational Tools
PAN	Personal Area Network

PAP	Policy Administration Point
PDCA	Plan – Do – Check – Act
PDP	Policy Decision Point
PEI	Port Environmental Index
PEP	Policy Enforcement Point
PIP	Policy Information Point
PIXEL	Port IoT for Environmental Leverage
PM_{10}	Particulate Matter 10µm
$PM_{2.5}$	Particulate Matter 2.5µm
PPA	Piraeus Port Authority
PRP	Policy Retrieval Point
KPI	Key Performance Indicators
RA	Reference Architecture
RAMI	Reference Architectural Model Industry
RDBMS	Relational Database Management System
REST	Representational State Transfer
RFID	Radio Frequency Identification
SMS	Short Message Service
SNS	Simple Notification Service
SOA	Service-oriented architecture
SQL	Structured Query Language
SSL	Secure Sockets Layer
STOMP	Simple Text Oriented Messaging Protocol
STS	Short-Term Storage
TLS	Transport Layer Security
THPA	Thessaloniki Port Authority
UI	User Interface
VMPS	VLAN Membership Policy Server
WP	Work Package
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol

1. About this document

The aim of the deliverable is to present the definitive architecture that will be used in PIXEL (after the first version in D6.1), as well as to identify and analyse the different modules at a high level view.

For the technological choices work of previous deliverables (D3.2 and D3.4, Definition of Requirements and Use Cases) has been considered.

This document complements the information reported in D6.1 and explains decisions of the software release of D6.3. The following sections have been newly included:

- Applied Agile concepts
- Integration aspects

While the following ones have been complemented or fixed:

- Reference Architecture
- Functional Architecture
- Information Model
- Deployment and Scalability

This document is intended to be used as a reference for software developers to have a general vision of the technical aspects of the different functional modules including the interaction between components, data flows, interfaces and design decisions. Although it is primarily intended for technical readers, its general perspective might be mostly understandable by non-technical readers.

1.1. Deliverable context

Table 1: Deliverable context

Keywords	**Lead Editor**
Objectives	*Objective 1: Enable the IoT-based connection of port resources, transport agents and city sensor networks,* This document represents the second and final part of the specification of the information system that will support PIXEL activities, including deployment in pilot sites.
	Objective 2: Achieve an automatic aggregation, homogenization and semantic annotation of multi-source heterogeneous data from different internal and external actors. The document describes the information system that enables the features of this objective. In particular, the **data acquisition layer** and the **information hub** have the responsibility of achieving this.
	Objective 3: Develop an operational management dashboard to enable a quicker, more accurate and in-depth knowledge of port operations. This document describes the architectural and design lines that fulfil this objective. The functional block in charge of it is the **dashboard and notifications** module.
	Objective 4: Model and simulate port operations processes for automated optimisation. This objective is technologically supported by the **operational tools** building block. Therefore, although this deliverable does not address directly the modelling and simulation, it sets the underpinning software

	elements that make possible the use of modelling and simulations in port sector.
	Objective 5: Develop predictive algorithms. Similar as for the models, the **operational tools** will support the predictive algorithms developed in WP4, so that port staff is able to configure and execute them.
Work plan	This deliverable reports about the work performed in task 6.1. The contents are a fundamental input for tasks T6.2, T6.3, T6.4 T6.5 and T6.6. The results reported in this document are also important for the work done in WP4, to develop the models and predictive algorithms to be integrated in the operational tools module.
Milestones	Direct contribution to MS7 (ICT solution developed, M26). Indirect contribution to MS5 (Predictive models/algorithms, M24.
Deliverables	Detected inputs: • D6.1: D6.2 is the second iteration of this document with the aim to describe the final architecture. • All the inputs of D6.1 are valid for document D6.2. These are: ○ D3.2: D6.1 considers all requirements listed in order to provide an initial supporting architecture. ○ D3.4: Use cases and scenarios manual v2: D6.1 considers the different defined use cases of scenarios to identify main building blocks and interactions. ○ D4.2 PIXEL Models v2: D6.1 includes the Operational Tools module responsible to import, configure and run models described in D4.1; D4.2 is the updated version of D4.1 releases in M18. Detected outputs: • D6.4 PIXEL Data Acquisition, Information Hub and Data Representation v2: Some functional blocks will be described in detail in separate documents and D6.1/D6.2 sets the base layer for this. • D6.5 APIs and documentation for software extension: D6.2 allows identifying the interactions among the functional blocks and therefore defining the APIs.
Risks	This deliverable helps mitigating the risks 6, 8, 14, 16, 17 and 18.

1.2. The rationale behind the structure

This document introduces the need of reference architecture (RA) and what are the characteristics that reference architecture should have?

The document also describes the architecture adopted and the different modules of PIXEL's RA:

- **PIXEL Data Acquisition** (see PIXEL Data acquisition).
- **PIXEL Information Hub** (see PIXEL Information Hub).
- **PIXEL Operational Tools** (see PIXEL Operational Tools).
- **PIXEL Integrated Dashboard and Notifications** (see PIXEL Integrated Dashboard and Notifications).
- **PIXEL Security and Privacy** (see PIXEL Security).

1.3. Version-specific notes

This document is the second version of the deliverable "**PIXEL Information System architecture and design**" (D6.2). This document aims to clarify the doubts that may remain outstanding in D6.1 and fix the architecture on which the development of PIXEL is based.

This document provides a final version of the architecture according to the works done until month 18. It will also contain some interfaces about the PIXEL design and will deepen the deployment in the different ports.

This document is the closing result of task 6.1, which is enhancing D6.1. Therefore, the results of D6.1 are essentially gathered in these two documents, which are complementary and, in case of update of any information, this is conveniently declared in D6.2.

2. Reference Architecture (RA)

The concept of reference architecture refers to the component and services layout and best practises of an IT system that is likely to be implemented recurrently with similar objectives but different contexts, constraints or business variations. It is not the scope of this document to explore comprehensively the concept of RA and its usefulness in IoT scenarios, to know more about this, there is extensive literature available[1] (Perry, 2018)(Fremantle, 2015).

PIXEL challenges such as the establishment of an IoT Platform valid for very heterogeneous conditions (different port sizes, different port operations, different areas and KPIs monitored…), makes very appropriate for the project the definition of reference architecture to keep a common technological and functional blueprint event when the deployments vary and the requirements are in some cases disparate.

Thus, establishing a RA and architectural patterns is a good practise in the scenarios faced in PIXEL. A RA for IoT in ports will lay the foundation for a common framework for the development of future systems and communications between market players. In addition, a RA is an important component of standardisation, contributing in the medium term to reducing costs for ports compared to each of the individual solutions currently available. Some other reasons for defining a RA before specifying the architecture are the following[2]:

- IoT devices are inherently connected. We need a way to uniformly interact with them.
- There are billions of devices in the market and the number is growing quickly. So, it becomes necessary to have a scalable architecture. Requirements vary among deployments even of the same technology and they also change throughout time. The need for adaptation is continuous.
- Management of devices (automatic updates, remote management) is needed, and these devices can change, evolve, be deprecated, substituted, etc.
- Security. These devices collect sensitive data thus it is necessary to establish a security layer that controls the communication between devices or with the platform that receives the data. Security protocols, patterns and technologies changes across devices and time.
- Provides a starting point for architects looking to create IoT solutions as well as a strong basis for further development (PIXEL).

According to these conditions, the use of a RA provides stability and reliability of the designed solution across multiple scenarios (as is the case in PIXEL) and throughout the time.

It is important to distinguish that a RA is more abstract than a system architecture that has been developed for a specific set of applications (PIXEL) with particular constraints and scenarios[3].

Due to the heterogeneity of concepts and technologies a RA for IoT is more complex than a traditional architecture due to relationships between the different technologies used.

It is important to understand the impact on scalability and other parts of the system when choosing a certain design aspect.

According to the book: **"Internet of Things for Architects"** (Perry, 2018), currently there are over 1.5 million different combinations of architectures to choose from.

[1] http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf

[2] Paul Fremantle. A Reference Architecture for the Internet of Things: https://www.researchgate.net/publication/308647314_A_Reference_Architecture_for_the_Internet_of_Things

[3] https://iotforum.org/wp-content/uploads/2014/09/120613-IoT-A-ARM-Book-Introduction-v7.pdf

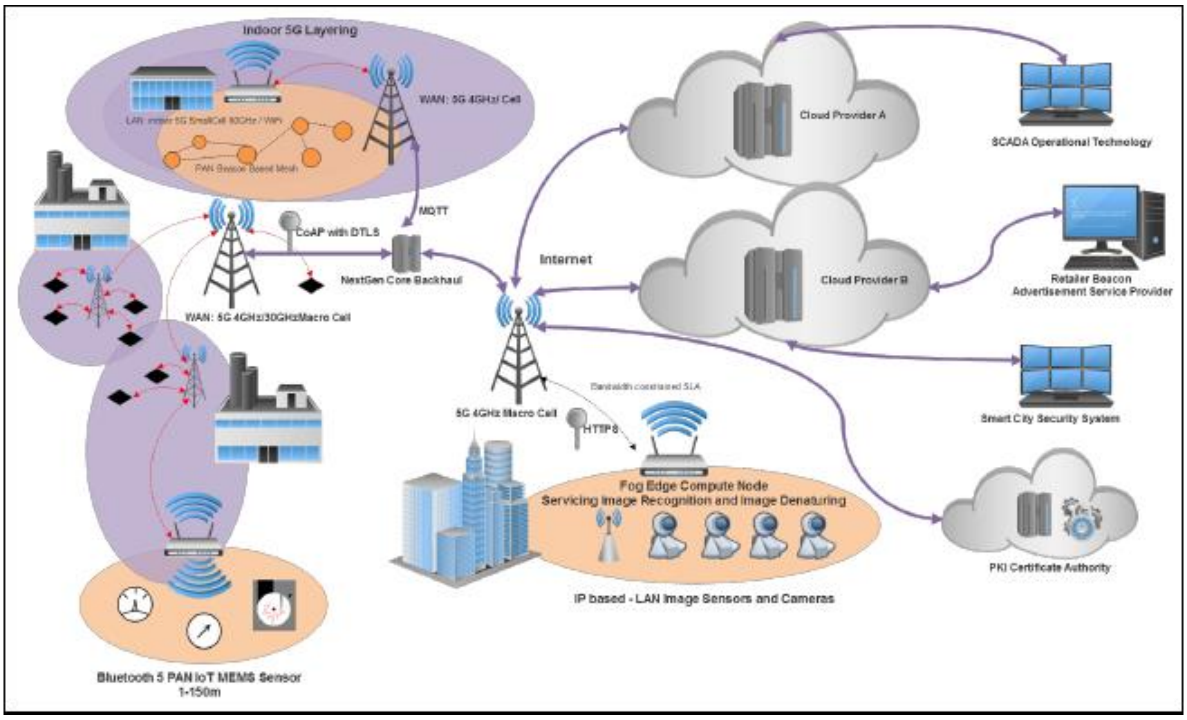

Indoor 5G Layering
WAN: 5G 4GHz/ Cell
LAN: Indoor 5G SmallCell 60GHz / WiFi
PAN Sensor Board Mesh
MQTT
CoAP with DTLS
NextGen Core Backhaul
WAN: 5G 4GHz/30GHzMacro Cell
Internet
5G 4GHz Macro Cell
Bandwidth constrained SLA
HTTPS
Fog Edge Compute Node
Servicing Image Recognition and Image Denaturing
IP based - LAN Image Sensors and Cameras
Bluetooth 5 PAN IoT MEMS Sensor
1-150m
Cloud Provider A
SCADA Operational Technology
Cloud Provider B
Retailer Beacon
Advertisement Service Provider
Smart City Security System
PKI Certificate Authority

Figure 1: IoT Design Choices. The full spectrum of various levels of IoT architecture from the sensor to cloud and back

Architectures, in general, cover different perspectives to describe the whole system and the internal interactions:

- **Functional elements**.
- **Interaction between elements**.
- **Information management**.
- **Operational features**.
- **Deployment of the system**.

These architectural viewpoints are described in the following sections, adapted for the PIXEL project and its main objectives.

With this brief introduction, the RA is defined as the tool expected by the systems architect to create the foundations of a reliable, secure, future-proof and resilient IoT platform. This is, in a modular way through blocks and flexible design options, able to cover and describe specific systems encompassing functional requirements, performance, and deployment, standardization of interfaces, security and connectivity with its environment[4].

[4] http://www.academia.edu/7197061/Estado_del_Arte_de_las_Arquitecturas_de_Internet_de_las_Cosas_IoT

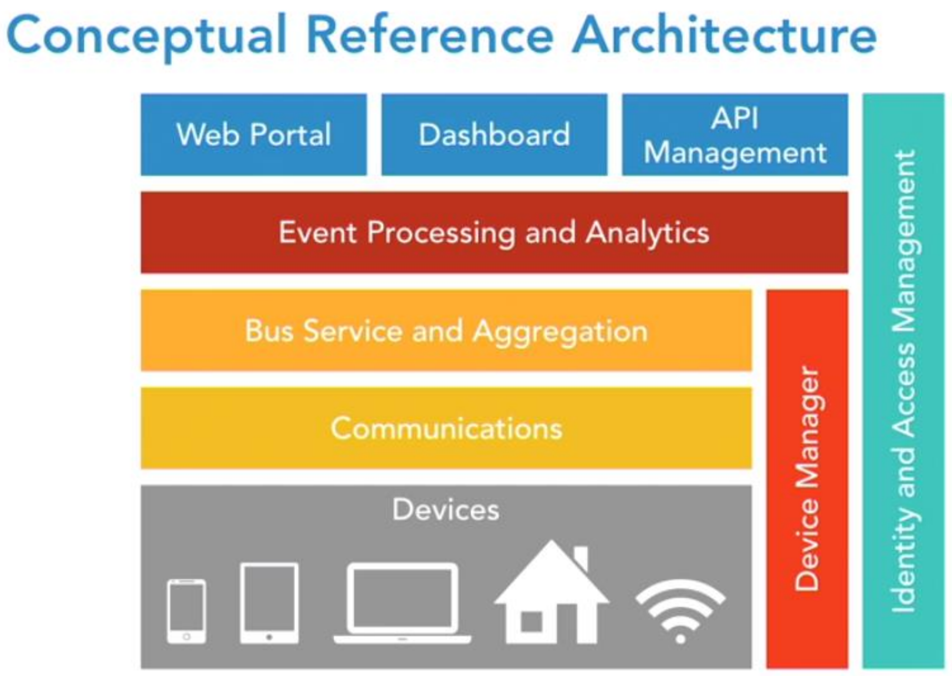

Conceptual Reference Architecture
Web Portal
Dashboard
API Management
Event Processing and Analytics
Bus Service and Aggregation
Communications
Devices
Device Manager
Identity and Access Management

Figure 2: Conceptual Reference Architecture

2.1.PIXEL RA

The aim of the PIXEL's RA is to modularly establish a series of components that meet the main needs/requirements of the PIXEL project.

Each of these components is intended to provide part or a complete solution to the different functional requirements of PIXEL. In addition, there is a transversal layer (as in Conceptual Reference Architecture) that manages security and access. In the following list the requirements needed to stablish a PIXEL's RA are described:

- *Connectivity and communications*. These features are related with the capacity to connect the different modules to each other and their interrelations. It spans all the components.
- *Device management*. It's important to have a centralized administration point to register devices (sensors, external platforms) that can interact with the platform and their special features (precision, range). This is located in the **PIXEL Information Hub**.
- *Data collection, analysis and actuation*. The data collection will take place in **PIXEL Data Acquisition module**s whereas the analysis and actuation will take place through the **PIXEL Operation Tools**.
- *Scalability*. This factor is critical in IoT platforms. Our RA needs to be scalable due to the great quantity of devices that exists now and will exist in the future, especially in the port sector.
- *Interoperability*. The PIXEL platform is defined as a **central data processing, warehouse and visualization point** for data from diverse sources in ports. This data can be generated by isolated devices, other IoT platforms, other IT systems or services or even document-based systems. This wide range of sources put the focus on the interoperability capabilities that the resulting architecture will support.
- *Security*. This aspect is one of the most important aspects of any IoT project. That is why our architecture will have a module for security (**PIXEL Security and Privacy**).

- *Predictive analysis*. This is one of the functionalities / tools that will be framed in the module **PIXEL Operational Tools**.
- *Integration*. According to the Perficient guide, The Why, What and How of IoT: 50+ examples across 11 industries[5], "Integration helps capture data from smart devices and move it into business applications to automate processes, support real-time monitoring and apply analytics for insights".
- *Visualization of the information*. It is important to have a good dashboard that allows understand correctly the data and the different simulations. For that we identify a module with a complete dashboard referenced as the **PIXEL Integrated Dashboard and Notifications**.

At the time to speak about RA it is possible to differentiate two types of IoT architectures: *generic* ones and *industrial* ones (**IIOT**).

In order to create a RA for PIXEL, an attempt has been made to reach a consensus on architecture with the best characteristics of both fields. For this reason, the chosen architecture presents characteristics of:

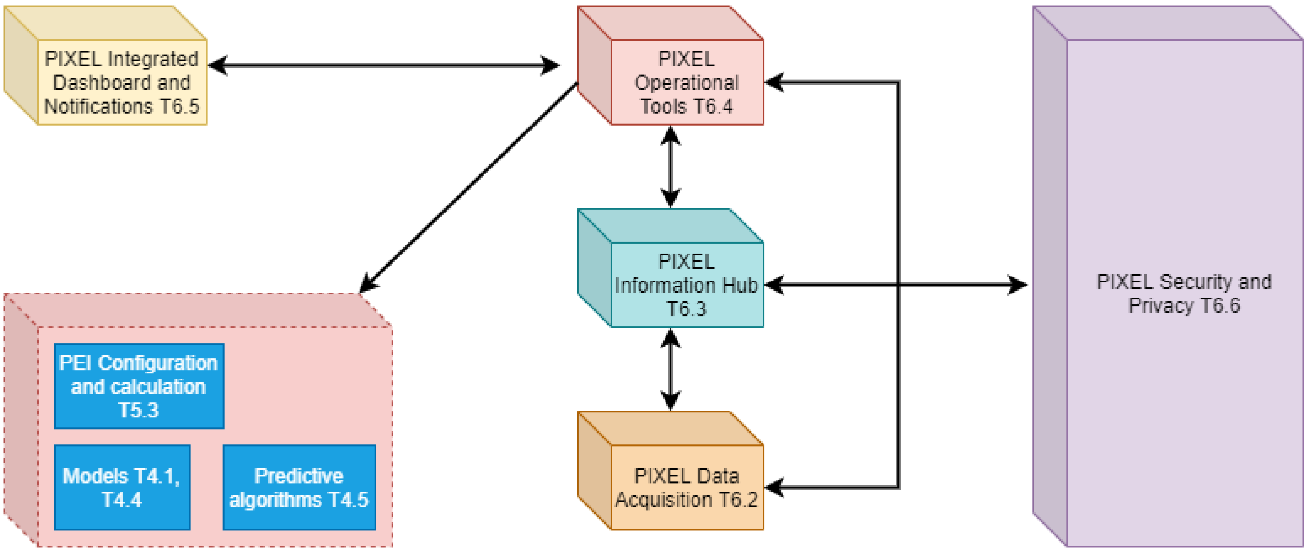
- **RAMI**
- **IIRA**
- **IoT-A**

RAMI and **IIRA** (both IIOT architectures) are complementary. IIRA analyses IIOT in all industries, with an emphasis on **homogeneity** and **interoperability** between industries, while RAMI 4.0 focuses on **manufacturing** and **related value chain life cycles**.

The great added value of both architectures consists in allowing **interoperability** between those IIOT systems that are built on the basis of these reference architectures.

IoT-A provides the understanding of an open architecture for the IoT and fully covers **security** and **privacy issues** as well as **scalability** and **interoperability** among other aspects.

Due to these features and aspects the following picture depicts modularly the RA of PIXEL:



PIXEL Integrated Dashboard and Notifications T6.5
PIXEL Operational Tools T6.4
PIXEL Information Hub T6.3
PIXEL Data Acquisition T6.2
PIXEL Security and Privacy T6.6
PEI Configuration and calculation T5.3
Models T4.1, T4.4
Predictive algorithms T4.5

Figure 3: PIXEL Reference Architecture

The next section will comment briefly the differences among **RAMI**, **IIRA** and **IoT-A**.

[5] https:// www.perficient.com/-/media/files/guide-pdf-links/the-why-what-and-how-of-iot.pdf

2.1.1. RAMI 4.0

RAMI define a service-oriented architecture, in which each of the components/modules provides services to the other components via a communication protocol across a network.

The principles of SOA architecture are independent of suppliers, products and technologies. The goal is to divide complex processes into packages that are easy to understand (as in **PIXEL**). This also includes data privacy and security (PIXEL Security).

RAMI promotes the principal aspects of the industry 4.0:

- *Interoperability*. Devices, machines and even people need to communicate between them.
- *Real-time data*. A Smart factory must be able to store data in real-time.
- *Service oriented*. Production is oriented to the client. The products are created following the specification of the clients.
- *Modularity*. The factories act as a module adapting it to the trends market, seasonality.

A major goal of RAMI 4.0 is to make sure that all participants involved in Industry 4.0 discussions and activities have a common framework to understand each other (as occurs in PIXEL).

