

# PIXEL Information System architecture and design - Version 1

<b>Deliverable No.</b>	D.6.1	<b>Due Date</b>	03/05/2019
<b>Type</b>	Report	<b>Dissemination Level</b>	Public
<b>Version</b>	1.0	<b>Status</b>	Release 1
<b>Description</b>	This document, released in two version (preliminary architecture and final architecture), reports about all the analysis and design activities to precisely specify the implementation tasks.		
<b>Work Package</b>	WP6		

## Authors

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

## History

Date	Version	Change
26-Nov-2018	0.1	ToC and task assignments
01-March-2019	0.2	First draft
31-March-2019	0.3	Second draft
12-April-2019	0.4	Third/final draft for Internal review
02-May-2019	1.0	Final version for final PIC review and delivery

## Key Data

<b>Keywords</b>	Reference Architecture, IoT, Data Acquisition Layer, Information Hub, Operational Tools, Dashboard, Notifications, Security
<b>Lead Editor</b>	Jose A. Clemente P02, PRO
<b>Internal Reviewer(s)</b>	CREOCEAN, GPMB, UPV

## Abstract

The present document contains the first version of the architecture for the information systems and its initial design. It reports about all the analysis and design activities performed during the first 9 months of activities in the task 6.1 and related WP6 activities.

The document describes the initial 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 different reference architectures for the IoT available, describing them briefly and selecting those more relevant given the scope of the project. These three are RAMI and IIRA for their European and global relevance and IoT-A for its orientation to interoperability, a main requirement of PIXEL. Once selected them, PIXEL functional blocks are mapped to them.

After this, a chapter is presented reviewing the main global requirements and use cases for the IoT platform. This review is not deeply described since the documents D3.2, D3.3 and D3.4 are the main references, avoiding unnecessary repetition of information.

The next section (section 4) presents the global architecture of the solution and the functional blocks with its components. Initially 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 datasources and persist them in a single storing point. The solution proposed for this is the creation of IoT Agents adapted to the different datasources generic formats, giving a 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 it 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 Operational Tools or Dashboards) and externals (API). The IH's main components are a high-performance data broker and a NoSQL database, although it contains accessory components that supports 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 include a model engine, a predictive algorithms engine, a complex event processor and a 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 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.

All the previous sections are described through components diagrams and interaction diagrams.

The information model is described in a separate section. It is performed a general review of the information models for IoT architectures, describing its characteristics and proposing a couple of candidates to follow. This chapter also contains the analysis of the starting data sets in the different ports, specific for their different domains. Data is classified and described in this chapter and it is intended to be a reference for future versions, where mapping between original data and PIXEL Information Model will be detailed.

The document also describes how PIXEL will lead with deployment and scalability. After reviewing the differences between multi-instance and multi-tenancy and the cloud and on-premises deployment. The choice of being multi-instance and proposing different hybrid deployments is made and justified, based on the scenarios and use cases identified in other work packages. Testing and maintenance strategies are also discussed in these sections, detailing the principles of multi-environment deployment and testing activities, and proposing different solutions (Nagios, Pandora, Prometheus) for the maintenance of the platform. These will be tested for the next deliverable.

The next section (State of the art and technological choices) describes the state of the art for each functional block, arriving finally to some design decisions based on the experience and the requirements of the project. For the DAL, the FIWARE stack will be used; in the case of the IH, a stack based on the open source projects Elastic, Kafka and Zookeeper has been selected, for the OT Flink, Perseo or Spark are proposed; in DN, Kibana, Grafana or Apache Superset are analysed, as well as notification solutions as Elsalert; finally, in the security block, an architecture based on XACML and FIWARE is selected.

The final section, Component interfaces, gives an initial idea on how the different functional blocks will interface with the rest of the components, internally in the PIXEL platform. This section is expected to grow in the next version including also the user interfaces, as this version only reports about the programming interfaces. The programming interfaces described are commonly HTTP REST like.

The final chapter has the conclusions and future work proposed for the next iteration of the document, and the depending activities in WP6, WP4 and WP7.

## Statement of originality

This document contains material, which is the copyright of certain PIXEL consortium parties, and may not be reproduced or copied without permission. This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

The information contained in this document is the proprietary confidential information of the PIXEL consortium (including the Commission Services) and may not be disclosed except in accordance with the consortium agreement.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the project consortium as a whole nor a certain party of the consortium warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, and accepts no liability for loss or damage suffered by any person using this information.

The information in this document is subject to change without notice.

The content of this report reflects only the authors' view. The Innovation and Networks Executive Agency (INEA) is not responsible for any use that may be made of the information it contains.

# Table of contents

Table of contents .....	5
List of tables .....	8
List of figures .....	9
List of acronyms .....	11
1. About this document.....	13
1.1. Deliverable context .....	13
1.2. The rationale behind the structure.....	14
1.3. Version-specific notes.....	15
2. Reference Architecture (RA).....	16
2.1. State of the art .....	18
2.2. PIXEL RA.....	22
3. Relation with use cases and scenarios .....	26
3.1. Requirements and scenarios identified.....	26
3.2. How use cases are addressed by the architecture .....	28
4. Functional architecture .....	31
4.1. Global architecture.....	31
4.2. Components diagrams.....	32
4.2.1. PIXEL Data acquisition .....	32
4.2.2. PIXEL Information Hub .....	33
4.2.3. PIXEL Operational Tools .....	36
4.2.3.1. Model Engine .....	37
4.2.3.2. Predictive Algorithms Engine .....	38
4.2.3.3. Complex Event Processing (CEP).....	38
4.2.3.4. Database (DB) .....	39
4.2.4. PIXEL Integrated Dashboard and Notifications .....	39
4.2.5. PIXEL Security.....	41
4.3. Component interaction diagrams .....	42
4.3.1. PIXEL Data Acquisition.....	42
4.3.1.1. Data acquisition .....	42
4.3.2. PIXEL Information Hub .....	43
4.3.2.1. Data acquisition .....	44
4.3.2.2. Retrieve data from PIXEL Information Hub.....	44
4.3.3. PIXEL Operational Tools .....	45
4.3.3.1. New Model in OT.....	45
4.3.3.1. New Rule (KPI) in OT .....	46
4.3.4. PIXEL Integrated Dashboard and Notifications .....	47
4.3.4.1. New Rule.....	47
4.3.4.2. Get rules.....	48
4.3.5. PIXEL Security.....	49

4.3.5.1.	Access to an API protected with OAuth2.....	49
5.	Information model.....	50
5.1.	Ontological model.....	50
5.1.1.	Heterogeneity.....	51
5.1.2.	Data understanding by programs .....	53
5.1.3.	Dynamicity of the environment .....	54
5.1.4.	Open ontologies review .....	54
5.1.4.1.	LOV4IoT .....	54
5.1.4.2.	FIWARE Data Models .....	55
5.1.5.	PIXEL Initial Data Model.....	55
5.2.	Data examples .....	59
5.2.1.	GPMB .....	59
5.2.1.1.	Electrical consumption data.....	59
5.2.1.2.	Activity data .....	60
5.2.1.3.	Weather.....	61
5.2.1.4.	Pollutants concentration data.....	62
5.2.2.	THPA.....	63
5.2.2.1.	Environmental data.....	63
5.2.2.2.	Traffic data .....	65
5.2.2.3.	Fuel Consumption .....	66
5.2.2.4.	Port Operations related data .....	66
5.2.3.	SDAG/ASPM .....	67
5.2.3.1.	SDAG Access Control and Management System .....	67
5.2.4.	PPA .....	69
5.2.5.	External Data .....	69
5.2.5.1.	Automatic Identification System (AIS) data .....	70
5.2.5.2.	European Space Agency Copernicus satellite imagery .....	70
5.2.5.3.	Thessaloniki traffic data .....	71
6.	Deployment and scalability .....	71
6.1.	Environments and testing.....	72
6.2.	Scalability. Multi-Instance vs Multi-Tenant .....	73
6.2.1.	Multi-instance deployment of PIXEL Information Hub.....	75
6.3.	Maintenance .....	76
6.4.	Deployment architecture. Cloud vs On-premises vs Hybrid.....	77
7.	State of the art and technological choices .....	79
7.1.	PIXEL Data acquisition layer .....	79
7.2.	PIXEL Information Hub .....	80
7.2.1.	Information Hub architecture approaches.....	80
7.2.2.	Technological choices.....	82
7.2.2.1.	Data/message broker .....	82
7.2.2.2.	Distributed coordination system.....	83

7.2.2.3.	Storage.....	84
7.3.	PIXEL Operational Tools .....	88
7.3.1.	Operational Tools SotA .....	88
7.3.2.	Technological choices.....	89
7.4.	Dashboard and notifications.....	89
7.4.1.	Dashboard SotA.....	89
7.4.2.	Technological choices.....	90
7.5.	Security .....	97
7.5.1.	IoT Security .....	97
7.5.1.1.	Global aspect .....	97
7.5.1.2.	The PIXEL perimeter .....	98
7.5.2.	PIXEL Security .....	98
8.	Component interfaces .....	100
8.1.	Data Acquisition Layer .....	100
8.1.1.	Programming interface .....	100
8.2.	Information Hub.....	100
8.2.1.	Programming interface .....	100
8.3.	Operational Tools.....	101
8.3.1.	Programming interface .....	101
8.4.	Dashboard .....	101
8.4.1.	Programming interface .....	101
8.5.	Security .....	101
8.5.1.	Programming interface .....	101
9.	Conclusion and future work .....	103
9.1.	Conclusion .....	103
9.2.	Future work.....	103
	References .....	104

## List of tables

Table 1: Deliverable context .....	13
Table 2: Use cases and scenarios identified .....	26
Table 3: Critical requirements from an architectural point of view .....	27
Table 4: How use cases are addressed by the Architecture .....	29
Table 5: Pollutants concentration data at Bassens station (manually downloaded) .....	62
Table 6: Wind data - small sample extracted from the database using SELECT * FROM WindData .....	63
Table 7: Sample Data of the second wind sensor (METEO STN RAW DATA).....	63
Table 8: For container management we have the following summary table: .....	78
Table 9: Shows the most popular message broker systems in 2018.....	83
Table 10: Shows the most popular distributed coordination systems in 2018.....	83
Table 11: Shows the most popular RDBMS systems in 2018 according to the DB-Engines ranking .....	85
Table 12: Shows the most popular Key-value stores in 2018 according to the DB-Engines ranking.....	85
Table 13: Shows the most popular Wide column stores in 2018 according to the DB-Engines ranking .....	86
Table 14: Shows the most popular Document stores in 2018 according to the DB-Engines ranking.....	86
Table 15: Shows the most popular Graph databases in 2018 according to the DB-Engines ranking.....	86
Table 16: Shows the most popular Search engines in 2018 according to the DB-Engines ranking.....	87
Table 17: Shows the most popular time series database management systems in 2018 according to the DBEngines ranking .....	87



## List of figures

Figure 1: IoT Design Choices. The full spectrum of various levels of IoT architecture from the sensor to cloud and back.....	17
Figure 2: Conceptual Reference Architecture .....	18
Figure 3: IoT-A's Views and perspectives.....	19
Figure 4: WSO2's reference architecture.....	20
Figure 5: Functional view of IoT RA .....	20
Figure 6: IoT reference architecture proposed by CCSA .....	21
Figure 7: Different layers in IoT World Forum Reference Model .....	21
Figure 8: Reference Architectural Model Industry 4.0 (RAMI 4.0).....	22
Figure 9: PIXEL Reference Architecture .....	24
Figure 10: Equivalence from RAMI and IIRA architecture to PIXEL RA .....	24
Figure 11: IoT-A's functional model to PIXEL RA .....	25
Figure 12: PDCA cycle .....	29
Figure 13: PIXEL Use Cases alignment with Architecture phases .....	30
Figure 14: Process axis of RAMI architecture ( see Figure 8 for more information).....	31
Figure 15: Global architecture.....	31
Figure 16: Shows the component architecture of the system .....	32
Figure 17: shows the component architecture of the system.....	34
Figure 18: Functional overview of the Operational Tools.....	37
Figure 19: Real Time invocation of models through OT API.....	38
Figure 20: Periodic invocation of models through OT API .....	38
Figure 21: CEP configuration through the OT API.....	39
Figure 22: Shows the component architecture of the system .....	40
Figure 23: PIXEL Security mechanism.....	42
Figure 24: Process of data acquisition the Oauth2 security mechanism .....	43
Figure 25: Process of data acquisition using the Security mechanism provided by the source .....	43
Figure 26: Acquiring data from PIXEL Data Acquisition.....	44
Figure 27: Client app complex data request .....	44
Figure 28: Change instance node configuration .....	45
Figure 29: Create new model in OT .....	46
Figure 30: Create new rule in OT .....	47
Figure 31: Create new rule in ElastAlert .....	48
Figure 32: Get rules from ElastAlert .....	49
Figure 33: Process of accessing to an API protected by OAuth2.....	50
Figure 34: Example of IoT cross domain applications.....	53
Figure 35: Example of semantic elevation .....	54
Figure 36: PIXEL Initial Data Model.....	57
Figure 37: Sample of raw electrical consumption data based on GPMB's bills.....	60
Figure 38: GPMB sample data concerning ship statistics .....	60
Figure 39: Example of one GPMB's web-service ( <a href="http://bordeaux.vigiesip.eu/vigiesip-webservice-bordeaux/nonauthent/export_demandes">http://bordeaux.vigiesip.eu/vigiesip-webservice-bordeaux/nonauthent/export_demandes</a> ).....	61
Figure 40: Weather data access through opendatasoft.com.....	62
Figure 41: Pointers to noise sensors locations (extractions from a .kmz file).....	64
Figure 42: Spatial data (extraction from a .dfx file) .....	65
Figure 43: Sample data of air quality measurements in THPA .....	65
Figure 44: Example of traffic monitoring in THPA .....	66
Figure 45: Example of THPA monitoring system of fuel .....	66
Figure 46: Screenshots of FRETIS modules (indicative): GIS / yard planning, XML messages, gate control. ....	67
Figure 47: Screenshot from Statistics software .....	67
Figure 48: How SDAG & ASPM systems work with PIXEL platform .....	68
Figure 49: Data frequency and format of SDAG system .....	69
Figure 50 Multi-tenancy .....	73

Figure 51: Characteristics of multi-Tenant and multi-Instance.....	74
Figure 52 Differences between Multi-Tenant and Multi-Instance architecture .....	75
Figure 53: Example of deployment for PIXEL Information Hub .....	75
Figure 54: Nagios Logo.....	76
Figure 55: Prometheus Logo .....	76
Figure 56: Pandora FMS logo .....	76
Figure 57: Integration between PIXEL Information Hub, PIXEL Data Acquisition and different Data sources .....	79
Figure 58: Example of a Kafka cluster architecture .....	80
Figure 59: Example of a WSO2 ESB architecture .....	81
Figure 60: Azure portal .....	91
Figure 61: Azure Dashboard .....	91
Figure 62: IBM Watson IoT platform .....	92
Figure 63: MindSphere. IoT platform from Siemens .....	92
Figure 64: Altair SmartWorks suite .....	93
Figure 65: FIWARE Lab .....	93
Figure 66: Kibana Logo.....	94
Figure 67: Kibana Dashboard.....	94
Figure 68: Grafana Logo .....	94
Figure 69: Tableau Logo .....	94
Figure 70: Tableau Dashboard .....	95
Figure 71: Apache Superset Dashboard .....	95
Figure 72: Graphite Dashboard .....	95
Figure 73: Visualization of Grafana to see the state of the alerts .....	96
Figure 74: Device interacting with several environment / applications .....	97
Figure 75: FIWARE Security solution .....	99

## List of acronyms

Acronym	Explanation
<b>ACID</b>	Atomicity, Consistency, Isolation, Durability
<b>API</b>	Application Programming Interface
<b>BI</b>	Business Intelligence
<b>CEP</b>	Complex Event Processing
<b>CLI</b>	Command Line Interface
<b>CPU</b>	Central Processing Unit
<b>CQL</b>	Cassandra Query Language
<b>CCSA</b>	China Communications Standards Association
<b>CSV</b>	Comma Separated Values
<b>DB</b>	DataBase
<b>FP7</b>	Seventh Framework Programme
<b>GB</b>	GigaByte
<b>GIS</b>	Geographical Information System
<b>GPMB</b>	Grand Port Maritime de Bordeaux - Port of Bordeaux
<b>HA</b>	Highly-available
<b>HTTP</b>	Hypertext Transfer Protocol
<b>IIOT</b>	Industrial IoT
<b>IMO</b>	International Maritime Organization
<b>IoT</b>	Internet of Things
<b>IoT-A</b>	Internet of Things Architecture
<b>IP</b>	Internet Protocol
<b>IIRA</b>	Industrial Internet Reference Architecture
<b>ITU</b>	International Telecommunication Union
<b>JSON</b>	JavaScript Object Notation
<b>KMZ</b>	Keyhole Markup Language
<b>LDAP</b>	Lightweight Directory Access Protocol
<b>LAN</b>	Local Area Network
<b>LTS</b>	Long-Term Storage
<b>MQTT</b>	Message Queuing Telemetry Transport
<b>M2M</b>	machine-to-machine
<b>NGSI</b>	Next Generation Service Interfaces
<b>OT</b>	Operational Tools
<b>PDCA</b>	Plan – Do – Check – Act
<b>PEI</b>	Port Environmental Index
<b>PIXEL</b>	Port IoT for Environmental Leverage
<b>PM<sub>10</sub></b>	Particulate Matter 10µm
<b>PM<sub>2.5</sub></b>	Particulate Matter 2.5µm
<b>PPA</b>	Piraeus Port Authority
<b>KPI</b>	Key Performance Indicators
<b>RA</b>	Reference Architecture
<b>RAMI</b>	Reference Architectural Model Industry

<b>RDBMS</b>	Relational Database Management System
<b>RFID</b>	Radio Frequency Identification
<b>SMS</b>	Short Message Service
<b>SNS</b>	Simple Notification Service
<b>SOA</b>	Service-oriented architecture
<b>SQL</b>	Structured Query Language
<b>SSL</b>	Secure Sockets Layer
<b>STOMP</b>	Simple Text Oriented Messaging Protocol
<b>STS</b>	Short-Term Storage
<b>T&amp;L</b>	Teaching and Learning
<b>TLS</b>	Transport Layer Security
<b>THPA</b>	Thessaloniki Port Authority
<b>UCC</b>	User Call Center
<b>UI</b>	User Interface
<b>VMPS</b>	VLAN Membership Policy Server
<b>WP</b>	Work Package
<b>XACML</b>	eXtensible Access Control Markup Language
<b>XML</b>	eXtensible Markup Language

# 1. About this document

The scope of the deliverable is to present the architecture that will be used in PIXEL, as well as identifying and analysing the different high-level modules. Besides, the first technological choices are reported here. These technological choices have been identified considering the requirements and use cases described in other work deliverables of the project (D3.2 and D3.4 respectively).

This document also includes the selection of a reference architecture oriented to IoT and logistics to guide and support the first technical stages of PIXEL.

This document is the first of two iterations and reports about the first phase of technical architecture and design definition of the project. The second and final iteration (D6.2) will provide a complete description of the PIXEL architecture as well as final design decisions to build the PIXEL IoT and data analytics platform.

The 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
<b>Objectives</b>	<p><i><u>Objective 1:</u> Enable the IoT-based connection of port resources, transport agents and city sensor networks,</i></p> <p>This document sets up the reference architecture of the IoT and data analytics platform that will be the basis for the connection of port resources, transport agents and sensors. The ICT infrastructure is the cornerstone for the achievement of most of the objectives in PIXEL.</p>
	<p><i><u>Objective 2:</u> Achieve an automatic aggregation, homogenization and semantic annotation of multi-source heterogeneous data from different internal and external actors.</i></p> <p>The IoT platform which is initially described in this report gives a technological support to this objective. Aggregation of data is performed in the PIXEL information hub, homogenization is performed both at Data Acquisition Layer level as well as in the information hub, the semantic annotation is done at data acquisition level.</p>
	<p><i><u>Objective 3:</u> Develop an operational management dashboard to enable a quicker, more accurate and in-depth knowledge of port operations.</i></p> <p>The ICT architecture of PIXEL provides a functional block for the dashboard and data analytics. This document describes the architectural foundations and technological choices of the solution that will be implemented.</p>
	<p><i><u>Objective 4:</u> Model and simulate port operations processes for automated optimisation.</i></p> <p>The architecture includes a functional block called operational tools. These tools give high-level technological support for the configuration and execution of predictive algorithms models developed in WP4.</p>
	<p><i><u>Objective 5:</u> Develop predictive algorithms.</i></p>

	Similar as for the models, the operational tools will support the predictive algorithms developed in WP4, so that port operators are able to configure and execute them.
<b>Work plan</b>	This deliverable reports about the works 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.
<b>Milestones</b>	Direct contribution to MS7 (ICT solution developed, M26). Indirect contribution to MS5 (Predictive models/algorithms, M24).
<b>Deliverables</b>	<p>Detected inputs:</p> <ul style="list-style-type: none"> <li>• D3.2: D6.1 obviously considers all requirements listed in order to provide an initial supporting architecture.</li> <li>• 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.</li> <li>• D4.1 PIXEL Models v1: D6.1 includes the Operational Tools module responsible to import, configure and run models described in D4.1.</li> <li>• D4.3 Predictive Algorithms v1: includes the Operational Tools module responsible to import, configure and run predictive models described in D4.3.</li> </ul> <p>Detected outputs:</p> <ul style="list-style-type: none"> <li>• D6.2 PIXEL information system architecture and design v2: D6.1 will be subject to a new iteration in order to describe the final architecture.</li> <li>• D6.3/4 PIXEL Data Acquisition, Information Hub and Data Representation v1/2: Some functional blocks will be described in detail in separate documents and D6.1 sets the grounding layer for this.</li> <li>• D6.5 APIs and documentation for software extension: D6.2 allows identifying the interactions among the functional blocks and therefore defining the APIs.</li> </ul>
<b>Risks</b>	This deliverable helps mitigating the risks 6, 8, 14, 16, 17 and 18.

## 1.2. The rationale behind the structure

As a first step, this document introduces the need of a reference architecture (RA) and presents the PIXEL's architecture. This RA must be the roadmap to follow along the WP6.

This first part of the document also aims to describe the characteristics of the architecture adopted.

The second part of the document is devoted to the different high-level modules existing in PIXEL:

- **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).

For each module, we present a state of the art and the existing related tools available in the market to be adapted for PIXEL. This allows us to choose the best option in each case.

### 1.3. Version-specific notes

This document is the first version of the deliverable "**PIXEL Information System architecture and design**" (D6.1). This document must be exhaustive in the explanation of the different modules / components and in the potential technological options because it must be the basis on which development will be based.

This document provides a preliminary version of the architecture according to the works done between months 4 and 12 of the project. This version contains the foundation of the architecture and the initial technological choices.

The second and final version of the document (D6.2), with due date in month 18, will contain any modification performed in the architecture requested, thanks to the feedback of the first months of development. It will also contain the graphical interfaces design for all the components and more details for the deployment alternatives.

## 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<sup>1</sup> (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 a 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<sup>2</sup>:

- 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 control 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<sup>3</sup>.

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.

---

<sup>1</sup> [http://cdn.iotwf.com/resources/71/IoT\\_Reference\\_Model\\_White\\_Paper\\_June\\_4\\_2014.pdf](http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf)

<sup>2</sup>Paul Fremantle. A Reference Architecture for the Internet of Things [https://www.researchgate.net/publication/308647314\\_A\\_Reference\\_Architecture\\_for\\_the\\_Internet\\_of\\_Things](https://www.researchgate.net/publication/308647314_A_Reference_Architecture_for_the_Internet_of_Things)

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



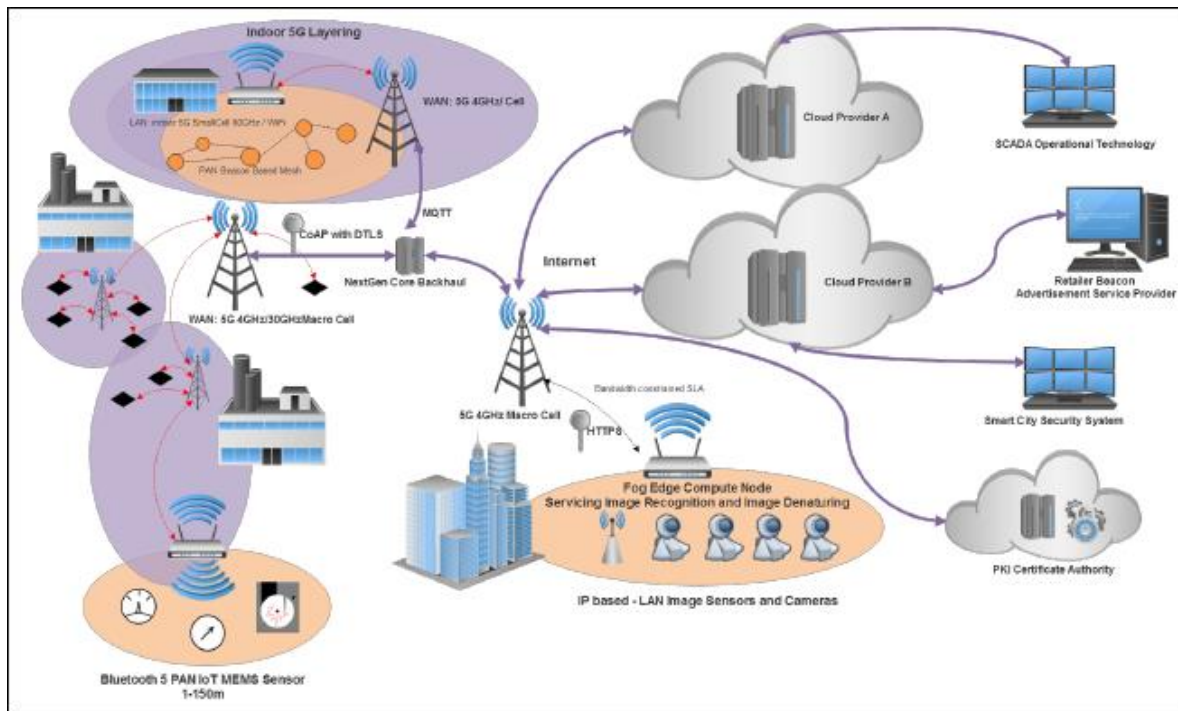


Figure 1: IoT Design Choices. The full spectrum of various levels of IoT architecture from the sensor to cloud and back

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<sup>4</sup>.

<sup>4</sup> [http://www.academia.edu/7197061/Estado\\_del\\_Arte\\_de\\_las\\_Arquitecturas\\_de\\_Internet\\_de\\_las\\_Cosas\\_IoT](http://www.academia.edu/7197061/Estado_del_Arte_de_las_Arquitecturas_de_Internet_de_las_Cosas_IoT)

## Conceptual Reference Architecture

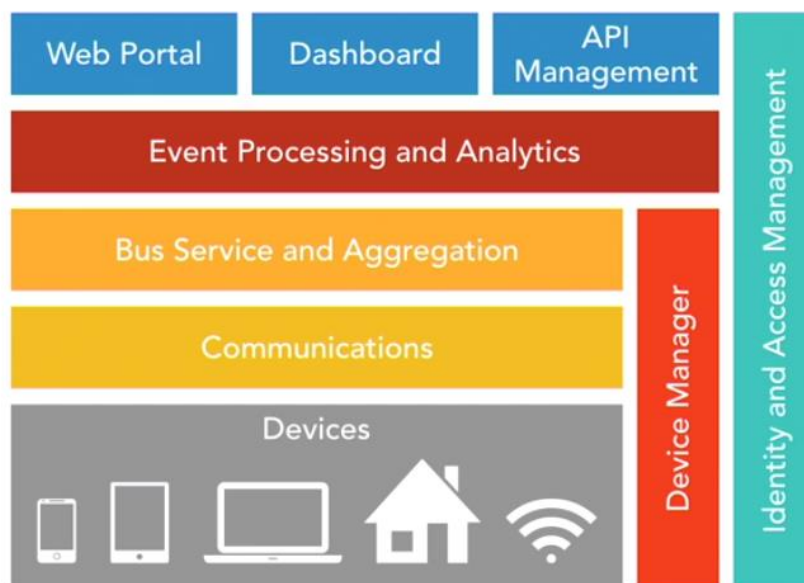


Figure 2: Conceptual Reference Architecture

### 2.1.State of the art

The rise of the IoT concept has led to an increase in the use of information and communication technologies. This also involves agreements on standardization as well as providing a new way of looking at society by interacting with a communications infrastructure person-to-machine or M2M (machine-to-machine) that will provide a new generation of services.

This entails the need to search for a valid reference IoT architecture that supports the different environments and contexts of the world of things; this has been a challenge since the very appearance of the IoT concept (Kevin Ashton, 2009). There have been a number of proposed architectures, many of them defined in specific contexts and providing solutions to a part of the "world of things". The concept, with a lot of related projects behind, that has most helped to spread IoT concepts or ideas in recent years are the Smart Cities.

IoT-A<sup>5</sup> was a lighthouse EU-funded project that established an Architectural Reference Model for the Internet of Things domain. The project ran from 2010 until 2013 and can be considered the foundation for all the EU efforts done in this area since then.