The following figure illustrates the RAMI 4.0 architecture:

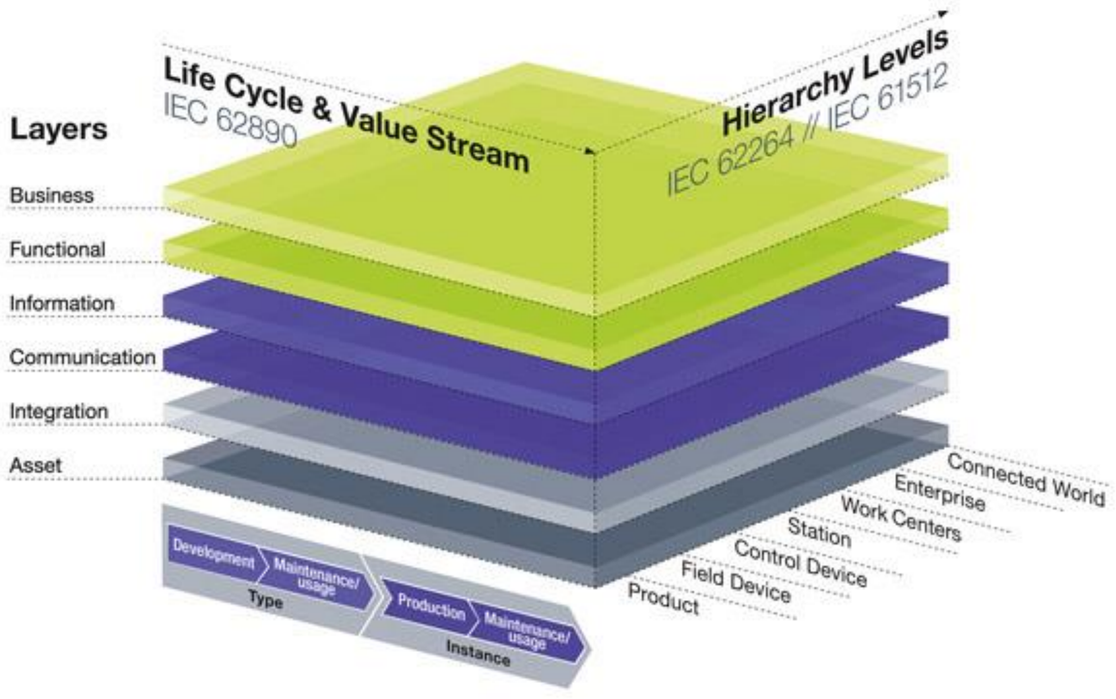


Figure 4: Reference Architectural Model Industry 4.0 (RAMI 4.0)

2.1.2. IIRA

IIRA is an architecture based on standards designed for IIOT system. The value of this architecture is its fast applicability (the life cycle of the product is taken into account).

The IIC[6] Architecture Task Group under the Technology Working Group is responsible for the IIRA.

It has been built and designed with a high level of abstraction with the idea to support the requisite broad industry applicability.

This architecture can be divided in four viewpoints:

- *Business*. Concerns to the identification of stakeholders and their business vision, values and objectives.
- *Usage*. Addresses the concerns of expected system usage.
- *Functional*. Focuses on the functional components, their interrelation and structure, the interfaces and interactions between them.
- *Implementation*. Deals with the technologies needed to implement functional components, their communication schemes and their lifecycle procedures. These components are coordinated by activities (Usage viewpoint) and supportive of the system capabilities (Business viewpoint).

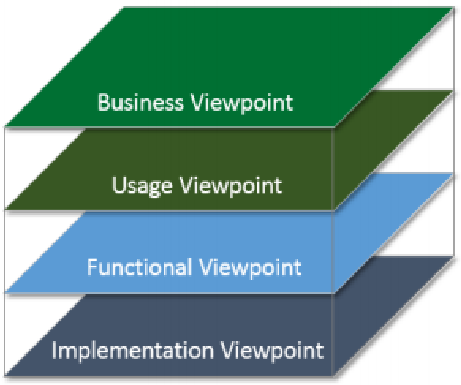


Figure 5: IIRA Architecture Viewpoints

2.1.3. IoT-A

IoT-A is an Architectural Reference Model proposed by European Commission (FP7[7]). The proposed RA introduces the concepts of views and perspectives. Its design is as follows:

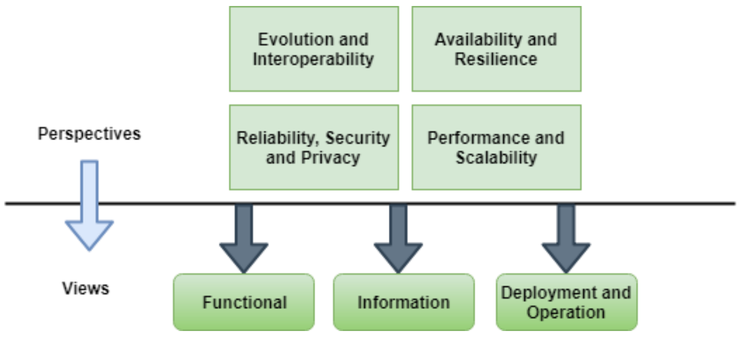

Evolution and Interoperability
Availability and Resilience
Perspectives
Reliability, Security and Privacy
Performance and Scalability
Views
Functional
Information
Deployment and Operation

Figure 6: IoT-A's Views and perspectives

- **Views**. "Different angles for viewing an architecture that can be used when designing and implementing it"[8]. The views include: Functional view, Information view, Deployment and operation view as seen in the image above.

[6] www.iiconsortium.org
[7] https://ec.europa.eu/research/fp7/index_en.cfm
[8] http://www.iot-a.eu/

- **Perspectives**. "Set of tasks, tactics, directives, and architectural decisions for ensuring that a given concrete system accomplishes one or more quality attributes"[9].

The image above illustrates the architecture from a functional point of view. The next figure shows the functional IoT-A model.

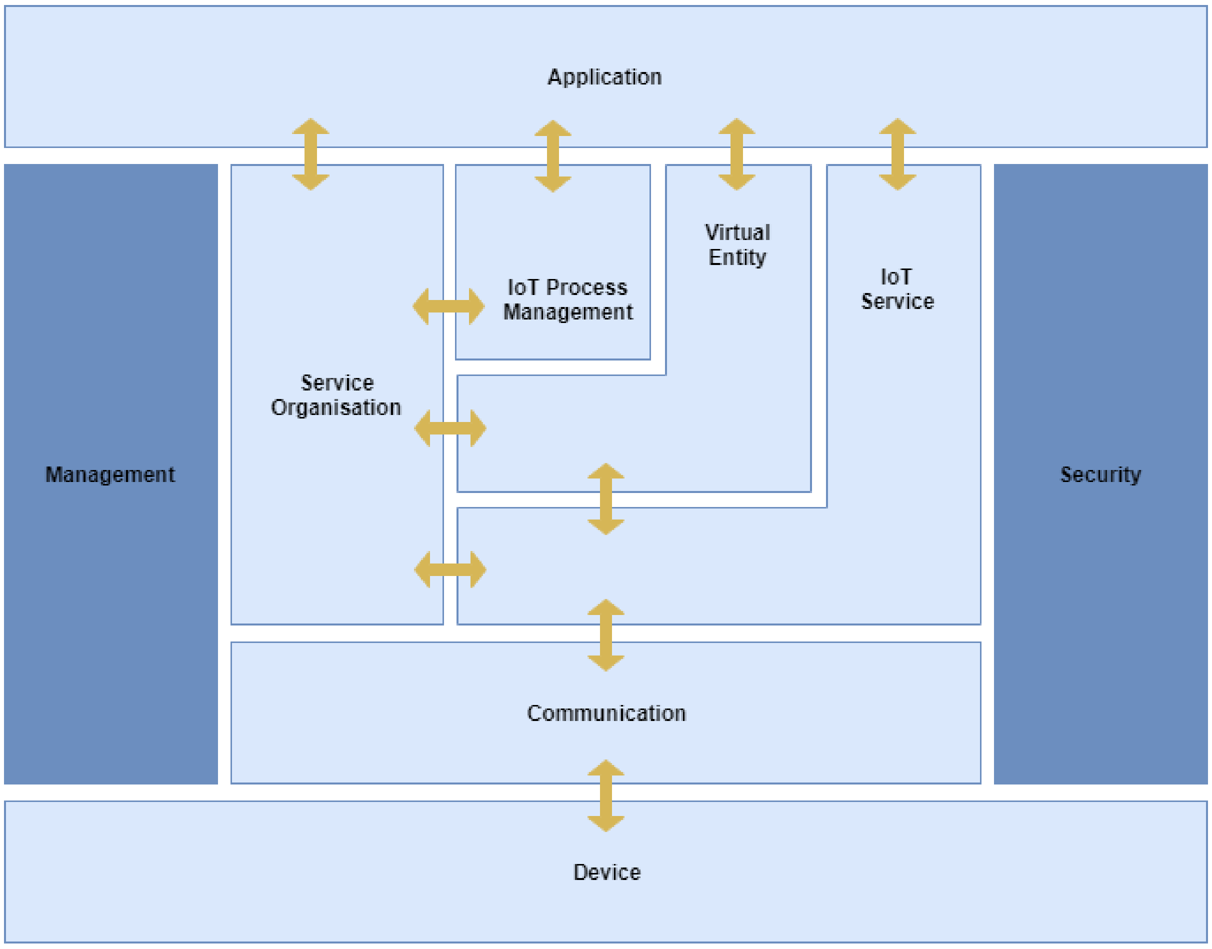


Figure 7: IoT-A's functional model

2.1.4. Conclusion

The reasons for the election of these three architectures as basis for PIXEL RA are:

- **Industrial focus** (*RAMI* and *IIRA*). The applications based on PIXEL will be developed in industrial environments, such as ports, and thus the specific requirements for industry will match better than generic architectures.
- Focus on **interoperability** (*IoT-A*), a challenge which is described as a major objective of the project.
- Follows a **European initiative** (*IoT-A*) that has been implemented in other projects from other domains. This way, the RA will accomplish one of their missions, to make the results more standardized and be less technology dependent.
- **Previous experience** of the project partners with the methodologies and views used.

The following figure depicts a mapping process among RAMI, IIRA and IoT-A architecture:

[9] http://www.iot-a.eu/

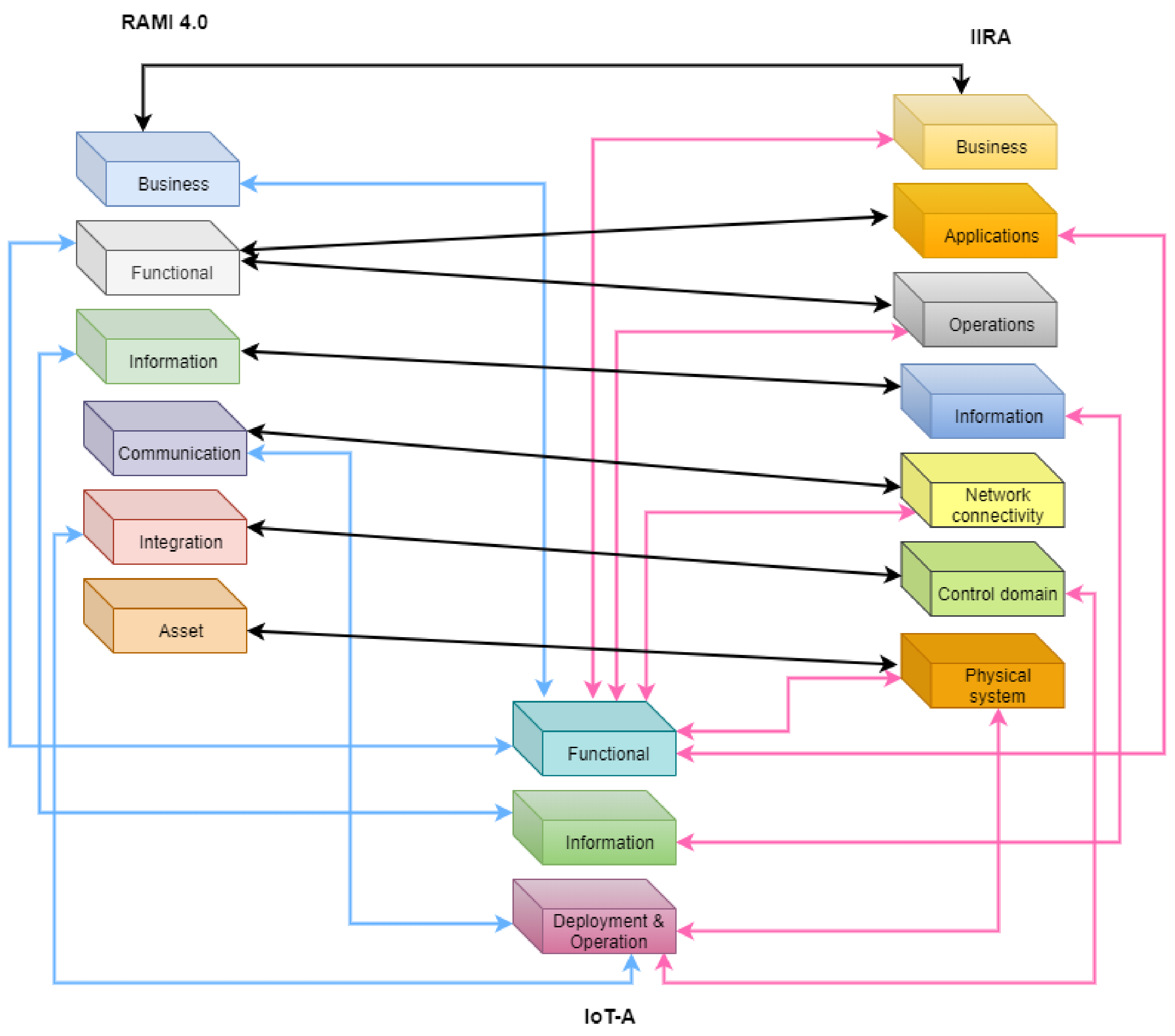


Figure 8: Mapping process among RAMI, IIRA and IoT-A

3. Applied Agile concepts

In the development of end-to-end IoT solutions, one of the biggest challenges is how to integrate IoT data with data streams from enterprise systems and external sources.

Currently, when connecting different systems and applications traditionally it is traditionally highlighted the use of a Service-Oriented-Architecture (**SOA**) and an Enterprise Service Bus (**ESB**). But these two possibilities have as main drawback their complexity and time consuming implementation cycle. For IoT it's extremely important the adaptability to digital business imperative (adaptability to changes) and interoperability. Due to this, an agile approximation / integration is considered in the project.

The term agile integration refers to a continuous integration / continuous delivery (**CI** / **CD**) process used for software releases. For this, it's very important to make use of the modern development technologies like containers and microservices which facilitates approximations as **CI** / **CD**.

The use of agile techniques facilitates the rapid creation of prototypes and iterations. Therefore, their adoption is very suitable for IoT projects (as in PIXEL).

There are three features that will be needed for a correct agile integration:

1. **Containers**.

2. **Distributed integration**.
3. **APIs**.

These three aspects will be commented in detail in section 6, ¡Error! No se encuentra el origen de la referencia..

The idea with these aspects is to obtain a constant interaction cycle of progress that will deliver the results incrementally as services. With this the development of the different use cases will follow the Deming **P**lan – **D**o – **C**heck –**A**ct cycles (**PDCA** cycle, see figure below). There will be a constant interplay between the use cases progress and the technology developments in PIXEL.

Next figure depicts the **PDCA** cycle commented below.

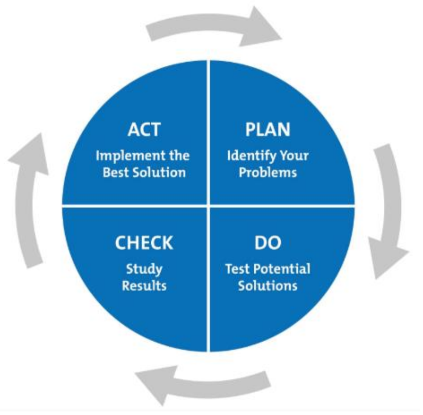


Figure 9: PDCA cycle

4. Functional architecture

4.1. Global architecture

In this section the architecture of PIXEL will be described from a functional point of view. For that, it's important to have a global vision of the architecture.

Following figure, **¡Error! No se encuentra el origen de la referencia.**, shows the global architecture including the interaction with the different data sources and the output to the devices that will work with PIXEL.

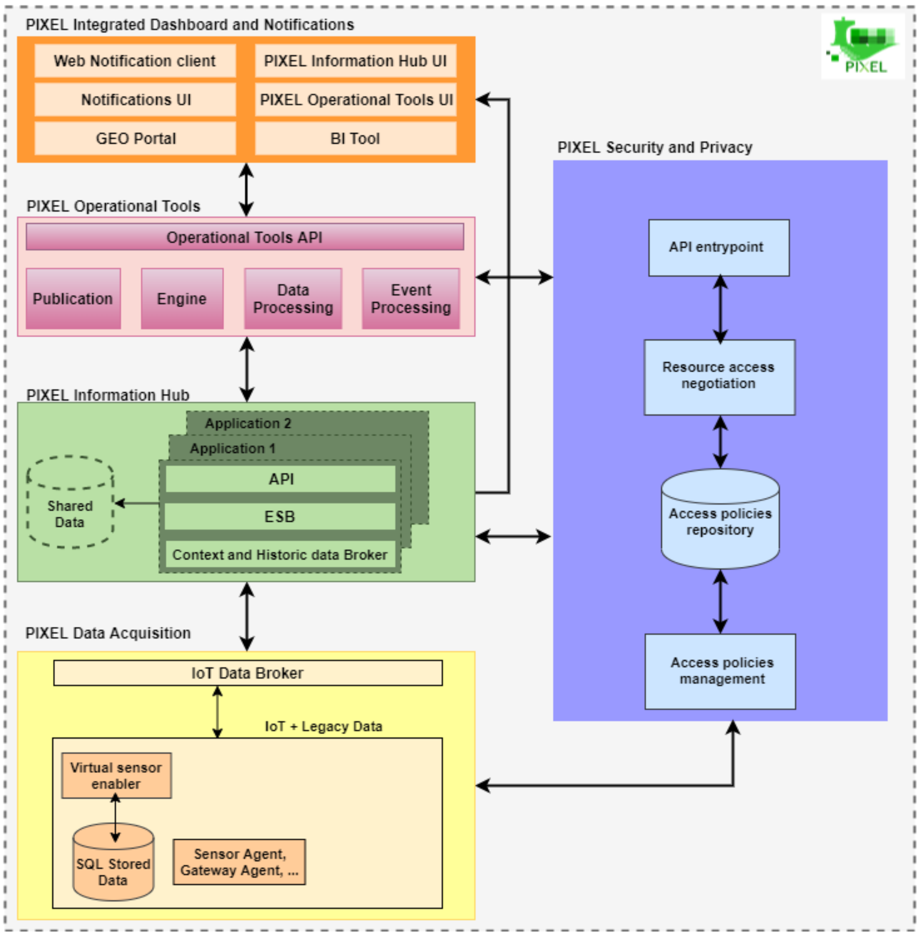


Figure 10: Global architecture

In the following subsections the different modules and their components will be further described.

4.2. Components diagrams

4.2.1. PIXEL Data acquisition

The PIXEL Data Acquisition Layer consists of several components designed to push data from the several data sources available and the PIXEL Information Hub. The solution provides a standard way to acquire data from different data sources that implements different kind of protocols and different data types.

The idea is to provide a standard way to import data into PIXEL Information Hub in order to allow an easy use of any kind of data sources available on each port.

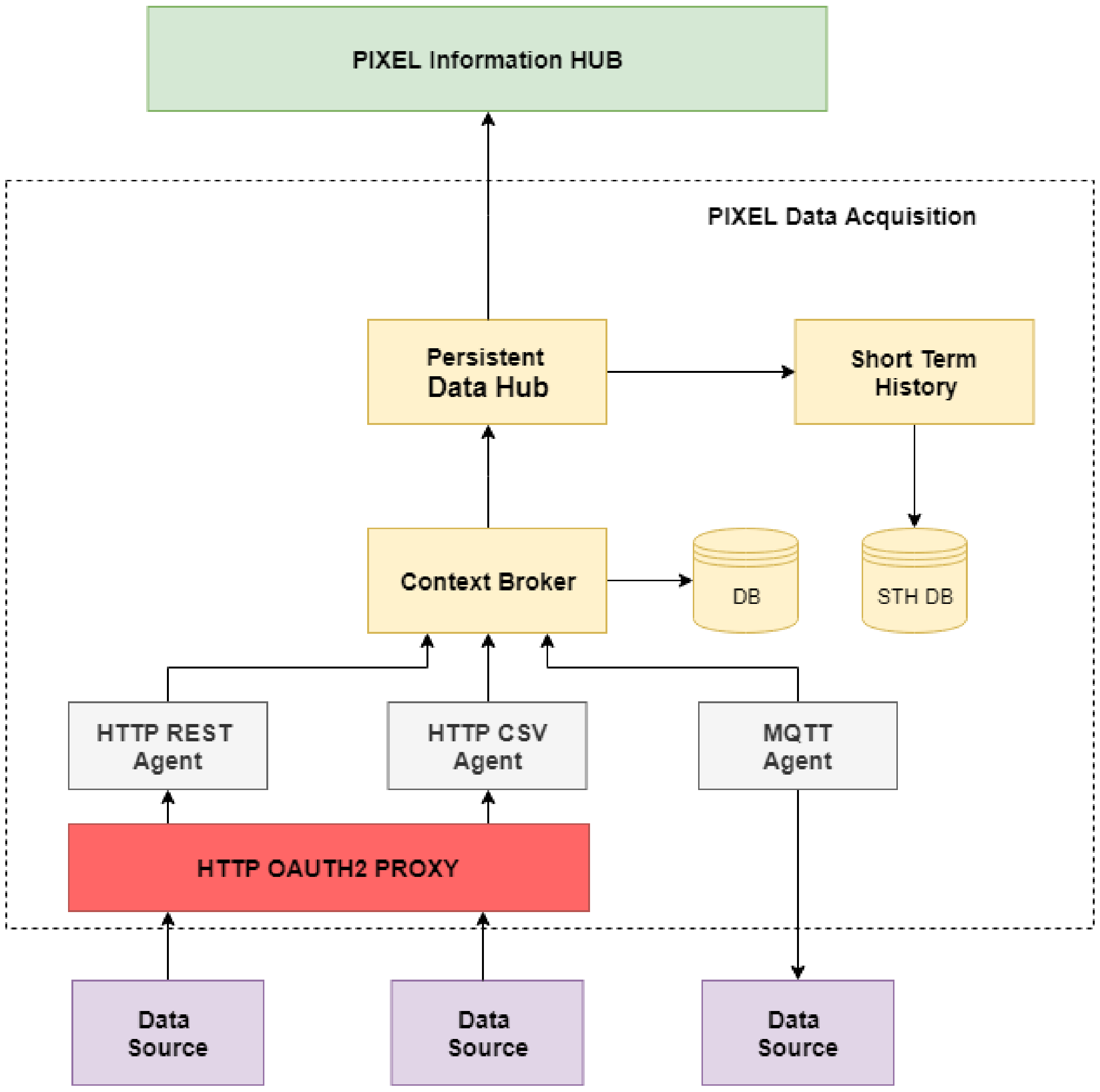


Figure 11: Component architecture of the system

Components involved for the Data Acquisition are:

- **Context Broker.** The context broker is responsible for the implementation of the standard data model used to import data on the PIXEL Information Hub. It collects the data from the data sources through specifics agents and pushes the new data to the Information Hub through the Persistent Data Hub component. It stores the last version of each entity (context data) in its own database (Context DB).
- **Persistent Data Hub.** The Persistent Data Hub is a connector used for persisting context data (managed by the Context Broker) into another third-party database and the PIXEL Information Hub. It could send data simultaneously to several storage systems so it will be able to send new data to the Data Broker of the Information Hub and also to the Short-Term History component.
- **Short Term History.** The Short-Term History component is used to keep track of the last imported data from the data sources but also to trace the activity of each Agent. The imported data should be pruned often, the main purpose here it just to be able to check that the data was well imported in the Information Hub.

- **Agent.** Agents are specific components developed to import data from a data source on the Context Broker. The Agent is in charge of reading the data source format and converting it to the shared data model used on PIXEL. There are two main kinds of Agents:
    - **Agent pulling data from the data source.** If the data source provides an API to retrieve the data, the Agent for this data source implements a client and connects periodically to the source to acquire new data.
    - **Agent listening to data pushed by the data source.** For some data source, it is easier to push the data rather implementing an API. For those cases, the Agent will implement an API to allow the data source to push its data whenever it wants. The security of the access will be performed by the OAuth2 Security Proxy.
- **OAuth2 Security Proxy.** The OAuth2 Proxy is a part of the security component

The Agents are in charge of protocol adaptation, so several standard agents will be provided to address each kind of data source that could be used in PIXEL:

- HTTP REST agent: Expose a REST API (JSON or XML) to allow data sources to push data to the Context Broker.
- HTTP POST File (CSV, JSON or XML) to allow data sources to send data file using an HTTP Post feature.
- MQTT Agent: This agent will allow connecting to MQTT channels in order to retrieve data.

4.2.2. PIXEL Information Hub

The main architectural approach for the PIXEL Information Hub is based on robust experience gained by XLAB during the design of a similar technical solution for the FAIR (Facility for Antiproton and Ion Research) [10] particle accelerator based in Darmstadt, Germany. FAIR is an international accelerator facility for the research with antiprotons and ions, which is being developed and built in cooperation with international partners.

PIXEL Information Hub consists of several parts conceptually divided into components that push data toward the database (downstream); components involved in stored data retrieval and further processing (upstream) and components responsible for data persistence and storage. In addition, the system provides supporting services for configuring, managing and monitoring the PIXEL Information Hub.

Refinements since D6.1: We have identified the requirement for near real-time integration of some models and predictive algorithms in the overall PIXEL data flow. For this reason, an interface has been defined to "plug-in" PIXEL operational tolls to the PIXEL Information Hub core data flow process. With this approach, we achieve scalability and performance for on-the-fly processing of incoming data streams. Furthermore, API access will be the managed centrally through PIXEL security components, so the diagram and description reflects this adaptation.

[10] https://fair-center.eu/

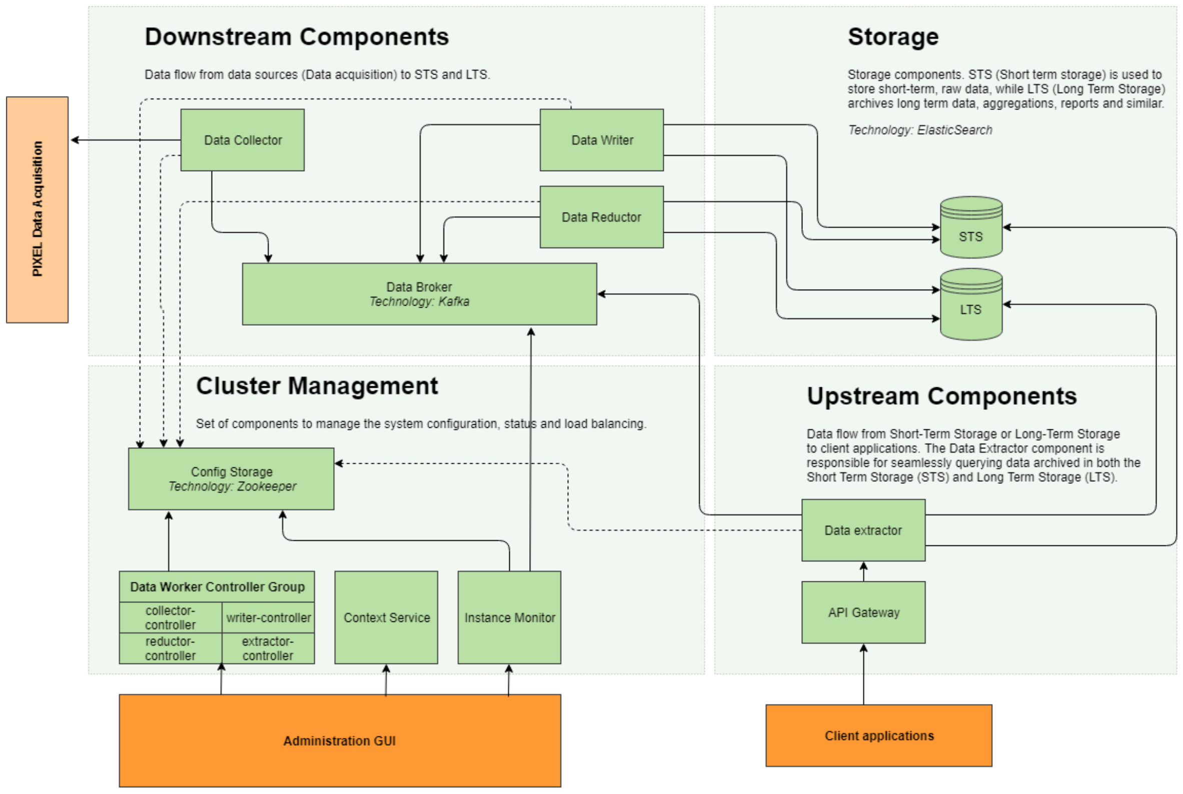

Downstream Components
Data flow from data sources (Data acquisition) to STS and LTS.
Storage
Storage components. STS (Short term storage) is used to store short-term, raw data, while LTS (Long Term Storage) archives long term data, aggregations, reports and similar.
Technology: ElasticSearch
PIXEL Data Acquisition
Data Collector
Data Writer
Data Reductor
Data Broker
Technology: Kafka
STS
LTS
Cluster Management
Set of components to manage the system configuration, status and load balancing.
Config Storage
Technology: Zookeeper
Upstream Components
Data flow from Short-Term Storage or Long-Term Storage to client applications. The Data Extractor component is responsible for seamlessly querying data archived in both the Short Term Storage (STS) and Long Term Storage (LTS).
Data extractor
API Gateway
Data Worker Controller Group
collector-controller
writer-controller
reductor-controller
extractor-controller
Context Service
Instance Monitor
Administration GUI
Client applications

Figure 12: shows the component architecture of the system

Components involved in the downstream flow are:

- **Data Collector**. The Data Collector component is responsible for obtaining data records from various devices and data sources through the PIXEL Data Acquisition layer, analysing and filtering their fields and pushing them downstream to the Data Broker for further processing.
- **Data Writer.** The Data Writer component acts as a bridge between the Data Broker, Short-Term Storage and Long-Term Storage components. It is responsible for pulling the records from the Data Broker, parsing their meta-info headers, initializing long or short term storage for the data source (if needed), and finally archiving them.
- **Data Reductor.** The Data Reductor collects and reduces data accumulated in the Short-Term Storage by applying different reduction algorithms, and permanently stores the reduced device data in the Long-Term Storage. Reduction operations are distributed between Data Reductor instances on a per Source basis, according to rules provided in the Configuration component. Reduction algorithms can be implemented and integrated into the Data Reductor service at build time or loaded at runtime. The status of the Data Reductor node is communicated to the rest of the system through the Data Broker.

Components involved in the upstream flow:

- **Data Extractor.** The Data Extractor component is responsible for seamlessly querying the data archived in both the Short-Term Storage and Long-Term Storage components. A dedicated Data Extractor Client provides a simple interface for accessing Data Extractor capabilities and facilitates client application development.
- **Client application(s).** Client applications use the provided REST API to obtain data maintained by the PIXEL Information Hub through the API Gateway. In the PIXEL context, client applications are primarily managed as part of the PIXEL Operational Tools and the PIXEL Dashboard. Nevertheless, a well-documented and efficient REST API allows development of additional, stand-alone, clients.

Storage and buffering components:

- **Short-Term Storage (STS).** Incoming data from devices and other sources is temporarily stored in the Short-Term Storage component in order to make it accessible from the Data Extractor. This provides the ability of browsing, exporting and correlating the data in full granularity and serves as a temporal persistent buffer and search engine for the Data Reductor component. On the other hand, the metadata are always stored directly in the Long-Term Storage.
- **Long-Term Storage (LTS).** The Long-Term Storage component is used to store reduced or raw data, provided by the Data Reductor and Data Writer components. The reduction enables the system to decrease storage requirements, thus lowering cost by avoiding storage of historically non-relevant data.
- **Configuration Service.** Configuration Service is the central configuration repository of the PIXEL Information Hub. Zookeeper is used in order to assure optimal availability, performance and configuration management. It is essentially a distributed hierarchical key-value store, which is used to provide a distributed configuration service, synchronization service, and naming registry for large distributed systems.
- **Data Broker.** The Data Broker component aggregates all data received from the Data Collector component, originating from the Data Acquisition platform. It provides a common interface for other components to retrieve and filter live data based on device/source name and property. In addition, it acts as an optimal consumer for the collector component underneath, reducing data flow congestion by buffering, effectively easing out any potential load peaks.

Supporting components:

- **Context Service.** The Context Service is the main back-end component responsible for providing information about the current context, managing global settings and Sources as well as applying complex settings that involve other components.
- **Instance Monitor.** Is a service which, using a shared library, collects metrics from all service instances. It enables clients to track metrics such as CPU and RAM consumption, data throughput and more. It is also responsible for triggering notifications in case of failures.
- **Data Collector Controller.** Main purpose of Data Collector Controller is to maintain the configuration of devices and deployed Data Collector nodes. It is connected to Data Collector instances by watching and updating common nodes of the Configuration Service.
- **Data Writer Controller.** Similar to Data Collector Controller, the purpose of Data Writer Controller is to maintain configuration of deployed Data Writer nodes. It is connected to Data Writer instances by watching and updating common nodes of the Configuration Service.
- **Data Reductor Controller.** The Data Reductor Controller is responsible for providing and configuring Data Reductor instances. It also provides monitoring for processing units; it allows configuration of load distribution and manages user provided or integrated reduction algorithms.
- **Data Extractor Controller.** The Data Extractor Controller is responsible for providing and configuring Data Extractor instances. It is connected to Data Extractor instances by watching and updating common nodes of the Configuration Service.
- **Administration GUI.** The Administration GUI is a web-based application exposing a user interface enabling admins to configure, monitor and control the PIXEL Information Hub. Admins can add or remove data sources, change instance node configurations and be notified about errors in the systems triggered by the Instance Monitor. They can also track load on different instance nodes, turn services on or off and manage reduction algorithms.

Components involved in the downstream flow together with the Data Extractor also belong to the Data Worker Group and their controllers to the Data Worker Controller Group. This notion is particularly useful when arranging and scaling components. In addition, Data Worker Group components together with storage are heavily involved in data processing and provide functionality of data acquisition, processing and retrieval.

4.2.3. PIXEL Operational Tools

The Operational Tools (OT) are mainly in charge of bringing closer to the user both the models and predictive algorithms developed in WP4. By user here we mean administrators and managers analysing port operations by means of simulation models and predictive algorithms. In order to reach that goal, a set of high-level operational tools are defined, these are:

- Publish models and/or predictive algorithms.
- Edit and configure the models and/or predictive algorithms
- Execute models and/or predictive algorithms.
- Schedule models and/or predictive algorithms to be executed at a specific time once or periodically.
- Define different operational (e.g. bottleneck detection) and environmental (e.g. PEI) KPIs, based on the data available in the information hub (from data sources, executions of models, etc.) for tracking and monitoring purposes.
- Establish some pattern detection mechanism. The most basic one is the use of triggers for a model and/or predictive algorithm. For example, when some event happens (e.g. some input changes), it may cause the relaunch of a scheduled model.
- Get the trends of a model and/or predictive algorithm. This implies:
  - Visualization of the latest historical values.
  - Possibility of making some basic regression process to get near future data, supposing the model can be characterized according to a certain distribution (e.g. linear, exponential, etc.).
- Detect anomalies and raise alarms. Specific rules can be established in order to trigger alarms and actions whenever some threshold is reached or some condition is reached.

  Note that the PEI is a particularisation of a model as well as a KPI; in fact the PEI is also composed of several KPIs (eKPIs and indexes, see deliverable D5.2).

The functional overview of the Operational Tools is depicted in Figure 13. Several internal components can be identified:

- OT UI: this is the graphical interface to access (most of) the underlying functionalities. This component provides independence and autonomy, but it can be later integrated as part of the PIXEL dashboard to provide a single-entry point for administrators.
- OT API: backend API implementing the functionalities needed. This component is aligned with PIXEL security framework in order to fulfil all required security policies (e.g. authentication, authorization, etc.).
- Publication component: it allows publishing both models and predictive algorithms. By publishing it may be necessary to deploy the models as services. Besides, the models ‘and predictive algorithms’ configurations can also be edited.
- Engine: this component is responsible for executing the different models and predictive algorithms. The execution can be invoked in real time or scheduled.
- Data processing: it is responsible for managing trends from models and/or predictive algorithms and also for some internal data adaptations required.
- Event processing: this component is responsible for real-time monitoring of indicators and conditions and trigger specific actions depending on previously configured rules. It includes a bridge to integrate with an external notification system.

Database: the database includes description of the models and predictive algorithms that can be used, KPI description, rules as well as other configuration and output related parameters necessary for the correct behaviour of the internal building blocks. Some execution results might be stored internally besides being provided to the information hub.

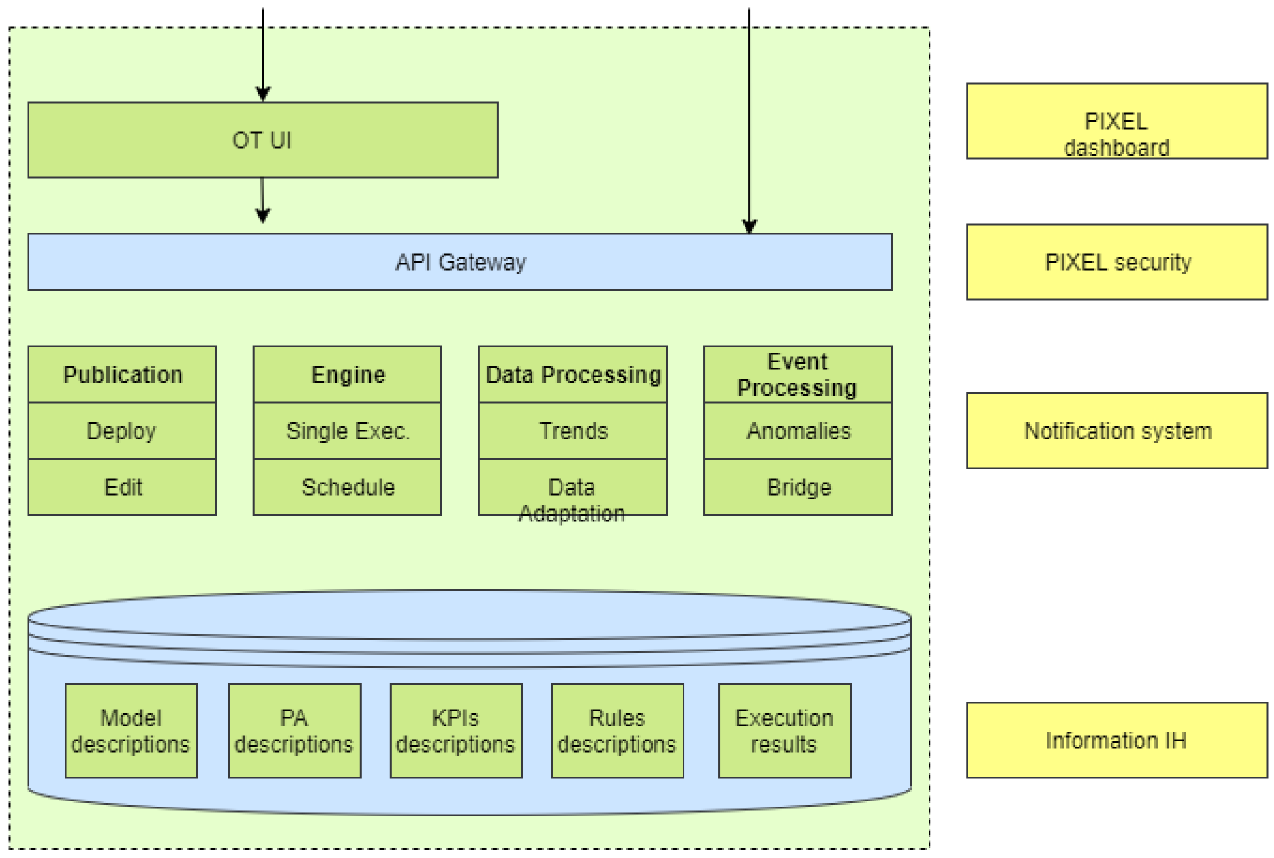


Figure 13: Functional overview of the Operational Tools aligned with other PIXEL modules

The engine can execute models and predictive algorithms according to configuration parameters, which are provided directly at runtime through the model/algorithm’s API, or obtained from the database for periodic executions.

The flow is depicted in Figure 14. An external component can invoke a model in three different ways:

- In real time (now), passing the necessary input parameters. This could be the case of what-if scenario.
- Scheduled for a certain point in the future (once). In this case, the input parameters should be stored in a database to a later feed the models. They can be static.
- Scheduled in a periodical way (periodic). Here the inputs might be stored as well in the database, but they can change (e.g. weather information changes every day for a daily schedule).

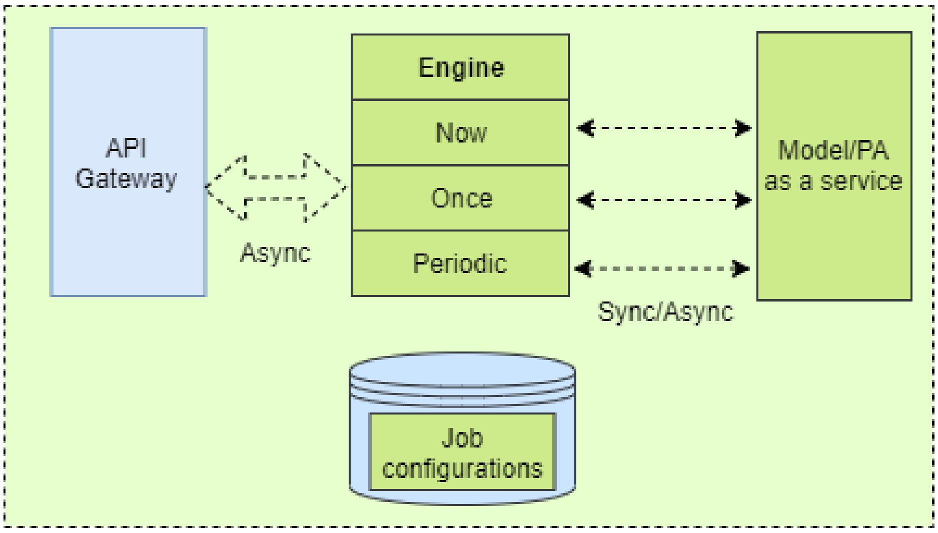


Figure 14: Real time invocation of models through OT API

From the point of view of the external invocation an asynchronous interface is provided, handled mainly by the Engine internal component. The interface with the specific model and/or predictive algorithm can be either synchronous or asynchronous; it depends on the final implementation of it and this approach covers both possibilities.

The database stores all necessary data that allows the configuration and execution of the different internal components building the OT module. Therefore, it will be necessary to create a data model aligned with the PIXEL data model. The OT API allows access to the database from external components, whereas internal components may directly invoke the database.

Some entities are identified for the data model:

1. *Model descriptions*. This mainly refers to a proper meta-data for the models that allows not only being described (name, description, data sources, etc.), but also being executed (service, endpoint, input parameters, output data format).
2. *Predictive algorithms description*. Similar as for models.
3. *KPIs description*. It allows categorization (operational, environmental) of KPIs, modes of getting this value from available data (typically IH), and establishing thresholds in order to later being able to monitor if they are exceeded.
4. *Rules*. Set of conditions to be monitored. This can be linked with the thresholds of the KPIs, or any other threshold established by the port operator. This is a logical repository, which might be shifted to an external notification system.
5. *Execution instances with related parameters* (configuration, status, result).
6. *Event history*. A table of historic events in order to keep track of them throughout time.

4.2.4. PIXEL Integrated Dashboard and Notifications

The PIXEL Integrated Dashboard and Notifications is designed to show the data available from the PIXEL Operational Tools and also from the IH. These data are the result of:

- **Retrieving data from IoT** and other information sensors
- Apply **predictive algorithms and models**.
- Calculate the **PEI**.
- **Notifications from Event Processing**. Whenever a rule is met, a notification will be received from the Operational Tools module.