The IoT-A project developed common tools and methodologies to achieve a complete reference architecture for the existing and forthcoming IoT scenarios. It presents a series of characteristics that must be taken into consideration for the analysis of IoT architectures:

- **Architecture description:** Addresses the vision of architecture in relation to the project / product.
- **Model and distribution of information:** Addresses the problem of how information is treated and how it is distributed in the system (more information in section 6, Deployment and Scalability).
- **Horizontality:** Ability to reuse the same blocks to provide different functionalities of the top layer.
- **Context knowledge and semantic capabilities:** Refers to the possibility of improving the information exchanged through semantic translators.
- **Technology Specification and Interoperability:** How much of the project / product depends on a particular technology and how to focus on interoperability.
- **Adaptation:** Capacity offered by the project / product reactivity to environmental changes.

<sup>5</sup> <https://cordis.europa.eu/project/rcn/95713/factsheet/en>

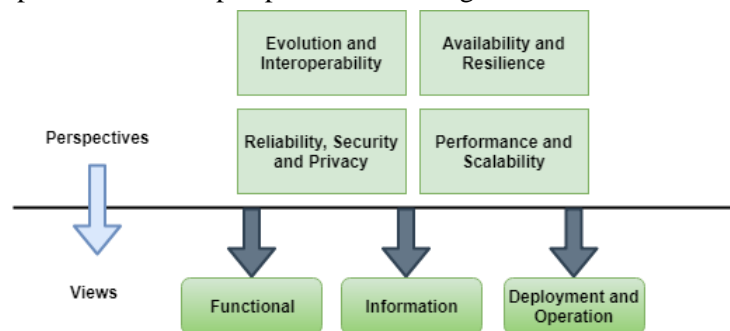
- **Programmability:** Defines APIs and specific standards for the development of an application.
- **Interface with the outside world:** Interfaces with which the end user interacts.
- **Work Plan:** Consideration of whether a product includes in its planning measures aimed at improving compatibility and / or development from the perspective of IoT.
- **Can the product interact with other articles?** Capacity or flexibility that a product presents when interacting with elements of the environment.
- **Obvious aspects of integration:** Evaluates whether a given product has taken into consideration basic aspects for integration into IoT architecture.

The IoT-A project has generated a series of results among which are the specifications of a reference architecture for the IoT (as defined in the previous section). This RA has provided 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.

Apart from research and private entities, as it will be shown below, standardization organisms have done work in this domain. ITU<sup>6</sup>, has proposed a comprehensive reference model in IoT environment. Their contribution has been the recommendation **ITU-T Y.2060**<sup>7</sup> that clarifies the concept and scope of the IoT. Identifying the fundamental characteristics and high-level requirements and describing the IoT reference model<sup>8</sup>.

Currently, there are few proposals with the intention of establishing RA in the domain of IoT. Among the existing ones, these four stand out:

- IoT-A Architectural Reference Model proposed by European Commission (FP7<sup>9</sup>). The proposed RA introduces the concepts of views and perspectives. Its design is as follows:



*Figure 3: IoT-A's Views and perspectives*

- **Views.** “Different angles for viewing an architecture that can be used when designing and implementing it”<sup>10</sup>. The views include: Functional view, Information view, Deployment and operation view as we can see in the figure above.
- **Perspectives.** “Set of tasks, tactics, directives, and architectural decisions for ensuring that a given concrete system accomplishes one or more quality attributes”<sup>11</sup>.
- IoT Reference Architecture developed by the **WSO2**<sup>12</sup> Company. Its proposal is based in its experience in the development of IoT solutions.

<sup>6</sup> <https://www.itu.int>

<sup>7</sup> <https://www.itu.int/rec/T-REC-Y.2060/en>

<sup>8</sup> <http://ijcsse.org/published/volume5/issue8/p1-V518.pdf>

<sup>9</sup> [https://ec.europa.eu/research/fp7/index\\_en.cfm](https://ec.europa.eu/research/fp7/index_en.cfm)

<sup>10</sup> <http://www.iot-a.eu/>

<sup>11</sup> <http://www.iot-a.eu/>

<sup>12</sup> <https://wso2.com>

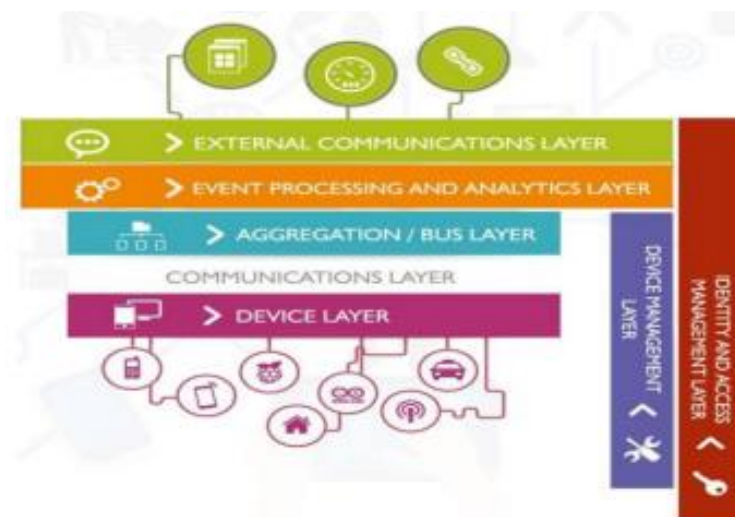


Figure 4: WSO2's reference architecture

- Korean IoT Reference Model. The Korean Study Group specification establishes a RA from communication and functional viewpoint.

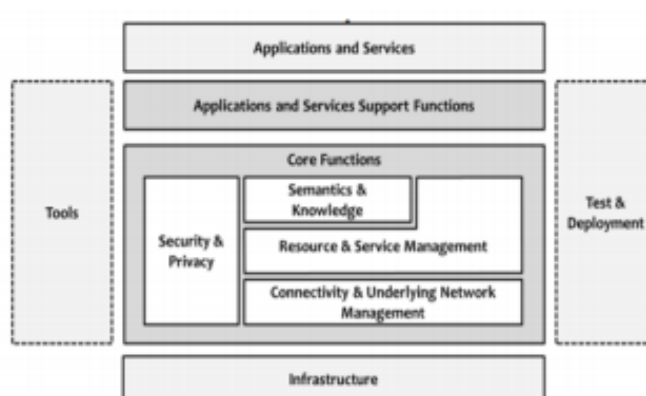


Figure 5: Functional view of IoT RA

- Chinese IoT Reference Model. CCSA<sup>13</sup> has proposed a RA which consists in several layers:
  - Sensing layer.
  - Network and business layer.
  - Application layer.

<sup>13</sup> <http://www.ccsa.org.cn/english/>

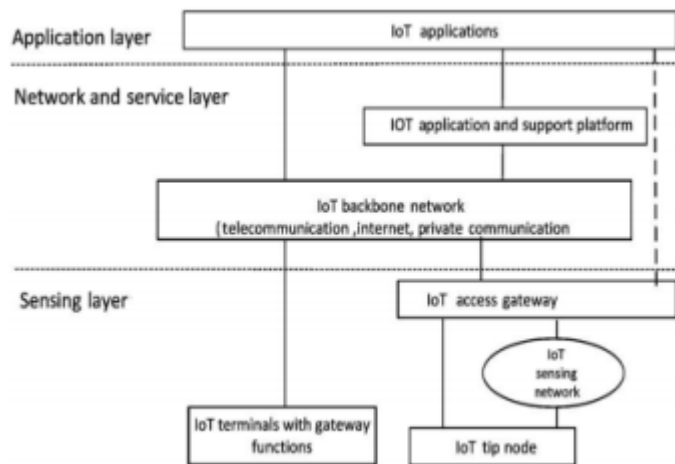


Figure 6: IoT reference architecture proposed by CCSA

In addition to these options there are other proposals that have emerged with the intention of establishing a RA for the Internet of Things, but from an industrial point of view (IIOT). However, none of them had a relevant acceptance. Among others, the following architectures for the IIOT have been considered:

- IoT World Forum's Seven Level Model<sup>14</sup>. The different layers are:



Figure 7: Different layers in IoT World Forum Reference Model

- Referenzarchitekturmodell Industrie 4.0 (**RAMI 4.0**)<sup>15</sup>. **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 a 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 industrie 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 modules adapting it to the trends market, stationality.

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:

<sup>14</sup> [https://www.researchgate.net/figure/IoT-World-Forum-Reference-Model\\_fig2\\_323525875](https://www.researchgate.net/figure/IoT-World-Forum-Reference-Model_fig2_323525875)

<sup>15</sup> [https://ec.europa.eu/futurium/en/system/files/ged/a2-schweichhart-reference\\_architectural\\_model\\_industrie\\_4.0\\_rami\\_4.0.pdf](https://ec.europa.eu/futurium/en/system/files/ged/a2-schweichhart-reference_architectural_model_industrie_4.0_rami_4.0.pdf)



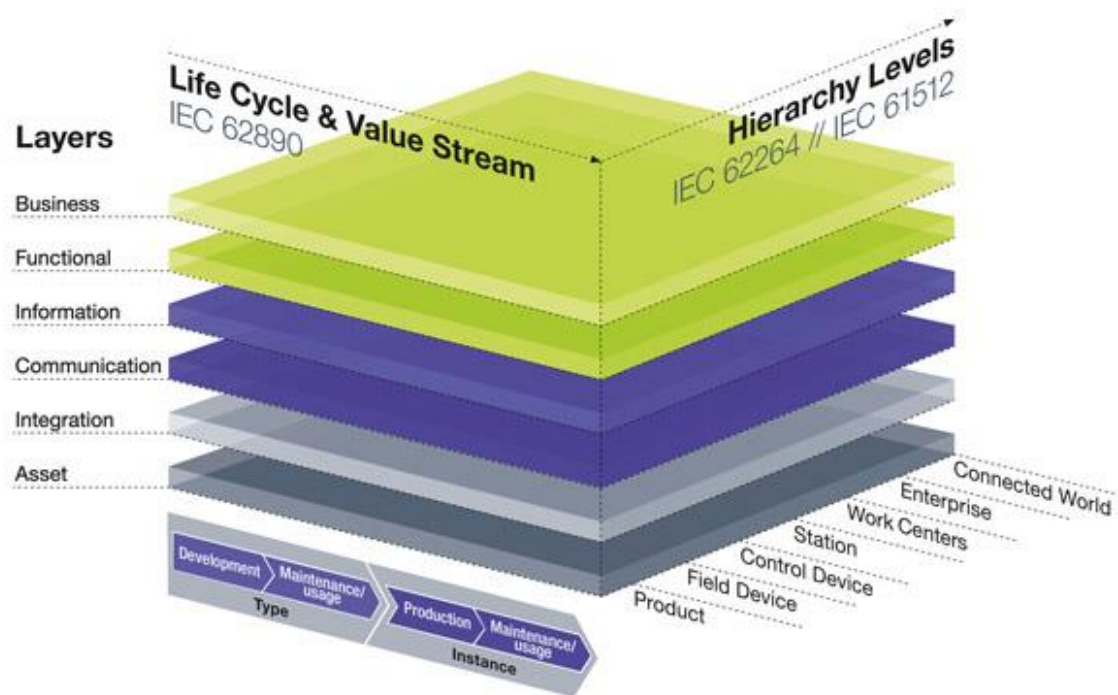


Figure 8: Reference Architectural Model Industry 4.0 (RAMI 4.0)

- Industrial Internet Reference Architecture (IIRA)<sup>16</sup>. 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).
- Reference Architecture Model for the Industrial Data Space<sup>17</sup>. This RA proposes a model for this specific capability including requirements for secure data exchange in business ecosystems. It has several layers: Business, Functional, Process, Information and System layer.

As we have seen, there is no a common one RA. There are several attempts to become a standard and great differences among them exist. Therefore, in the next section we will see which RA is proposed to PIXEL.

## 2.2.PIXEL RA

The aim of the PIXEL's RA is (as defined in the introduction of section 2, Reference Architecture (RA)), 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, we have a transversal layer (as in Conceptual Reference Architecture) that manages security and access. In the following list we describe the requirements needed to establish a PIXEL's RA:

- Connectivity and communications: These features are related with the capacity to connect the different modules to each other and how their interrelations are. It spans all the components.

<sup>16</sup> <https://www.iiconsortium.org/IIRA-1.7.htm>

<sup>17</sup> [https://www.fit.fraunhofer.de/content/dam/fit/en/documents/Industrial-Data-Space\\_Reference-Architecture-Model-2017.pdf](https://www.fit.fraunhofer.de/content/dam/fit/en/documents/Industrial-Data-Space_Reference-Architecture-Model-2017.pdf)

- 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**. While 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. In chapter 6 “Deployment and scalability” we give a definition of scalability.
- 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 Perficent guide, The Why, What and How of IoT: 50+ examples across 11 industries<sup>18</sup>, “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**.

Among the architectures seen the most relevant for PIXEL are: **RAMI**, **IIRA** and **IoT-A**. The reasons for this choice 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 an **European initiative** (RAMI) 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.
- **Experience** of the project partners with the methodologies and views used.

---

<sup>18</sup> [https:// www.perficent.com/-/media/files/guide-pdf-links/the-why-what-and-how-of-iot.pdf](https://www.perficent.com/-/media/files/guide-pdf-links/the-why-what-and-how-of-iot.pdf)

The following figure illustrates modularly the RA of PIXEL:

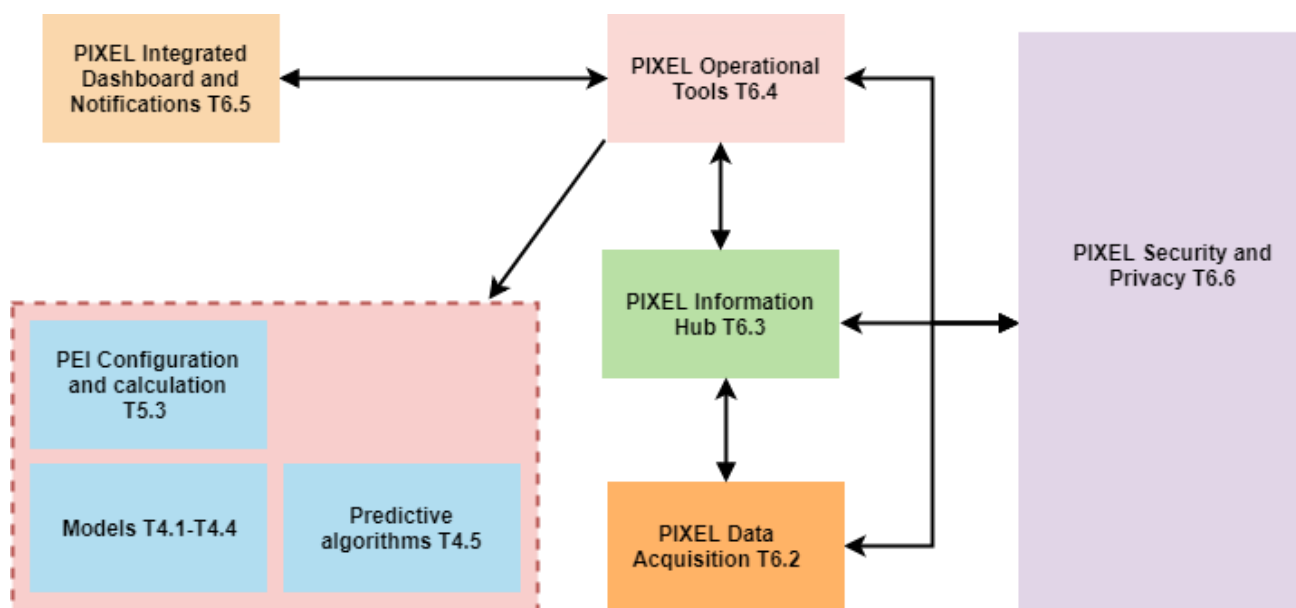


Figure 9: PIXEL Reference Architecture

In the following figure we show a mapping process from a RAMI architecture to an IIRA architecture and from these architecture to the PIXEL architecture.

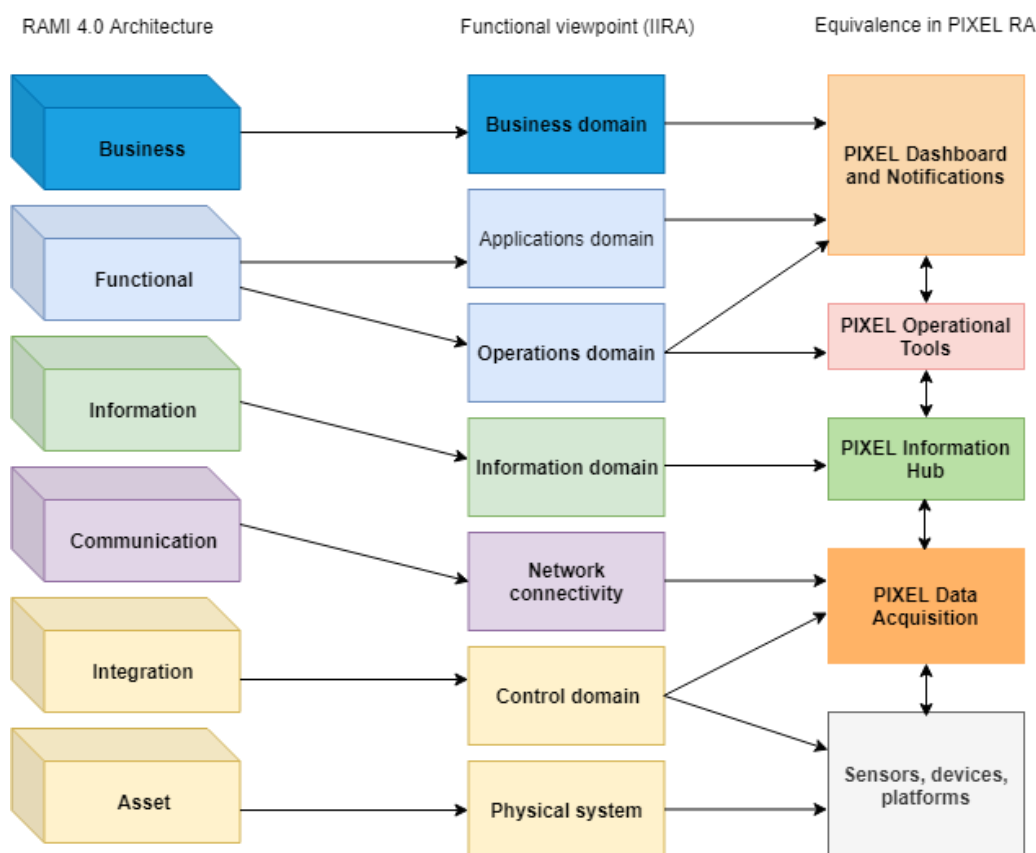


Figure 10: Equivalence from RAMI and IIRA architecture to PIXEL RA

The following figure illustrates the equivalence between the IoT-A functional model and PIXEL RA.



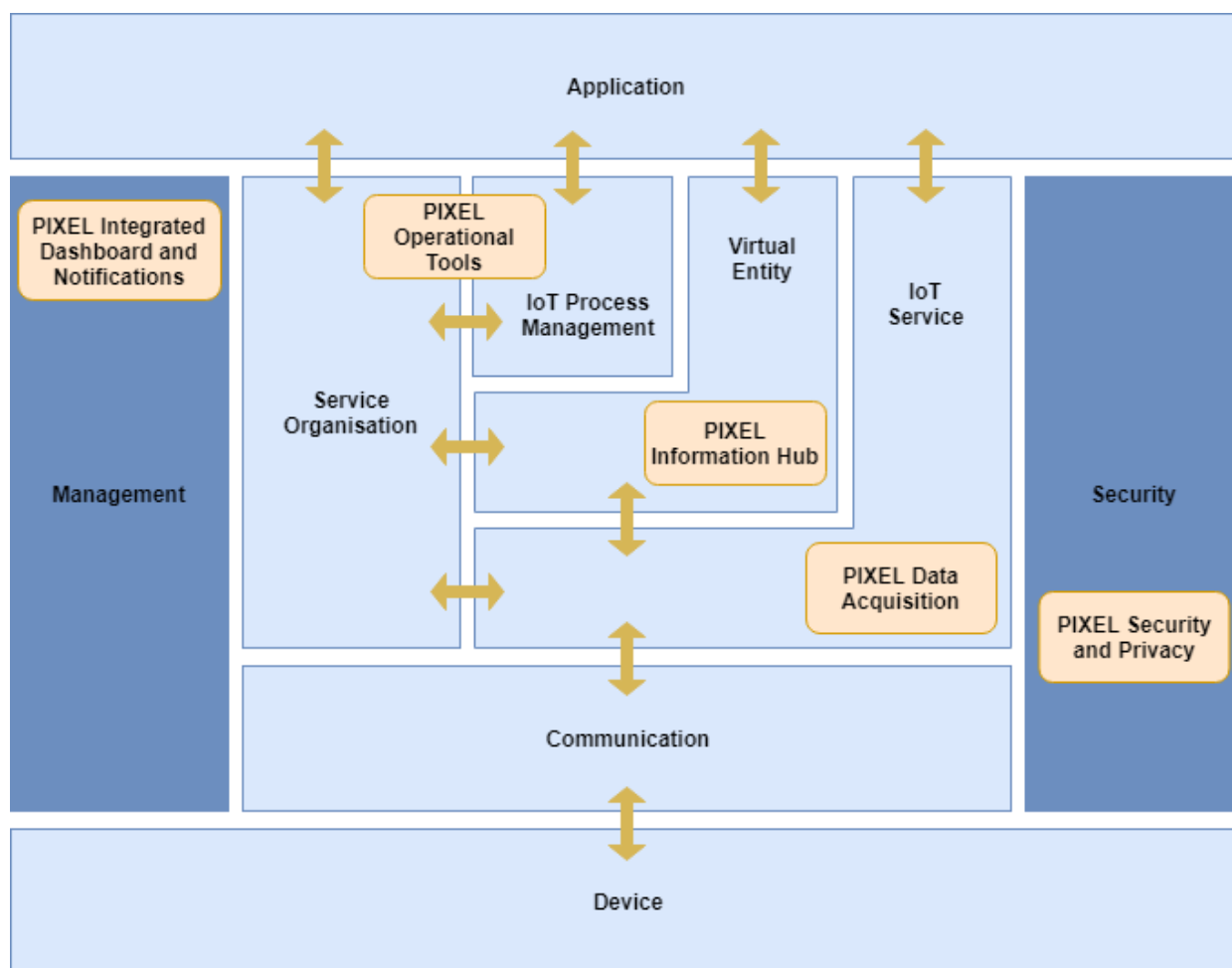


Figure 11: IoT-A's functional model to PIXEL RA

We will explain the different functions performed by each of the modules of the IoT-A's functional model to compare it with PIXEL as indicated in the image above:

- **Security.** Among their responsibilities are: Authorisation, authentication, identity management. So, there is a correspondence with PIXEL Security.
- **Management.** It is responsible of: Configuration, State, Reporting. Corresponds to PIXEL Integrated Dashboard and Notifications.
- **IoT Process Management.** Responsible for: Process Modeling and Process Execution.
- **Service Organization.** Responsible for: Service Orchestration.
- **IoT Service.** Module responsible for: Registration of devices, Historic data access, etc. Owing to the capabilities of the last three modules there is a correspondence to the PIXEL Operational Tools.
- **Virtual Entity.** It is responsible to handle the relations between the virtual entities (devices, sensors, etc) and domain entities. Provides the tools to access the entities. Corresponds to the PIXEL Information Hub.
- **Communication.** Corresponds to PIXEL Data Acquisition.

In the following chapters, the different modules of PIXEL's RA will be explained as well as their features and functionality.

## 3. Relation with use cases and scenarios

### 3.1. Requirements and scenarios identified

Requirements and scenario identification and analysis have been performed in detail during WP3 and described in D3.2 and D3.4, aimed respectively on requirements identification and use cases and scenarios definition.

Both deliverables are the result of an iterative process, which involved all different stakeholders of the PIXEL projects, in particular the ports that will adopt the output of the project. Use cases, scenarios and related requirements have been collected and validated in order to represent in an effective way the different ways each actor will interact with the PIXEL platform in order to fulfil a specific set of goals and expectations. Moreover, by means of use cases and requirements, deliverables D3.2 and D3.4 focus, for each port, on available sensors, data and information systems. Information provided by such deliverables must be carefully considered in order to properly design an architecture able to cope with integrability and extendibility.

D3.4 is the second version of the specifications of the use cases; it extends and completes D3.3 by providing a fully functional specification of the use cases with a common structure. In particular D3.4 describes in a detailed way the four use cases of the PIXEL project and their respective scenarios (several scenarios may concur to define a larger and more complex use case). In particular, the following scenarios mapped with the user stories have been identified:

*Table 2: Use cases and scenarios identified*

Partner	User story	Scenario
<b>GPMB</b>	<b>Statistics Manager</b>	GPMB-StM-1
	<b>Energy Manager</b>	GPMB-EM-1
		GPMB-EM-2
	<b>IT Manager</b>	GPMB-IT-1
	<b>Environmental Manager</b>	GPMB-EnvM-1
	<b>Port Manager</b>	GPMB-PM-1
	<b>Software Editor</b>	GPMB-SE-1
	<b>Port Agent</b>	GPMB-PA-1
<b>ASPM</b>	<b>Gate/Access Manager</b>	PoM-GM-1
	<b>Environmental Manager</b>	PoM-EM-1
	<b>Software editor</b>	PoM-SE-1
<b>SDAG</b>	<b>Parking area manager</b>	SDAG-PM-1
		SDAG-PM-2
		SDAG-PM-3
<b>ThPA</b>	<b>Terminal Operator</b>	ThPA-TO-1
	<b>Environmental Manager</b>	ThPA-EM-1
		ThPA-EM-2
<b>PPA</b>	<b>Environmental Manager</b>	PPA-EM-1
		PPA-EM-2

Each use case targets special issues or goals and is particularized in each corresponding port, according to its context and environment; such goals relate to the topics of energy management (GMPB), intermodal

transportation (ASPM, SDAG) and port-city integration (ThPA, PPA). Moreover, as one of the pillars of the whole project, all partners are interested in understanding and monitoring the environmental impact of the daily port activities: such challenge is considered crucial and allows shaping the ports of the future, and is related to the PEI (Port Environmental Index).

D3.2, on the other hand, provides an effective overview of both functional and non-functional requirements needed in order to provide functionalities and properties expected by project's stakeholders. The identification, communication and management of the requirements within PIXEL project follow the VOLERE methodology<sup>19</sup>. VOLERE has been used by thousands of organizations around the world in order to discover, define, communicate and manage all the necessary requirements for any type of system development (e.g. software, hardware, commodities, services, organizational, etc.). By looking at non-functional requirements described in D3.2, some particularly critical requirements, from an architectural point of view, emerge:

*Table 3: Critical requirements from an architectural point of view*

<b>Compliance [38]</b>
PIXEL must respect all different compliances, laws and regulations concerning port facility management, goods transportation and GDPR. Information access must be limited only to authorized users and in accordance with their respective roles. Information concerning dangerous transportations and environmental risks need to be carefully managed in order to keep them private if required.
<b>Interoperability [59]</b>
PIXEL platform must be interoperable with existing platforms in the port. As there are virtually infinite possible platforms, PIXEL must ensure good interfaces to integrate the already existing ICT systems and be compatible with well-known communication standards and message formats.
<b>Scalability [60]</b>
Pixel should be scalable in terms of growth needs or the large amount of data you can work with in the future.
<b>Security communications between components [68]</b>
Due to the privacy and variety of data involved in the platform is very important security within the platform. PIXEL must ensure end-to-end security.
<b>Access Security [97]</b>
Access Security is the extent to which the system is safeguarded against deliberate and intrusive faults from internal and external sources.
<b>Availability [98]</b>
Availability is the degree to which users can depend on the system to be up (able to function) during “normal operating times.” PIXEL must grant availability, in order to allow data collection, model calculation and PEI evaluation to take place, by minimizing data loss and downtime.
<b>Integrity [99]</b>
Integrity is the degree to which the data maintained by the software system are accurate, authentic, and without corruption. PIXEL Hub must provide data integrity for each collected set of data, in order to improve trust perceived by stakeholders in: <ul style="list-style-type: none"> <li>• Model evaluation;</li> <li>• PEI calculation.</li> </ul>
<b>Portability [103]</b>

<sup>19</sup> “VOLERE Requirements: How to Get Started” <http://www.volere.co.uk/pdf%20files/VolereGettingStarted.pdf>

PIXEL components need to be generic enough to be easily deployable for (transferred to) any port. This should be supported through well-documented extension mechanisms (like plug-ins, generic interfaces, abstract classes and similar).

Architectural design must be exploited in order to cope with such requirements; in particular, according with the large and heterogeneous set of sensors and information system which characterizes each partner port, architectural design must focus on interoperability and portability. Such requirements, in fact, are critical for the final results of the PIXEL project and the exploitation activities. A generic solution can be more easily deployed and integrated, by allowing several ports worldwide to adopt it.

Several use cases defined in D3.4 and analysed in detail in D3.2 in terms of related requirements, provides valuable and critical functionalities to ports, deeply connected with safety and security (e.g.: ADR freight re-routing, city-port traffic management). In order to deal with such use cases in a proper way and effectively achieve partners' goals, PIXEL tools' architecture must address scalability and availability. From an architectural point of view solutions aimed at minimizing the risks related with the PIXEL tools must be exploited and properly integrated.

Functional requirements described in D3.2. provide, in a common way across the different use cases, several other suggestions which should be handled by the architectural design task. In particular:

- Need of tools aimed at collecting, validating, cleaning and archiving large sets of data coming from real-time sensors (in a typical IoT scenario) or from historical data (e.g., locally available databases, legacy systems, etc.);
- Need of modelling and forecasting tools, aimed at providing a significant added value to each use case according with the specific needs and constraints (e.g., efficient and effective energy management, etc.);
- Need of an effective notification tool, which can be used by the different modules of the solution as a unique (and multichannel) way to interact with the user and provide alerts, warning, messages and errors.

Need of a simple but, at the same time, rich way to present visually aggregated data (e.g.: KPI, PEI), by providing filtering and drill-down capabilities.

### 3.2. How use cases are addressed by the architecture

This is a management and monitoring task for all four PIXEL use cases. The concept is to provide the test-bench to the stakeholders, in order to be able to evaluate the PIXEL innovations with realistic data (in terms of content, but also in terms of using large data volumes, to test non-functional parameters of the system, such as performance, scalability, responsiveness, usability). The process in developing the use cases and their scenarios, will be agile, so that a constant interaction cycle of progress will deliver the results incrementally as services. To this end, the use cases will be following the Deming **Plan – Do – Check – Act** cycles (PDCA cycle, below picture). There will be a constant interplay between the use cases progress and the technology developments in PIXEL.

The following figure depicts the PDCA cycle commented above.

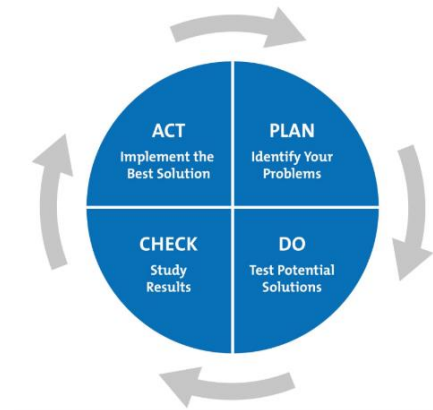


Figure 12: PDCA cycle

### PIXEL Architecture methodology and implementation approach through Use Cases

During the development of the PIXEL Platform Architecture, every use case will deploy a layered testing practice to drive development, involving significant industry representative tests, verifications and validations:

1. Using simulation, small models, or partial models, at the laboratory level, typically for all or parts of functionalities.
2. Tests using specific planning and optimization use cases.
3. Tests, measurements and validation of the use cases.

The PIXEL use cases have been presented in WP3, highlighting the actors, specific objectives, ambitions and benefits. PIXEL's agile and integrated approach allows the results learned from use cases to influence the development of the framework. This framework will be refined and redistributed for testing, keeping users in production cycles.

The design and implementation activities in PIXEL will be driven by the *Co-operative Inquiry Methodology*. This is a type of research action methodology focused on performing research with users. *Co-operative Inquiry* uses focused workshops that engage researchers and users ('co-researchers') in the development of a framework of inquiry and the exploration of their experiences and impressions of a research topic. The details of the three PIXEL project phases are listed in the following table.

Table 4: How use cases are addressed by the Architecture

Summary on how use cases are addressed by the architecture	
Objectives and Outcomes	Technical Approach
<b>Specifications &amp; Design Phase</b>	
<ul style="list-style-type: none"> <li>To establish requirements based on T&amp;L planning contexts analysis for the design of the PIXEL platform.</li> <li>To define collaborative Ports planning innovation use cases based on improved environmental models.</li> <li>To design environmental models for improvement of Port operations and T&amp;L planning contexts and a context driven method for orchestrating the planning.</li> <li>To create simulation-based investigation of collaborative planning and execution of T&amp;L operations.</li> </ul>	Phase 1 is concerned with gathering stakeholder requirements to produce the detailed low-level design of the collaborative T&L planning platform and its constituent components.

<ul style="list-style-type: none"> <li>To create the design specification for the planning platform and its constituent components.</li> </ul>	
<b>Development &amp; Integration of the Use Cases with the Architecture</b>	
<ul style="list-style-type: none"> <li>To develop the prototype PIXEL platform components including the relevant Publish/Subscribe APIs, interfaces and adaptors to enable the integration to existing planning systems as well as to external IoT devices and other data feeds.</li> <li>To develop an innovative set of core value-added services.</li> </ul>	Step 2 focuses on the development and integration of all platform components with external planning systems, presenting an environment with a basic functionality of all features/value-added services, in which user-specific cases will be tested and validated.
<b>Maturity Phase</b>	
<ul style="list-style-type: none"> <li>To deploy the PIXEL platform as a Service, test and validate it in a number of representative Use Cases.</li> <li>To adapt and refine the specification requirements, so as to enhance PIXEL platform functionality, and increase the impact in Use Cases, through a continuous cycle of planning, monitoring, assessment, evaluation and continuous learning.</li> <li>To focus on dissemination and exploitation activities to ensure the comprehensive and sustained impact of the project outputs.</li> </ul>	The final step will focus on broadening the validation and applicability of the PIXEL technologies, which will be further developed and enhanced based on the findings made in the previous step and on new end-user requirements in Use cases. The evaluation of the final system will further feed the dissemination and exploitation activities with the creation of research publications and validated industry relevant technologies.

The following figure shows the alignment between the PIXEL use cases and the phases of the architecture.

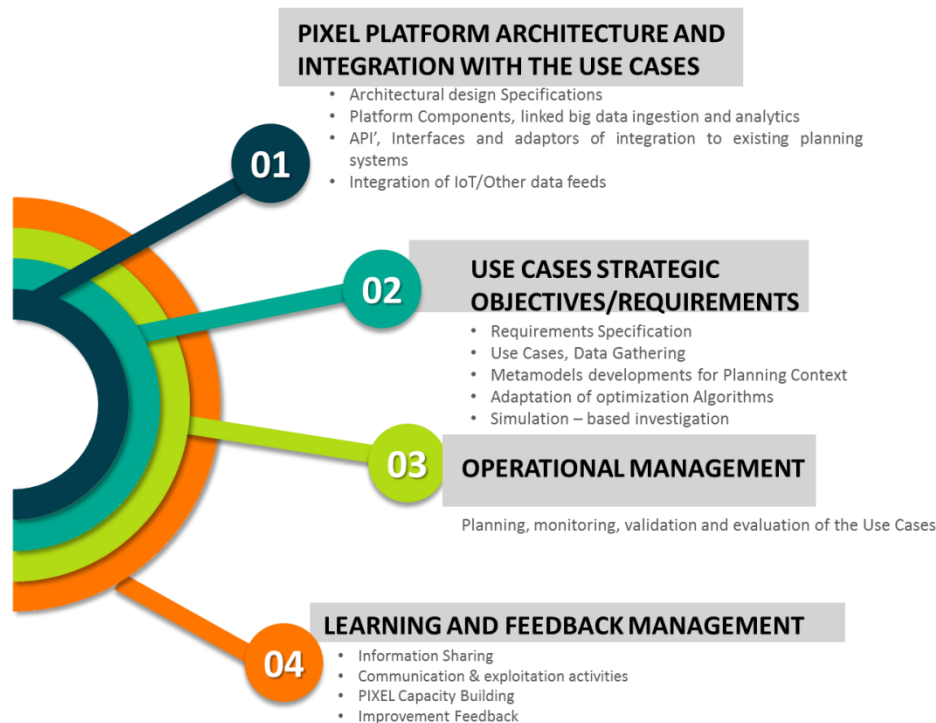


Figure 13: PIXEL Use Cases alignment with Architecture phases

These four stages can be compared with the Process axis that RAMI has. It divides the development / life cycle of the product in four stages which are the following:



Figure 14: Process axis of RAMI architecture ( see ¡Error! No se encuentra el origen de la referencia. for more information)

## 4. Functional architecture

### 4.1. Global architecture

In this section we will describe the architecture of PIXEL from a functional point of view. So, it's important to have a global vision of the architecture. The following figure shows the global architecture including the interaction with the different data sources and the output to the devices where we can work with PIXEL.

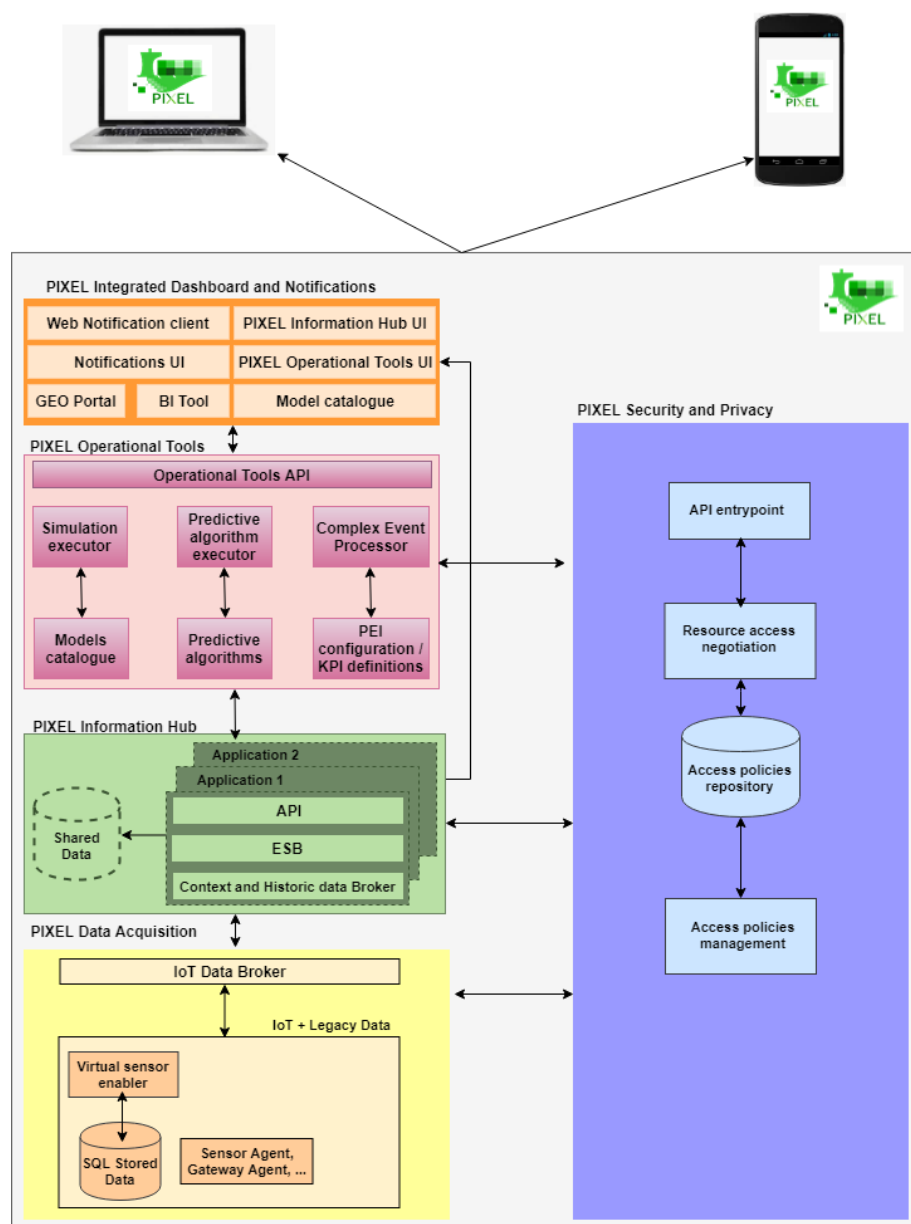


Figure 15: Global architecture



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 source that implements different kind of protocol and different data type.

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.

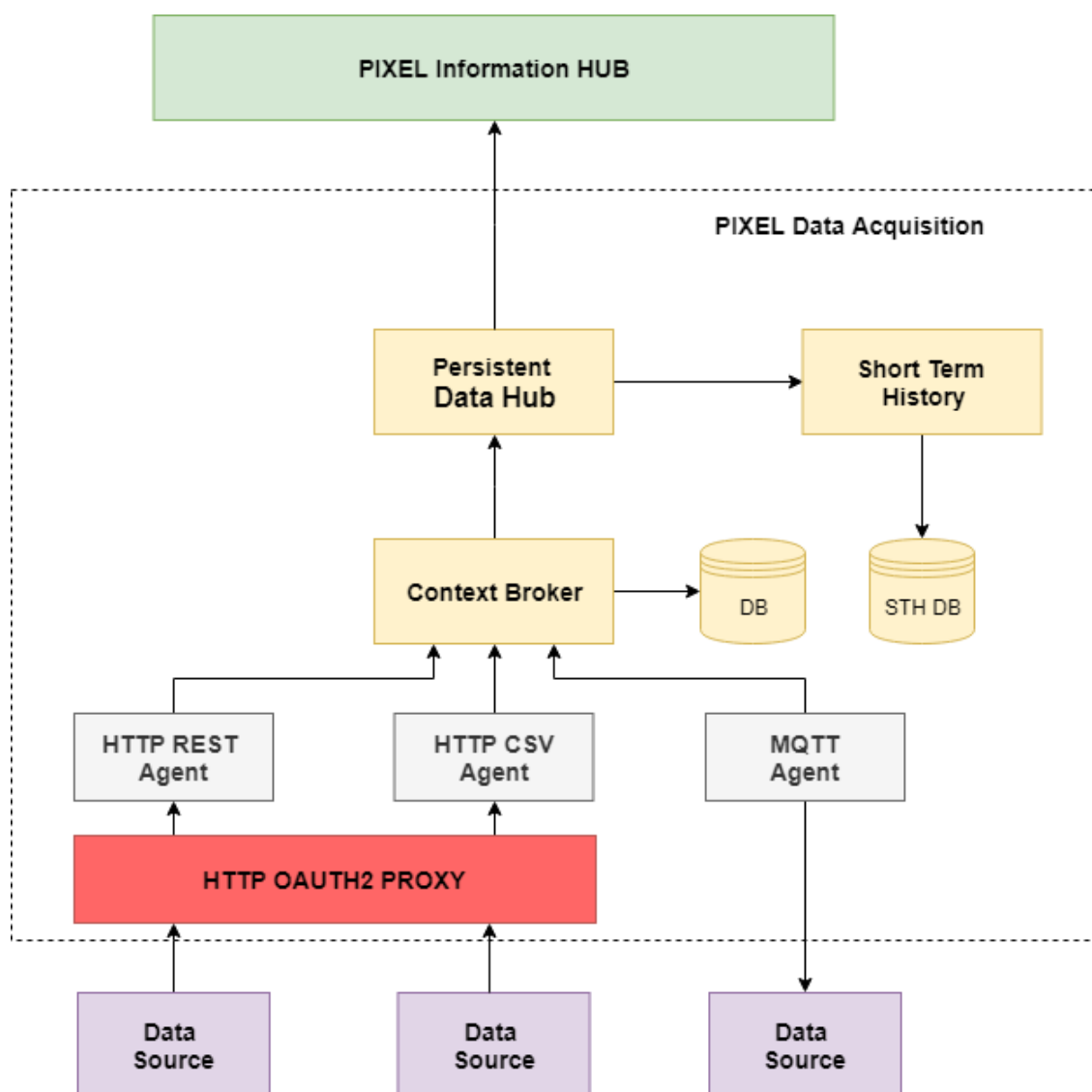


Figure 16: Shows the component architecture of the system

Component involved for the Data Acquisition:

- **Context Broker.** The context broker is responsible of the implementation of the standard data model used to import data on the PIXEL Information Hub. It collects the data from the data source through



specifics agents and pushes the new data to the Information Hub through the Persistent Data Hub component. It stored 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 in track the last imported data from the data source but also to trace the activity of each Agent. The imported data should be pruned often, the 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 we will provide several standard agents to address each kind of data source that could be used on 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)<sup>20</sup> 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. At the time of the writing of this deliverable, a technical and legal feasibility assessment is being carried out in order to identify if any of the components built for the FAIR Archiving System project could be adapted for use in PIXEL.

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.

---

<sup>20</sup> <https://fair-center.eu/>

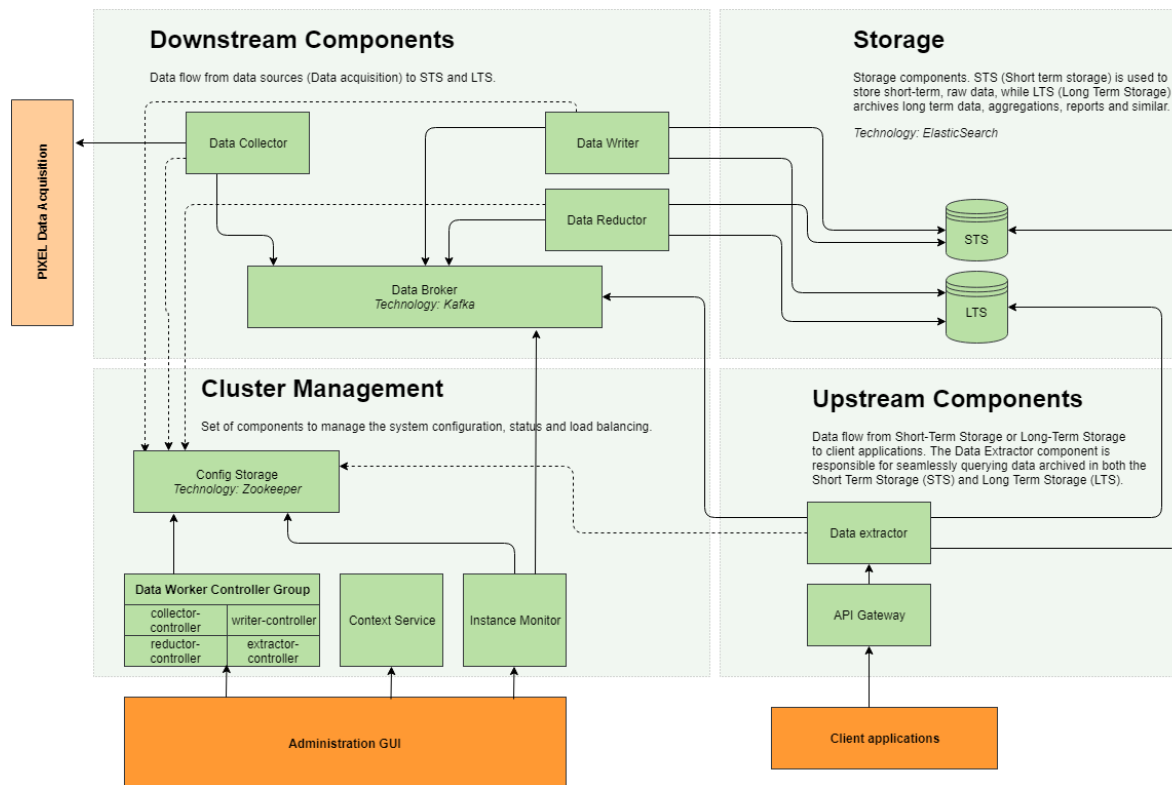


Figure 17: shows the component architecture of the system

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.
- **API Gateway.** The API Gateway takes all the requests from the client, determines which services are needed, and combines them into a synchronous experience for the user. It acts as a single point of entry for a defined group of APIs. It allows developers to encapsulate the internal structure of an application, depending on the use case. In addition to exposing APIs, popular API Gateway features include

authentication security policy enforcement, load balancing, cache management and more. Further assessment of requirements for an API Gateway implementation is needed. Depending on those requirements possible solutions are: a separate API Gateway for PIXEL Information Hub, or a shared API Gateway with the rest of the PIXEL platform.<sup>21 22</sup>

- **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.

---

<sup>21</sup> <https://whatis.techtarget.com/definition/API-gateway-application-programming-interface-gateway>

<sup>22</sup> <https://www.nginx.com/learn/api-gateway/>

- **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 need to be defined, so that:

- Port administrators are able to configure the models.
- Port administrators and other port entities are able to (easily) access to the results of the models.
- Internal components are smoothly integrated in the overall PIXEL architecture in a modular way, clearly identifying interactions with other architecture components.

Furthermore, the Operational Tools include a Complex Event Processor (CEP) able to monitor, at least, different aspects of the overall operations in ports:

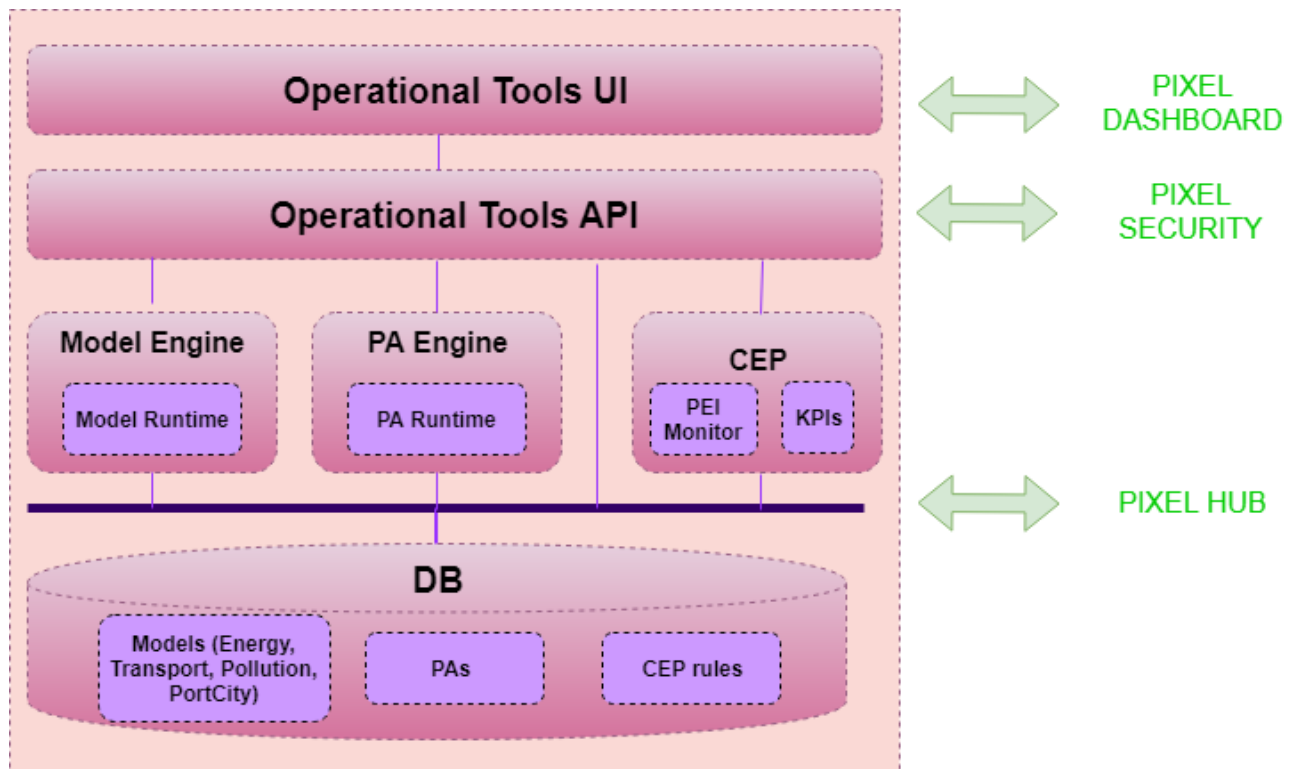
- Port Key Performance Indicators (KPIs): they refer to the ones described in deliverable D3.4, which are a particularization of the overall KPIs defined in the Grant Agreement (GA). Some of them are merely qualitative (e.g. local IoT platform implementation), but others are quantitative (e.g. number of sensors connected) and one can track or monitor the evolution throughout time.
- PEI monitoring: Even if the PEI is still under formal definition within the Consortium, it is envisioned that it will be an aggregated metric depending on environmental factors. Such environmental factors will be measured by means of one or more measurable indicators. By continuously monitoring such indicators, every port will be able to pre-emptively detect situations that will impact the final PEI and determine proper actions for mitigation.
- Specific alarms: Considering that an IoT platform is integrated in each port, some specific rules can be established in order to trigger alarms and actions whenever some threshold is reached or some complex condition is reached.

The functional overview of the Operational Tools is depicted in Figure 18. 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.).

- **Model Engine:** this component is responsible for executing the different models available and provides a result, either directly to the user at invocation time or storing the result in the internal database for later or real-time monitoring.
- **PA Engine:** analogous to the previous one, this component is responsible for executing the different predictive algorithms available and provide a result, either directly to the user at invocation time or storing the result in the internal database for later or real-time monitoring.
- **CEP:** this component is responsible for real-time monitoring of indicators and conditions and trigger specific actions depending on previously configured rules.

**Database:** the database includes description of the models and predictive algorithms that can be used, the CEP rules as well as other configuration and output related parameters necessary for the correct behaviour of the other building blocks. Both the models and the predictive algorithms should be previously imported from a catalogue available in the PIXEL catalogue.



*Figure 18: Functional overview of the Operational Tools*

In the following subsections a more detailed view of the functionality will be provided.

#### 4.2.3.1. Model Engine

The model engine should be able to execute models according to configuration parameters, which can be provided directly at runtime through the API, or obtained from the database for periodic executions.

The flow for direct invocations is depicted in Figure 19. An external component invokes a model passing the necessary input parameters. The model is executed by the Model Engine within an execution environment and the result is sent back to the invocation entity.

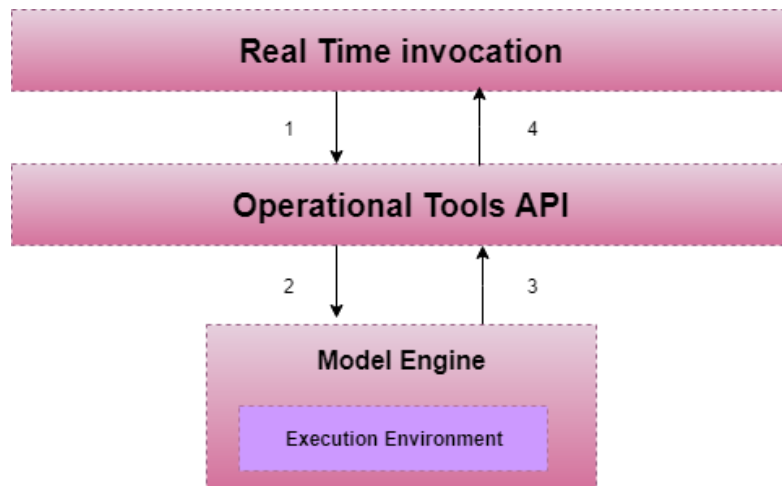


Figure 19: Real Time invocation of models through OT API

The flow for periodic invocations is depicted in Figure 20. For periodic execution of models, let's suppose that the port administrator has already configured in the database the corresponding model. Then, the Model Engine will periodically query the database for such models (1), execute them and store the result in the database (2). Whenever an invocation for such model occurs (3), there is no need to use the Model Engine and the information can be directly retrieved from the database (4) and offered to the client (5).

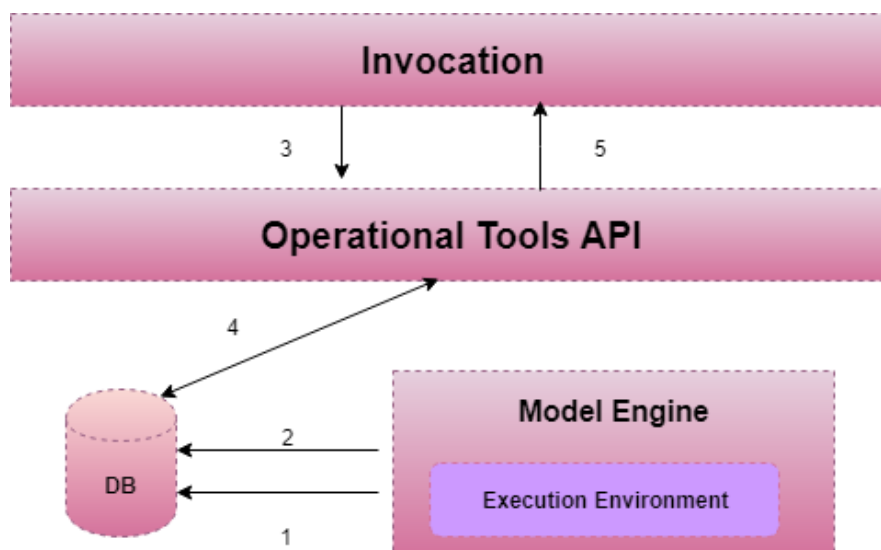


Figure 20: Periodic invocation of models through OT API

#### 4.2.3.2. Predictive Algorithms Engine

This component is analogous to the previous one but intended for algorithms instead of models. In fact, it is possible that both can be physically merged into one single component; however, conceptually it is better to conceive them differently.

#### 4.2.3.3. Complex Event Processing (CEP)

The Complex Event Processing overview is depicted in next figure. The flow is as follows:

1. The port administrator creates the rules (e.g. through the OT UI).
2. CEP rules are stored in the database.



3. The CEP accesses the database to get the rules.
4. The CEP monitors the corresponding variables according to the rules, which are typically data available in the PIXEL hub.
5. Whenever any rule condition is met, the CEP triggers the notification system, which may be user or another entity in the system. Additionally, the action can be stored in the database for keeping an historic record.