These data are represented to the platform user via three main channels:

- **Charts & Dashboard**. Visualization of the data received from the different data sources (devices, sensors), results of simulations, predictions, etc.
- **GIS**. Geolocated data (sensors, devices, services, data result) is represented in a map view, which offers the geospatial interpretation of the data and allows interpolation of information, detecting potential sources of anomalies, etc.
- **Notifications**: Coming from executions of predefined rules/conditions, this channel allows the transmission of high-importance or actionable information to the appropriate addressee(s), doing it in the specific moment when the information is useful for informed decision-making, establishing prevention mechanisms or doing meaningful reports.

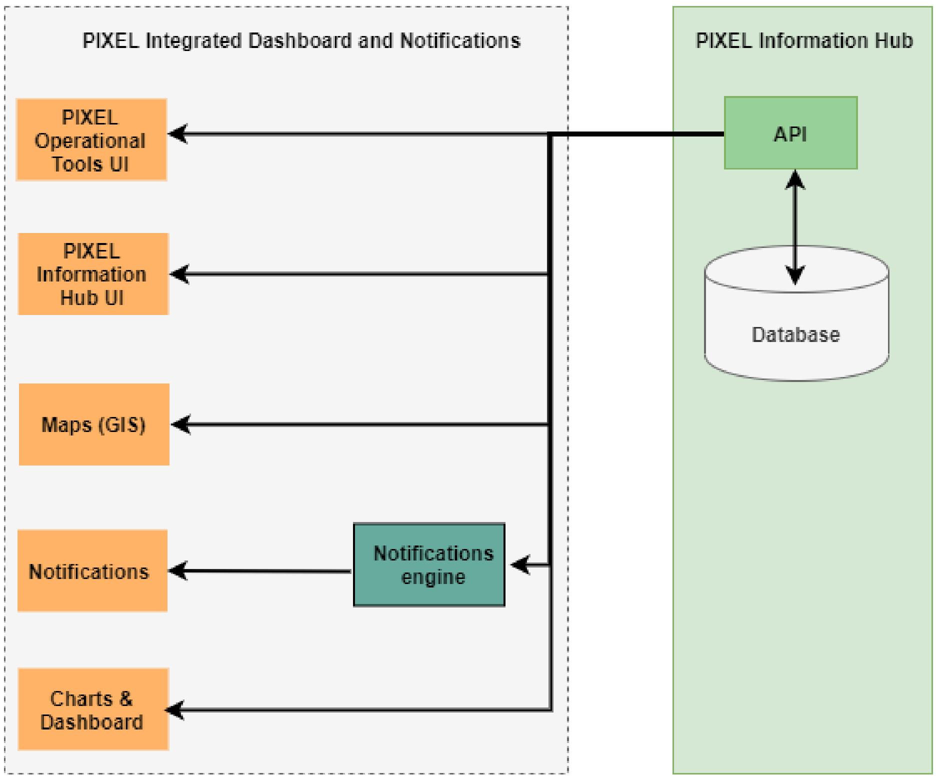


Figure 15: Shows the component architecture of the system

There are two kinds of elements involved in the PIXEL Integrated Dashboard and Notifications:

1. **Inputs**. Data entries that will be shown on dashboards and other types of visualizations.
   a. **PIXEL Information Hub.** The data comes from PIXEL Information Hub thanks to its API. These data come from the following inputs.
      i) **Database**. The database stores all that comes from PIXEL Data Acquisition.
   b. **Notifications engine**. It is the engine where will be created the rules that in case of being fulfilled will end up triggering the alerts / notifications that will be seen in the dashboard. In the cases of PIXEL, alerts are typically generated by the analysis of data inside the information hub.
2. **Outputs**. Visualization of data offered to the end user.
   a. **Charts & Dashboards**. Component responsible for the visualization of data in real-time via graph/chart representation a cluster of several charts that show information of a specific problem or an aspect of the port (for instance vehicle activity, lighting status) in a single screen, as long all that charts are related and show a specific view of the general problem.
   b. **Notifications**. Will be the result of a met rule (for instance, a value is over a threshold, a sequence is met, the average of a value is below a threshold, the number of values over a threshold in a certain period, etc.). The result of a notification can be an email, a ticket in JIRA or a conversation

in Slack for example (**Command (custom)**, **Email**, **JIRA**[11], **SNS**[12], **HipChat**[13], **Slack**[14], **Telegram**, **Google Chat and STOMP**[15]).

c. **GIS**. Dashboard will include a map section where visualize the port area where the data comes from and the information of the different devices / sensors that are in the area.

d. **PIXEL Operational Tools UI**. UI for planning, configuring, managing and visualize the results of executing the PIXEL Operational Tools (PEI, predictive algorithms).

e. **PIXEL Information Hub UI**. UI for see the data stored in the PIXEL Information Hub.

4.2.5. PIXEL Security

The PIXEL Security solution is in charge of providing a solution to identify and authenticate users that could be connected to existing identity management solutions already deployed in ports, and also of providing a solution to control the access of the data managed by the PIXEL Solution.

The solution provides an API Gateway based on OAuth2 mechanism in order to protect the access of the different API exposed by PIXEL. The Gateway also implements XACML rules in order to define access rules based on API URL and VERBS and the user roles.

The security solution provides also an Identity Management solution that can be used by other PIXEL components to share the same user identity across the all platform.

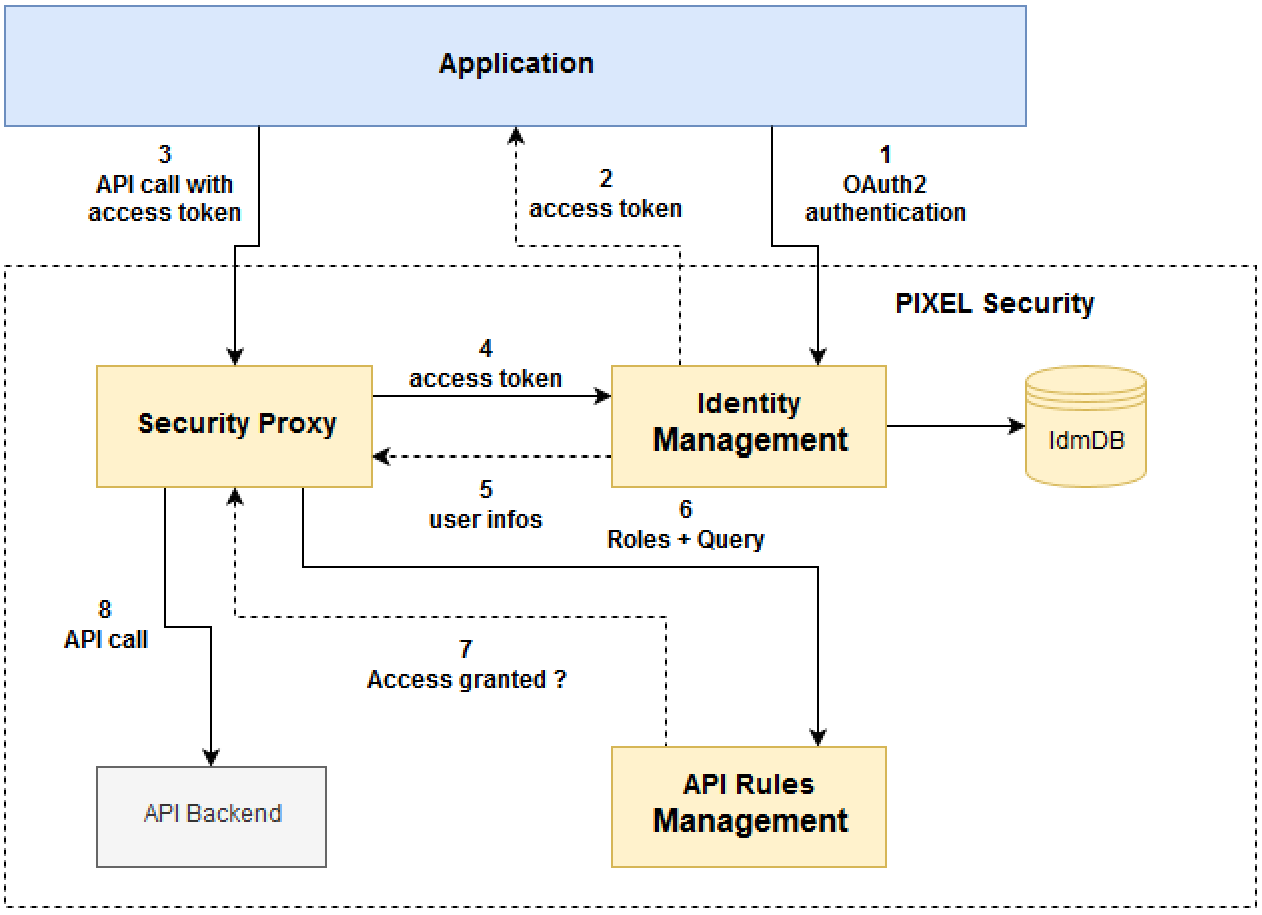


Figure 16: PIXEL Security mechanism

[11] https://es.atlassian.com/software/jira
[12] https://aws.amazon.com/sns
[13] https://www.atlassian.com/software/hipchat/downloads
[14] https://slack.com
[15] https://stomp.github.io

The security will rely on 3 mains components:

- **Identity management**: This component manages the user database and connects to existing Identity Management solutions. It also implements the OAuth2 standard protocol (signing, signup, authenticate…).
- **API Rules Control Management**: This server is in charge of managing API Control access rules based on XACML in order to control if a user is allowed to access specifics part of the API exposed.
- **Security Proxy**: This proxy check that the user is allowed to make the API call he/she requested. It checks the OAuth2 token with the Identity Management Server and the API rules against the XACML server.

The security in PIXEL is based on the XACML reference architecture. This architecture allows setting fine-grained authorization rules at API level, so that all resources in the platform are secured and their access is restricted to users in a per-resource basis.

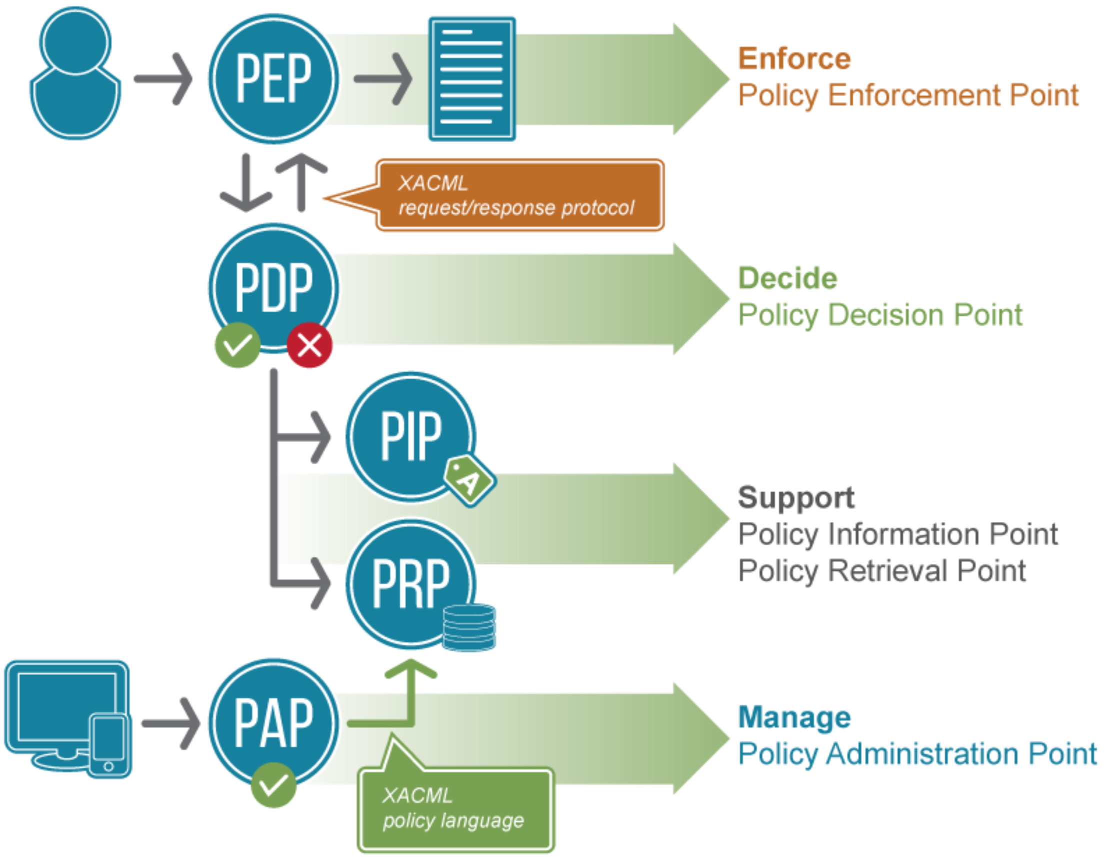


Figure 17: XACML Reference Architecture. Source: Axiomatics.com

As mentioned, the use of XACML-based architecture (FIWARE security stack is based on it) allows filtering all resources and features requests on the platform (in the PEP) and set up intelligent rules (via PDP and PAP/PIP/PRP) that can discriminate users, context data, time frames, etc. XACML enables, therefore, a flexible and complete control over the access to the platform data and, at the same time, stablish a well-defined manageable permission hierarchy. In PIXEL, the permission on resources and data affects not only to what a user can see on the platform view (menus, resources, actions) but also to the API action that can perform. Consequently, when authorization rules are defined, these protect the management and the programming side of the platform, being consistent and robust again attacks such as reverse engineering.

4.3. Selected technological options

The aim of this point is to have located in a section the technological option chosen for each module.

- Data Acquisition Layer will use the **FIWARE** stack.
- IH will use a stack based on the open source projects **Elastic**, **Kafka** and **Zookeeper**.
- For OT the development will be **ad hoc**. As the Event Processing Engine **ElastAlert** will be used; for the execution and scheduling of models and predictive algorithms, supporting asynchronous mode and adapted version of the **Agenda** project will be used (*https://github.com/agenda/agenda*).
- Dashboard and Notifications will use **Elasticsearch** for the storage. **Kibana** will be the Dashboard tool and **ElastAlert** the notifications tool. The UI will be built with **vue.js/vuetify** and **leaflet.js** (GIS)
- For Security layer the option chosen is based on **XACML** and **FIWARE**.

Next picture depicts the architecture indicating the technologies used in each layer.

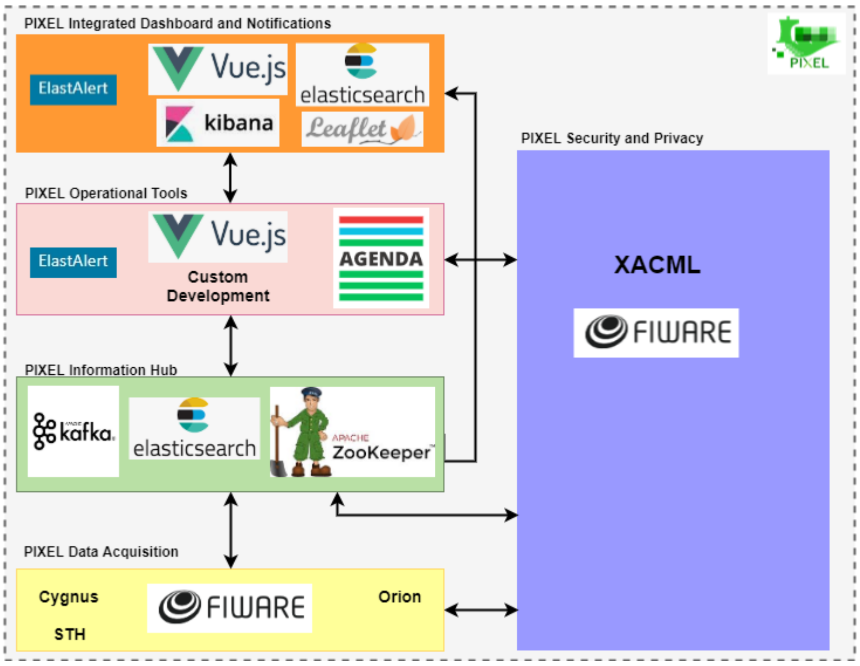


Figure 18: PIXEL's architecture with technology by module

4.4. Integration aspects

4.4.1. Linking Data Acquisition Layer with the Information Hub

The core component of DAL is the Orion Context Broker which is responsible for the implementation of the standard data model used to collect data and store it into the PIXEL Information Hub. This Generic enabler is not only a way to store the data provided by the Data Source with a generic API to query them (NGSI API). Through the Orion Context Broker context elements representing entities from the PIXEL Information Model (see PIXEL Initial Data Model) are created and managed. In addition, DAL provides a functionality to retrieve the list of all entity types and entities. Furthermore, an external module (PIXEL IH in this case) can subscribe to context information, so when some condition occurs (e.g. the context elements have changed) a notification is received through HTTP call-back.

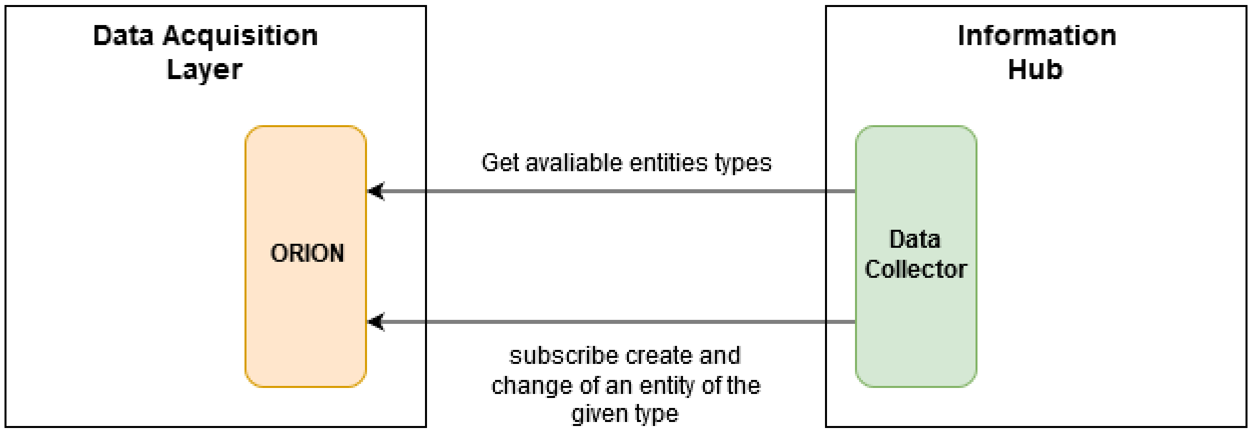

Data Acquisition Layer
ORION
Information Hub
Data Collector
Get avaliable entities types
subscribe create and change of an entity of the given type

Figure 19: Integration between DAL and IH

These usage scenarios allow the implementation of a discovery service that provides all data sources and exposes them to the PIXEL IH administrator through the management UI. Those sources can then be switched on, which means the subscription gets activated and all data changes are gathered by the IH and stored to a new Elasticsearch index as timeseries representing changes of all entities from that source, where each PIXEL IH source represents an Orion entity type.

Next flow diagram depicts the synchronization of data sources between DAL and Information Hub as well as flow of data from external data sources to the data storage within Information Hub.

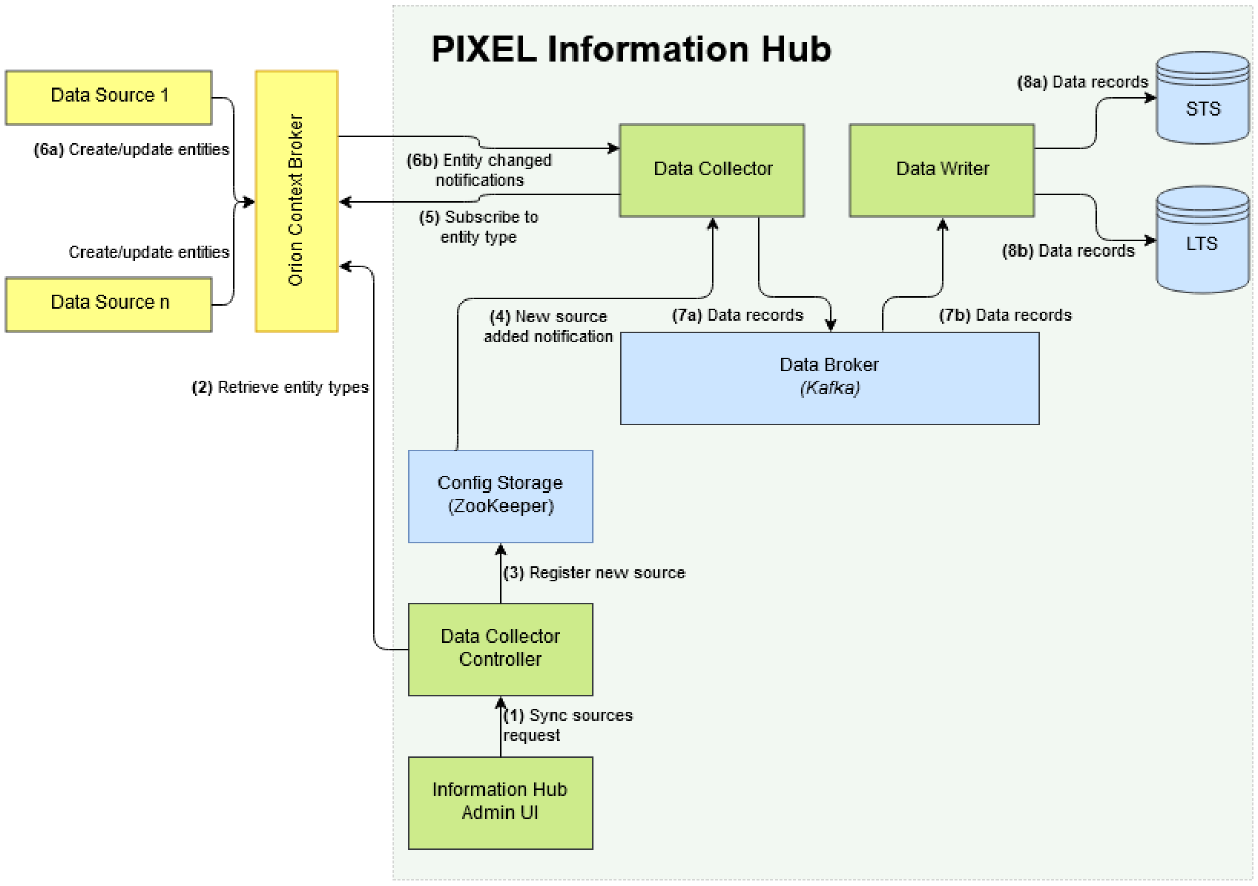


Figure 20: Synchronization of data sources between DAL and IH

1. After connecting new data source to the Orion Context Provider, the system administrator opens the Information Hub dashboard and preforms the 'Sync sources' action to synchronise the list of sources registered within Information Hub with sources provided by DAL. The Information Hub dashboard sends 'Sync sources' request to the Data Collector Controller REST API.

2. Data Collector Controller retrieves a list of all entity types that exist in the DAL by sending 'GET /types' request to Orion REST API. Each entity type corresponds to one data source within Information Hub.
3. Data Collector Controller registers new data source in the Information Hub Config Storage (ZooKeeper). Data source identifier equals to the entity type.
4. Data Collector receives a notification from the Config Storage that new data source has been added.
5. Data collector subscribes to the given Orion entity type by sending a subscribe request to the Orion Context Broker.
6. When the given entity type is updated at the Orion Context Broker by an external data source, Orion sends a notification message to the Data Collector. The notification message contains all attributes of one or more updated entities. For each updated entity the Data Collector generates a corresponding Archive Record object.
7. Data collector sends Archive Record objects through the Data Broker (Kafka) to the Data Writer component.
8. Data Writer component stores records to the short-term (STS) or long-term (LTS) storage.

4.4.2. Linking Information Hub and Operational Tools

The Operational Tools have several execution modes, which affect the way of interfacing with the PIXEL IH. OTs may execute on demand, at specified time intervals or in a near-real time mode, when the execution is triggered on predefined changes of data source attributes. In addition to execution modes, OTs need access information (metadata) about data available in the PIXEL IH.

On-demand and on-timer execution

The main integration component for on demand/on timer execution of OT models and predictive algorithms is through the PIXEL IH Data Extractor component and the native Elasticsearch interface.

The Data Extractor supports REST API calls for querying meta-data and querying actual measurements stored in the PIXEL IH. There is no need for the client to know whether data comes from a meta-repository, STS or LTS, as the Data Extractor engine is responsible for distributing calls to relevant subsystems and merging the results. In addition to a registry (discovery) functionality that provides a list of available data sources, clients can request a list of time periods for which data is available for a given query. In this case a list of time periods (time-stamp from/to) is returned with the meta-data attached to each measurement. Based on available time periods OTs can then request data for a specific time series.

In addition to the functionality provided by the Data Extractor, OTs can use the Elasticsearch interface to execute queries using the native Elasticsearch interface.

There are two ways how OTs can write modelling or Predictive Algorithms results back to the PIXEL Information Hub. Either a custom-developed PIXEL IH Data Collector can be developed, or the standard Elasticsearch REST API can be used. Based on performance and functional requirements of OTs models and PAs to be integrated, the most suitable approach will be used, as the open architecture defined in this document supports both.

Integration in the data flow

A more advanced integration approach is needed when a model or PA has been invoked (through scheduling) and there is a specific change in a data source. For example, a predictive algorithm to estimate the time of departure of a vessel may be executed each time there is a new ship entering the port. For this reason, an interface has been defined to integrate models/PAs directly into the PIXEL IH flow by exposing system setup and message broker interfaces through a special “OT connector” library that will be integrated with those models that need this functionality. This library will be used during the integration of models as described in the following chapter (Integration models in the Operational Tools).

4.4.3. Models integration through Operational Tools

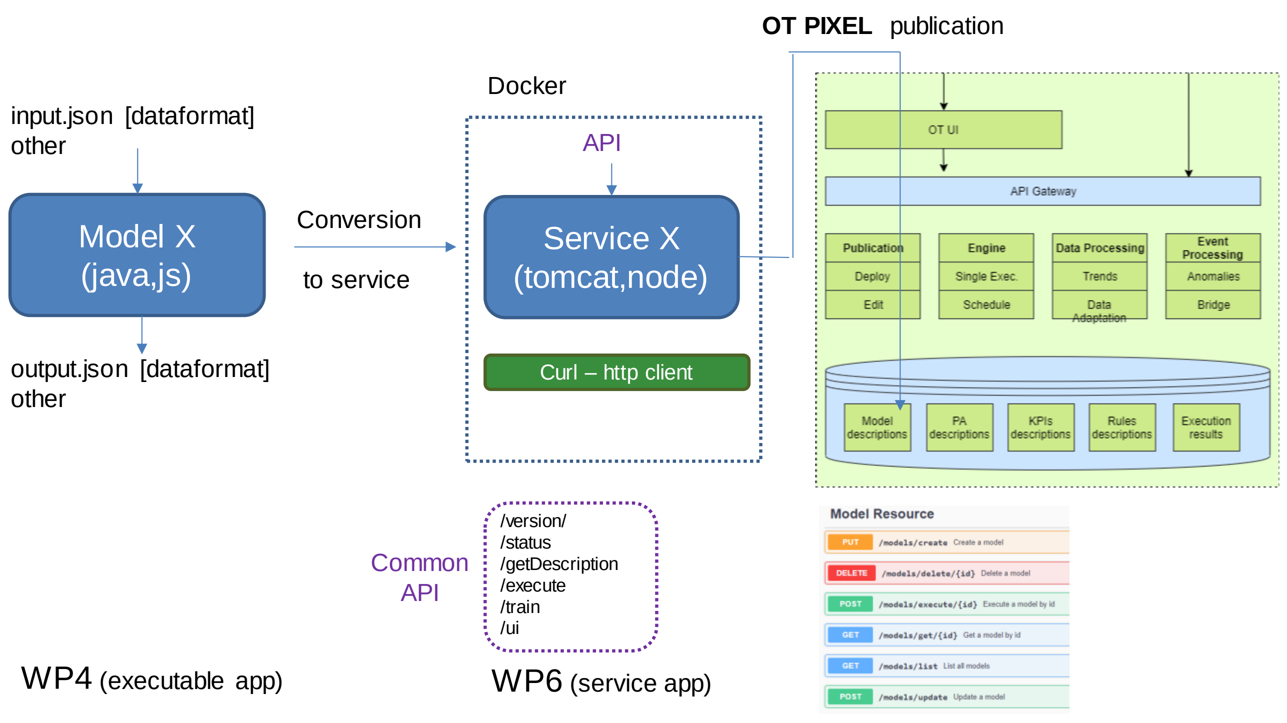
The Operational Tools have as mission the adaptation, execution and orchestration of the models developed in WP4 to let the rest of platform components and PIXEL users to manage, control and obtain results out of the models. The same applies for the predictive algorithms and also for the PEI that is conceived also as a model. This implies the exposition of two interfaces: one interface to wrap the heterogeneous models into a common format (standardization) and a second to expose the module features for the rest of components and users. However, the whole process requires to be decomposed into several steps to guarantee modularity and flexibility and this chapter is devoted to describe it from an architecture perspective.

The process is depicted in figure Integration models and OT. It has been already introduced in a previous deliverable (D6.3) but here more details will be provided to better understand the interaction with the architecture.

At a WP4 level, the initial phase of the model is provided as a typical executable file. This initial model (**model X** in figure below) can be developed in different programming languages (Java, JavaScript). Each model is supposed to interpret one JSON file as input and provide one JSON file as output, with a specific data format. Working with JSON formats is widely accepted good practise to work with open interfaces due to its conciseness and readability. However, some proprietary models may require a special treatment where the conversion from formats may be inefficient; therefore, it is possible to support other formats and encapsulate the proprietary format (part or all the input) also into a JSON format as raw data. Therefore, the system is provided with flexibility to manage different input formats.

At a WP6 level, the initial model is first converted into a service (see **service X** in figure below) that encapsulates the functionality and allows to be invoked with a REST API. For maximum flexibility, the service is further encapsulated into a Docker instance, able to be deployed anywhere (not only in the PIXEL platform). A common API has been specified for such service to support a wide range of functionalities: versioning, monitoring, execution, training (mainly for predictive algorithms) and user interface. A detailed specification is outside the scope of this deliverable and will be described in deliverable D6.5. The use of Docker for containerisation also allows including certain libraries at OS level that can facilitate extended functionalities to the service. As example, the CURL library is able to provide HTTP access to external services (therefore, the service could potentially query any external service to get some intermediate processing).

The containerised version of the service enters then in the PIXEL platform through the OT, at publication phase. A new entry in the PIXEL data model is added, representing the model (as a service). The entity includes different attributes (name, inputs, outputs, endpoints, etc.) that can be later queried. The publication API in the Operational Tools allows publishing the service as a Docker image (still to be deployed within PIXEL) or as an external service (the Docker instance is already deployed). The latter case will not be the case for the PIXEL deployed models, but allows that third party services (models) could in the future be published in the PIXEL platform.



Model Resource

PUT	`/models/create`	Create a model
DELETE	`/models/delete/{id}`	Delete a model
POST	`/models/execute/{id}`	Execute a model by id
GET	`/models/get/{id}`	Get a model by id
GET	`/models/list`	List all models
POST	`/models/update`	Update a model

Figure 21: Integration models and OT

The publication API in the Operational Tools allows publishing the service as a Docker image (to be later deployed within PIXEL) or as an external service (the Docker instance is already deployed). The latter case will not be the case for the PIXEL deployed models, but allows that third party services (models) could in the future be published in the PIXEL platform. The process is depicted in the Figure below. The first model (model 1) is converted into a service and deployed into an external service repository; through the Model-as-a-Service publication (MaaS). In this case, the Operational Tools do not need to deploy the service internally in the PIXEL platform, but just to store the details in the PIXEL data model. The second model (model 2) is also converted into a service and dockerized, placed into a public Docker repository; through the Model-as-a-Docker publication (MaaD), the Operational Tools deploys the service within the PIXEL platform, and inserts the service endpoint as part of the details in the PIXEL data model.

Invocation to the deployed services will not be performed directly from end users, but through the Operational Tools, that act as proxy and provide an additional management layer able to run and schedule these services, even providing asynchrony. This functionality is mainly performed by the Engine subcomponent of the Operational Tools.

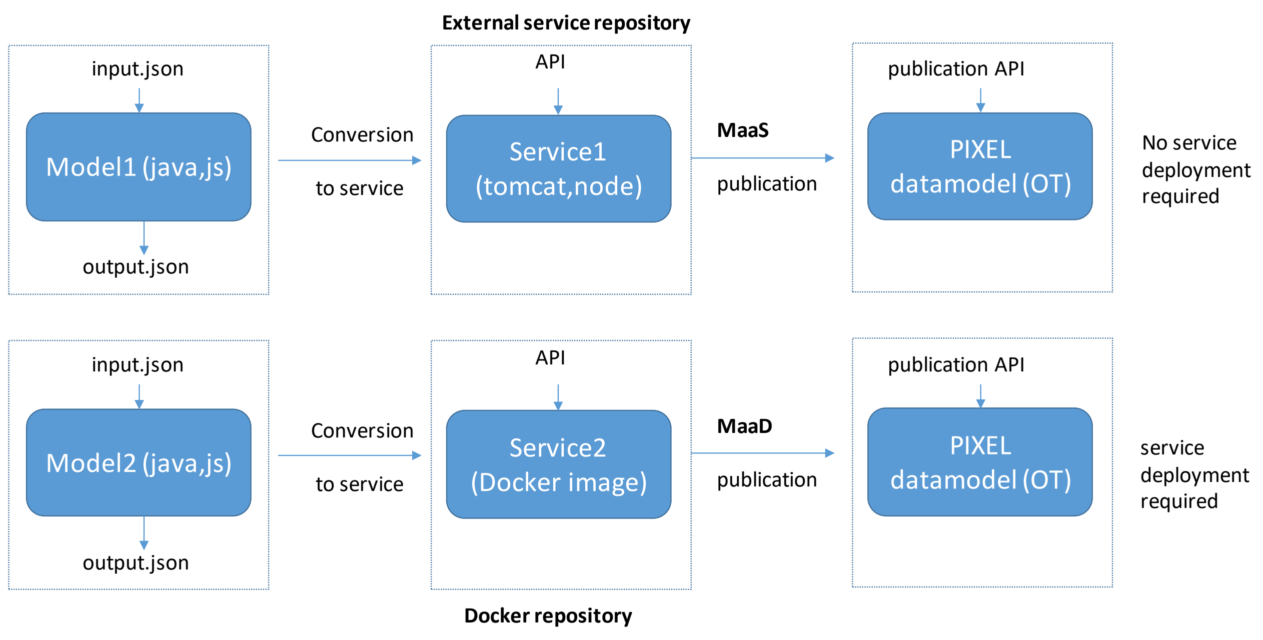


Figure 22: Engine subcomponent

4.4.4. Dashboard and Notifications

4.4.4.1. Integration of UI with Alerts

ElastAlert is the component responsible for the creation of alerts in the module of Dashboard and Notification. It is a framework to alert about anomalies, spikes or other patterns of interest in Elasticsearch data. It is a tool that complements Kibana to alert about data inconsistencies.

It works by combining Elasticsearch with two types of components: rule types and alerts.

Elasticsearch is consulted periodically and the data is passed to the rule type, which determines when a match is found. When one occurs, one or more alerts are given, which take action based on the match.

It is configured through a set of rules, each of which defines a query, a type of rule and a set of alerts.

Several types of rules are included with common monitoring paradigms with ElastAlert:

- Match where there are X events in time Y (frequency type).
- Match when the rate of events increases or decreases (spike type).
- Match when there are less than X events in time Y (flat line type)
- Match when a given field matches a blacklist and whitelist type.
- Match any event that matches a given filter (any type).
- Match when a field has two different values within some time (change type).

ElastAlert is a plugin installed over Kibana. The creation of alerts with ElastAlert forced the user to know its syntax when he wanted to create alerts / rules. With the aim to avoid this **Praeco** will be used as UI.

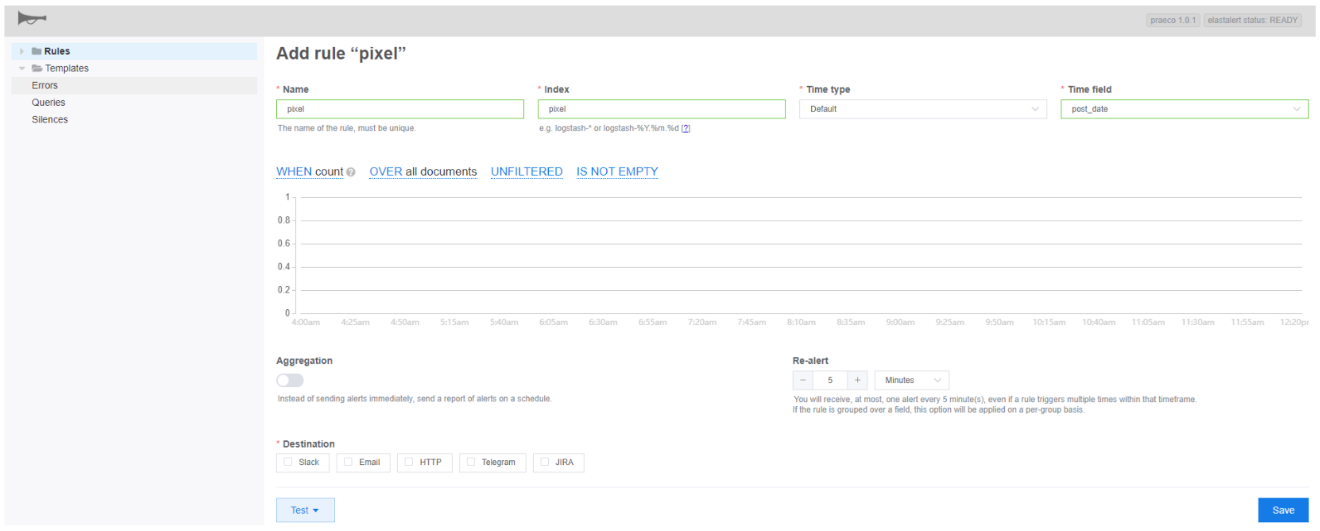
Praeco is an alerting tool for Elasticsearch. It is a UI for ElastAlert. Praeco uses the API of ElastAlert.

Among the features of Praeco are:

- Supports notifications to Slack, Email of HTTP POST.
- Use templates to pre-fill commonly used rule options.
- Supports Any, Blacklist, Whitelist, Change, Frequency and Spike ElastAlert rule types.
- Interactively build alert rules using a query builder.

- Test your alerts against historical data.

Next picture depicts the aspect of PRAECO's UI.



praeco 1.0.1 | elastalert status: READY

- Rules
- Templates
  - Errors
  - Queries
  - Silences

Add rule "pixel"

* Name: pixel — The name of the rule, must be unique.

* Index: pixel — e.g. logstash-* or logstash-%Y.%m.%d [?]

* Time type: Default

* Time field: post_date

WHEN count   OVER all documents   UNFILTERED   IS NOT EMPTY

Aggregation — Instead of sending alerts immediately, send a report of alerts on a schedule.

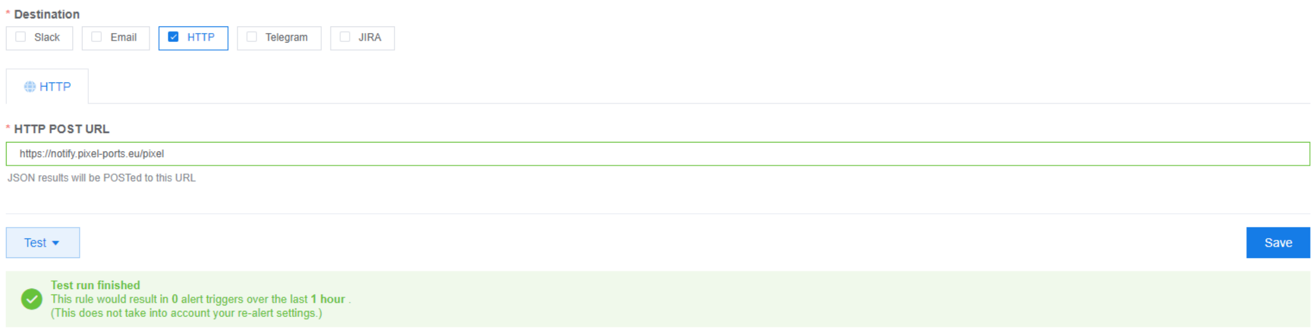
Re-alert: 5 Minutes — You will receive, at most, one alert every 5 minute(s), even if a rule triggers multiple times within that timeframe. If the rule is grouped over a field, this option will be applied on a per-group basis.

* Destination: Slack | Email | HTTP | Telegram | JIRA

Test ▾   Save

Figure 23: PRAECO's UI

This component has a region in order to test / check the newly created rule.



* Destination: Slack | Email | [✓ HTTP] | Telegram | JIRA

HTTP

* HTTP POST URL: https://notify.pixel-ports.eu/pixel

JSON results will be POSTed to this URL

Test ▾   Save

Test run finished
This rule would result in **0** alert triggers over the last **1 hour** .
(This does not take into account your re-alert settings.)

Figure 24: Check the newly created rule

4.4.4.2. Integration of the notifications in the dashboard

Among the functions involved in the Dashboard is the management of notifications. These notifications can come from executions of predefined rules / conditions.

Next picture illustrates the process since a rule is met until it arrives to the UI:

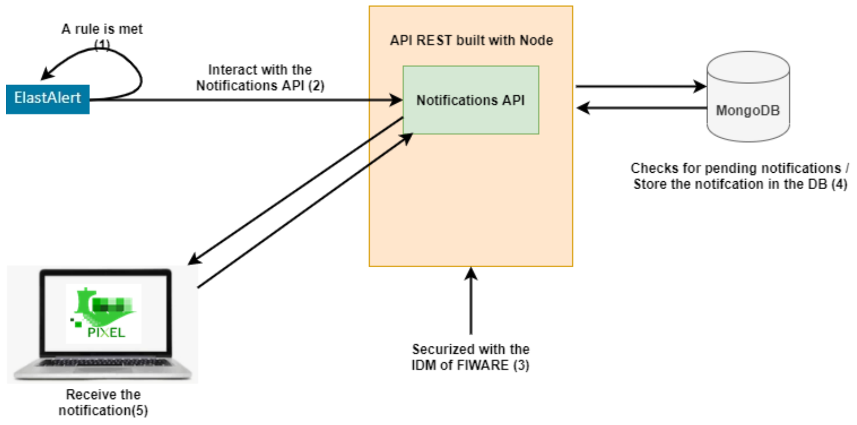

A rule is met (1)
ElastAlert
Interact with the Notifications API (2)
API REST built with Node
Notifications API
MongoDB
Checks for pending notifications / Store the notifcation in the DB (4)
Securized with the IDM of FIWARE (3)
PIXEL
Receive the notification(5)

Figure 25: Workflow related the reception of notifications

1- ElastAlert is the engine responsible to create and validate if a rule is met.

2- Once a rule is met ElastAlert will interact with the Notifications API. This API has been built with Node. This API will be securitized with the IDM of FIWARE.
3- After validated the access to the API throughout IDM of FIWARE, the API will insert the notification in the DB (MongoDB).
4- The API will check if exists any Notification pending to send to the UI.
5- UI will receive the notifications.

4.4.5. Privacy and Security

FIWARE modules KeyRock with Wilma and AuthzForce are the chosen option to implement the main security access mechanism on PIXEL.

KeyRock offers a unique endpoint to manage users, roles and organizations. All those information are available through the Identity Manager KeyRock API and the other PIXEL modules could rely on this API to manage user, roles and organization. Modules could share the authentication using a common PIXEL cookie (on / *.pixel-ports.eu for example) containing the access token provided by the API.