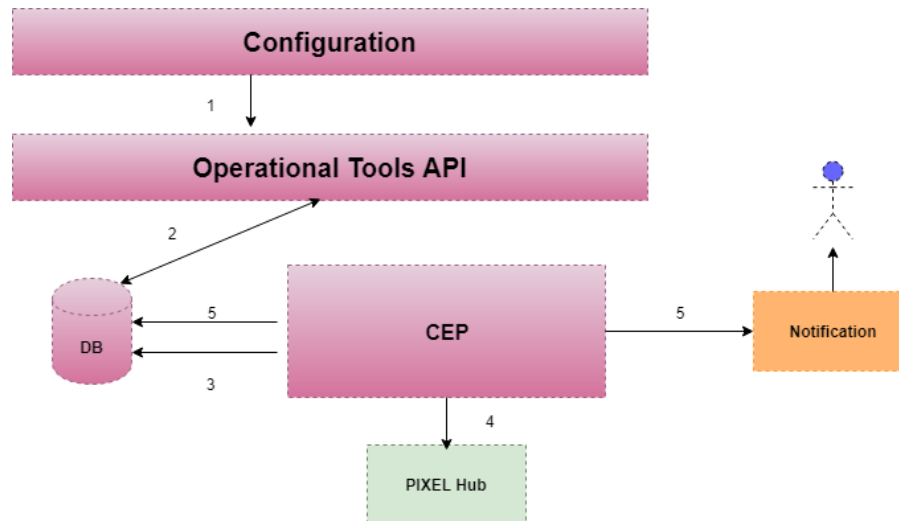


Figure 21: CEP configuration through the OT API

#### 4.2.3.4. Database (DB)

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

Some entities can be easily identified for the data model:

1. Model descriptions/catalogue: This is somehow similar to the catalogue, but more focussed on deployment issues rather than business issues.
2. Predictive algorithms description/catalogue: Similar as for models.
3. CEP rules: set of conditions that the CEP will be monitoring.
4. Execution instances with related parameters (configuration, status, result).
5. Event history: it is basically a table of historic events in order to keep track of them throughout time.
6. Alarms (actions): a set of action templates specific for ports that the CEP may triggered when some condition (CEP rule) is reached.

#### 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. These data are the result of:

- Apply **predictive algorithms and models**.
- Calculate the **PEI**.
- **Notifications from CEP**. Whenever a rule from CEP is met, we receive a notification from the Operational Tools module.

We also show data from:

- **Notifications:** Coming from executions of predefined rules/conditions.
- **GIS.** Geolocated data (sensors, devices, services, data result) can be visualized in a GIS.
- **Charts & Dashboard.** Visualization of the data received from the different data sources (devices, sensors).

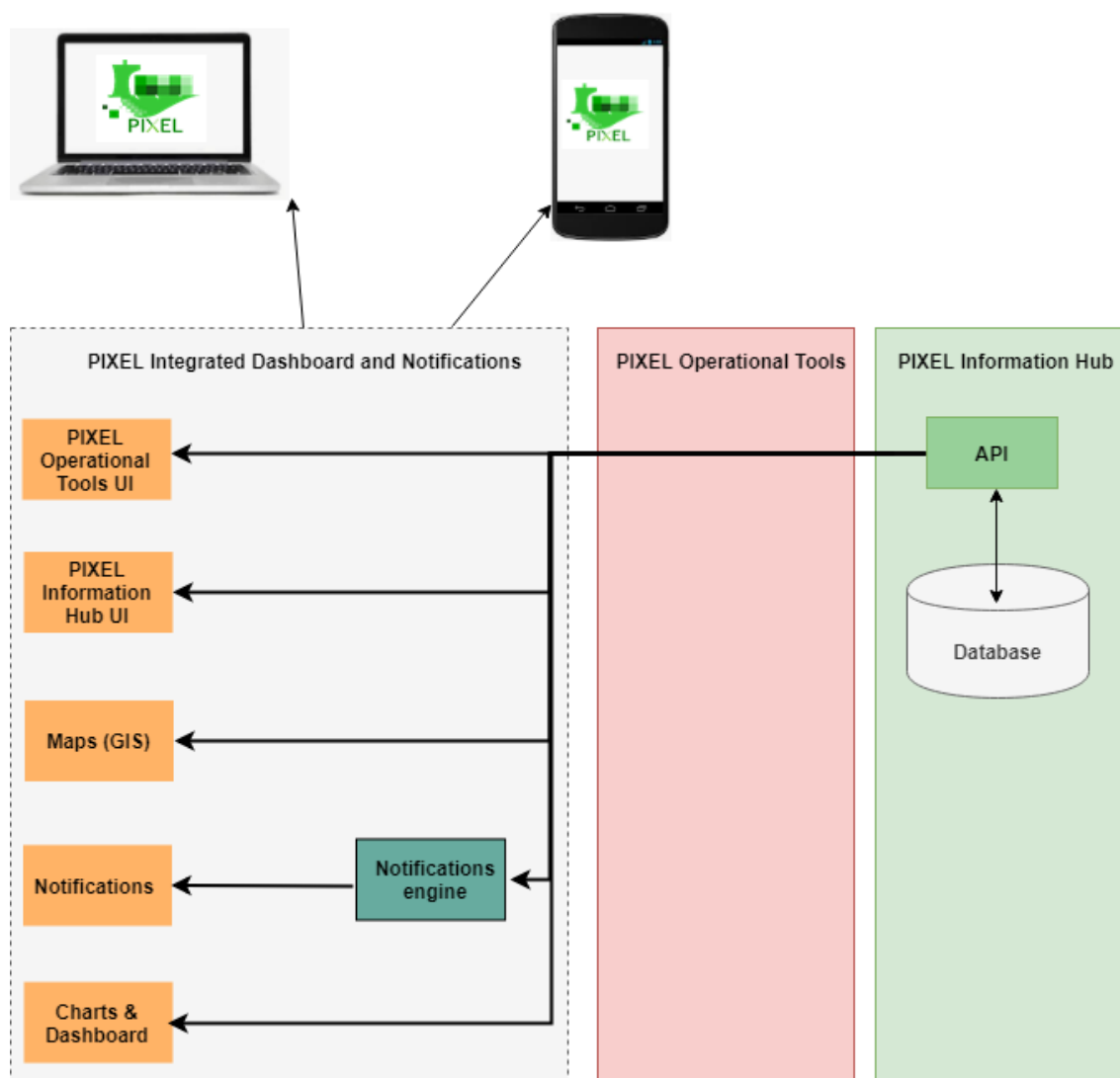


Figure 22: Shows the component architecture of the system

We can distinguish 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 will store 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. Among the supports



for the alerts that have the engines of notifications are the following ones: **Command, Email, JIRA<sup>23</sup>, SNS<sup>24</sup>, HipChat<sup>25</sup>, Slack<sup>26</sup>, Telegram, GoogleChat, STOMP<sup>27</sup>**.

2. **Outputs.** Visualization of data offered to the end user.

- a. **Charts & Dashboards.** Component responsible for the visualization of data in real-time. The users can create different kind of diagrams (bar, line or scatter plots).
- b. **Notifications.** Will be the result of a met rule. The UI of a notification can be an email, a ticket in JIRA or a conversation in Slack for example (the engine for the alerts supports different types of notifications as we have seen in the component **Notifications engine**). Also, we will visualize an advice of the notifications in the dashboard.
- c. **Maps.** Our 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.** In this interface we can visualize the results of execute the tools of PIXEL Operational Tools (PEI, predictive algorithms).
- e. **PIXEL Information Hub UI.** UI where we can see the data stored in the PIXEL Information Hub.

### 4.2.5. PIXEL Security

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

---

<sup>23</sup> <https://es.atlassian.com/software/jira>

<sup>24</sup> <https://aws.amazon.com/sns>

<sup>25</sup> <https://www.atlassian.com/software/hipchat/downloads>

<sup>26</sup> <https://slack.com>

<sup>27</sup> <https://stomp.github.io>

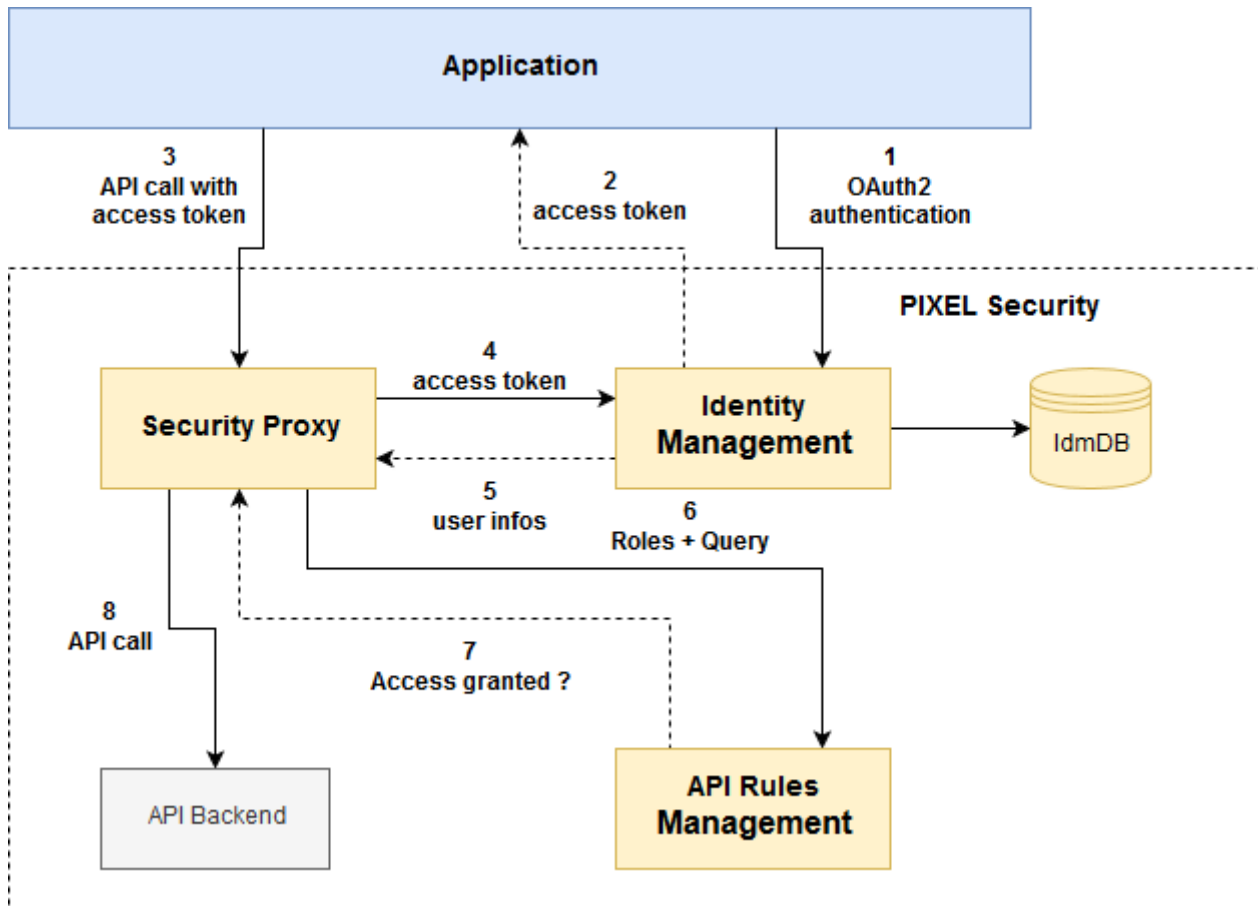


Figure 23: PIXEL Security mechanism

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.

## 4.3. Component interaction diagrams

### 4.3.1. PIXEL Data Acquisition

In this section we are going to show and describe two sequence diagrams in order to describe the process of data acquisition.

#### 4.3.1.1. Data acquisition

The diagram below illustrates the process of data acquisition. Data is pushed from the data source with the appropriate protocol, to the Data Collector of the Information Hub. The Agent exposed an API protected with the Oauth2 security mechanism.

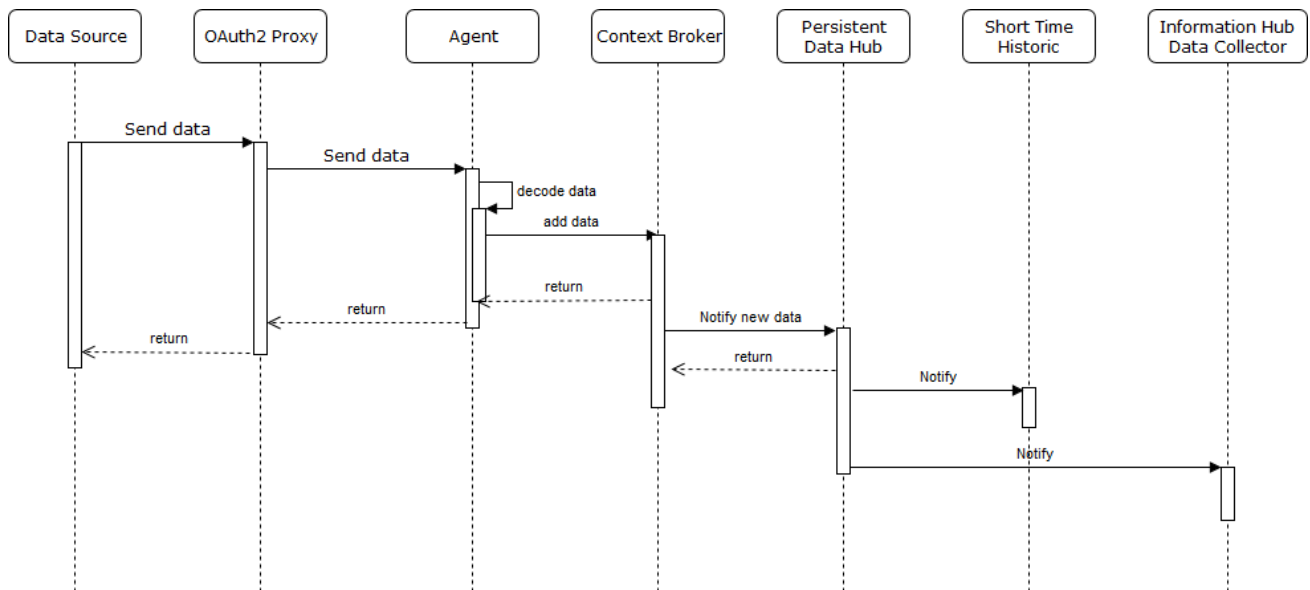


Figure 24: Process of data acquisition the Oauth2 security mechanism

The diagram below illustrates the process of data acquisition. Data is pulled from the data source with the appropriate protocol, to the Data Collector of the Information Hub. The Agent calls the data source using the security mechanisms provided by the source.

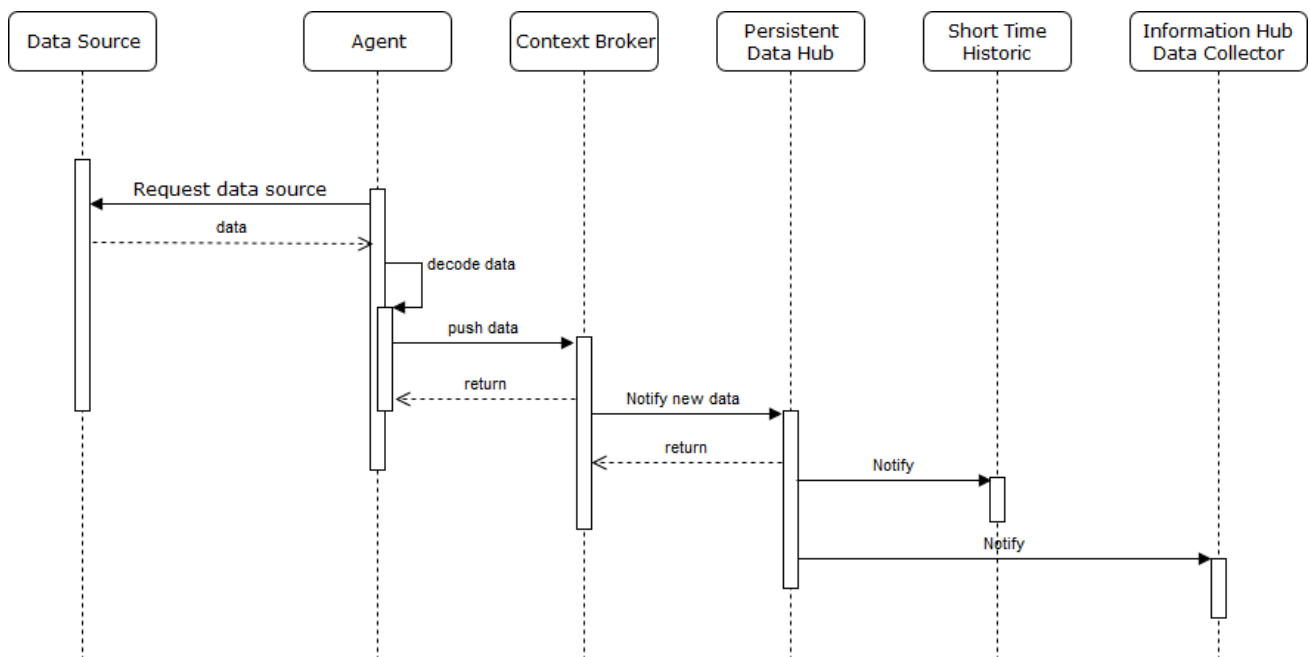


Figure 25: Process of data acquisition using the Security mechanism provided by the source

### 4.3.2. PIXEL Information Hub

In this section we are going to show two sequence diagrams related to processes that occur within this module:

- Data acquisition.
- The process of retrieving data from the PIXEL Information Hub.

### 4.3.2.1. Data acquisition

The diagram below illustrates the process of data acquisition. Data is delivered via a PUSH or PULL mechanism to the Data Collector. In this process, the data is saved in its raw form in the Short-Term Storage along, with the additional option of storing it also in the LTS.

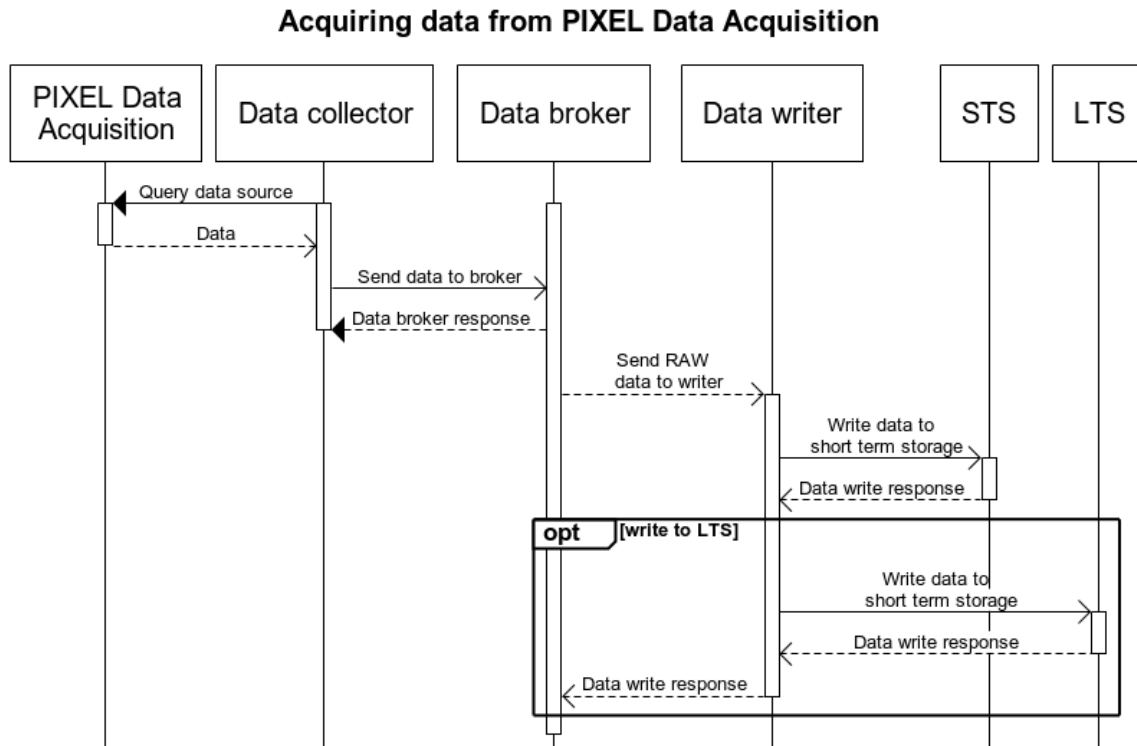


Figure 26: Acquiring data from PIXEL Data Acquisition

### 4.3.2.2. Retrieve data from PIXEL Information Hub

Diagram below illustrates the process of retrieving a set of complex data from the PIXEL Information Hub. Complex data in this case means, it is collected from multiple sources or from multiple different queries, combined and returned to the Client Application.

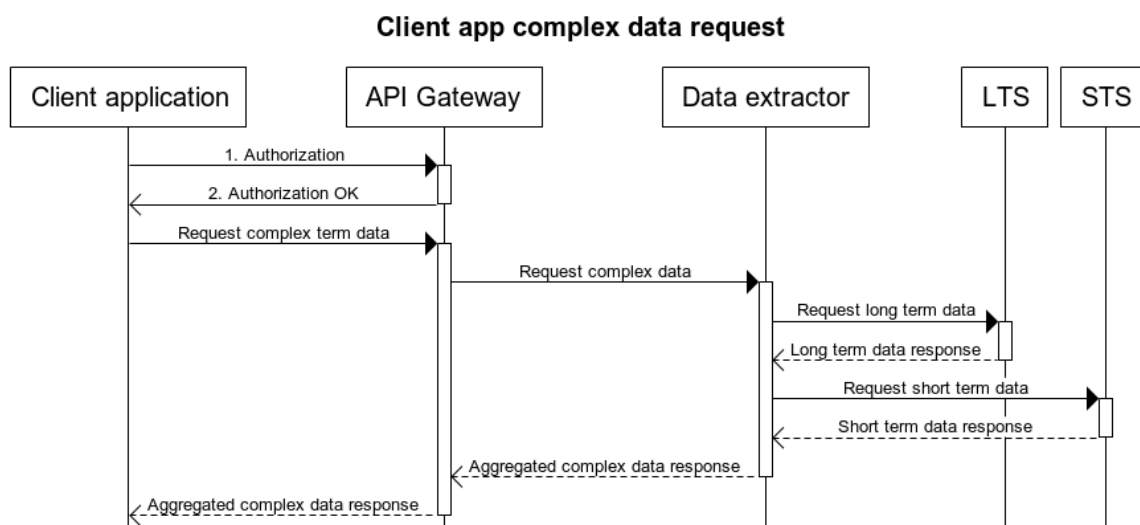


Figure 27: Client app complex data request

The diagram below illustrates a process of changing an instance node configuration. This could be changing the active services of an instance, how many threads the services use or other settings. In this process configuration

changes are applied to the Zookeeper node, which then through its “watch” mechanism propagates those changes to the corresponding nodes. All nodes are being watched by the Instance Monitor, which provides metric data for the Administration GUI as well as notifications in case of errors.

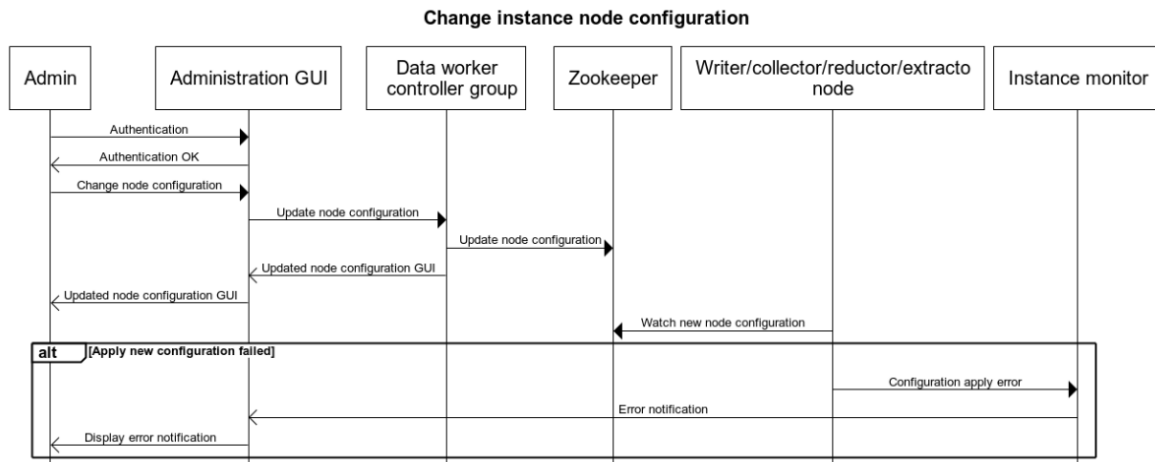


Figure 28: Change instance node configuration

### 4.3.3. PIXEL Operational Tools

In this section we are going to show and describe two sequence diagrams that serve as example of how a user (administrator) is able to make use of the Operational Tools:

- Deploy and run a new model in the OT. The example is analogous for the predictive algorithm.
- Configure a rule (e.g. related to a KPI) and let the CEP monitor it and trigger a notification if a threshold is reached.

#### 4.3.3.1. New Model in OT

The sequence diagram is depicted in the following figure:

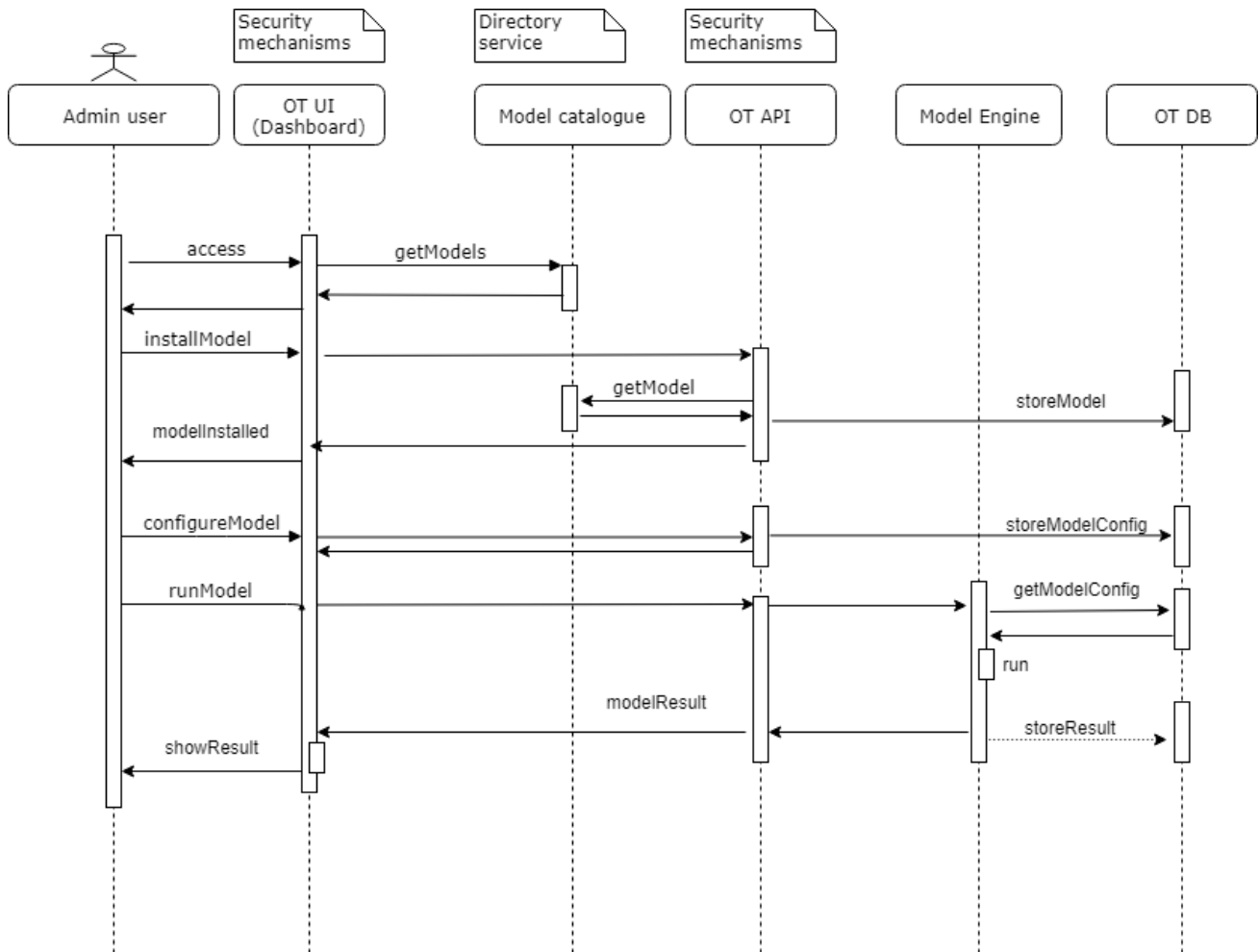


Figure 29: Create new model in OT

The flow is as follows:

1. The user accesses the Dashboard as entry management point. Once authenticated, it will switch to the OT user interface.
2. The model is already published in the PIXEL catalogue but needs to be incorporated into the OT environment; the first step is to search for that model in a directory service.
3. The user selects the model and presses the 'Install' button. This invokes the OT API which is in charge of (i) getting model information from the model catalogue, and (ii) storing the model in the OT internal database.
4. Once the model is installed, it is time to configure the model, according to the input data and input format described in the definition of the model. This information is entered or selected from the user and later on stored in the OT database.

After set up, the model can be executed. This can be launched from the OT user interface (if not scheduled), which will invoke the OT API. The API will take the input data stored in the OT database (if not given at launch time) and run the model. After that, the result (i) may be stored internally in the OT database and, (ii) will also provide the response to the user. The OT user interface should parse the obtained data (e.g. a JSON file) and show it in a graphical way. As this seems more a part of the dashboard (show graphical results), it is likely that the OU user interface invokes a dashboard template.

#### 4.3.3.1. New Rule (KPI) in OT

The sequence diagram is depicted in the following figure:

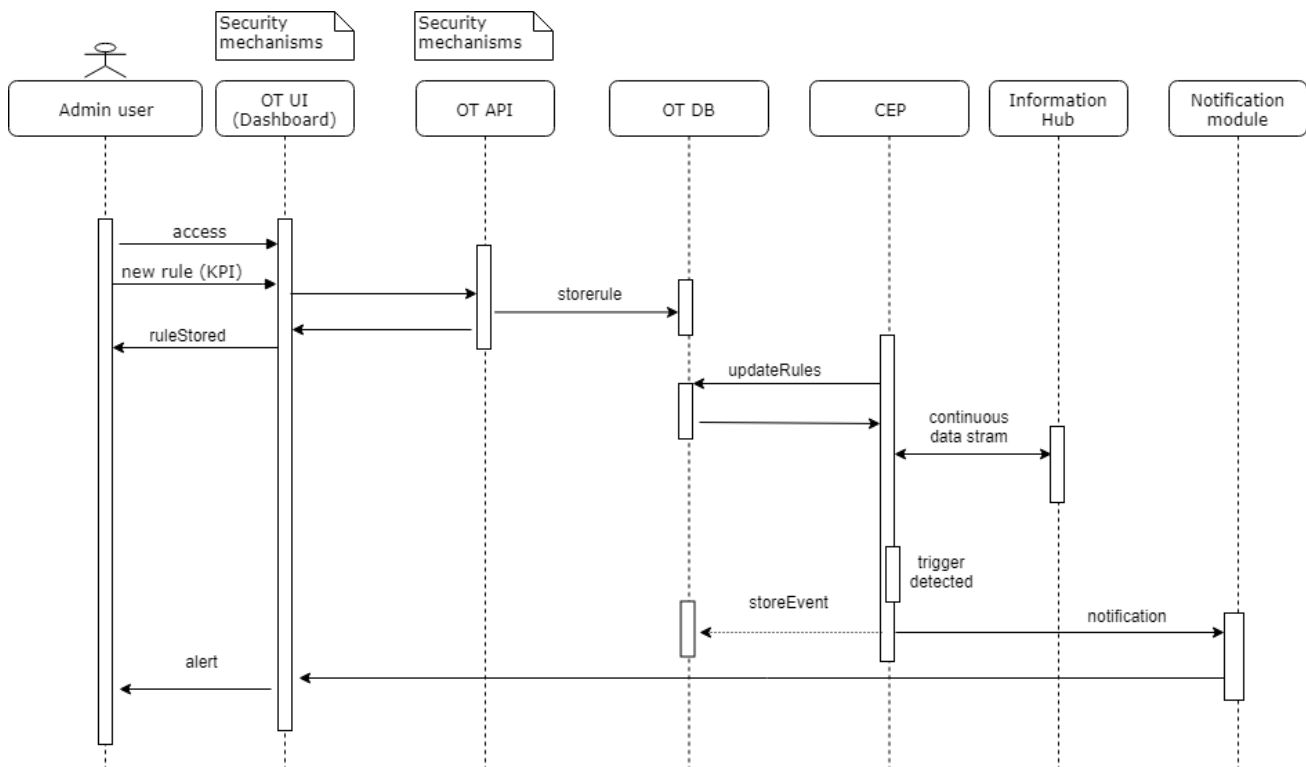


Figure 30: Create new rule in OT

The flow is as follows:

1. The user accesses the Dashboard as entry management point. Once authenticated, it will switch to the OT user interface.
2. The OT user interface will have a dedicated section for entering (CEP) rules that may potentially correspond to defined KPIs for a port.
3. Once the CEP rule is created by the administrator, it is stored in the OT database, with a confirmation message to the user. No further action from the user is needed.
4. The CEP is a continuous process that periodically updates the active rules by querying the OT database. Considering this rule, and any other one previously activated, the CEP obtains data streams from the PIXEL information hub.
5. At a certain rule, a rule is triggered, and a notification is sent to the notification module. Optionally, this event can also be stored in the OT database.

The notification module will send an alert to the user by some predefined configuration; typically, it can happen directly through the dashboard (e.g. pop-up message).

### 4.3.4. PIXEL Integrated Dashboard and Notifications

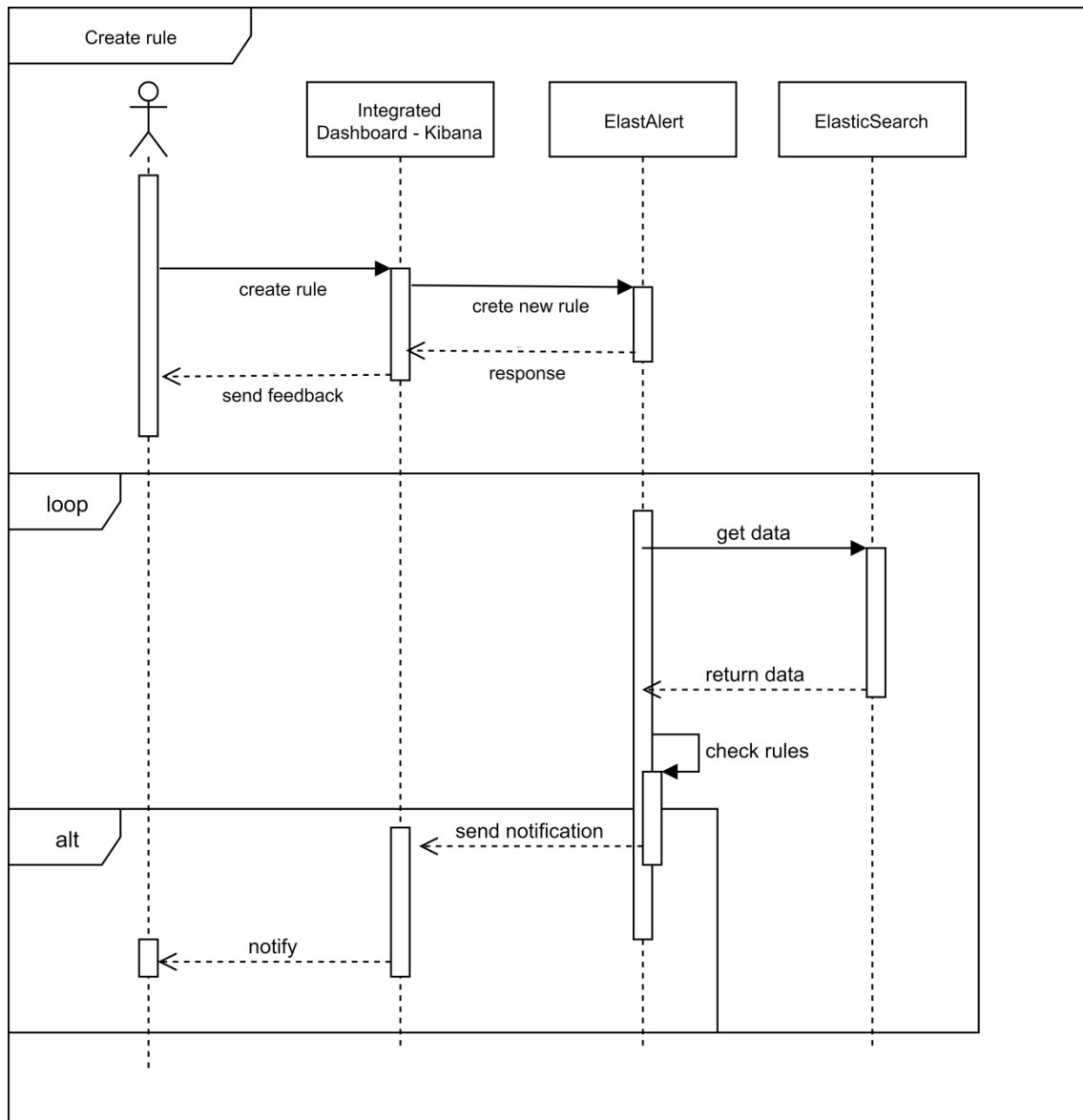
In this section we are going to see the diagrams of two use cases that can occur in this module:

- Create a new rule in Kibana and follow the validation process to see if trigger a notification.
- Recover the list of rules from Kibana.

#### 4.3.4.1. New Rule

The following figure illustrates the sequence of steps:





*Figure 31: Create new rule in ElastAlert*

1. The user interacts with Kibana to create a new rule.
2. The new rule is created in ElastAlert and Kibana receives a response.
3. The user receives feedback from Kibana indicating that the rule is created.
4. ElastAlert check the data until the rule is met.
5. Once time the rule is met elastAlert send a notification to Kibana.
6. The user receives a notification from Kibana.

#### 4.3.4.2. Get rules

The following figure illustrates the sequence of steps:

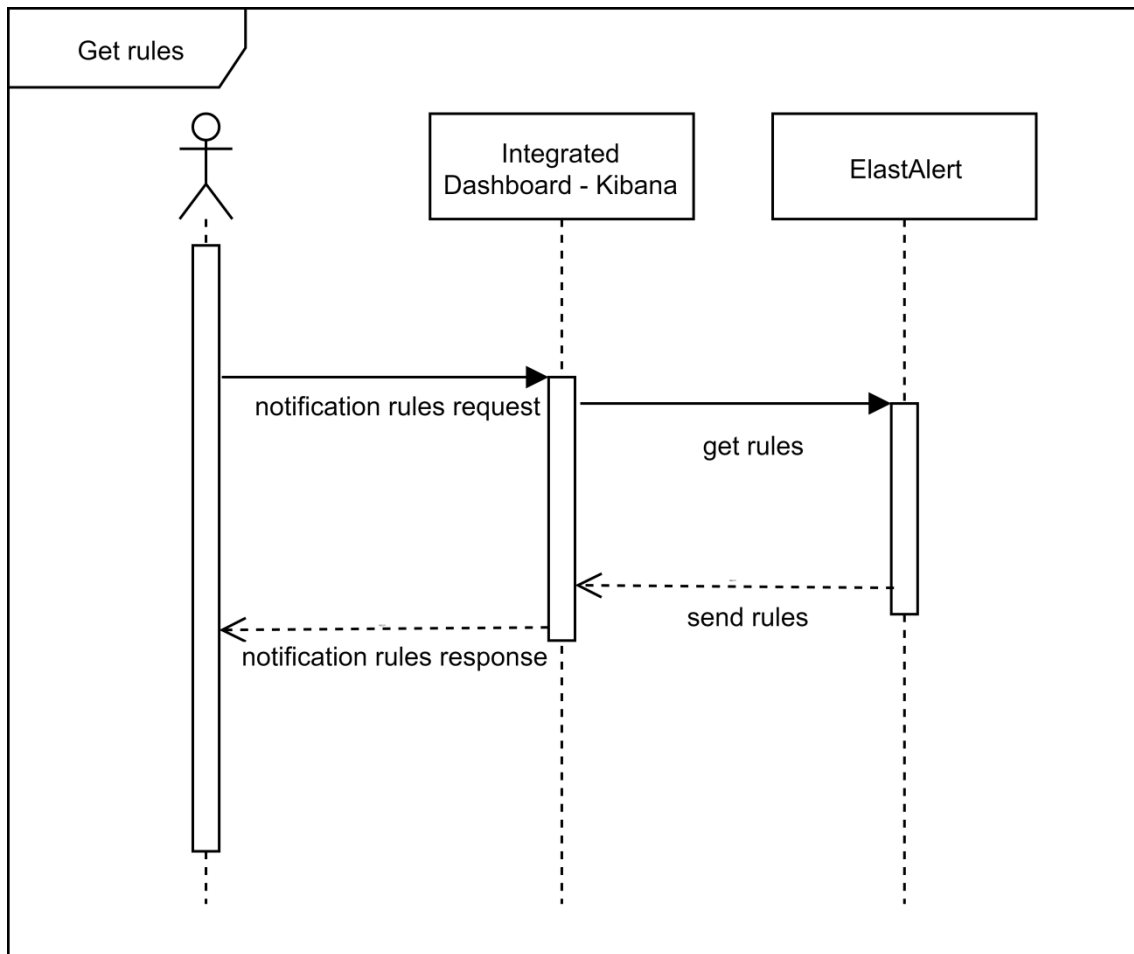


Figure 32: Get rules from ElastAlert

1. The user interacts with Kibana to recover a list of rules.
2. Kibana send a petition to get the rules.
3. ElastAlert send the rules to Kibana.
4. Kibana recover the list of rules.
5. The user can see the rules in Kibana.

### 4.3.5. PIXEL Security

In this section we are going to show a sequence diagram related to a process that occurs within this module:

- Access to an API protected with OAuth2.

#### 4.3.5.1. Access to an API protected with OAuth2

Diagram below illustrates the process of accessing an API protected by the OAuth2 mechanism provide by PIXEL Security solution to access a REST API exposed by PIXEL.

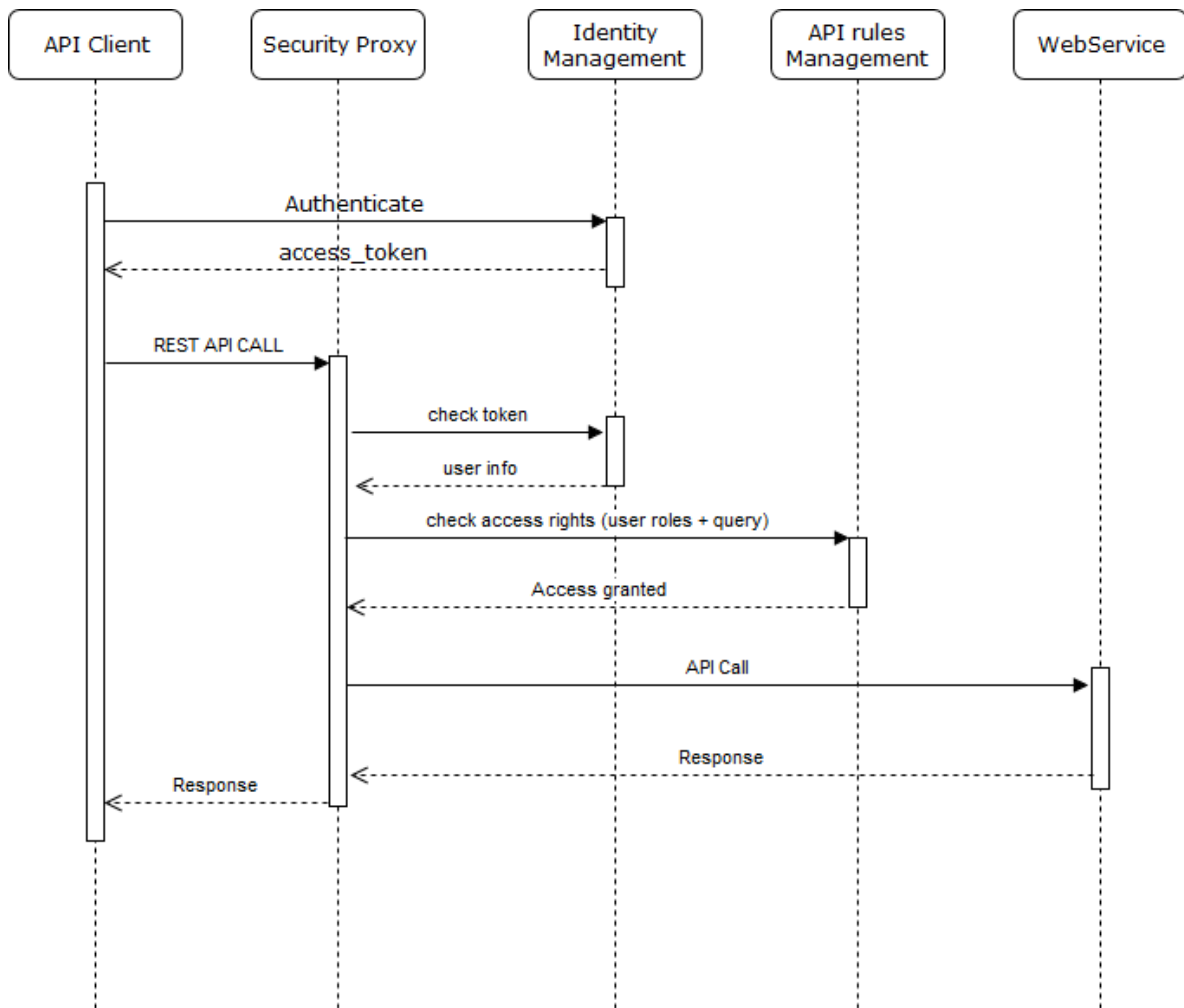


Figure 33: Process of accessing to an API protected by OAuth2

## 5. Information model

An information model<sup>28</sup>, 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.

### 5.1. Ontological model

The evolution of new technologies and their use has resulted in a rapid increase of the quantity of data produced each day. The democratization of services such as YouTube, Facebook, Flickr, etc. has resulted in the production of huge amounts of data every day. It is therefore increasingly difficult for the users to manipulate and process these large volumes of data. Set off by Tim Berners Lee, the Semantic Web is a set of technologies whose main goal is to make available to machines the meaning of data. The Semantic Web was promoted in order to use the increasing processing capacity of machines to process the volumes of data on the Web by providing the means to the machines to process this data efficiently and correctly. This enables to assist the users in the processing of the data. The Semantic Web can be considered as a sub domain of knowledge engineering as it brings elements of response to several problems inherent in this area:

<sup>28</sup> [https://en.wikipedia.org/wiki/Information\\_model](https://en.wikipedia.org/wiki/Information_model)

- The representation of knowledge in sectors where the need to collect and manage large volumes of knowledge is omnipresent.
- Search for information through advanced search engines. In this area, the use of semantics allows the engines to give more relevant answers to the users.
- The sales websites where the Semantic Web can meet the needs inherent to user profiling and recommendation of items based on these profiles.

The Semantic Web technologies can also be used in a non-Web context to address problems related to many use cases. However, the philosophy of the Semantic Web is preserved if the use case consists to give to analysis processes some understanding of the data to process, in order to provide advanced functionalities to assist the users.

### 5.1.1. Heterogeneity

Due to the large and wide variety of device types (different vendors, languages and techniques used) the devices are not interoperable. This heterogeneity of devices causes problems to federate knowledge / get a big picture (for example, to monitor a house, several sensors producing heterogeneous data are needed) to correlate the information. In the 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. The temporal heterogeneity is due to the use of different sources of information from different machines and devices. First, there are some issues due to the use of different time zones and unsynchronized clocks. Second, the temporal heterogeneity can be due to the use of different formats or granularities (e.g. 2 seconds in FAT file systems, 100 nanoseconds in NTFS file systems). Then, the autonomy of data sources is another type of heterogeneity. The autonomy problems occur when data is stored in different storing systems. This leads to the use of different methods to access and query the data (different APIs and query languages for example). Finally, the semantic heterogeneity occurs when data inherent to a same domain are produced by independent producers. The semantic heterogeneity is mainly due to the information encoding which is not the same among sources due to formatting and the fact that same information can be represented or interpreted in different ways. Semantic heterogeneity can itself be split in multiple categories first defined in (Charnyote Pluempitiwiriyaewej n.d.) and refined in (Bergman 2006).

The conceptual heterogeneity (also called structural heterogeneity) occurs when the schemas of the information sources to integrate show differences while represented concepts are similar. This type of heterogeneity encompasses several issues:

- Naming: data sources can use different naming conventions for their attributes, concepts and properties. This is due to case sensitivity (uppercase vs lower case vs camel case), synonyms (meters vs m, owner vs proprietary), acronyms (EU vs European Union), misspellings, etc.
- Generalization / Specialization: occurs when a single concept in one schema is related to several concepts in another schema. For example, a field “address” in one schema can be decomposed in “street”, “town” and “postal code” in another schema.
- Aggregation: occurs when the same individuals are grouped differently in two schemas (e.g. census vs federal regions in USA).
- Internal path discrepancy: this type of heterogeneity means that there are differences between the concept hierarchies of two schemas. For example, a same concept is at different levels in the two hierarchies.

- Missing item: the scope of the two schemas can be different. Therefore, an attribute or a concept can be present in a schema and absent in another one.
- Item equivalence: two concepts of two different schemas are admitted as being the same but they are not (Charles de Gaulle the French president and Charles de Gaulle the aircraft carrier)
- Type mismatch: the same concept subsumed different types. For example, in one schema, a person is an animal and another schema a person is a human being.
- Constraint mismatch: the cardinalities and assertions on two schemas are different.
- Domain heterogeneity occurs when the semantic of the data sources is different. This type of heterogeneity includes the following issues:
  - Schematic discrepancy.
  - Scale or units: the measurement system is different (metric vs English system) or the units used are different (meters vs millimeters).
  - Precision: the values do not have the same precision (e.g. a value of 5.2kg in one data source and a value of 5.254kg in another one).
  - Data representation: this sub-category encompasses every difference in the representation of data including the use of different date formats, the use of a period instead of a comma for decimals, the use of different datatype (a postal code can be represented using a string or a number), etc.

Data heterogeneity is observed when similar data are represented in a different form in information sources. This type of heterogeneity includes several issues:

- Naming: data sources can use different naming conventions for the values they store. As in conceptual heterogeneity, this is due to case sensitivity, acronyms, misspellings, etc.
- 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.
- Missing data.
- Element ordering: if set of data are used, the order can differ between two data sources.

The language heterogeneity includes all the differences due to the languages used in the sources and variations introduced by the latter (grammar, ambiguity, etc.):

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

Build IoT cross domain applications and ensure interoperability: why and give example of such applications?  
→ Aggregation, mashup, facilitate communication between devices

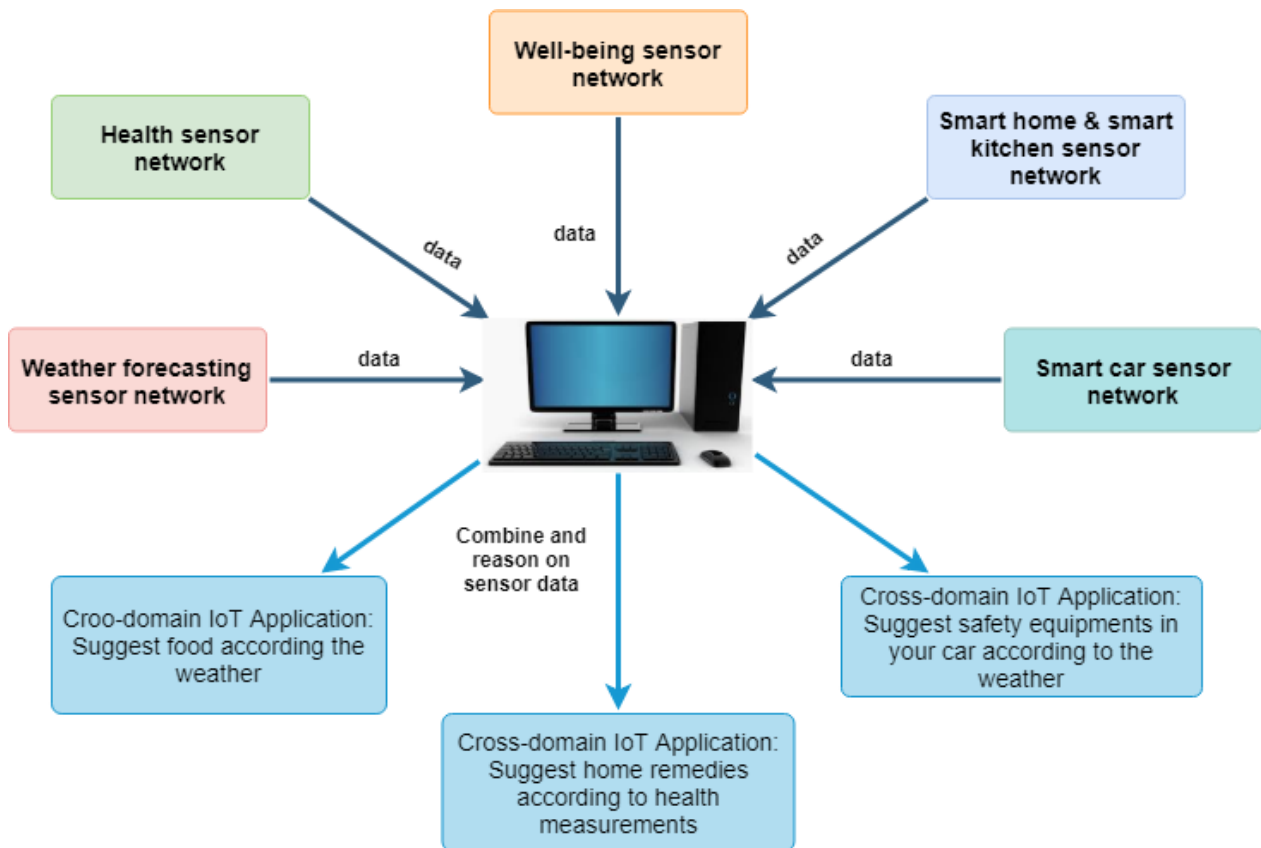


Figure 34: Example of IoT cross domain applications

### 5.1.2. Data understanding by programs

The connected devices largely contribute to the production of a huge amount of data on which the techniques of big data are used (4.4 billion GB produced in 2013). The main goal is to create value from the data collected by the IoT sensors which means make informed decisions from a big picture built using the information generated by them. To achieve this goal, it is necessary to analyse (time-consuming processes) these large volumes of data. For a human, the handling and understanding of this large amount of information is a tedious and complex task. In addition, this can lead to omissions or mistakes and therefore, the results can be false or information potentially relevant is not taken into account.

Automated processes can be leveraged to assist the human user. For this, it is necessary to give a certain understanding of data to the algorithms. Thus, they no longer work at the level of raw data but on knowledge represented by this data. The goal is to simulate cognitive processes to replace / assist humans in the interpretation and analysis of data.

This semantic elevation is illustrated in the following figure. This example shows the implementation of a crawling tool used to treat the information contained in the homepage of the Orange portal. For a human, the content and the structure of this webpage are understandable. For a machine, the language used and the layout of the elements are not understandable (a). A first step of the semantic elevation may be to use XML-based languages such as XHTML to allow the machine to understand the structure of the webpage (b). However, the meaning of the content remains out of reach of the machine. The objectives of the semantic technologies are to provide languages and techniques to describe the meaning of the content. This description should be formal and structured to allow the machine to treat it.



Figure 35: Example of semantic elevation

Make the data understandable by machines means explicitly associate a meaning to every data according to the context (for example, a temperature does not have the same meaning if it is about a room, a body, water, etc.), using a formal vocabulary and in a unified way.

### 5.1.3. Dynamicity of the environment

Devices / Equipments are integrated continuously in the environment or disappears from it (for example, in an urban environment, signalling equipment such as traffic lights or signs are added or deleted, the traffic rules evolve, etc.).

It is necessary to be able to adapt to change: automatic configuration plug and play, taking into account the changing environment in intelligence layers.

The two types of dynamicity are:

- The dynamicity due to the reconfiguration of the environment (the graph of devices is modified. For example, one node (a device) is added to the graph).
- The dynamicity due to real time system.

### 5.1.4. Open ontologies review

A recommended way for using ontologies is trying to reuse existing ones rather than creating your own one, and extend or adapt it. Some examples are provided below

#### 5.1.4.1. LOV4IoT

LOV4IoT<sup>29</sup> (Linked Open Vocabularies for Internet of Things) provide an Ontology Catalogue for Reusing Domain Knowledge Expertise

<sup>29</sup> <http://lov4iot.appspot.com/?p=ontologies>



Before reinventing the wheel, one can reuse the following existing ontologies referenced in this ontology catalogue with minor modifications. Sometimes these domain ontologies, although very interesting, are not referenced yet on the Linked Open Vocabularies (LOV)<sup>30</sup> since they need to follow more Semantic Web best practices<sup>31</sup>.

The LOV4IoT ontology catalogue references 499 ontology-based research projects for IoT and its applicative domains.

#### 5.1.4.2. FIWARE Data Models<sup>32</sup>

FIWARE<sup>33</sup> provide several data models easy to reuse and to contribute. They cover several domains and are built together with the FIWARE community. These data models 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<sup>34</sup>.

They define several domains that cover PIXEL needs:

- ENVIRONMENT: Enable to monitor air quality and other environmental conditions for a healthier living.
- TRANSPORTATION: Transportation data models for smart mobility and efficient management of municipal services.
- DEVICE: IoT devices (sensors, actuators, wearables, etc.) with their characteristics and dynamic status.
- PARKING: 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 in 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.

#### 5.1.5. PIXEL Initial Data Model

According to Princeton University<sup>35</sup>, “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 four-step process that is outlined below:

---

<sup>30</sup> <https://lov.linkeddata.es/dataset/lov/>

<sup>31</sup> <http://perfectsemanticweb.appspot.com/>

<sup>32</sup> <https://fiware-datamodels.readthedocs.io/en/latest/>

<sup>33</sup> <https://www.fiware.org/developers/>

<sup>34</sup> <https://www.fiware.org/2016/06/08/fiware-ngsi-version-2-release-candidate/>

<sup>35</sup> <https://cedar.princeton.edu/understanding-data/what-data-model>

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. This will be the first approach to the data model of PIXEL global platform. As it is usual in software engineering, this data model will be modified and improved as the execution of this task (T6.1 PIXEL Architecture) evolves. The idea is to have a common grounding to be shared with all the Consortium and stakeholders to advance on ICT development. This specification will be also completed as soon as more detailed information is received from other work packages different than WP6 (models definition in WP4, KPIs for evaluation in WP8, etc.).

For this iteration, we have kept the tables without relation arrows. This responds to the current status of technological choices. In case that we finally use no-relational databases (which is most likely), there will be no need of specification, as this will be performed by foreign keys correlation.

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**.

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 **KPIs** will be evaluated.

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

- Port.
- Stakeholder.
- User.
- Use-case.
- Data source.
- Service.
- Model / PA (Predictive Algorithm).