But for a better user experience, modules could also rely on the OAuth2 mechanism supported by KeyRock and Wilma. That way whatever URL was used to exposed the different portals of PIXEL, the user will authenticate only once on the identity endpoint. As OAuth2 is a common standard on the Internet it will be also quite easy to extend the PIXEL functional perimeter and to add new component on PIXEL.

The most standard way to manage identity and SSO on PIXEL is to use the OAuth2 mechanism. Later if PIXEL wants to integrate with the identity management system available on the port, it will only be necessary to build an OUAth2 interface on top of it.

5. Information Model

An **Information model**[16], in software engineering, is a representation of concepts, relationships, restrictions, business rules with the intention of specifying the data semantics of a domain (business area) in question. The model labels information according to the ways it will be accessed.

The model provides a framework where the information is accessible to experienced and inexperienced seekers alike.

Due to the large and wide variety of device types (different vendors, languages and techniques used) the device are not interoperable. In our context (IoT domain), the heterogeneity can be explained by the following reasons:

- The languages/techniques/protocols used for M2M processes and communication are often proprietary.
- Vertical fragmentation of IoT to cover all the needs of the different fields/applications/use cases (smart home, smart building, smart city, smart industry, smart health, etc.).
- Devices are not interoperable as they do not use common vocabularies to describe interoperable IoT data (Gyrard 2015). The languages used do not have common terms to describe the same things.

Several types of heterogeneity are encountered in the IoT field as for example:

1. **Temporal heterogeneity**. Due to the use of different sources of information from different machines and devices (different time zones and unsynchronized clocks).
2. The **autonomy** of data sources is another type of heterogeneity.
3. **Data heterogeneity**. When similar data are represented in a different form in information sources. This type of heterogeneity includes several issues:

[16] https://en.wikipedia.org/wiki/Information_model

a. **Naming**. Data sources can use different naming conventions for the values they store (case sensitivity, acronyms, misspellings, etc.).

b. **ID mismatch or missing ID**. The ID used in two different databases can differ. This is particularly true when the data source used URIs for identification. In this case, the namespaces or the use of prefixes may introduce differences between the data sources.

c. **Element ordering**. If set of data are used, the order can differ between two data sources.

4. **Language heterogeneity.** Includes all the differences due to the languages used in the sources and variations introduced by the latter (grammar, ambiguity, etc.):

   a. **Encoding**. This type of heterogeneity is introduced when different encoding mechanisms are used (ANSI, UTF-8, etc.).

   b. **Language**. This encompasses script mismatch (variations during the process of stemming for example), parsing errors (Arabic language vs romance language), ambiguous sentences, semantic errors, etc.

Due to this reasons is for what it's so important the definition of the Information model.

A good information model must ensure interoperability among IoT cross domain applications. Next figure, Example of IoT cross domain applications, will illustrate better the relevance of this concept.

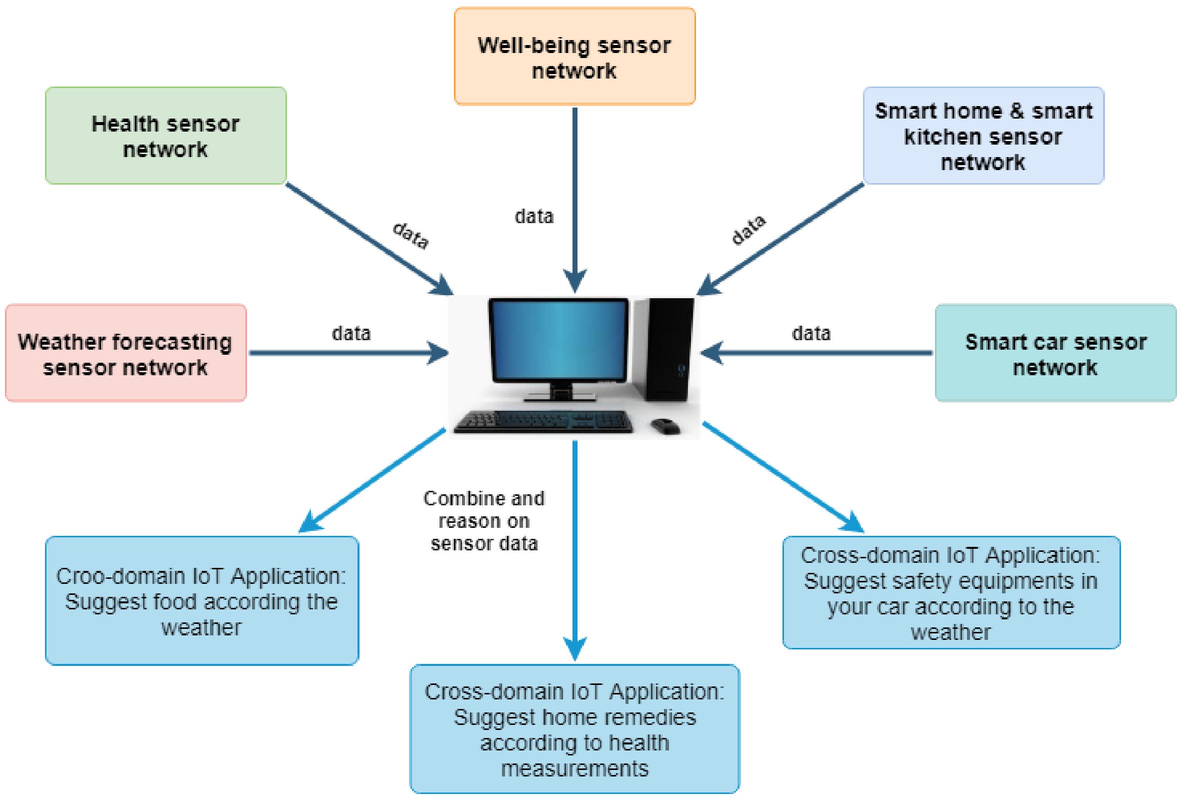

Well-being sensor network
Health sensor network
Smart home & smart kitchen sensor network
data
data
data
Weather forecasting sensor network
data
data
Smart car sensor network
Combine and reason on sensor data
Croo-domain IoT Application: Suggest food according the weather
Cross-domain IoT Application: Suggest safety equipments in your car according to the weather
Cross-domain IoT Application: Suggest home remedies according to health measurements

Figure 26: Example of IoT cross domain applications

5.1. Review of existing Data Models

A recommended way for using ontologies is trying to reuse existing ones rather that creating an own one, and extend or adapt it. So, the idea in this section is to explain one of the most known ontologies (FIWARE) and explain why (or why not) use the models offered.

5.1.1. FIWARE Data Models[17]

FIWARE[18] provides several data models easy to reuse and to contribute. They cover several domains and are built together with the FIWARE community. These data models[19] have been harmonized to enable data portability for different applications including, but not limited, to Smart Cities. They are intended to be used together with FIWARE NGSI version 2[20].

They define several domains that cover PIXEL partially needs:

- **ENVIRONMENT**[21]: Enable to monitor air quality and other environmental conditions for a healthier living.
- **TRANSPORTATION**[22]: Transportation data models for smart mobility and efficient management of municipal services.
- **DEVICE**[23]: IoT devices (sensors, actuators, wearables, etc.) with their characteristics and dynamic status.
- **PARKING**[24]: Real time and static parking data (on street and off street) interoperable with the EU standard DATEX II.

FIWARE Data models have been identified in PIXEL as the starting point to build data formats regarding IoT sensors to be integrated in the PIXEL Data Acquisition Layer (DAL). Considering that the employed software DAL technology is also composed by FIWARE enablers, it seems a suitable approach and potentially identifies specific contribution from PIXEL towards the FIWARE community by potentially extending models, or adding new ones.

5.1.1.1. Data Model

Templates have been created for the creation of the different data models. At the time to defining the models there will be a series of generic fields and others that will depend on the nature of the model.

Within the generic fields you will find the following:

Table 2: Generic field for Data Model

Field	Mandatory	Type	Format	Description
id	YES	String		Unique identifier
type	YES	String		Entity type it must be equal to DATAMODEL NAME
dataProvider	YES	URL		Specifies the URL to information about the provider of this information
source	YES	String		The ID of the PIXEL Data Source
dateModified	AUTO	DateTime		Last update timestamp of this entity (Read-Only Automatically generated)
dateCreated	AUTO	DateTime		Entity's creation timestamp. (Read-Only Automatically generated)

[17] https://fiware-datamodels.readthedocs.io/en/latest/
[18] https://www.fiware.org/developers/
[19] https://www.fiware.org/developers/data-models/
[20] https://www.fiware.org/2016/06/08/fiware-ngsi-version-2-release-candidate/
[21] http://fiware-datamodels.readthedocs.io/en/latest/Environment/doc/introduction/index.html
[22] http://fiware-datamodels.readthedocs.io/en/latest/Transportation/doc/introduction/index.html
[23] http://fiware-datamodels.readthedocs.io/en/latest/Device/doc/introduction/index.html
[24] http://fiware-datamodels.readthedocs.io/en/latest/Parking/doc/introduction/index.html

name		String		A name to describe this entity
location	or address	GeoJSON		Location of the data represented by a GeoJSON geometry
address	or location	String		Civic address of the data
refDevice		RelationShip		A reference to the device(s) which capture this data

5.2. PIXEL Initial Data Model

According to Princeton University[25], "*A **data model** organizes data elements and standardizes how the data elements relate to one another. Since data elements document real life entities, places and things and the events between them, **the data model represents reality***". Thus, what the Data Model definition in PIXEL has been tackled considering:

- What elements form part of the PIXEL system from a reality-representation point of view?
- Which and how many different entities will be used from different parts of the system?
- How to express them in a common fashion in order to store the global knowledge that the system must have?
- Which are the structures and relations that could be inferred about them?

In order to answer these questions, an **inductive** methodology has been followed. Particularly, the Data Model definition team conducted a three-step process that is outlined below:

1. To envision several short scenarios of usage of the system from different perspectives. The aim in this step is to cover the basics of PIXEL usability and realise which kind of entities would be needed to embed the information required to satisfy them. In this step the crucial words are highlighted. The words highlighted will become entities of the data model.

   The idea is to keep the level of deepness short, as we are aiming to define a **global Data Model**. It is planned that each sub-component of the architecture (Operational Tools, Acquisition Layer, Notification, Models) will have their own data model and database organization.

2. To create a tabular graphical definition of the Data Model. Following classic approaches, the first structure of data per entity has been created in order to represent the reality. The idea is to have a common grounding to be shared with all the Consortium and stakeholders to advance on ICT development.

3. To enrich the explanation of the Data Model by adding a brief description of each entity and its possible representation (e.g. field options).

Hereafter we include the activity and results of those steps:

1. Key sentences or scenarios of usage

One **user** of the system connects to PIXEL platform. He/she belongs to a **stakeholder** within the environment of a **port**.

[25] https://cedar.princeton.edu/understanding-data/what-data-model

This user requests a **service** from the system. This service can be a prediction (**predictive algorithm**) over an operation of the port, the simulation of a **model** of one use-case, retrieving information from the hub or visualizing the PIXEL dashboard.

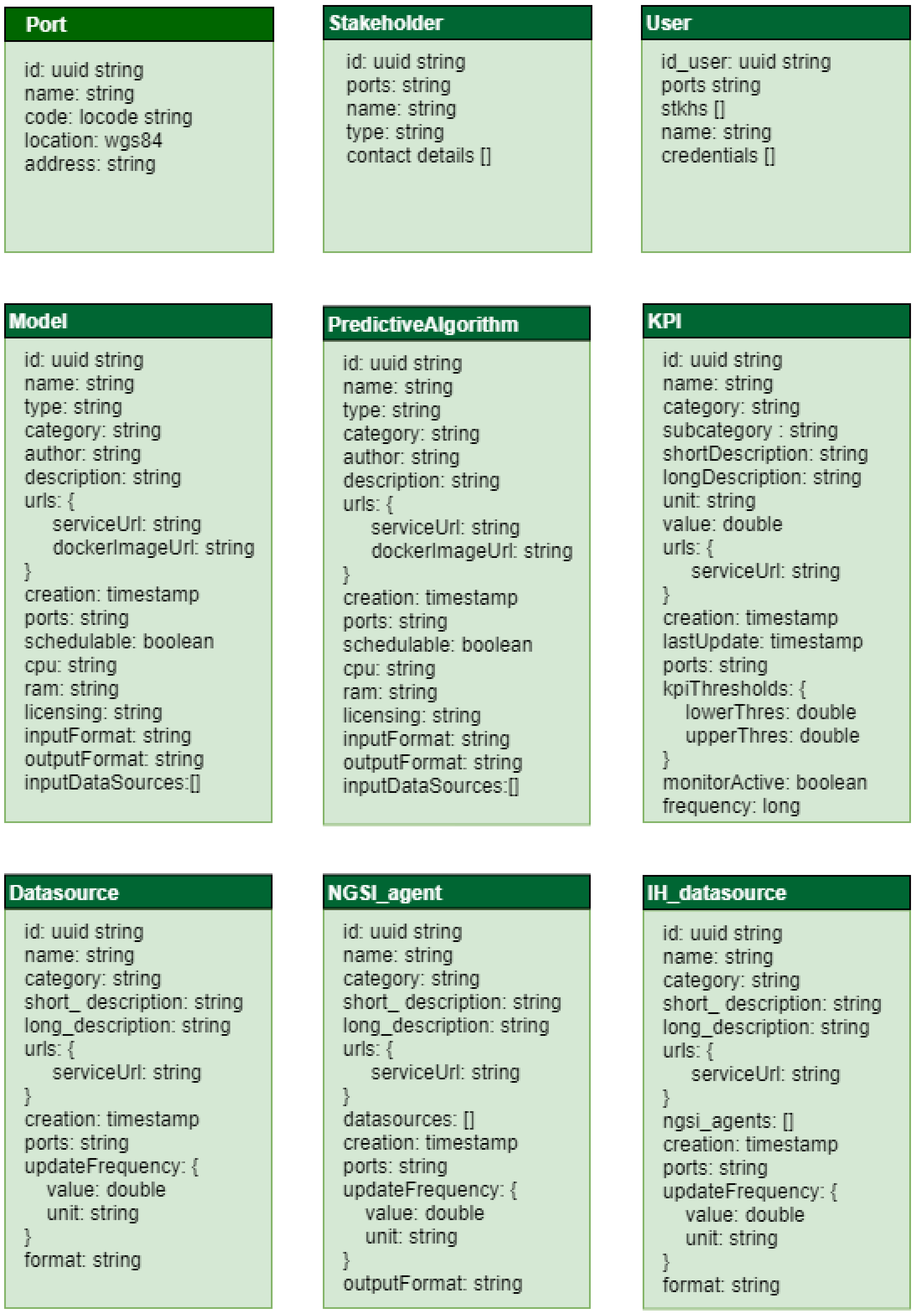
This service will rely on performing several operations over data coming from heterogeneous **data sources** within a port: sensors, legacy ICT systems or others. The most innovative service of PIXEL is the **PEI**.

In order to validate and demonstrate the PIXEL concept and its usefulness in the pilots, several **KPI**s will be evaluated.

Thus, the initial Data Model will be comprised of the following entities:

- Port.
- Stakeholder.
- User.
- Data source.
- Model / PA (Predictive Algorithm). PEI will be considered as a special model. Both models and Pas are invoked as a service.
- KPI.

2. Global Initial PIXEL Data Model



```
Port
id: uuid string
name: string
code: locode string
location: wgs84
address: string
```

```
Stakeholder
id: uuid string
ports: string
name: string
type: string
contact details []
```

```
User
id_user: uuid string
ports string
stkhs []
name: string
credentials []
```

```
Model
id: uuid string
name: string
type: string
category: string
author: string
description: string
urls: {
    serviceUrl: string
    dockerImageUrl: string
}
creation: timestamp
ports: string
schedulable: boolean
cpu: string
ram: string
licensing: string
inputFormat: string
outputFormat: string
inputDataSources:[]
```

```
PredictiveAlgorithm
id: uuid string
name: string
type: string
category: string
author: string
description: string
urls: {
    serviceUrl: string
    dockerImageUrl: string
}
creation: timestamp
ports: string
schedulable: boolean
cpu: string
ram: string
licensing: string
inputFormat: string
outputFormat: string
inputDataSources:[]
```

```
KPI
id: uuid string
name: string
category: string
subcategory : string
shortDescription: string
longDescription: string
unit: string
value: double
urls: {
    serviceUrl: string
}
creation: timestamp
lastUpdate: timestamp
ports: string
kpiThresholds: {
   lowerThres: double
   upperThres: double
}
monitorActive: boolean
frequency: long
```

```
Datasource
id: uuid string
name: string
category: string
short_ description: string
long_description: string
urls: {
    serviceUrl: string
}
creation: timestamp
ports: string
updateFrequency: {
   value: double
   unit: string
}
format: string
```

```
NGSI_agent
id: uuid string
name: string
category: string
short_ description: string
long_description: string
urls: {
    serviceUrl: string
}
datasources: []
creation: timestamp
ports: string
updateFrequency: {
   value: double
   unit: string
}
outputFormat: string
```

```
IH_datasource
id: uuid string
name: string
category: string
short_ description: string
long_description: string
urls: {
    serviceUrl: string
}
ngsi_agents: []
creation: timestamp
ports: string
updateFrequency: {
   value: double
   unit: string
}
format: string
```

Figure 27: PIXEL Initial Data Model

3. <u>Enriched explanation of Data Model entities:</u>

PIXEL aims at creating a solution for (mainly) small and medium ports. Therefore, the basic entity that will need to be stored in a global context of the system is the **Port.** Initially this data representation will be composed by id, name, location and code (LOCODE[26] code). Whenever the system becomes more sophisticated this entity will be enhanced, including, for example: type of port (according to D3.1 definitions).

Ports are usually managed by a Port Authority which, in turn, tends to rely some activities to stakeholders that work within its environment. PIXEL contemplates several users that potentially may come from any of those stakeholders. For that reason, the **Stakeholder** entity will be associated to at least one port. Other fields are type (whether it is a terminal company, truck operator, etc.), name and contact details. This can be enhanced in the future as it is supposed to be a document based database.

The **User** is the basic entity interacting with the system. It is necessary to include a user as a separate entity in the data model as this will be used for personalisation, security and configuration means. One user will be always part of at least one stakeholder of at least one port.

The **models** and **predictive algorithms** will be the reference services invoked by users. By the time of publication of a service (or PA), the admin may specify either a Docker image or a service endpoint. Other important mandatory parameters are the input and output formats, in order to be able to feed properly the models (or Pas) at execution time. Other optional parameters, but helpful for deployment and execution is information related to required CPU (vCPU) and RAM. The **PEI** is the most innovative proposal of PIXEL, but from the technical perspective, it is conceived as another model.

KPIs are also an important entity in order to evaluate the port operations. According to the GA, PIXEL should be able to provide both operational and environmental KPIs, which must be defined by each port. Obviously, a native environmental KPI is represented by the PEI, which is fact, is a composed KPI encompassing indexes, and eKPIs. Both of them will be modelled as KPIs according to this datamodel. As can be observed in the previous Figure, it is possible to define some thresholds in order to automatically monitor the status and detect patterns or anomalies.

Data Source is a basic entity in PIXEL, but must be adapted to the architecture proposed in PIXEL. Therefore, one must distinguish:

- The original data source (**Datasource**), representing typically the information offered by ports, sensors, third parties, etc.
- The final data source (**IH_datasource**), representing the information offered by the information, with a certain data format and capable of being understood and interpreted by models and predictive algorithms.

In order to transition from the Datasource to the IH_datasource, an NGSI agent is required to capture the data and convert it properly into the common destination data format. This is the basis for syntactic and semantic interoperability. FIWARE data models are the basis for this, as explained before.

5.3. Data examples

The aim of this section is to show an example of data since they reach the data acquisition layer until the data are pushed to Orion.

5.3.1. Integrated Surface Data

The example described will be Integrated Surface Data for that it's necessary to run the Air Pollution Model. The Data are provided by the NOAA (National Oceanic and Atmospheric Administration - US) all over the

[26] https://www.unece.org/cefact/locode/service/location

world using the ISH data format describe by this document: https://www1.ncdc.noaa.gov/pub/data/ish/ish-format-document.pdf.

In this case, the Data Acquisition Layer has to first retrieve the raw data exposed by the public source, then decode it and transform it to fit a PIXEL Data Model and finally it has to push it to Orion in order to make the data available.

5.3.1.1. Raw Data

The data are available on a public FTP server: ftp://ftp.ncdc.noaa.gov/pub/data/noaa/2019/167160-99999-2019.gz. So you have to choose the one matching your location and date period, and then download it.

The file contains multiple records encoded with the ISH format.

```
0186167160999992019010100004+37883+023750FM-
12+004599999V0203601N006719999999N999999999+00731+00571101171ADDAA106024031AW1411AZ
141021AZ241021MA1999999100651MD1710141+9999OD139901441999REMSYN09816716    17///    /3613
10073 20057 30065 40117 57014 60241 74144 333 90730 91128 555 60000 60163 60300=
```

Figure 28: Multiple records in ISH format

5.3.1.2. Transformations

To decode the data we rely on an open source parser that allow us to retrieve each record of data: https://github.com/haydenth/ish_parser. After that the data will be transformed to make it available in a PIXEL Data Format that is close to the FIWARE Data Model WeatherObserved[27] (see next table).

Table 3: PIXEL Data format for this example

Model Field	Source Field	Format	Transformation
Id			NOAA:ISH:USAF:WBAN:yyyy-MM-dd'T'HH:mm:ss
Type			NoaaIsh
dataProvider			ftp://ftp.ncdc.noaa.gov/pub/data/noaa/2019/167160-99999-2019.gz
source			NoaaIsh
Location			Type: "point", coordinates: [+0.23.735, +37.882]
weatherStation			USAF
elevation	decoded from ISH		
dataObserved	decoded from ISH	"yyyy-MM-dd'T'HH:mm:ss"	The date and time of this observation in ISO8601 UTCformat. It can be represented by a specific time instant or by an ISO8601 interval.
Temperature	decoded from ISH	Default unit: Celsius degrees	
windSpeed	decoded from ISH	Default unit: meters per second	

[27] https://fiware-datamodels.readthedocs.io/en/latest/Weather/WeatherObserved/doc/spec/index.html

windDirection	decoded from ISH	Default unit: Decimal degrees	
rawISHData	The raw ISH data trame		

5.3.1.3. Final Data

Finally, the data is pushed to Orion using the corresponding JSON structure. The rawISHData is kept in order to facilitate the integration with the Air Pollution model that is used to deal with this format.

```
{
    "id": "NOAA:ISH:167160:99999-2019-01-01T00:00:04Z",
    "type": "NoaaIsh",
    "dataProvider": "ftp://ftp.ncdc.noaa.gov/pub/data/noaa/2019/167160-99999-2019.gz",
    "source": "NoaaIsh",
    "dateObserved": "2019-01-01T00:00:04Z",
    "location": {
        "type": "Point",
        "coordinates": [+023.735, +37.882]
    },
    "elevation": "200",
    "temperature": 3.3,
    "windDirection": -45,
    "windSpeed": 2,
    "rawISHData": "0186167160999992019010100004+37883+023750FM-
12+004599999V0203601N006719999999N999999999+00731+00571101171ADDAA106024031AW1411
AZ141021AZ241021MA1999999100651MD1710141+9999OD139901441999REMSYN09816716 17///
/3613 10073 20057 30065 40117 57014 60241 74144 333 90730 91128 555 60000 60163
60300="
}
```

After that the JSON is pushed to ORION, and the Information Hub can be notifying that this information is available.

5.3.2. Vessel Call

Another important data source we have to import is the vessel call provide by the ports. Each could provide the same kind of information, but not the same way, same format and same temporality.

5.3.2.1. Raw Data

5.3.2.1.1. Piraeus

The data is provided every month with the past data through a CSV file sent to an NGSI Agent.

```
"SHIP NAME","COMPANY","AGENT","Î™ÎœÎŸ","SCHEDULED ARRIVAL DATE","SCHEDULED ARRIVAL TIME","ACTUAL
ARRIVAL DATE","ACTUAL ARRIVAL TIME","SCHEDULED DEPARTURE DATE","SCHEDULED DEPARTURE TIME","ACTUAL
DEPARTURE DATE","ACTUAL DEPARTURE TIME","CAPACITY","PASSENGERS TO ARRIVE","TRANSIT PASSENGERS","TOTAL
PASSENGERS ARRIVED","ARRIVAL RATIO","PASSENGERS TO DEPART","TRANSIT PASSENGERS DEPARTURED","TOTAL
DEPARTURED PASSENGERS","DEPARTURE RATIO","HOME PORT FLAG"
```

5.3.2.1.2. Bordeaux

A special component is built on Bordeaux side to push the vessel call data every time they arrive on the PIXEL platform.

```
 {
    "id": "FR_BAS:9137870:4",
    "type": "VesselCallType",
    "amountOfHandle": 2010,
    "arrivalTime": "5 Jan 15 17:07:33",
    "berth": 449,
    "cargoType": "E.SORGHO VRAC",
    "loadingDirection": "unloading",
    "shipName": "LEZHEVO",
  }
```

5.3.2.2. Transformation

The idea is that the model developed on PIXEL works for every port that is able to provide the input data necessary to run it. So, it will be necessary to transform those data to fit in a common Data Model.

For PIXEL has been defined a common Data Model for Vessel Call matching each data source but also each type of vessel (cargo or passengers). The transformation job of the NGSI agent is to fulfil the target data source with the information provided by the source.

Table 4: Header fields

Field	Mandatory	Type	Description
id	YES	String	Unique identifier
type	YES	String	Entity type. It must be equal to VesselCall
dataProvider	YES	URL	Specifies the URL to information about the provider of this information
source	YES	String	The ID of the PIXEL Data Source
dateModified	AUTO	DateTime	Last update timestamp of this entity (Read-Only. Automatically generated)
dateCreated	AUTO	DateTime	Entity’s creation timestamp. (Read-Only, Automatically generated.)
name		String	The ship name
location	or address	GeoJSON	Location of the data represented by a GeoJSON geometry
address	or location	String	Civil address of the data
refDevice		Relationship	A reference to the device(s) which captured this data

Table 5: Specifics Data fields

Field	Mandatory	Type	Description
IMO	YES	String	Ship international identification
journeyid	YES	Number	It identify the rotation of the boat to detect different vessel Call from the same ship and same day
company		String	Ship company name
agent		String	
unlocode		String	https://www.unece.org/cefact/locode/service/location.html
arrival_port_area		DateTime	Ship arrives at port area

arrival_dock		DateTime	Ship on dock
scheduled_arrival_dock		DateTime	Scheduled date for ship on dock
start_work		DateTime	Work start
end_work		DateTime	Work finished
leave_dock		DateTime	Ship leaves dock
scheduled_leave_dock		DateTime	Scheduled date for ship leaves dock
leave_port_area		DateTime	Ship leaves port area
dock		String	The dock identifier
work_descr		String	Work description
operation	YES	String	The type of operation
cargo_type	YES	String	Cargo description or PASSENGERS
cargo_fiscal_type		String	Cargo Fiscal type (Only for French port?)
empty_container		Number	The number of empty container
full_container		Number	The number of full container
capacity	YES	Number	The weight of the cargo (in tons) or the capacity of passengers
passengers_to_arrive		Number	
passengers_to_depart		Number	
transit_passengers		Number	

The fields of Table 5 are generic and not necessary fully applicable for all ports.

5.3.2.3. Final Data

For each entity generated by the transformation process, the NGSI will push it on the Orion database to make it available for the information Hub

For example the data generate for Piraeus look like this:

```
{
 "id":"GRPIR:VesselCall:9228356:2018-08-01T07:30:00+02:00:1",
 "type":"VesselCall",
 "source":"GRPIR:VesselCall",
 "dataProvider":"CSV File",
 "location":{
  "type":"point",
  "coordinates":[
   23.635979,
   37.939013
  ]
 },
 "imo":"9228356",
 "name":"JEWEL OF THE SEAS",
 "company":"ROYAL CARIBBEAN CRUISES",
 "agent":"ΙΝΣΚΕΙΠ ΝΑΥΤ/ΚΕΣ ΕΠΙΧΕΙΡΗΣΕΙΣ Α.Ε",
 "journeyid":1,
 "unlocode":"GR PIR",
 "arrival_dock":"2018-08-01T07:30:00+02:00",
 "scheduled_arrival_dock":"2018-08-01T07:30:00+02:00",
 "leave_dock":"2018-08-01T20:00:00+02:00",
 "scheduled_leave_dock":"2018-08-01T20:00:00+02:00",
 "operation":"both",
 "cargo_type":"passengers",
 "capacity":2700,
 "passengers_to_arrive":2,
 "passengers_to_depart":7,
 "transit_passengers":2670
}
```

6. Deployment and scalability

In software development, deployment refers to the phase when, once all developments fulfilling the features are ready, functional and non-functional requirements, the software is installed in the final servers where it will run in production conditions to offer the benefits that was conceived for. This phase not only consists of the process of installing the software in the final destination, but also organising the number of nodes where the solution will be installed, how these nodes communicate together, which communications they have with external resources or services and what are the maintenance protocols, among others.

In PIXEL, the deployment deserves special attention since the same architecture will be deployed in four different scenarios that will provide different resources and will work with different setups. It will be necessary to put the focus on the different physical deployment options available for PIXEL (both at the level of solution and at the level of hosting of its infrastructure and/or data. For that, section *Ports deployments* will speak about the deployment in the different ports. Also concepts that were mention in: *Applied Agile concepts* (Containers, CI/CD and APIs) will be presented in first sections. Obviously, it will be necessary to speak about *Environments and testing* (in order to introduce the work that will be made in D7.1). Finally, linked to the concept of deployment we will analyse the **scalability** and the **deployment architecture**.

6.1. Containers

Due to the ever-increasing number of devices that requires connectivity to applications or platforms it becomes necessary a way for scale these applications independently. The answer is **containers**.

Containers allow applications to be deployed reliably and migrated quickly between various computing environments by packaging code, configuration settings and dependencies into a single object.

The use of containers is very characteristic in Agile Developments. They approach to the focus that IoT solutions require.

The aim of containers is to divide complex systems into microservices (see Figure 29, different services included in the container used for PIXEL Dashboard and Notifications).

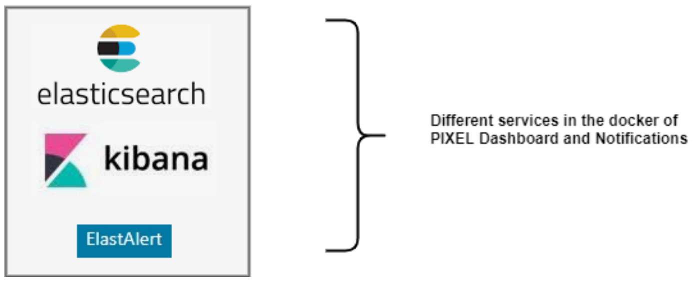

elasticsearch
kibana
ElastAlert
Different services in the docker of PIXEL Dashboard and Notifications

Figure 29: Docker for PIXEL Dashboard and Notifications

Advantages of the use of containers:

- **Portability**. Possibility to deploy the application in any environment on any operating system.
- **Security**. Containers act as an isolated system. They do not interact among them.
- **Distributed Integration**. Containers facilitate **CI** (Continuous integration) and **CD** (Continuous delivery).

6.1.1. Docker

Among container managers, **Docker** (www.docker.com) is the most widely used to encapsulate applications into software packages called containers, that are more lightweight than virtual machines. Docker offers several features: cross-platform, versioning, reusability, shared libraries and repository. There is a huge community of Docker, and many applications are already available for an easy installation as images in the Docker hub (https://hub.docker.com/). One of the main outputs from the PIXEL project is also to distribute the developed components as open source, possibly using GitHub or Docker hub (or both).

Docker can be considered as an evolution of **LXC** (LinuX Containers), also known as microcontainers and also linked to LXD (manager of LXC). Though the approach in LXC is similar and the memory footprint can be quite reduced, it is only designed for Linux environments and does not have a wide community behind compared to Docker. Kubernetes, and other orchestration engines, typically only support Docker.

6.2. Distributed integration

The architecture of PIXEL (PIXEL RA) is a series of interconnected modules. The development of applications / platforms more modular satisfy the need for faster and simpler integration of new services and also facilitates continuous integration (**CI**).

This is the same trend that is followed in IoT development. It has moved from centralized control to a distributed model and collaborative participation.

Continuous delivery (**CD** also known as **CI**) is the ability to get changes of all types (including new features, configuration changes, bug fixes and experiment) into production, or into the hands of users, safely and quickly in a sustainable way. Next pictureFigure 30 depicts the cycle follow for a CI in a development.

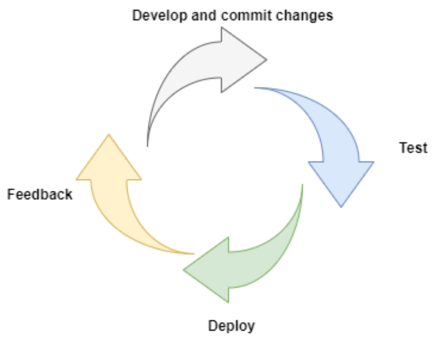


Figure 30: Cycle of Continuous Integration

6.3. APIs

The capacities of communications for existing applications increase daily. Among the reasons are the following:

- Increase in service providers (public / private services).
- Increase in the number of clients.

This implies that point-to-point integration is not sustainable. So, how will be possible the communication / interaction among all actors? The answer of this is with the use of **APIs**.

An API[28]: "Is an interface or communication protocol between a client and a server intended to simplify the building of client-side software".

With the use of APIs the UI can get access to IoT device data without need to know the hardware interfaces or device drivers.

The use of APIs allows access to applications, thus improving access to data. This has favored the creation of applications in unrelated environments as well as increasing the exploitation of data by being accessible to third parties.

APIs allow to use software (platforms, applications) written in several programming languages. This is because the APIs have a unified architectural style, REST.

PIXEL will have APIs in each layer with the aim to facilitate the communication among it.

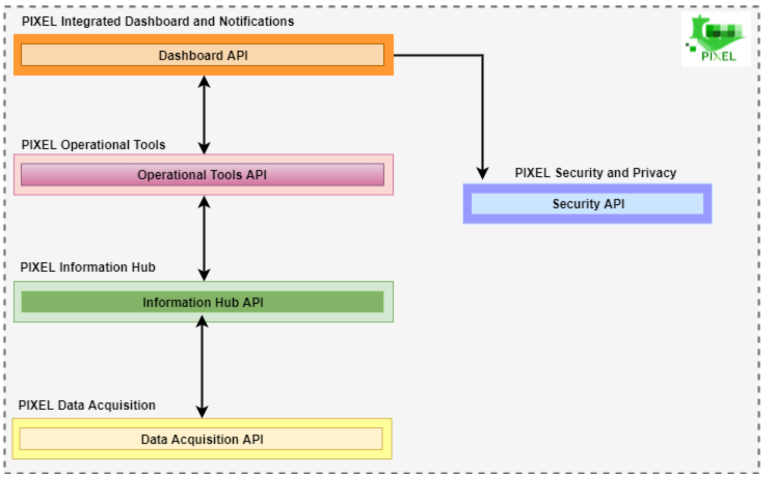


Figure 31: PIXEL APIs

These APIs will be used in WP7 task 7.1 to test the different modules.

Next section pretends to comment very briefly the APIs that will have each layer.

6.3.1. PIXEL APIs

6.3.1.1. Data Acquisition

The PIXEL Data Acquisition is composed of several components. Each of them will expose one or more APIs:

- **FIWARE Context Broker Orion**. This component exposes several APIs.
  - NGSI v2 API[29]. Used to manipulate the data. It is a HTTP REST API used to create, update, delete and request the entities and their attributes.

[28] https://en.wikipedia.org/wiki/Application_programming_interface
[29] http://fiware.github.io/specifications/ngsiv2/stable/

  - Subscription API[30]. It allows other components to notify by the Context Broker when entities are modified.
- **NGSI Agent**. The NGSI Agents are designed to connect external data to the Data Acquisition Layer of PIXEL through the Context Broker. To communicate with the Context Broker NGSI Agents use the NGSI v2 API exposed by Orion. To connect to the Data Source, it will use any kind of relevant API or communication protocol that the data source can communicate with.
- **FIWARE Cygnus**. Cygnus is designed to send data from the Context Broker to several solution of data persistence. To communicate with Orion, Cygnus will expose Notify API that will be register to the Context Broker with the Subscription API.
- **FIWARE Comet STH**. Like Cygnus, Comet implement a Notify API in order to allow Orion to push data it in, but it is better to use Cygnus for that purpose. Comet proposes its own HTTP REST API to access raw[31] and aggregate[32] data.

6.3.1.2. Information Hub

The PIXEL Information Hub API is responsible for the interaction of the outside world with the Information Hub. Because of the Information Hubs data archiving nature, the information will flow both upstream and downstream. The Information Hub will have to handle simultaneous reading and writing from LTS and STS, for instance a constant inflow of data from sensors while simultaneously updating client applications with data. This is why operations of writing and reading will be segregated into multiple API entry points. The three components providing an API are:

- **Data collector.** Is responsible for handling the inflow of data into the PIXEL Information Hub from the PIXEL Data Acquisition system. The collector interface allows for custom implementations based on capabilities of the source connected. Data collector API will mostly use the PUSH mechanism via an exposed REST API endpoint for acquiring data, but will also be able to fall-back to old-style periodic PULL system for data sources that do not support PUSH.

- **Data extractor**. Is responsible for handling the outflow of data from the PIXEL Information Hub to any third party applications connected to it. Access to the Data Extractor instance node REST API will be routed through the API Gateway leveraging its authentication and authorization systems among others. Generally, the Data Extractor will be responsible for retrieving data from database storage systems connected to the Information Hub. Because of some limitations of the connected database sources the Data Extractor instance node will also have the ability to join data from multiple queries in memory, execute simple transformations and enrich the extracted data and pass it to the connected client application through the API Gateway.

- **Data redactor**. Is responsible for execution of reduction and aggregation algorithms, which can be provided by the customer. This component is not exposed through REST API, but provides an interface for the implementation of custom Data reductors. This allows implementation of custom reduction, aggregation or other data transformation algorithms provided by customers.

6.3.1.3. Operational Tools

The OT API is responsible for providing all implemented functionalities to the outside world. The OT UI is just an example for a visual client, but for automating executions (no direct user involved) it should be possible to invoke directly the API. For example, VIGIEsip in the Port of Bordeaux may directly interact with the OT API and provide its own UI to the user. Other components as part of the PIXEL architecture may also use directly the OT API. A complete set of features is not the aim of this deliverable, as this will be provided in D6.5. However, some high-level features have been identified that this API will have to cope with:

[30] https://fiware-orion.readthedocs.io/en/master/user/walkthrough_apiv2/index.html#subscriptions
[31] https://fiware-sth-comet.readthedocs.io/en/latest/raw-data-retrieval/index.html
[32] https://fiware-sth-comet.readthedocs.io/en/latest/aggregated-data-retrieval/index.html

- Publishing models and predictive algorithms in the PIXEL platform. Even if this can be a service by its own, it should be somehow assisted by or synchronized with the Operational Tools.
- Creation, edition and execution of models and predictive algorithms, both in runtime or as scheduled jobs.
- Access to the OT data model, at least to part of it (e.g. model configuration, model result, etc.).
- Creation and edition of Event Processing rules.
- Self-documented API. By means of providing an open API (e.g. Swagger file), developers will easier and better familiarize with the interface, facilitating internal development throughout the project as well as transferability to other ports.

6.3.1.4. Dashboard

This API will be in charge of communicating the interface with the rest of the components. It is secured (making use of the implementation of FIWARE IDM in PIXEL). In order to have access it will be necessary to have a valid user, so all calls to the rest of the components that come from the API are considered secure. The main features of this API are:

- *User Management*. User authorization and authentication, creation of users, etc.
- *Display Management*. Module in charge of creating visualizations to manage them.
- *Manage visual dashboard configurations*. Manage visual aspects related to the dashboard (Zoom, Widgets, etc.).
- *Communication with OT module*. UI responsible for executing models, show simulations, etc.

6.3.1.5. Security

In this section we are going to see the APIs that expose each one of the components included in this module:

- **FIWARE OAuth2 Server – Keyrock**. Keyrock offers the full functionalities to manage users, organizations, roles. Keyrock exposes all of these features through an HTTP REST API[33].
- **FIWARE – AuthzForce**. Provides the following APIs:
    - PDP API (Policy Decision Point in the XACML terminology). Provides an API for getting authorization decisions computed by a XACML – compliant access control engine.
    - PAP API (Policy Administration Point in XACML terminology). Provides an API for managing XACML policies to be handled by the Authorization Service PDP.

  The full API (RESTful) is described by a document written in the Web Application Description Language format (WADL) and associated XML schema files available in Authzforce rest-api-model project files[34].

  XACML is the main international OASIS standard for access control language and request-response formats, that addresses most use cases of access control. AuthzForce supports the full CORE XACML 3.0 language; therefore it allows enforcing generic and complex control policies.

- **FIWARE – Wilma PEP Proxy**. Wilma is an HTTP Reverse Proxy component that can communicate with Keyrock and AuthzForce to ensure that sends a request to Wilma is allow executing this API call.

6.4. Environments and testing

Before deploying an application in production, it is necessary to test it. For this reason it is important to have different environments. An **environment** can be defined as a configuration of a set of nodes or even one single node that respond to particular conditions that are in favour on different stages in the development

[33] https://keyrock.docs.apiary.io/#reference/keyrock-api
[34] https://github.com/authzforce/rest-api-model/tree/release-5.3.1/src/main/resources

phases. There are several ways to organise the environments in a software engineering project. A common standard way is typically:

- **Local:** the workstation of the developer/programmer. It is typically customised by the developer following common guidelines of the project.
- **Development:** a server where a development teams have common development resources such as databases, acting as a common sandbox for multiple purposes.
- **Integration:** a common server where the integration works (test, builds, etc.) is performed in an independent (usually scripted) way. The integration server is used to detect side effects in developments that imply different functional groups that interact among them.
- **Pre-production or staging:** a mirror of the production environment that is used to test the software in the closest conditions to the real environment where it will be operating (production). The primary use of a staging environment is to test all the installation/configuration/migration scripts and procedures before they are applied to a production environment. Other common use is load testing to check the performance of the system in near-real conditions.
- **Production:** this environment is where the real user or real user application interact with the new software. This environment is the most critical part since it interacts with all the working systems of the user, so that any error or problem introduced by the new software can lead to a complete stop or general malfunction of the systems.

To minimise the effects of switching context and ensuring the proper accomplishment of the requirements, it is advisable to prepare software test in the different environments and with different objectives. In PIXEL, these tests will be linked to the different use cases.

In PIXEL, there will be deployed at least the local, development and integration environments, as well as the production provided by ports. In first iteration of this document (D6.1) only the two first are already being used. But now, the third is ready to use because it will be necessary for the WP7 task 7.1 (month 18).

The tests can be divided into two large blocks: Functional and Non-Functional. PIXEL will focus on functional testing (included in the scope of D7.1). Within the functional we find the following types of tests:

- Unit tests.
- Module tests.
- Integration tests.

In this project, the testing principles followed will be[35]:

- **Usability**. Due to the great complexity and the number of devices this factor is very relevant. Tests will ensure the ease of use, reception of notifications, etc.
- **Security**. It is a key factor in IoT. All the devices are connected between them and the data must be accessible. Communication protocols must ensure data integrity and privacy (this can be solved for instance by using protocols that support **TLS[36]/SSL**[37]).
- **Connectivity**. Check if the system is available as well as the different networks which is composed of.
- **Compatibility**. Due to the number of devices involved in an IoT platform and the different features employed. Compatibility tests (multiple operating system versions, browser type and respective versions, communication modes) are mandatory.
- **Pilot testing**. The best manner to test an IoT application is in its specific use case. Pilot testing is a need. The application must be exposed to a limited number of users in the real field. Moreover, this can help the production deployment. This will be performed in the different phases of the PIXEL pilots.