- PEI.
- KPI.

## 2. Global Initial PIXEL Data Model

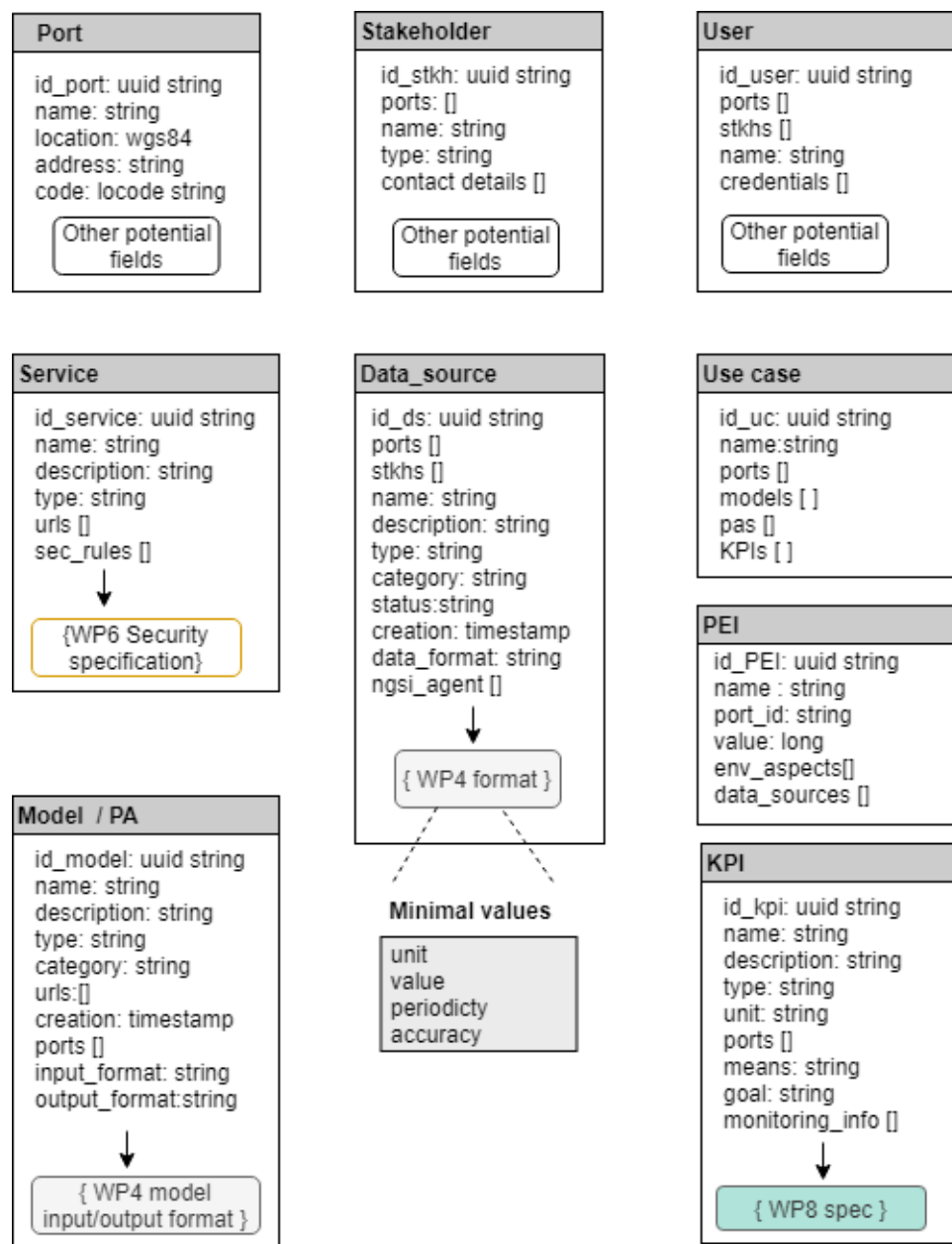


Figure 36: PIXEL Initial Data Model

## 3. Enriched explanation of Data Model entities:

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<sup>36</sup> code). Whenever the system becomes

<sup>36</sup> <https://www.unece.org/cefact/locode/service/location>

more sophisticated this entity will be enhanced, including, for example: type of port (according to D3.1 definitions). We leave this open now for further indications.

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 (ports array). Other fields are type (whether it is a terminal company, truck operator, etc.), name and contact details. Again, this might be enhanced later on so we have specified that other potential fields will be included.

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. Again, this might be enhanced later so we have specified that other potential fields will be included.

The PIXEL platform is composed of different components, which in the end may be considered as services: some of them are managed internally, some others may be even accessed by end users. In order to handle all these services (components) we need a directory of them so that it is possible to manage, describe and access them (endpoints). Initially, the type of services identified to be stored are the following: (i) models, (ii) predictive algorithms, (iii) operational tools, (iv) dashboards, (v) Data Acquisition Layer and (vi) Information hub. Security related services may also be included here (gateway, Identity Manager, etc.). Each service that would be developed/added to the system will imply (at least) a new entry in this central database following this entity data model. As PIXEL intends to be scalable and flexible, all (or nearly all) services will be callable through a REST API accessible from an endpoint. Furthermore, if the particular service has an associated HMI, this should be accessed through a different endpoint URL. Different users may have different security rules, so the final implementation of this entity in the Data Model may be completed with specification coming from task T6.6.

The **Data Source** is a basic entity in PIXEL. Every data gathered by the system to be exploited via services must have its source represented and stored in the central database of the system. With this aim, an initial approach has been defined. In PIXEL context, each data source will be associated to a port and to a stakeholder (owner of the sensor/ICT system). Type (physical/virtual), status (running/stopped/...), category (environmental/traffic/...) and register timestamp are others of the main attributes already identified. Additionally, more fields will be needed after further advance on WP4 and WP6. From our perspective at this point of the project, some fields that will be needed to represent accurately one data source should be, at least: measure unit, current value, periodicity and accuracy rate. For this means, we have included a “box” of new fields that will be completed later on.

In PIXEL we have several **use cases** defined. According to D3.4 and D4.1, different pilots will include different use-cases (energy, intermodal transport, port-city relation, environment). This use-cases will involve some inherent models in order to perform simulations or predictive algorithms. Finally, to measure the impact of the use-cases in the pilots some KPIs will be needed and defined.

The **PEI** is the most innovative proposal of PIXEL. Currently (month M11 of the project), the exact methodology of calculation, format of storage of values, etc. is under definition in WP5 (task T5.1 and T5.2). For that reason, limited description is described yet, but it is subject to change as WP5 work evolves.

The **models** and **predictive algorithms**, despite being a subset of services, deserve an independent entity within PIXEL Data Model. These services will need a particular input and output format of data that is worth to be stored to be consumed by different parts of the platform. Again, this field composition will be completed later, so we have specified that other potential fields will be included. Particularly, WP4 is working on specifying input and output formats as well as other features relevant for a global storage of the models.

Since the proposal stage, PIXEL has intended to create a grounding for evaluating its impact and relevance for ports. PIXEL has a specific work package (WP8) in which the Consortium is making an effort to particularise different **KPIs** for business-based and technical evaluation of the system. Besides, other KPIs related with parameters of port operations are being defined so that different agents can validate PIXEL concept and its power. This entity is aimed at storing at our central database all the KPIs and to clarify their relation to operations. Each KPIs will have its own measure unit, means of verification and goal (configurable by the port). Again, this field composition will be completed later, so we have specified that other potential fields will be included. Particularly, WP4 is working on specifying this point.

## 5.2.Data examples

In order to collect real data samples coming for ports, a specific folder has been created in the OnlyOffice tool (internal project repository). Each data sample is coupled with a readme.txt file describing data specification and format. In the following subsections, we are describing some of these data.

In this section there are also some examples of external data that will be collected (Task 6.2 Data acquisition), stored (Task 6.3 Information Hub) and then processed by the Operational Tools (Task 6.4) for predictive algorithms (Task 4.5). We refer readers to read deliverable D4.3 Predictive Algorithms v1 for a detailed description of open data sources related to predictive algorithms.

### 5.2.1. GPMB

#### 5.2.1.1. Electrical consumption data

GPMB already has the electrical consumption on a monthly basis from January 2014 to April 2017 in Bassens's terminal. The data are coming from bills and can be made available in an excel file. These data are aggregated data. Thus, we can only extract the whole electrical consumption of Bassens's terminal on a monthly basis. The sample data are available in the OnlyOffice tool.

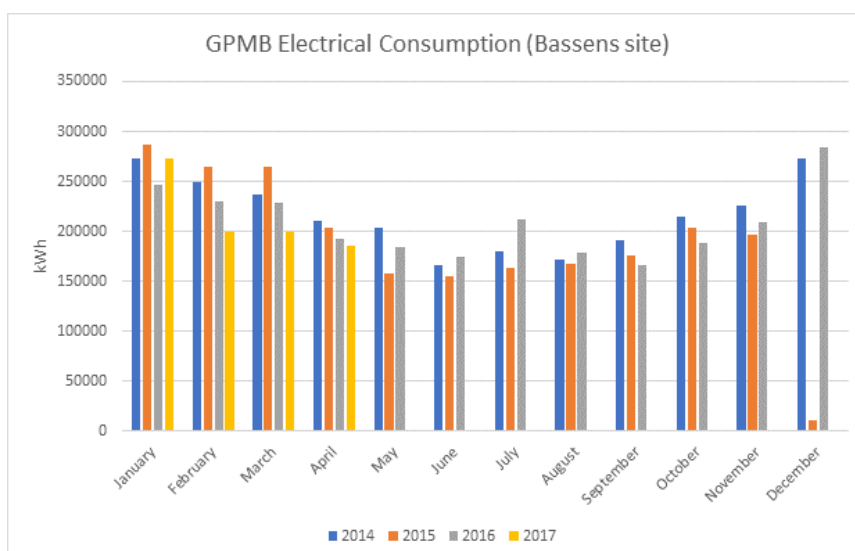


Figure 37: Sample of raw electrical consumption data based on GPMB's bills

### 5.2.1.2. Activity data

Raw vessels' calls logs are available through an extract of the VIGIESip database (GPMB's port management information system). The main source for this data is FAL forms. GPMB has made available an extract of its VIGIESip database from 2010 to 2017 (available in an excel file). This extracted data contains 9 230 records with 11 features for each.

This data will be used to obtain statistics and prediction about vessels 'calls.

Année Statistique	Numéro d'escale	Sens	Poste à quai	Nom Navire	Libellé Type Cargaison	Tonnage définitif	Date de poste à quai	Libellé du type de construction	Code type fiscal Navire	Libellé type fiscal Navire
2010	0003	Entrée	449	SP VIKING		0	1 janv. 10 17:15:00	CARGO DE PLUS DE 500 TJB	06	SOL.V
2010	0003	Sortie	449	SP VIKING	E.MAIS VRAC	4 370	4 janv. 10 19:57:00	CARGO DE PLUS DE 500 TJB	06	SOL.V
2010	0005	Entrée	432	ARKLOW RAIDER	I.URÉE VRAC	2 344	2 janv. 10 05:20:00	CARGO DE PLUS DE 500 TJB	06	SOL.V
2010	0005	Sortie	449	ARKLOW RAIDER	E.TRTX TOURNESOL	3 294	15 janv. 10 20:50:00	CARGO DE PLUS DE 500 TJB	06	SOL.V
2010	0006	Entrée	436	STRAITVIEW	I.HUILE COLZA	2 999	2 janv. 10 19:00:00	PRODUITS CHIMIQUES (CHEMICAL)	05	LIQ.V
2010	0006	Sortie	436	STRAITVIEW	E.HUILE COLZA	3 000	8 janv. 10 23:35:00	PRODUITS CHIMIQUES (CHEMICAL)	05	LIQ.V
2010	0007	Entrée	436	LISTRAUM	I.METHANOL	3 432	2 janv. 10 19:25:00	PRODUITS CHIMIQUES (CHEMICAL)	05	LIQ.V
2010	0007	Sortie	436	LISTRAUM		0	3 janv. 10 19:00:00	PRODUITS CHIMIQUES (CHEMICAL)	05	LIQ.V
2010	0008	Entrée	433	ARKLOW RAVEN	I.URÉE VRAC	4 123	3 janv. 10 07:55:00	CARGO DE PLUS DE 500 TJB	06	SOL.V
2010	0008	Sortie	449	ARKLOW RAVEN	E.MAIS VRAC	4 151	7 janv. 10 12:10:00	CARGO DE PLUS DE 500 TJB	06	SOL.V
2010	0009	Entrée	431	FEHN CARTAGENA		0	3 janv. 10 19:35:00	CARGO DE MOINS DE 500 TJB	06	SOL.V
2010	0009	Sortie	431	FEHN CARTAGENA	E.TERRE REFRACT.	1 516	5 janv. 10 20:35:00	CARGO DE MOINS DE 500 TJB	06	SOL.V
2010	0014	Entrée	415	KARIN	I.PROD.METALL.AUTRES	1 385	5 janv. 10 09:10:00	CARGO DE PLUS DE 500 TJB	12	AUTRES
2010	0014	Sortie	415	KARIN		0	9 janv. 10 12:10:00	CARGO DE PLUS DE 500 TJB	12	AUTRES
2010	0019	Entrée	433	BRATISLAVA	I.ENGR.MANUF.VRAC	3 300	6 janv. 10 21:45:00	CARGO DE PLUS DE 500 TJB	06	SOL.V
2010	0019	Sortie	433	BRATISLAVA		0	8 janv. 10 12:35:00	CARGO DE PLUS DE 500 TJB	06	SOL.V
2010	0020	Entrée	432	DINA TRADER	I.CONTENEURS	3 376	7 janv. 10 09:15:00	PORTE-CONTENEURS INTEGRAL	09	P.C.
2010	0020	Sortie	432	DINA TRADER	E.CONTENEURS	5 084	8 janv. 10 22:25:00	PORTE-CONTENEURS INTEGRAL	09	P.C.
2010	0021	Entrée	415	MEKHANIK TYULENEV	I.BOIS SCIES DU NORD	1 688	7 janv. 10 10:10:00	CARGO DE PLUS DE 500 TJB	12	AUTRES

Figure 38: GPMB sample data concerning ship statistics

There is also some web-services that are made available by GPMB.



JSON	Données brutes	En-têtes
Enregistrer	Copier	Tout réduire
		Tout développer
▼ 0:		
ANNONCE:		20187396
IMO:		"9531038"
▼ MOUVEMENTS:		
▼ 0:		
IDMVT:		"64049"
TYPE:		"DPQ"
ETA:		"26/12/2018 19:00:00"
ETD:		"01/03/2019 00:00:00"
BPM:		true
DESTINATION:		"700 Pauillac"
MDP:		false
AGENT:		"SEAINVEST"
TE:		340
MVTREEL:		true
▼ 1:		
ANNONCE:		20197706
IMO:		"7108930"
▼ MOUVEMENTS:		
▼ 0:		
IDMVT:		"64503"
TYPE:		"DPQ"
ETA:		"04/04/2019 02:45:00"
ETD:		"05/04/2019 20:00:00"
BPM:		false
DESTINATION:		"124 Bordeaux"
MDP:		false
AGENT:		"HIIMANN"

Figure 39: Example of one GPMB's web-service ([http://bordeaux.vigiesip.eu/vigiesip-webservice-bordeaux/nonauthent/export\\_demandes](http://bordeaux.vigiesip.eu/vigiesip-webservice-bordeaux/nonauthent/export_demandes))

### 5.2.1.3. Weather

Weather measurements are available through Météo France (official French weather forecasting service). This data is open-data<sup>37</sup>. These data are in ASCII format with a frequency of 3h. These data are observational data from international surface observation messages (SYNOP) circulating on the Global Telecommunication System (GTS) of the World Meteorological Organization (WMO). The atmospheric parameters measured are temperature, humidity, wind direction and force, atmospheric pressure and precipitation height. The atmospheric parameters observed are sensitive time, cloud description, visibility from the Earth's surface. Depending on instrumentation and local specificities, other parameters may be available (snow depth, soil condition, etc.).

<sup>37</sup> Weather data are accessible following this link:

[https://donneespubliques.meteofrance.fr/?fond=produit&id\\_produit=90&id\\_rubrique=32](https://donneespubliques.meteofrance.fr/?fond=produit&id_produit=90&id_rubrique=32) The Bordeaux station code is 07510 (filter rows on this ID).



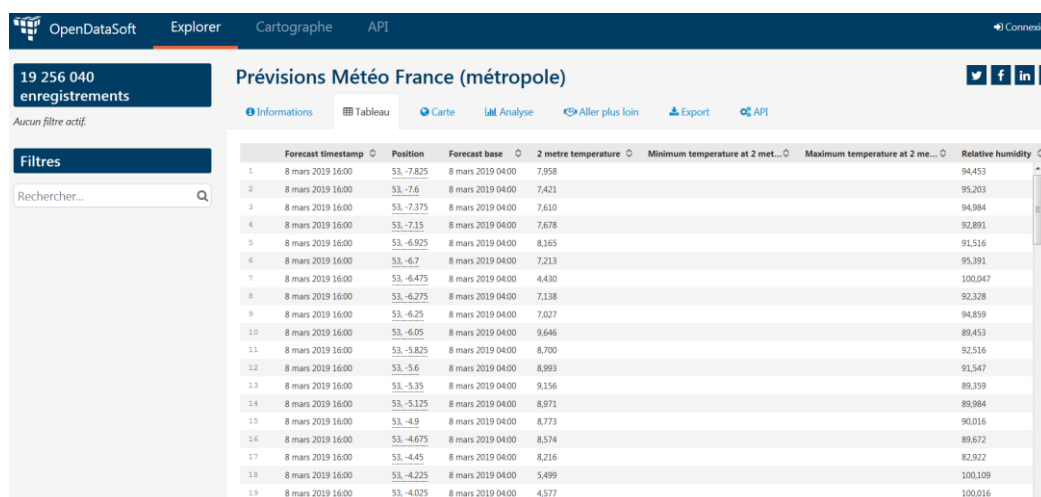


Figure 40: Weather data access through opendatasoft.com<sup>38</sup>

The direct Météo France's open-data part (free services) does not seem suitable because there is a need to manually download the data. Subscription services with API are proposed. However, an indirect API access using opendatasoft.com<sup>39</sup> may be interesting.

#### 5.2.1.4. Pollutants concentration data

Pollutants concentration data are available through ATMO Nouvelle-Aquitaine<sup>40</sup>. Data is updated every hour. ATMO monitors five pollutants' concentrations at Bassens station. This station is not located inside the port but in the city nearby. A sample data is shown in the following table. The data need to be manually downloaded in a .csv format. An open data platform is also available to access all data monitored by ATMO: <https://data-atmo-na.opendata.arcgis.com/>.

Table 5: Pollutants concentration data at Bassens station (manually downloaded)

Station	Pollutant	Measure	Unit	01/03/2017 00:00	01/03/2017 01:00
Bassens	Nitrogen dioxide	NO <sub>2</sub>	µg/m <sup>3</sup>	17	13
Bassens	Sulphur dioxide	SO <sub>2</sub>	µg/m <sup>3</sup>	0	1
Bassens	Ozone	O <sub>3</sub>	µg/m <sup>3</sup>	73	76
Bassens	Suspended matters	PM <sub>10</sub>	µg/m <sup>3</sup>	18	20
Bassens	Fine particles	PM <sub>2,5</sub>	µg/m <sup>3</sup>	14	10

<sup>38</sup>

[https://public.opendatasoft.com/explore/dataset/arome-0025-sp1\\_sp2/table/?location=22,44.85002,-0.575&basemap=jawg.streetshttps:%2F%2Fhelp.opendatasoft.com%2Fapis%2Fods-search-v1%2F#search-api-v1](https://public.opendatasoft.com/explore/dataset/arome-0025-sp1_sp2/table/?location=22,44.85002,-0.575&basemap=jawg.streetshttps:%2F%2Fhelp.opendatasoft.com%2Fapis%2Fods-search-v1%2F#search-api-v1)

<sup>39</sup> API services for weather data:

[https://public.opendatasoft.com/explore/dataset/arome-0025-sp1\\_sp2/table/?location=6,46.00538,2&basemap=jawg.streetshttps://help.opendatasoft.com/apis/ods-search-v1/#search-api-v1](https://public.opendatasoft.com/explore/dataset/arome-0025-sp1_sp2/table/?location=6,46.00538,2&basemap=jawg.streetshttps://help.opendatasoft.com/apis/ods-search-v1/#search-api-v1)

<sup>40</sup> <https://www.atmo-nouvelleaquitaine.org/donnees/telecharger>

## 5.2.2. THPA

### 5.2.2.1. Environmental data

#### 5.2.2.1.1. Wind data

The wind data are obtained by using two different sensors. The first sensor is located on gantry crane in the container terminal. An application communicates with the sensor and dumps raw data into a database (see next table). This database has only one table.

A second sensor is located on the rooftop of the technical services building. For this sensor, the data are downloaded manually on a daily basis. The objective is to give access to the downloaded file in a public repository in order to be able to upload this data to the PIXEL Data Acquisition Layer or Hub. The raw data is text in CSV format. This wind sensor is also connected with temperature and humidity measurements.

*Table 6: Wind data - small sample extracted from the database using SELECT \* FROM WindData*

id	timestamp	speed (m/sec)	angle (degrees)	dates
75072833	0x000000000482F853	6.08	99	2019-02-23 00:06:04.460
75072834	0x000000000482F854	5.83	102	2019-02-23 00:06:06.520
75072835	0x000000000482F855	5.75	101	2019-02-23 00:06:08.520
75072836	0x000000000482F856	5.78	99	2019-02-23 00:06:10.570
75072837	0x000000000482F857	5.74	97	2019-02-23 00:06:12.650
75072838	0x000000000482F858	6.35	97	2019-02-23 00:06:14.663
75072839	0x000000000482F859	6.16	95	2019-02-23 00:06:16.710
75072840	0x000000000482F85A	5.93	99	2019-02-23 00:06:18.757
75072841	0x000000000482F85B	5.66	100	2019-02-23 00:06:20.763
75072842	0x000000000482F85C	5.21	93	2019-02-23 00:06:22.800
75072843	0x000000000482F85D	5.81	90	2019-02-23 00:06:24.803
75072844	0x000000000482F85E	5.77	96	2019-02-23 00:06:26.887
75072845	0x000000000482F85F	5.33	92	2019-02-23 00:06:28.910
75072846	0x000000000482F860	5.23	93	2019-02-23 00:06:30.957

*Table 7: Sample Data of the second wind sensor (METEO STN RAW DATA)*

datetime	CH1	VAL1	CH2	VAL2	CH3	VAL3	CH4	VAL4
15:00:00 02.01.19	00	046.4	01	010.45	02	002.160	03	00300
15:30:00 02.01.19	00	046.2	01	010.38	02	002.133	03	00290
16:00:00 02.01.19	00	045.3	01	010.61	02	001.466	03	00295
16:30:00 02.01.19	00	046.7	01	010.36	02	001.386	03	00290
17:00:00 02.01.19	00	046.9	01	010.39	02	000.906	03	00300
17:30:00 02.01.19	00	052.7	01	009.21	02	001.600	03	00305

18:00:00 02.01.19	00	056.9	01	008.05	02	002.480	03	00000
18:30:00 02.01.19	00	058.4	01	007.50	02	001.680	03	00000
19:00:00 02.01.19	00	059.5	01	007.22	02	002.880	03	00000
19:30:00 02.01.19	00	060.6	01	006.93	02	002.960	03	00000

Units:

- VAL1: hum%
- VAL2: temp (°C)
- VAL3: wind speed (m/sec)
- VAL4: wind dir (degrees)

#### 5.2.2.1.2. Noise and spatial data

In THPA, there are noise measurements at different locations. These locations are pointed in a .kmz file, that could be opened using Google Earth. Other geographical data are available in .shp and .dxf format.



Figure 41: Pointers to noise sensors locations (extractions from a .kmz file)

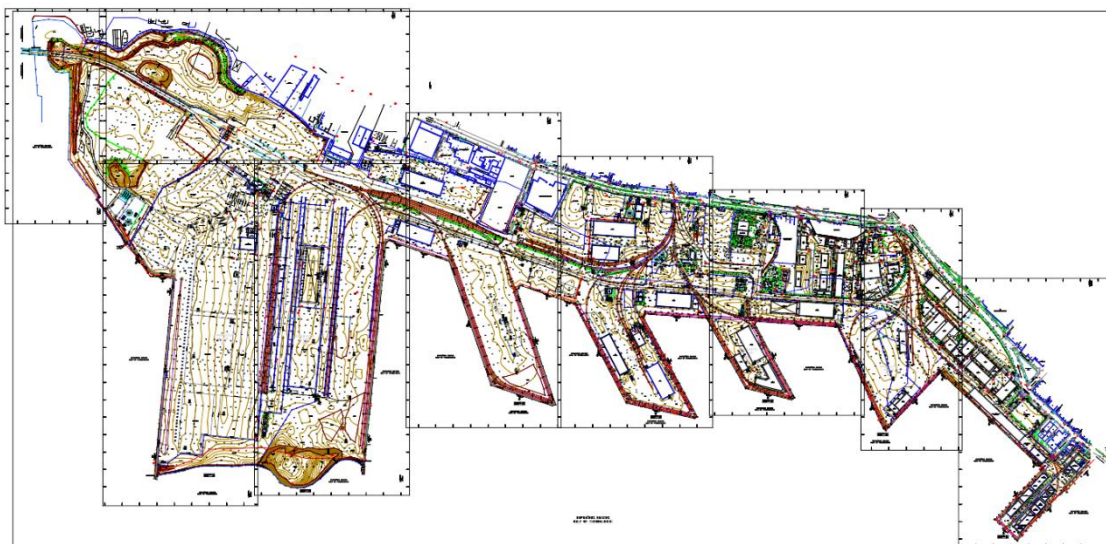


Figure 42: Spatial data (extraction from a .dxf file)

### 5.2.2.1.3. Air quality data

Four analysers of air quality are available in THPA. The analyser sends data directly to the logger website –the data can be accessed by logging in and downloading the data in CSV format, as a static file. THPA cannot give the credentials for the site. Moreover, the website does not have an API and the data need to be manually downloaded. A sample data (after importing .csv data into an excel sheet) is showed in the figure below. The data that are analysed are: NO, NO<sub>2</sub>, NO<sub>x</sub>, SO<sub>2</sub>, CO, PM<sub>10</sub>, PM<sub>2.5</sub>, temperature and relative humidity.

	A	B	C	D	E	F	G	H	I	J
	TimeBeginning	TimeEnding	946100_NO_1_Scaled	946100_NO_1_Unit	946100_NO_1_Status	946100_NO2_3_Scaled	946100_NO2_3_Unit	946100_NO2_3_Status	946100_NOx_2_Scaled	946100_NOx_2_Unit
1	01/01/2018 00:00	01/01/2018 00:05	84 211	ppb	Valid	35 096	ppb	Valid	119 307	ppb
2	22/11/2018 00:30	22/11/2018 00:35	0.663	ppb	Valid	4 248	ppb	Valid	4.91	ppb
3	22/11/2018 00:35	22/11/2018 00:40	0.603	ppb	Valid	3 903	ppb	Valid	4 506	ppb
4	22/11/2018 00:40	22/11/2018 00:45	0.627	ppb	Valid	4 059	ppb	Valid	4 685	ppb
5	22/11/2018 00:45	22/11/2018 00:50	0.573	ppb	Valid	4 466	ppb	Valid	5 039	ppb
6	22/11/2018 00:50	22/11/2018 00:55	0.797	ppb	Valid	4 813	ppb	Valid	5.61	ppb
7	22/11/2018 00:55	22/11/2018 01:00	0.489	ppb	Valid	4 423	ppb	Valid	4 912	ppb
8	22/11/2018 01:00	22/11/2018 01:05	0.009	ppb	Valid	4 014	ppb	Valid	4 023	ppb
9	22/11/2018 01:05	22/11/2018 01:10	0.065	ppb	Valid	4.74	ppb	Valid	4 805	ppb
10	22/11/2018 01:10	22/11/2018 01:15	0.129	ppb	Valid	4 696	ppb	Valid	4 825	ppb
11	22/11/2018 01:15	22/11/2018 01:20	0.212	ppb	Valid	4 584	ppb	Valid	4 796	ppb
12	22/11/2018 01:20	22/11/2018 01:25	0.258	ppb	Valid	4.51	ppb	Valid	4 767	ppb
13	22/11/2018 01:25	22/11/2018 01:30	0.331	ppb	Valid	4 369	ppb	Valid	4.7	ppb
14	22/11/2018 01:30	22/11/2018 01:35	0.398	ppb	Valid	4 772	ppb	Valid	5.17	ppb
15	22/11/2018 01:35	22/11/2018 01:40	0.515	ppb	Valid	5 011	ppb	Valid	5 526	ppb

Figure 43: Sample data of air quality measurements in THPA

Since direct access to data is not possible. It is planned to upload the downloaded file in a public repository that can be access by PIXEL partners.

### 5.2.2.2. Traffic data

Inbound and outbound traffics in the Port of Thessaloniki are monitored by a system comprising of the following:

- RFID readers (CS203ETHER), one per lane. In Gate 16 (main gate) there are three (3) lanes for entry, and one (1) for exit. In Gate 10A there is one (1) lane for entry and one (1) for exit. All vehicles have an RFID tag located on their windshield.
- A web application, developed by a private company, visualizing (and storing) the readings of the sensors, is shown below.



- Data is stored on a MySQL relational database. The data can be shared via a web call (JSON, direct CSV download, etc.).