[35] https:// www.softwaretestinghelp.com/internet-of-things-iot-testing/
[36] https://www.globalsign.com/en/blog/ssl-vs-tls-difference/
[37] https://www.digicert.com/ssl/

- **Regulatory or legislative testing**. It is important to check that an application passes the different regulatory / compliance checkpoints.
- **Upgrade testing**. This aspect is related with the compatibility feature. When we upgrade the application, we must be sure that the overcome upgrade related issues (multiple protocols, devices, operating systems, firmware and hardware) have disappeared.

6.5. Scalability. Multi-Instance vs Multi-Tenant

Scalability is the ability of a system to preserve its average performance as the number of target elements (requests, nodes, services, etc.) increases. Another definition could be: the way to manage the growth of a system so that the quality of it (services offered) is not affected.

How a system will be deployed have consequences in their scalability. Exist two terms that refer to architectural principles related to deployment: Multi-Instance and Multi-Tenant. What are the characteristics of each of them?

- **Multi-tenancy**[38]: Is the ability to offer the same service to different customers from a single instance of software. In other words, a single code development can serve multiple users by separating sensitive information from each other that is only visible to them.

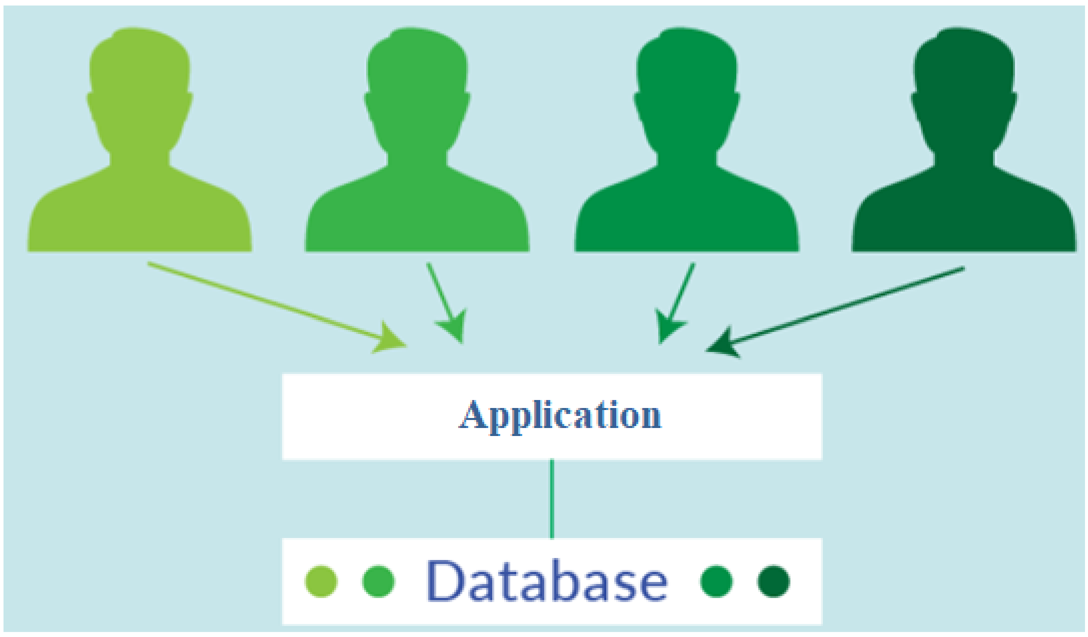


Figure 32: Multi-tenancy

Each client of the service is considered as a tenant, this allows administrators to customize elements of the application (such as interface colours), but does not customize the code as such.

In this type of architecture, it is important to emphasize that a client is not necessarily a unique user; it can be a group of users.

Among the advantages of this type of architecture we can find the following:

1. **Profitability**. Costs distributed among all customers.
2. **Easy updating**. Only one instance needs to be updated.
3. **Information security per client**. It has a separate schema for each user.
4. **Optimizes** the use of server resources.

Among its disadvantages we find the following:

1. **Personalization**. Difficulty in the use of specific characteristics for a client.

[38] https:// platzi.com/blog/multi-tenant-que-es-y-por-que-es-importante/

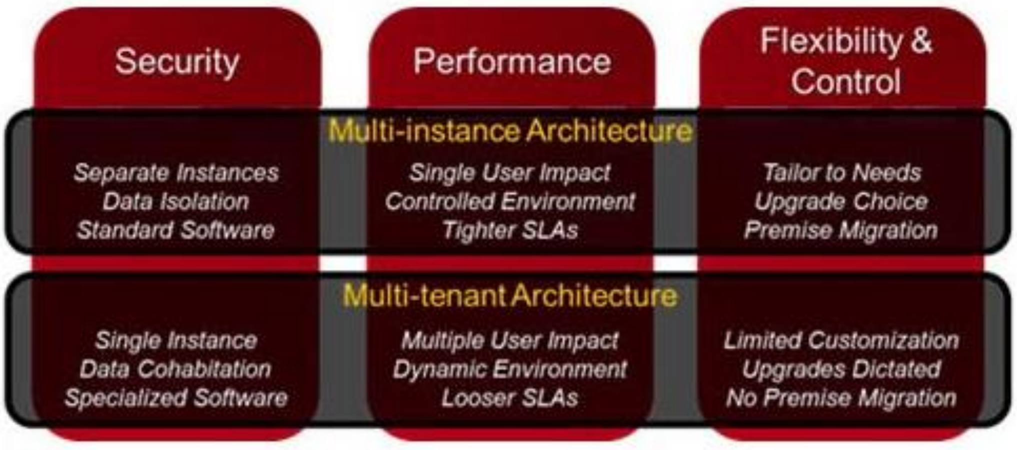
2. **Isolation**. Since different tenants share the same instance of software and hardware, it is possible for one tenant to affect the availability and performance of software from other tenants.
3. **Simultaneous updating**. Despite of being an advantage because, it only implies to update an instance, it can become an inconvenient since the simultaneous updating of software may not be desirable for all the tenants.
4. **Single point of failure**. If the application has an error, it will fail for all clients.

- **Multi-Instance**: Unlike the previous option in this case each client runs its own instance separate from the application.

  The advantages of this type of architecture are the following:

  - **Safer** than the previous case, it uses isolated environments.
  - It allows **greater flexibility** and control of configuration, customization, and updates.
  - **Less risks of attacks** affecting data security.
  - **Ability to migrate the instance** to a local server or other cloud provider.
  - The architecture allows for **greater growth and flexibility of deployment**.

The following image illustrates the characteristics of each option:



	Security	Performance	Flexibility & Control
Multi-instance Architecture	Separate Instances Data Isolation Standard Software	Single User Impact Controlled Environment Tighter SLAs	Tailor to Needs Upgrade Choice Premise Migration
Multi-tenant Architecture	Single Instance Data Cohabitation Specialized Software	Multiple User Impact Dynamic Environment Looser SLAs	Limited Customization Upgrades Dictated No Premise Migration

Figure 33: Characteristics of multi-Tenant and multi-Instance

Reasons to select multi-instance over multi-tenancy in PIXEL

Following the use cases and requirements of the project, the deployment of PIXEL will be done following the multi-instance architectural principles. The reason behind this decision is that multi-tenancy has demonstrated to be more demanding in resources and be potentially less secure and lack of privacy[39]. Considering the different contexts (ports) where PIXEL is designed to be deployed, having full control on each instance is an asset in order to be more adaptable to the circumstances of each scenario. This will be tested in the four different pilots and the scale-up actions envisaged in the project.

Also, having multi-instance is a better decision in terms of sales and marketing, since it is more acceptable from the point of view of a company to have the data completely separated from other user that can incidentally be the competition. Multi-instance architecture allows for easy migration from and to the cloud, or from and to one cloud hosting provider to another. With a multi-tenant architecture, it is not possible to move instance from the cloud to the premise, and vice versa, they are blocked.

[39] https:// fayebsg.com/2013/05/multi-tenancy-vs-multi-instance/

The conclusion is that multiple instance architecture is of great benefit to companies that value control over their systems, customization capability and flexibility, as well as the agility to respond to market changes and evolving business needs.

The following image illustrates with a diagram the comments throughout the section.

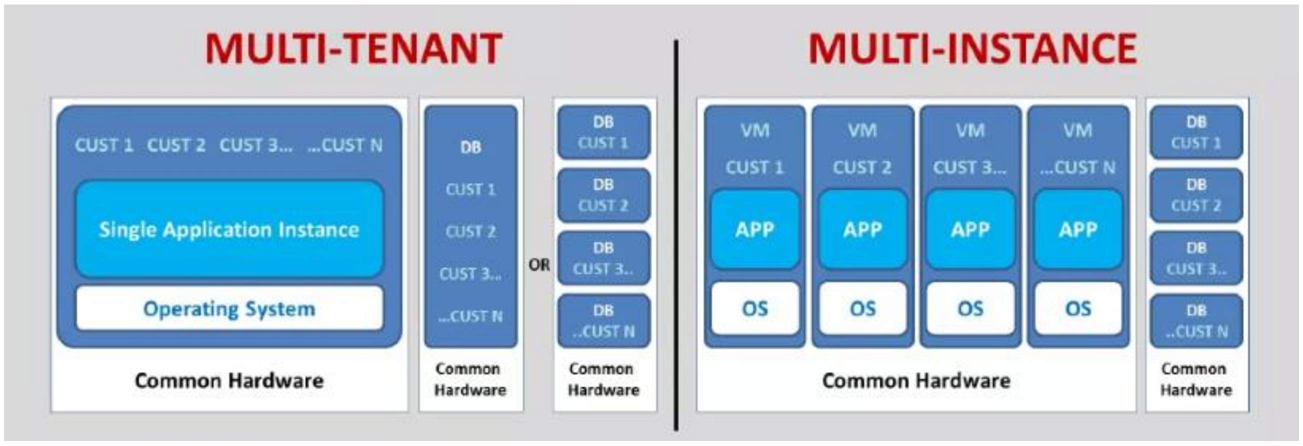


Figure 34 Differences between Multi-Tenant and Multi-Instance architecture[40]

6.5.1. Deployment of PIXEL platform

The way to execute the deployments in PIXEL will be with the use of containers (dockerization). In this way it is possible to simplify the cost in time of the deployments (they will be faster) and that the different modules of PIXEL are kept isolated.

The final objective is to achieve one container per module that contains all the elements that are part of that component.

In the case of the Dashboard and Notifications component, the following image illustrates the composition of that container:

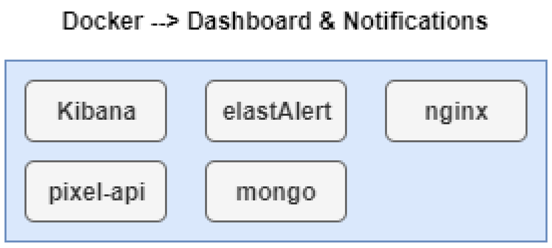


Figure 35: Elements in the Docker of Dashboard and Notifications

The ideal way to Docker the different PIXEL modules is with Docker Compose. It is a tool that allows us to create multiple container applications. It is very easy to start / stop / build a platform / application consisting of groups of related Docker containers.

Docker-compose allow us to define each one of the containers that we are going to implement, as well as characteristics for each container implementation.

An advantage of Docker-Compose is that it allows us to create configuration variants according to the environment. Therefore, in a very simple way, different container compositions can be created according to the environment.

[40] http:// www.contactcenterarchitects.com/dont-make-the-common-mistake-of-believing-multi-tenancy-is-the-same-as-multi-user-or-multi-enterprise-clouds-multi-tenancy-vs-multi-instance-in-ccaas-ucaas-clouds/

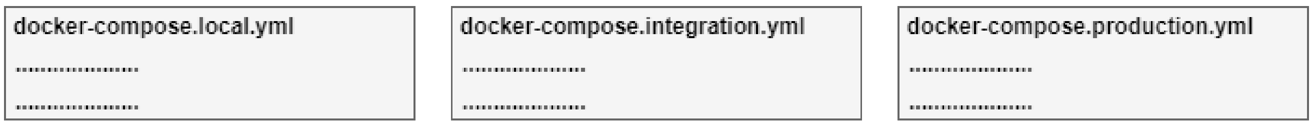


Figure 36: Different Docker-compose per environment

6.6. Deployment architecture. Cloud vs On-premises vs Hybrid

One of the first steps in any development should be to select the type of implementation[41]: On premise (own servers) or Cloud.

On premises: This refers to the private data center that companies operate and maintain 'at home'. Widely known as "On-Premise", the key benefit is the control it offers.

On-site hosting software means that companies can decide exactly what systems they want and where they are located, giving them the flexibility to create and ad-hoc system that exactly suits their needs.

The downside is the cost. The costs associated with installation, maintenance and various upgrades add up. Together with costs for electricity, physical space, cooling equipment and power supply tools this can be quite costly.

A viable alternative is the **Cloud**. It involves renting virtual server space that is hosted on the Internet and accessed remotely.

Cloud solutions tend to be more expend than internal servers[42]. But the pay-as-you-go offer makes it ideal for smaller businesses that do not want to invest in IT costs during their initial development. The ability of cloud users to scale up and down relatively quickly is ideal for businesses that are on the verge of rapid transformation. It also adapts to those with a mobile or flexible workforce, as it can be accessed from anywhere as long as there is an internet connection.

Both implementations have advantages and disadvantages. In PIXEL, the development strategy is going to be flexible to adapt to different port needs. The following criteria have been identified to decide on the deployment strategy I each instance of the platform:

- *Financial capacity*. The ability of the port to invest in new resources (required infrastructure, software licenses, support)
- *Technical knowledge*. The capacity of the IT team or port stakeholders to maintain and/or evolve the systems.
- *Customizations needed*. Possibility to customize my application according to my business.
- *Integrations expected*. Possibility to integrate our system with other solutions.

There would be a third way that would be **Hybrid**. Companies do not have to choose just one of the above options to satisfy 100% of their demands. The hybrid solution would be one in which an organization uses a combination of on-premises and cloud services. It can offer flexibility by allowing workloads to shift between the two when capacity and costs change. Workloads and confidential data can be hosted in the private cloud, with less critical workloads than the public cloud. If an enterprise has regulatory requirements for data handling and storage, this can be provided in the private cloud.

What deployment option is safer?

There is not an easy answer to this question, it depends in great measure of the security application and how they control their access. The reason is that the ports have different policies for their data protection, as we as very different strategies for integration or service sharing.

Next sections will bring more light on how the deployments in the ports will be.

[41] https:// www.telehouse.net/resources/blog/september-2018/on-premises-vs-cloud-vs-colocation
[42] https:// servicemuse.com/cloud-vs-on-premises-vs-hybrid/

6.6.1. Ports deployments

6.6.1.1. Server requirements

PIXEL platform is designed to be completely modular and to avoid coupling between components. As commented in section 4 and 6.1, one of the drivers of this modularity is the use of Docker. Although the use of containers allows for other server topologies (such as cluster-based topologies with container orchestration), at PIXEL the approach has been a classic one in which services (or modules) are separated into different nodes, and these nodes run on different servers. The migration to cluster architecture is simple and intuitive as long as all the services and component are already defined in containers as atomic units.

In PIXEL there are different deployment environments:

- Development: This environment is used for developing the different modules; it comprises a varying number of servers created for programming and unit testing specific purposes.
- Integration and Test. These environments keep the different software releases and are used to test the modules integration. This methodology allows the decoupling of the different development teams of the project, since the releases can be done separately among modules.
- Demo. This environment holds an online demo with the latest platform features. It contains some real data integrated facilitated from participating ports and some simulated or differed data for those services that still are not ready. This environment allows live-demonstrations in dissemination events, congresses or stakeholders interviewers.

These three environments are used for port-generic purposes and they have data, fixtures and configurations specific for their purposes. They are hosted in the cloud IaaS service FIWARE Labs, liaised by the partner ORANGE to be used free of charge.

In the pilots, however, the deployments will be a production environment and therefore, the configuration must ensure the expected level and quality of service, as well as provide appropriate concern division in case of troubleshooting.

Thus, a recommended and minimum set of resources has been defined for the port deployments:

Table 6: Recommended and minimum set of resource for port deployments

		Minimum			Recommended			Kind
		VCPU	RAM	Disk	VCPU	RAM	Disk	
DAL	Orion + Agents	4x	4GB	80GB	4x	8GB	80GB	APPLICATION
IH	IH + Kafka	4x	8GB	80GB	4x	8GB	80GB	APPLICATION
	Elastic	4x	8GB	160GB	8x	16GB	1TB	DATABASE
OT	Backend	8x	16GB	80GB	8x	8GB	80GB	WEB SERVER, APPLICATION
	Algorithms (1 per algorithm or model)				8x	8GB	80GB	APPLICATION
DN	Portal, dashboards and alerts				8x	16GB	80GB	WEB SERVER, APPLICATION
Security	Keyrock	4x	16GB	80GB	4x	8GB	40GB	APPLICATION
Pilot apps	Pilot specific applications				8x	8GB	80GB	APPLICATION

The server must have visibility among them, this is, network access and for configuration and management a VPN access is required.

PIXEL ambition is to have results capable to be extendible, generalizable and affordable for all-sized ports in Europe and the world. To achieve this, use cases and configuration must be considered, in order to address a high share of future requirements and constraints in different organisation. The benefit of having 5 ports organisations (4 ports + 1 logistics hub) in the project is that they can bring different policies and IT strategies that can enrich the vision of the ICT developing staff of the project. This is the case on solution deployment. Although in general all ports have their way to address the deployment of new IT applications, their strategies are completely different. The following sub-sections describe briefly how these deployments will be performed in each participant port.

6.6.1.1.1. Bordeaux

In Bordeaux, the port will choice between two solutions to host PIXEL virtual machines:

1. On an external datacenter (like VIGIEsip).
2. On their own datacenter.

This means that the port will create a set of virtual machines with the dimensioning provided and deploy the containerized solution on a datacenter.

6.6.1.1.2. Piraeus

In Piraeus, the servers will be deployed internally in port premises. They will rely on existing infrastructure at the port.

6.6.1.1.3. Monfalcone

In Monfalcone port + SDAG pilot, a configuration near to minimum will be initially deployed. This will let the project to test the platform in constrained conditions and assess that the planned performance is achieved. In this pilot, a CENTOS base distribution will be used for servers, instead of Ubuntu. As for the containerisation, this is not an issue for PIXEL.

In case of experiencing limitations in performance due to the lack of resources, the pilot partners (in particular INSIEL, who will be the partner providing the servers) will commit more resources to ensure the validity of the pilot and the achievement of the objectives.

6.6.1.1.4. Thessaloniki

The port of Thessaloniki will use a hybrid deployment, since some of the resources will be hosted at the port and some other will be provided as a cloud service. The cloud service will be FIWARE Labs (the same used for the development and integration purposes). This will allow the project to assess the performance of the solution in hybrid deployments.

7. Conclusion and future work

7.1. Conclusion

This document contains the second version of the PIXEL IoT platform architecture and design. While the first version, released in M12 (April 2019) contained an analysis of the prominent reference architectures, the description of the building modules and the sequence diagrams according to the requirements gathered, this document is more focused in the integration and data processing and exchange inside the platform.

The document revises the concept of reference architecture and functional architecture of the platform, complementing the information already provided in D6.1. Integration between the different architecture modules and the external systems is explained, as well as the conditions for deployment in ports, as an input for WP7.

The current design will be, together with D6.1 the foundations for the developments and integrations performed during the remaining tasks in WP6. This design shows a modular platform that fully covers the requirements gathered from the consortium, stakeholders and the Grant Agreement. The modules are designed to be independent and are integrated via APIs, which will be reported in D6.5. The four modules that complete the data extraction, processing and visualization process (data acquisition layer, PIXEL information hub, operational tools and dashboards and notifications) are complemented by a security layer that provides security by design to all the data and users in the platform, enabling advance rules and fine-grained authorization access to resources. The document also revises the main data models currently analysed, where some others are still under consideration for the different data sources.

Finally, a general vision of the deployment strategies envisaged a review of the current deployment arrangements and the specific particularities at the ports.

7.2. Future work

This document has been written based on 6 months of work (12 – 18) after the release of D6.1 and is the definitive version of the PIXEL architecture.

This second version also contains architectural details such as the standard deployment schemas proposed for the different ports, APIs of the different functional modules and changes that have occurred since the first version of this deliverable (D6.1).

Although this document is the closing result of the task T6.1, the technical work at work package level continue and, given the nature of the project, minor changes (refinements) could occur at platform level, once the pilots start and on-site validation is performed. If those changes exist and are significant, the document will be extended with an annex of changes by the time of finishing the pilots.

The implementations derived from the documents D6.1 and D6.2 will last until month 26 and will be reported in D6.4 and D6.5.

References

[1] IoT-A (Internet of Things Architecture), “Project Deliverable D1.1 – SOTA report on existing integration frameworks/architectures for WSN, RFID and other emerging IoT related Technologies”, 2011.

[2] I. Ishaq, D. Carels, G. K. Teklemariam, J. Hoebeke, F. Van den Abeele, E. De Poorter, I. Moerman and P. Demeester, "IETF standardization in the field of the internet of things (IoT): a survey," Journal of Sensor and Actuator Networks, vol. 2, pp. 235-87, June 2013, 2013.

[3] "The Internet of Things Reference Model", 2014, [online] Available:

http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf

[4] Perry Lea.Internet of things for architects. IoT Architecture and Core IoT Modules. Birmingham: Packt Publishing Ltd; 2018. p. 26-38.

[5]Fremantle, Paul. (2015). A Reference Architecture for the Internet of Things. 10.13140/RG.2.2.20158.89922.

[6] Krčo, Srdjan, Boris Pokrić, and Francois Carrez. "Designing IoT architecture (s): A European perspective." 2014 IEEE World Forum on Internet of Things (WF-IoT). IEEE, 2014.

[7] "IoT Security", [online] Available:

https://hellofuture.orange.com/app/uploads/2019/01/181220_OrangeTGI_LivreBlanc_Ill367_VA.pdf