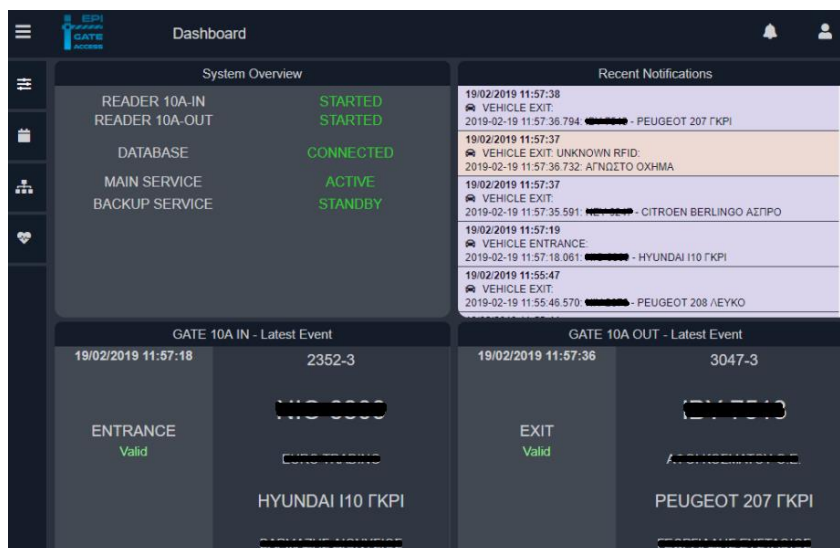


Figure 44: Example of traffic monitoring in THPA

### 5.2.2.3. Fuel Consumption

ThPA monitors the consumption of fuel in real-time, using a system comprising:

- RFID rings & sensors installed on fuel nozzles and vehicles' reservoirs.
- The application GAS Station developed by the company Logicom. The application includes a number of modules, related to syncing the nozzle data, the fuel availability, linking to SAP (see below), etc.

Connector

Απόκρυψη Αναφορές & Διαχείριση Προεπισκόπηση Εξαγωγή σε Excel Αντιστοίχιση Σχήματος Πλοήγηση Συμβάντων Κλειδωμένα Όλα Εκτύπωση Αποθεμάτων

Πρατήριο: Όχημα: Συμ. Ποσότητα: 6.436,460  
 Server: Όλα τα οχήματα Συμ. Ποσότητα 15°C: 6.436,460  
 BYTIO Καύσιμο: Από: Έως: Τώρα  
 ΠΡΑΤΗΡΙΟ η Ανανέωση Όλα τα καύσιμα 18/02/2019 00:00 19/02/2019 23:59  
☒ Απόκρυψη LOG

Ημερομηνία/ώρα	#	Αν.	Καύσιμο	Λίτρα	°C	Λιτ. 15°C	Όχημα	Αρ. Κυκλοφορίας	Πρατήριο	Ώρες Κιν.	Π. Ώρες Κ.	Λίτρα/Ώρα
18/02/2019 15:02:31	19088	6	Diesel	10,180		10,180	CATERPILLAR	ME 87671	ΠΡΑΤΗΡΙΟ	5,0	5252,0	
18/02/2019 18:30:23	19089	3	Diesel	61,470		61,470	VOLKSWAGEN AMAROK	ΚΗΗ 5816	ΠΡΑΤΗΡΙΟ			
19/02/2019 07:53:22	19090	1	Diesel	13,260		13,260	KALMAR	ME 110983	ΠΡΑΤΗΡΙΟ	1,0	1,0	
19/02/2019 07:53:23	19091	2	Diesel	173,820		173,820	KALMAR	ME 110981	ΠΡΑΤΗΡΙΟ	1,0	1,0	
19/02/2019 07:56:19	19092	1	Diesel	76,780		76,780	FRONTLIFT	ME 134929	ΠΡΑΤΗΡΙΟ	5,0	5,0	
19/02/2019 07:56:36	19093	2	Diesel	11,210		11,210	KALMAR	ME 110983	ΠΡΑΤΗΡΙΟ	5,0	1,0	2,80
19/02/2019 07:57:48	19094	6	Diesel	44,360		44,360	CATERPILLAR	ME 87620	ΠΡΑΤΗΡΙΟ	888888,0	123,0	
19/02/2019 07:57:57	19095	2	Diesel	52,730		52,730	KALMAR	ME 110983	ΠΡΑΤΗΡΙΟ	2,0	5,0	
19/02/2019 08:02:02	19096	6	Diesel	39,330		39,330	CATERPILLAR	ME 87669	ΠΡΑΤΗΡΙΟ	888888,0	1,0	
19/02/2019 08:03:43	19097	1	Diesel	176,270		176,270	KONECRANES	ME 137474	ΠΡΑΤΗΡΙΟ	1,0	236,0	
19/02/2019 08:04:45	19098	6	Diesel	46,740		46,740	FIAT	ΚΗΗ 6484	ΠΡΑΤΗΡΙΟ			
19/02/2019 08:05:36	19099	1	Diesel	4,140		4,140	BOBCAT	ME 72525	ΠΡΑΤΗΡΙΟ	1,0	5,0	
19/02/2019 08:14:26	19100	2	Diesel	123,240		123,240	KALMAR	ME 110982	ΠΡΑΤΗΡΙΟ	1,0	6085,0	
19/02/2019 08:18:21	19101	1	Diesel	149,510		149,510	SIZU	ME 57642	ΠΡΑΤΗΡΙΟ	235,0		0,64
19/02/2019 08:23:04	19102	1	Diesel	125,000		125,000	CAT 980.H	ME 103725	ΠΡΑΤΗΡΙΟ	9580,0	9576,0	31,25
19/02/2019 08:25:48	19103	1	Diesel	57,940		57,940	TERBERG11	ME 138400	ΠΡΑΤΗΡΙΟ	423,0	418,0	11,59
19/02/2019 08:36:49	19104	1	Diesel	66,800		66,800	TERBERG6	ME 138402	ΠΡΑΤΗΡΙΟ	369,0	364,0	13,36

Figure 45: Example of THPA monitoring system of fuel

### 5.2.2.4. Port Operations related data

#### 5.2.2.4.1. Environmental data

The two terminals of ThPA SA, store all data related to port operations, as follows:

As far as the Container Terminal (CT) is concerned, all data about vessel calls, arrivals, unloading/loading etc., is handled by the FRETIS Terminal Operating System (TOS), developed by a private company. The TOS includes a number of modules, depending on the business area; for example: GIS (positioning containers on yard), receipt of XML messages for vessel operations such as calling (CALINF, COARRI), invoicing, gate control (related paperwork), etc. The invoicing module includes a “bridge” (link) to SAP (analysed below), to sync financial data. All data is stored in a Microsoft SQL Server relational database.

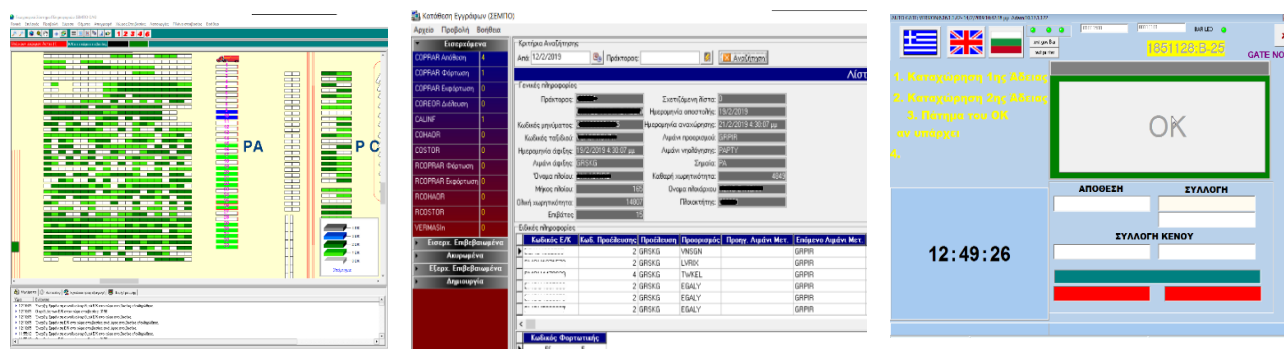


Figure 46: Screenshots of FRETIS modules (indicative): GIS / yard planning, XML messages, gate control

For conventional cargo, most relevant data is stored in a custom application called Statistics, developed by a private company. In it, vessel calls, duration of works, type and quantity of cargo handled, etc., is (manually) entered. All data is stored in a Microsoft SQL Server relational database.

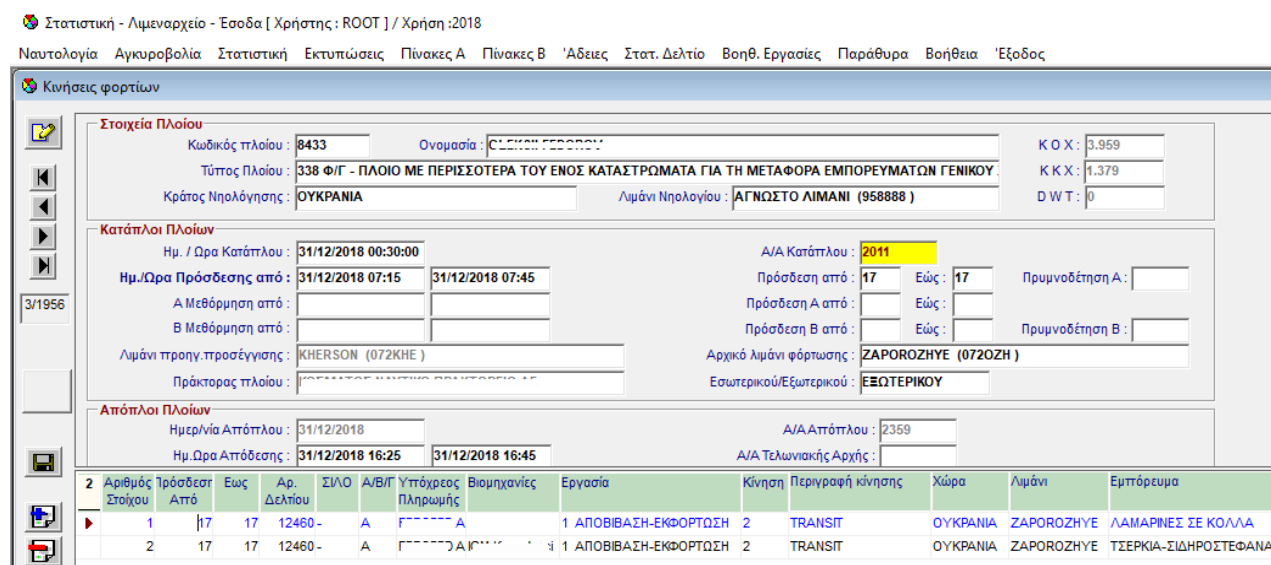


Figure 47: Screenshot from Statistics software

### 5.2.3. SDAG/ASPM

#### 5.2.3.1. SDAG Access Control and Management System

SDAG parking area has got three informatic systems:

- For the trucks flow management.
- For the payment of parking tickets.
- For the security in parking areas.

All areas are managed by SDAG and are connected by optical fibre, and in such system, there are several data. Mainly there are two systems:

- Truck flows and ticket payments management.

The systems, named CAME PsOne, is based on a protocol that is composed by:

- Entrance station for the tickets, the management of membership tickets or the acceptance of the parking booking code.
- The Exist station to check the payment or the registration of memberships tickets.
- Automatic cash register for payments management and for pre-paid phonecards.
- Manual cash register for the management of payments.
- Server of the configuration and maintenance of the system, management of payments and memberships administration.

- Video-surveillance and truck number plates reading.

The system allows the video surveillance in parking areas and to manage the access of vehicles in SDAG areas (both SDAG and other Entities). It is entirely based on IP and is it composed by:

- VMPS Genetec Security Center for the management of the system configuration, video archives and real-time areas checks.
- Cameras dome for the control of the areas.
- Badge reader installed in the crossing access areas.
- Sub-system based on caleras SELEA that allows the reading of number plates and Kemler Codes (for ADR transport) both in access and exit areas. This system is connected online with a server with which you can request data.

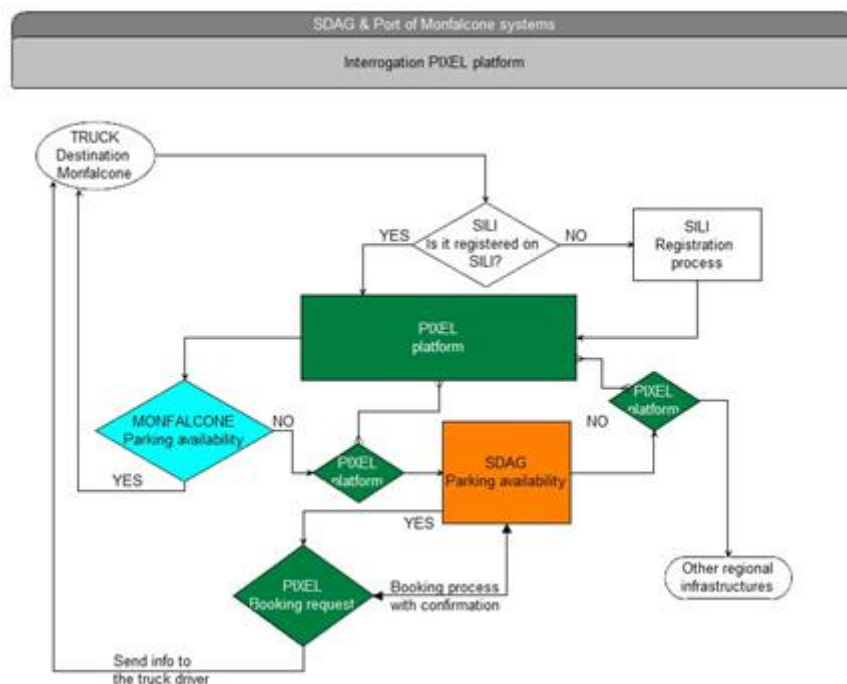


Figure 48: How SDAG & ASPM systems work with PIXEL platform

Regarding the data that SDAG system can provide, they are:

- Historic data from access in parking areas;
- Free lots in parking areas;



- Remote Booking with return of the booking code;
- Communication of access in the parking area;
- Communication of exit in the parking area.

The CAME system accepts HTTP calls with JSON format. Eventually, on request, the instructions can be personalized.

Data	Frequency		Format
Historic data of access	SDAG software	Manual	Word,Excel,RTF, RPT
Free lots availability	real time		JSON
Remote booking of parking lots	real time		JSON
Communication of truck entrance in parking area (with booking code)	real time		JSON
Communication of truck exit from parking area (with booking code)	real time		JSON

Figure 49: Data frequency and format of SDAG system

## 5.2.4. PPA

Vessels calls: one semester of data (4<sup>th</sup> semester 2018).

- Cruise ship data:
  - Ship name, Company, Agent, IMO, Scheduled arrival date and time, actual arrival date and time, Scheduled departure date and time, actual departure date and time, capacity, passengers to arrive, transit passengers, total passengers, ...
  - One year of data.
- Weather:
  - Temperature, Humidity, Wind speed, Wind direction, precipitation.
  - One month of data (March 2019).
- AIS Data:
  - This data was extracted from Marine Traffic.
- Pollutants:
  - One month of data (August 2018).
  - NO, NO<sub>2</sub>, CO, SO<sub>2</sub>, O<sub>3</sub>, BTEX, PM<sub>10</sub>, wind direction and speed.

These data are stored in PPA using ORAMA, EXPRESSJ and the internal PP ERP, and they have been downloaded manually. The raw format of data are: .csv and .xlsx. For cargo in/out, there is a monthly report generated; for cruise ships past data are available 15 days after last call with complete passenger's data back to 1-2 years. Concerning cruise ships arrivals, future data are available for the next year but without any data related to passengers. The AIS data can be provided by third parties such as Marine Traffic.

## 5.2.5. External Data

In the following section, we provide some examples of external data that will be collected (Task 6.2 Data acquisition), stored (Task 6.3 Information Hub) and then processed by the Operational Tools (Task 6.4) for predictive algorithms (Task 4.5). We refer readers to read deliverable D4.3 Predictive Algorithms v1 for a detailed description of open data sources related to predictive algorithms.

### 5.2.5.1. Automatic Identification System (AIS) data

AIS data was imposed by the International Maritime Organization (IMO) to improve safety on the sea. Each vessel above 300GT is obligated to have an AIS system installed. In EU this applies to all the vessels exceeding 15m. AIS data is exchanged between nearby ships by VHF radio and to coastal AIS stations. On open seas satellite AIS is used instead.

AIS messages contain different data that is transmitted at different time intervals. Bellow, we summarize some of the data sent every 2 - 10 seconds when a ship is underway and is mostly gathered from the AIS transponder itself and is as such more accurate and data that is entered manually by the crew and as such prone to errors, sent every 6 minutes. AIS messages contain pieces of information such as:

- MMSI number
- IMO ship identification number
- Navigation status (manually entered)
- Radio call sign
- Rate of turn
- Name
- Speed over ground
- Type of ship/cargo
- Positional information (Longitude, Latitude)
- Dimensions
- UTC seconds
- Location of GPS antenna
- True heading
- Type of GPS
- True bearing
- Draught
- Destination
- ETA

Raw AIS messages are transmitted in a NMEA 0183<sup>41</sup> format. An example of raw AIS data can be gathered from MarineTraffic<sup>42</sup>. Freely available raw AIS data in NMEA format is available through AISHub<sup>43</sup>.

### 5.2.5.2. European Space Agency Copernicus satellite imagery

European Space Agency (ESA) has been developing 5 Sentinel satellite missions as part of the Global Monitoring for Environment and Security (GMES) Operational Services. For predictive analytics, Sentinel-1 and Sentinel-2 constellations are relevant. Sentinel-1 uses Synthetic Aperture Radar (SAR) imaging which is particularly useful in maritime domain due to its independence from weather conditions and is also utilized in operational tools in maritime domain. Sentinel-2 uses multispectral camera which is mostly utilized in land-cover analysis. The use in maritime applications is underutilized and will be explored as part of T4.5. For detailed description we refer readers to deliverable D4.3.

---

<sup>41</sup> <https://www.tronico.fi/OH6NT/docs/NMEA0183.pdf>

<sup>42</sup> <https://help.marinetraffic.com/hc/en-us/articles/215626187-I-am-an-AIS-data-contributor-Can-you-share-more-data-with-me->

<sup>43</sup> <http://www.aishub.net/>

ESA Sentinel data is available through Open Access Hub<sup>44</sup>, though there exists commercial providers such as Sinergise<sup>45</sup> which provide API access that is easy to use and to integrate into existing solutions at an affordable price. Data that is available through Open Access Hub is provided in Sentinel-SAFE format<sup>46</sup> where imagery is available in JPEG2000 format.

### 5.2.5.3. Thessaloniki traffic data

Hellenic Institute of transport (CERTH-HIT) provides plethora of open available data about the traffic in Thessaloniki<sup>47</sup>. Historical data is available from 2017 onwards. An example of a message is provided bellow. Interested readers are referred to deliverable D4.3 for detailed description and visualizations of the data.

```
{
  "recorded_timestamp": "2019-03-12 10:15:01.950",
  "lon": "22.94646",
  "lat": "40.63049666666667",
  "altitude": "3.3",
  "speed": "4",
  "orientation": "314.600006103516"
}
```

## 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.

In this section, we intend to focus on the different physical deployment options that we can adopt for PIXEL, both at the level of solution and at the level of hosting of its infrastructure and/or data. First, we will introduce the concept of **environment**. Secondly, we will overlook the **maintenance** of our application. Finally, linked to the concept of deployment we will analyse the **scalability** and the **deployment architecture**.

By scalability we mean the ability of a system to preserve its average performance as the number of target elements (requests, nodes, services, etc.) increases. Another definition is the way to manage the growth of a system so that the quality of it (services offered) is not affected. Thus, the scalability of the PIXEL solution will be a determining factor when talking about the different points.

<sup>44</sup> <https://scihub.copernicus.eu/>

<sup>45</sup> <https://www.sentinel-hub.com/>

<sup>46</sup> <https://sentinel.esa.int/web/sentinel/user-guides/sentinel-2-msi/data-formats>

<sup>47</sup> <http://opendata.imet.gr/organization/hit>

## 6.1. Environments and testing

In traditional developments we 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 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.) are 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 malfunctioning of the systems.

In PIXEL, there will be deployed at least the local, development and integration environments, as well as the production provided by ports. By the time of writing this document the two first are already being used, and the third will start as soon as the work package 7 will kick-off (month 13).

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. As the scope of an IoT development is wide and complex traditional test are not enough (different environments) because of connectivity, usability and security reason among others.

In this project, the testing principles followed will be<sup>48</sup>:

- **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**<sup>49</sup>/**SSL**<sup>50</sup>).
- **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.
- **Regulatory or legislative testing.** It is important to check that an application passes the different regulatory / compliance checkpoints.

<sup>48</sup> [https:// www.softwaretestinghelp.com/internet-of-things-iot-testing/](https://www.softwaretestinghelp.com/internet-of-things-iot-testing/)

<sup>49</sup> <https://www.globalsign.com/en/blog/ssl-vs-tls-difference/>

<sup>50</sup> <https://www.digicert.com/ssl/>

- **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.

## 6.2. Scalability. Multi-Instance vs Multi-Tenant

These two terms refer to architectural principles related to deployment. What are the characteristics of each of them?:

- **Multi-tenancy**<sup>51</sup>: 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.

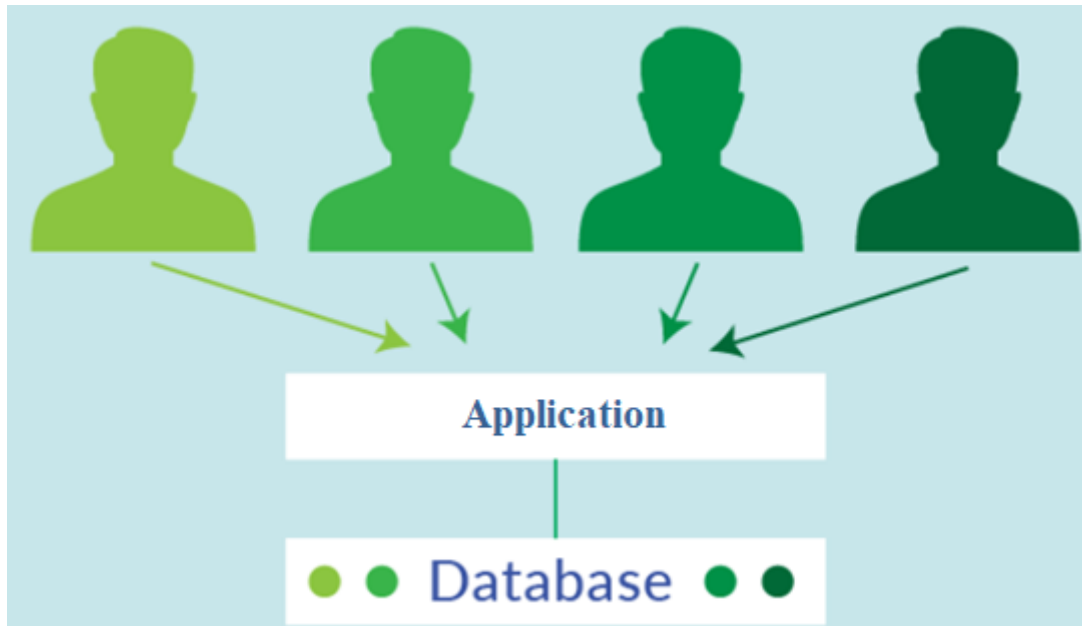


Figure 50 Multi-tenancy

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.
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.

<sup>51</sup> [https:// platzi.com/blog/multi-tenant-que-es-y-por-que-es-importante/](https://platzi.com/blog/multi-tenant-que-es-y-por-que-es-importante/)

- **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:

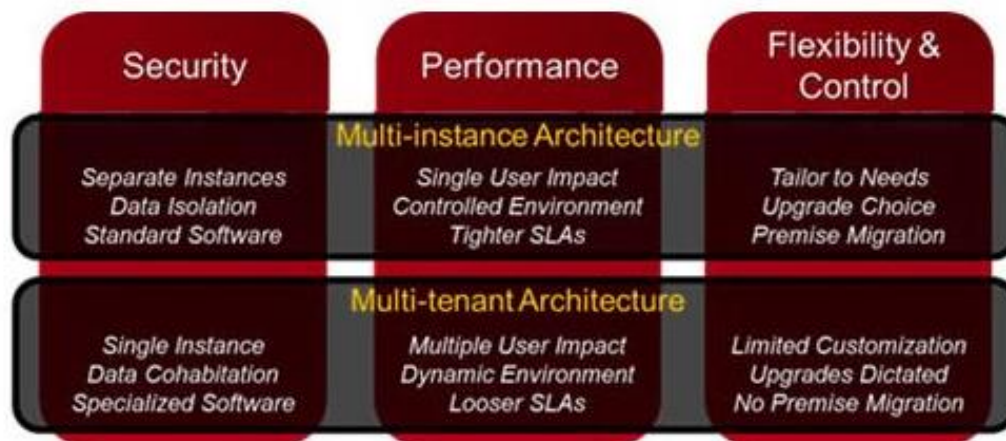


Figure 51: Characteristics of multi-Tenant and multi-Instance

### 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<sup>52</sup>. 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.

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.

<sup>52</sup> <https://fayebg.com/2013/05/multi-tenancy-vs-multi-instance/>



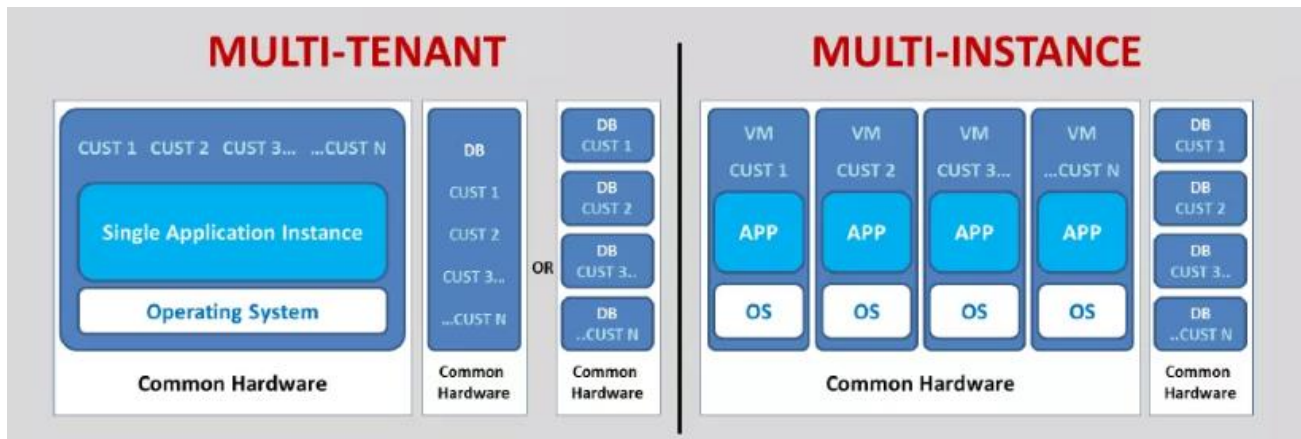


Figure 52 Differences between Multi-Tenant and Multi-Instance architecture<sup>53</sup>

### 6.2.1. Multi-instance deployment of PIXEL Information Hub

As an example of the multi-instance strategy, here is the deployment diagram of the PIXEL Information Hub, which bases the isolation and multi instance in containerization.

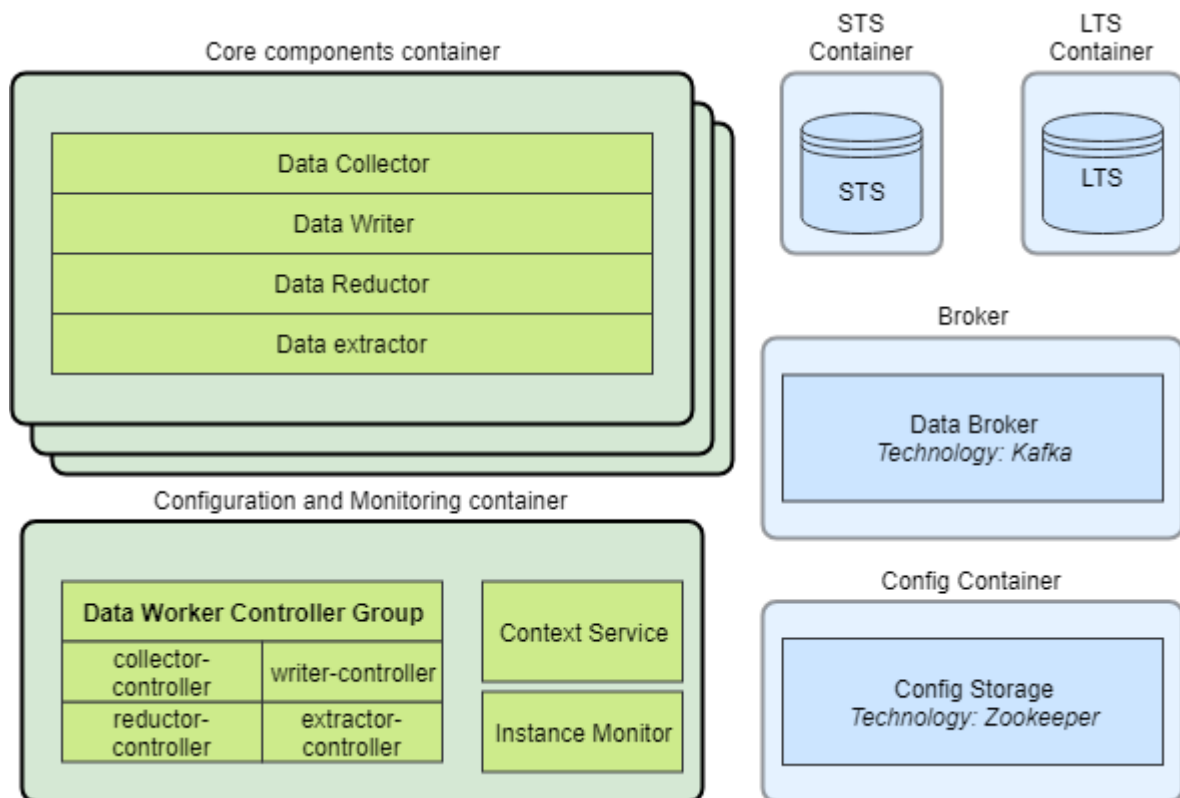


Figure 53: Example of deployment for PIXEL Information Hub

<sup>53</sup> [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/](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/)




## 6.3.Maintenance

The maintenance of an application does not finish when the application is on production; on the contrary, this is the moment when the maintenance processes are started to be more demanding in terms of operations and sometimes optimization.

There are several types of maintenance:

- **Corrective/Reactive maintenance.** Bug fixing, emerging patches (updates).
- **Preventive maintenance.** Monitoring system, installing updates, incident logging.
- **Environment support.** Release planning, back up.
- **User support (UCC).**

We are going to focus on preventive maintenance, and which are the main tools to use:

- **Nagios<sup>54</sup>.** Monitoring system released under the GNU General Public License. Nagios is an alert system based on the execution of scripts and their exit codes. Nagios doesn't store data. There are plugins which can store data for visualization (historic data). Nagios provides monitoring of all mission critical infrastructure components including applications, services, operating systems, network protocols, systems metrics and network infrastructure.  *Figure 54: Nagios Logo*
- **Prometheus<sup>55</sup>:** Open source systems monitoring, and alerting toolkit originally built at SoundCloud. It collects metrics from configured targets at given intervals, evaluates rule expressions, displays the results, and can trigger alerts if some condition is observed to be true. Prometheus is composed of multiple tools: *exporters* (to export local metrics), *Alertmanager* (to trigger alerts based on those metrics), *Grafana* (for the dashboards), *PromQL* (Query language used to create dashboard and alerts).  *Figure 55: Prometheus Logo*
- **Sensu<sup>56</sup>.** Monitoring framework that aims to be simple, malleable and scalable. Sensu takes the results from the execution of scripts. If certain conditions are met this will raise alerts to notify the user.
- **Pandora FMS.** Open Source software that monitorize and measures all kind of elements (systems, applications or network devices). It allows to know the state of each element of a system along the time since it has historical data and event. Pandora FMS is oriented to large environments, and allows to manage with and without agents, several thousands of systems, so it can be used in large clusters, data centers and networks of all kinds.  *Figure 56: Pandora FMS logo*

For all this, we have selected Nagios as tool for the primary predictive maintenance of PIXEL. However, other solutions as the listed above will be tested when the development is more advanced to have a complete assessment of the current state of the art and looking for a better user experience.

<sup>54</sup> <http://www.nagios.com>

<sup>55</sup> <https://prometheus.io>

<sup>56</sup> <https://sensu.io>



## 6.4. Deployment architecture. Cloud vs On-premises vs Hybrid

One of the first steps in any development should be to select the type of implementation<sup>57</sup>: 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 an 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 expensive than internal servers<sup>58</sup>. 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 in 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-premise 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.

In PIXEL, we are going to use very typically mixed or hybrid solutions. The reason for this is that the ports have different policies for their data protection, as well as very different strategies for integration or service sharing. Developing with the objective of being hybrid implies that most of the modules can be deployed either in the cloud or on premises without affecting the features of the platform. This provides a wider flexibility needed by a transversal software such as the PIXEL platform, that can be used for several different purposes and

<sup>57</sup> <https://www.telehouse.net/resources/blog/september-2018/on-premises-vs-cloud-vs-colocation>

<sup>58</sup> <https://servicemuse.com/cloud-vs-on-premises-vs-hybrid/>

be attached to very heterogenous data sources. To achieve this, PIXEL will strongly rely in containerization in order to be as much system agnostic as possible, using market standards as Docker<sup>59</sup> or Kubernetes<sup>60</sup>.

Among container managers, **Docker** ([www.docker.com](http://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 on top 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.

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.

**RKT** was another container manager developed by CoreOS which was purchased by Red Hat in 2018.

**Apache Mesos** (<http://mesos.apache.org/>) is an open source cluster manager with high scalability, supporting various kinds of workloads (e.g. Hadoop, cloud native apps, etc.). The main components are: Agent nodes, master, zookeeper and frameworks. Mesos typically runs with Marathon (<https://mesosphere.github.io/marathon/>), which provides scheduling functionalities and building therefore a container orchestration platform.

However, the most widely used implementation for container orchestration is **Kubernetes (K8s)** (<https://kubernetes.io/>), an open-source system for automating deployment, scaling and management of containerized applications. This software requires relatively high infrastructure requirements (various powerful hosts with plenty of memory) as it is devoted to scale for hundreds/thousands of nodes. For small environments or test scenarios with a small set of nodes, it is possible to use a reduced version of Kubernetes, called **Minikube** (<https://kubernetes.io/docs/setup/minikube/>).

Other container orchestration alternatives, such as **Docker swarm** (<https://docs.docker.com/engine/swarm/>), are not widely used and do not have the same amount of functionalities as Kubernetes. **Amazon ECS** (EC2 Container Service), on the contrary, is not open and can only be used when purchasing Amazon services. Anyway, and considering the wide adoption of Kubernetes, Amazon developed in the recent years a Kubernetes-compliant version called Amazon EKS (Elastic Container Service for Kubernetes).

The following table summarize all we have seen about container tools:

*Table 8: For container management we have the following summary table:*

	Containers	Containers orchestration
<b>Function</b>	Keep software isolated	Connect containers among each other and to the outside world
<b>Alternatives</b>	Virtual machines Software installation within the same OS	Scripts Manual configuration between static containers
<b>Packages vendors</b>	Docker RKT LXC/LXD MESOS	<b>Kubernetes</b> Docker swarm Amazon ECS MESOS

<sup>59</sup> <https://www.docker.com/>

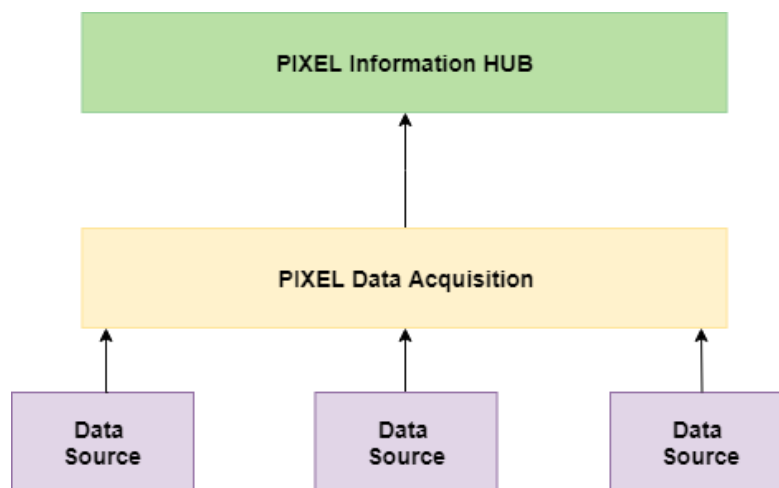
<sup>60</sup> <https://kubernetes.io/>

As the project has different deployment models in the different ports, we will leverage this circumstance to propose deployment standards based on the examples of the participant ports to have precise guidelines for typical scenarios. These standards templates will be reported in the final version of the document.

## 7.State of the art and technological choices

### 7.1.PIXEL Data acquisition layer

The purpose of the data acquisition layer is to provide a standard way to import the data from existing data sources and to push them on the PIXEL Information Hub.



*Figure 57: Integration between PIXEL Information Hub, PIXEL Data Acquisition and different Data sources*

Data sources could be any kind of information useful for PIXEL:

- Port ICT legacy. Ports have their own information solution that could provide a lot of usable data for PIXEL.
- IoT Platform. To measure environmental information, ports could deploy some sensors managed by the IoT Platform of their choice.
- External Services. Services provide some information about ship position, environmental data like air quality ...

As we could have several kinds of sources that provide the same data with different formats, the Data Acquisition layer has to provide a standard view of each data type in order to import them in the same way on the PIXEL Information Hub.

In the meantime, it should also allow to connect standard data source quickly on the system and to share data acquisition solution between ports and cities.

FIWARE provides software components, data model and a community to achieve those objectives. The FIWARE solution is designed to facilitate the sharing of information between several sources to allow the build of smart services. The FIWARE community works together to define shared data model for several applications. They already have defined some data models for environmental applications and transportation on road. The models are easy to extend for the PIXEL purpose and the sharing with the community will allow improving them.

The FIWARE Marketplace will also allow using existing NGSI Agents to connect to standard data sources and to share them with the community.

## 7.2. PIXEL Information Hub

### 7.2.1. Information Hub architecture approaches

This section provides information about architectural patterns and architectures utilizing those patterns. Architectural patterns have been analysed from the information flow perspective e.g. how different parts of a larger system will communicate between each other. Two architectural patterns may be suitable for the PIXEL Information Hub system.

First is the **Message Broker** (“hub”) architectural pattern, also known as the Integration Broker or Interface Engine. Message broker is a mediator for communication between applications. It is used for message validation, transformation and routing between multiple applications. It minimizes mutual awareness between applications, which exchange messages among themselves.

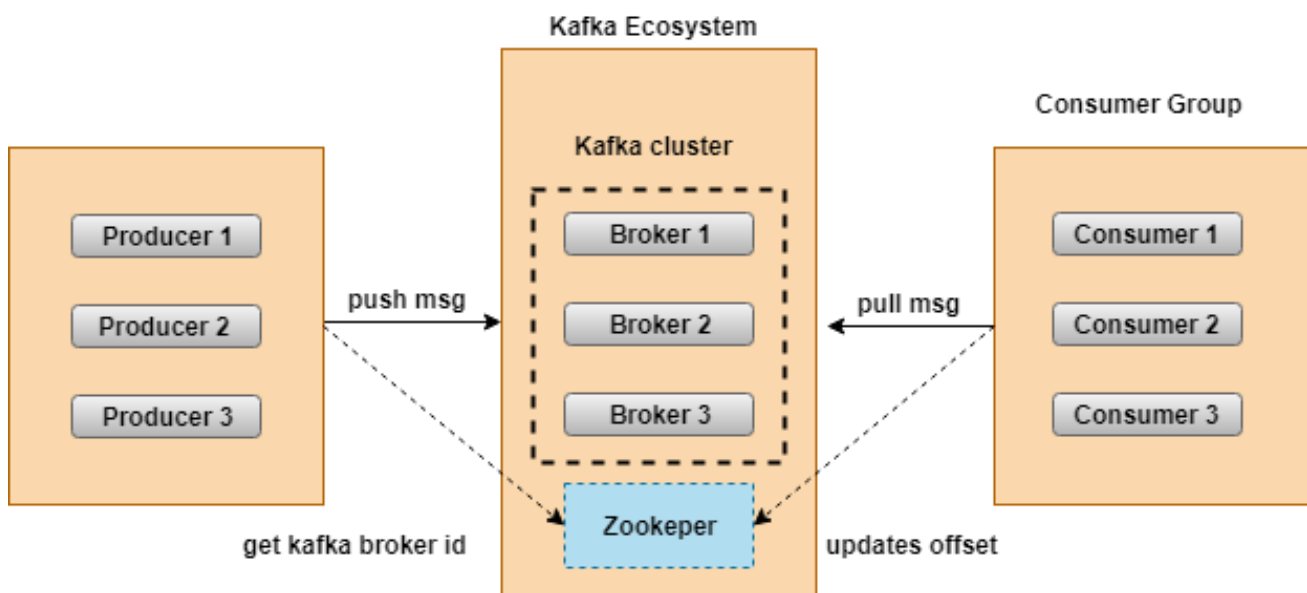


Figure 58: Example of a Kafka cluster architecture<sup>61</sup>

Although message routing and point-to-point communication is the main function of message brokers, they usually provide additional benefits. They can route messages to more than one destination, which covers a wide array of real-life use cases. They can also transform messages to an alternative representation if needed. They can perform aggregations, decompose messages into multiple ones, send them to their destinations and then recompose the responses into one message and return it to the user. Some brokers can interact with external repositories to augment messages with additional data or to store them. They can also invoke web services to receive data. All these features make message brokers suitable for a wide variety of use cases, which is also evident from their popularity.

Typical benefits of message brokers are better performance as Enterprise Service Bus<sup>62</sup> (ESB), message brokers are always asynchronous, and they are reliable, as the data is persistent in its message queues. Message brokers<sup>63</sup> are very well suited to granular scalability as well as they simplify decoupling. However, they do introduce some operational complexity because of creation, configuration and/or monitoring of queues. In some cases, message delivery is not guaranteed by default and has to be taken into account by the developer<sup>64,65</sup>.

<sup>61</sup> <https://www.digitalvidya.com/blog/kafka-architecture/>

<sup>62</sup> [https://en.wikipedia.org/wiki/Enterprise\\_service\\_bus](https://en.wikipedia.org/wiki/Enterprise_service_bus)

<sup>63</sup> [https://en.wikipedia.org/wiki/Message\\_broker](https://en.wikipedia.org/wiki/Message_broker)

<sup>64</sup> [https://en.wikipedia.org/wiki/Message\\_broker](https://en.wikipedia.org/wiki/Message_broker)

<sup>65</sup> <https://historicalmodeling.com/distributed-systems/message-queues.html>

The second architectural pattern is the Enterprise Service Bus (ESB). ESB is analogous to the bus concept found in computer hardware architecture. General purpose of the ESB is integration of loosely coupled software components, or services, which can be independently deployed and running.

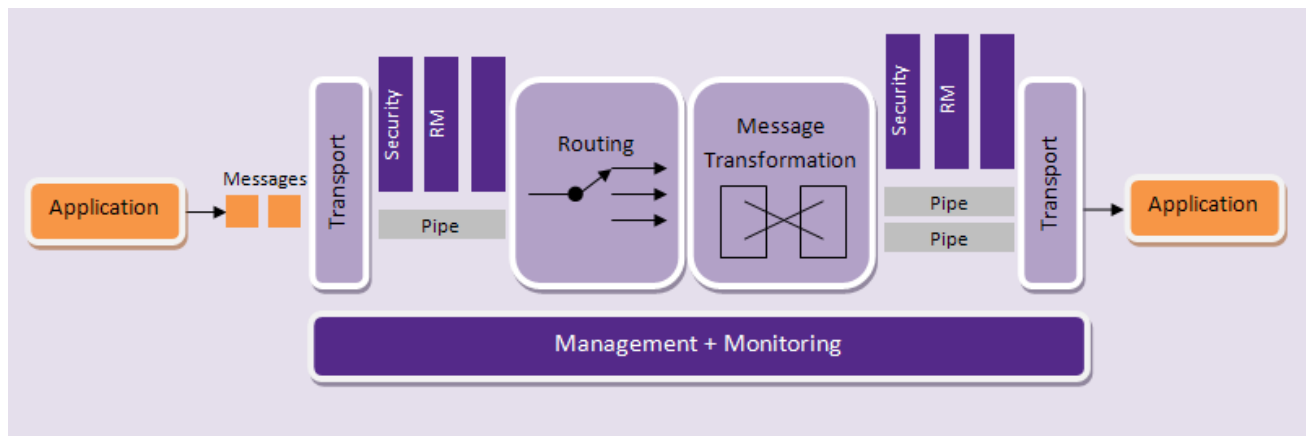


Figure 59: Example of a WSO2 ESB architecture<sup>66</sup>

**ESB** implements a communication system between mutually interacting software applications, which conform to a special variant of the client-server model, where generally any application can behave as a server or client. Agility and flexibility are promoted with regard to high-level protocol communication between integrated applications. This high-level protocol communication allows for implementation of the enterprise application integration (EAI) architecture. Besides EAI architecture, ESB is a common pattern found in Service-Oriented Architecture <sup>67</sup>(SOA).

There are several primary duties of ESB. They include monitoring and controlling messages between services, resolving contention between communicating service components, control deployment and versioning of services and marshal use of redundant services. ESB implementations usually provide commodity services like event handling, data transformation and mapping, message and event queueing and sequencing, security or exception handling and protocol conversion among others<sup>68 69</sup>. One would benefit from ESB by being able to scale from point-solutions to enterprise-wide deployments without major trouble. Using ESB to integrate your services offers much higher configurability than using traditional integration coding. Another benefit would be not utilizing a central broker or central rules engine. Generally, ESB is a loosely coupled system which allows easy plug-in and plug-out of services<sup>70 71</sup>. However, it does not offer the communication speeds of other solutions, such as message brokers. It also acts as a single point of failure, which can bring down communication across the whole Enterprise, which is further emphasised by high configuration and maintenance complexity.

These two architectural patterns are most commonly leveraged in the development of distributed systems in form of two architectures: Service-oriented Architecture (SOA), and Enterprise Application Integration (EAI).

**EAI**<sup>72</sup> is an integration framework composed of a collection of technologies, which form a middleware framework to enable integration of systems and applications. As many types of business software are unable to communicate with each other in order to share data or business rules, EAI is the process of linking such

<sup>66</sup> <https://docs.wso2.com/display/ESB460/ESB+Architecture>

<sup>67</sup> [https://en.wikipedia.org/wiki/Service-oriented\\_architecture](https://en.wikipedia.org/wiki/Service-oriented_architecture)

<sup>68</sup> [https://en.wikipedia.org/wiki/Enterprise\\_service\\_bus](https://en.wikipedia.org/wiki/Enterprise_service_bus)

<sup>69</sup> <https://aws.amazon.com/message-queue/benefits/>

<sup>70</sup> <https://www.tibco.com/blog/2015/03/25/integration-broker-or-enterprise-service-bus/>

<sup>71</sup> <https://www.linkedin.com/pulse/enterprise-service-bus-vs-message-brokers-eai-soa-anuj-varma/>

<sup>72</sup> <https://www.tibco.com/blog/2015/03/25/integration-broker-or-enterprise-service-bus/>

applications with a single organization. Applications can be linked either at the back-end via APIs or (rarely) via front-end (GUI).<sup>73 74</sup>

**SOA** offers point-to-point interactions between applications. Services are provided to other components through a communication protocol over a network. The basic principles are independent from vendors, products and technologies. Services (discrete units of functionality) can be accessed remotely and acted upon, updated independently. Moreover, different services can be used in conjunction to provide the functionality of large software applications.<sup>75</sup>

PIXEL Information Hub will use a message broker (“hub”) architectural pattern to implement the integration of different components. This approach would allow better performance than an ESB approach.

## 7.2.2. Technological choices

### 7.2.2.1. Data/message broker

Message brokers are important components of modern web technology. They allow asynchronous communication between components of a software architecture and thus enhance the scalability of the aforementioned architecture. They achieve this by a publish-subscribe paradigm. This means both sender and receiver are loosely coupled and do not need to be online at the same time to communicate.<sup>76</sup> There are a lot of message brokers to choose from: Apache Kafka, RabbitMQ, ActiveMQ, ZeroMQ and more. When comparing message brokers there are three important points to look at. First is the message broker behaviour when there is a backlog of messages. Second is the ability to create a cluster. Third is the ability to protect data without blocking publishers if there is a failure on a node in the cluster.<sup>77</sup> When comparing message brokers for use in the PIXEL project we focused on the two most popular ones: Apache Kafka and RabbitMQ. These two message brokers are quite different to each other, but still serve the same purpose. We will go through some pros and cons of both message brokers.

**Apache Kafka** is a message broker best suited for scenarios where less complex routed streaming with high throughput is required. This usually means A → B streaming with throughputs in excess of 100k messages per second.<sup>78</sup>

Apache Kafka is also an obvious choice when stream replays are needed and when access to stream history is required, as it keeps the history of messages, at least to some point. It can achieve this by using offset based message consumption. Kafka implements the “dumb broker/smart consumer” model, which means the broker does not track which messages were read by consumers. It keeps all messages in queues for a fixed time duration, or until a storage limit is exceeded. This also means Kafka does not have any overhead operations of tracking consumer states.

However, there are some downsides of Apache Kafka. For one, it requires Apache Zookeeper for its operation, especially for clustering. This also raises the hardware requirements when running a Kafka cluster, as it needs running Zookeeper instances. For example, a three node Kafka cluster would consist of ~8 nodes, as ~4 nodes will be required for Zookeeper. It has no web frontend for management, so all the configuration has to be done through CLI and configuration files. Because of the “dumb broker/smart consumer” approach, it needs external tools for monitoring. As it is written in Java, adoption of clients in other languages is still in early stages, so use of Java clients is advisable.

**RabbitMQ** on the other hand, is focused on consistent delivery of messages.

---

<sup>73</sup> [https://en.wikipedia.org/wiki/Enterprise\\_application\\_integration](https://en.wikipedia.org/wiki/Enterprise_application_integration)

<sup>74</sup> [https://en.wikipedia.org/wiki/Business\\_rules\\_engine](https://en.wikipedia.org/wiki/Business_rules_engine)

<sup>75</sup> [https://en.wikipedia.org/wiki/Service-oriented\\_architecture](https://en.wikipedia.org/wiki/Service-oriented_architecture)

<sup>76</sup> <https://medium.com/@aselarbd/message-brokers-and-brief-comparison-of-them-part-1-4c573cdcc50f>

<sup>77</sup> <http://teckhike.com/message-broker-comparison-rabbitmq-kafka-activemq-kestrel/>

<sup>78</sup> <https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka>



It is most suited for applications where complex routing to consumers is needed, usually integration of multiple services or apps with non-trivial routing. It also supports multiple protocols, such as AMQP 0-9-1, STOMP, MQTT, AMQP 1.0. RabbitMQ, unlike Kafka, employs the “smart broker/dumb consumer” approach.<sup>10</sup> This means the broker makes sure the messages are delivered to the consumers and dequeues them only when it gets acknowledgement from all the consumers that were reading that message. Hence, it offers finer-grained consistency control/guarantee on a per-message basis. This can also be achieved in Apache Kafka, but with use of external libraries for transaction support. RabbitMQ is easy to setup and easy to migrate. It has a large and growing community, so support and documentation are one of the best. It has a large pool of available plugins, which allow RabbitMQ to handle even more use cases and different integration scenarios. It does provide a HTTP API, along with a browser-based management UI. Besides that, it also provides CLI tools and an API for monitoring, audit and troubleshooting. It includes support for LDAP and other external HTTPS-based providers. It also supports x509 certificate authentication, while other forms can be easily developed by using plugins.

RabbitMQ does have some downsides. A major one is a lack of stream history. If a message is marked as read by all consumers, it is deleted. This can be overcome by using external tools, but is not a part of RabbitMQ itself. In addition, RabbitMQ does not perform as well as Kafka does. In some tests, it reportedly achieves lower throughputs than Kafka does, 20k/s vs 100k/s message throughput. One more drawback compared to Kafka is that RabbitMQ does not support multiple consumers for one message without routing that message to multiple queues. When developing a complex system, this can complicate things even further.<sup>79</sup>

*Table 9: Shows the most popular message broker systems in 2018*

MBS	Vendor	URL	License
Apache Kafka	Apache Software Foundation	<a href="https://kafka.apache.org/">https://kafka.apache.org/</a>	Apache License 2.0
RabbitMQ	Pivotal Software	<a href="https://www.rabbitmq.com/">https://www.rabbitmq.com/</a>	Mozilla Public License
ActiveMQ	Apache Software Foundation	<a href="http://activemq.apache.org/">http://activemq.apache.org/</a>	Apache License 2.0
ZeroMQ	iMatix	<a href="http://zeromq.org/">http://zeromq.org/</a>	LGPL with static linking exception

For the needs identified in the PIXEL project, the decision made was to use Apache Kafka. The main reasons relate to a relatively simple routing scenario of the Information Hub architecture, while high throughput may be required. As Apache Zookeeper is going to be used as configuration storage for the system, it would not pose additional burden on system requirements.

#### 7.2.2.2. Distributed coordination system

The norm in building large software systems is high availability. This usually means that the system is distributed across many nodes, even across the globe, but acts as one single machine to the outside world. Communication and information gathering in such distributed systems of nodes can be a pain and can flood the internal network. This is the reason for implementing a mediator into the system, which will act as a hub for communication. To avoid that hub being a single point of failure it is usually made highly available through replication. These hubs are called Distributed Coordination Systems and they handle all the complexities added by different nodes providing (same) services. These systems enable nodes to share configuration, variables, locks and distributed data stores, all in runtime.<sup>80</sup>

*Table 10: Shows the most popular distributed coordination systems in 2018*

DCS	Vendor	URL	License
-----	--------	-----	---------

<sup>79</sup> <https://itnext.io/kafka-vs-rabbitmq-f5abc02e3912>

<sup>80</sup> <https://medium.com/@Imesha94/distributed-coordination-5eb8eabb2ff>

Zookeeper	Apache Foundation	Software	<a href="https://zookeeper.apache.org/">https://zookeeper.apache.org/</a>	Open Source
Consul	HashiCorp. Inc.		<a href="https://www.consul.io">https://www.consul.io</a>	Mozilla Public License 2.0 and commercial
Etdcd	CoreOS		<a href="https://coreos.com/etcd/">https://coreos.com/etcd/</a>	Open Source
Serf	HashiCorp		<a href="https://www.serf.io/">https://www.serf.io/</a>	Open Source
Eureka	Netflix		<a href="https://github.com/Netflix/eureka/wiki">https://github.com/Netflix/eureka/wiki</a>	Open Source

While there are several options available, for the PIXEL Information Hub Zookeeper has been selected as a most suitable solution, due to its bundling with Kafka, its maturity, robustness and feature richness.

### 7.2.2.3. Storage

There are several categorisations<sup>81</sup> used for different storage/database mechanisms, but from the point of view of modern trends in database developments<sup>82</sup>, the SQL/NoSQL distinction is the most prominent feature when selecting a database.

Relational databases (SQL) are a mature technology that is widely known and understood. There is a multitude of products available on the market. They are mostly general-purpose, however, they may be tuned for specific usage scenarios.

While No-SQL databases are a relatively novel technology, they have been gaining attraction because in many scenarios they overcome limitations posed by traditional SQL approaches. Most importantly, they provide much more flexibility as what regards stored data structures and types.

Following sections provide a comparison of those two database types, with particular focus on features relevant for the PIXEL Information Hub. The comparison is based on *A Review of Different Database Types: Relational versus Non-Relational*<sup>83</sup> and documentation provided by database vendors.

SQL databases store data as rows and columns in tables. Each row in a table has its own unique key. Rows in a table can be linked to rows in other tables by adding a column for the unique key in the linked row. Such columns are known as foreign keys. When using relational databases, we want to be able to select or modify one and only one row in a table. Relational databases allow this by implementing a unique primary key for each row in a table. When a new row is written to the table, a new unique value for the primary key is generated. This key is used by the system primarily for accessing the table. These primary keys are also used within a database to define the relationships among tables.

SQL databases have several positive qualities emerging from their maturity. They are well-documented, mature technologies with a well-defined SQL standard. They are widely accepted, becoming a synonym for “a database”. For this reason, there is a large pool of knowledge and use support available.

Another important property is that they are ACID compliant, which means that there is a guarantee the database is always kept in a consistent state, even in case of transaction.

(or other) failures. This property comes with cost in complexity and performance, thus even in some SQL implementations there are options to relax the ACID rule in order to gain on performance.

The main shortcoming of SQL databases is the ability to work well only with well-structured data, while unstructured or semi-structured poses big challenges due to schema and type constraints. These constraints are also related to a problem with mapping to object and classes representing the data. As the mapping is often not to one-to-one, this presents difficulties in data interpretation as well as implementation of persistence

<sup>81</sup> Types of databases: <https://www.tutorialspoint.com/Types-of-databases>

<sup>82</sup> The Types of Modern Databases: <https://www.alooma.com/blog/types-of-modern-databases>

<sup>83</sup> <https://www.dataversity.net/review-pros-cons-different-databases-relational-versus-non-relational>



mechanisms for classes defined in programming languages. Because of all these constraints, complex datasets are difficult to handle.

Another shortcoming is scalability, where they do not scale well horizontally. Horizontal scaling is in essence adding more machines to your pool of resources. In the database world this usually means partitioning the data in such a way, each node only contains a part of the whole data. On the other hand, vertical scaling means adding resources to a single machine in form of CPU and RAM. Scaling vertically is limited by the expandability of a single machine (max CPU cores/slots, max supported RAM). Scaling horizontally is generally easier to achieve dynamically by adding machines into the existing pool. This can be done without downtime and any other major impacts on performance. Vertical scaling on the other hand usually means some downtime, at least for the machine being upgraded.

Both constraints, handling only structured data and demanding approaches to horizontal scalability, are the reason why the **SQL databases are not suited for large analytics or IoT workloads**.

*Table 11: Shows the most popular RDBMS systems in 2018 according to the DB-Engines ranking<sup>84</sup>*

DBMS	Vendor	URL	License
Oracle	Oracle	<a href="http://www.oracle.com/-database">www.oracle.com/-database</a>	Commercial and restricted free version
MySQL	Oracle	<a href="http://www.mysql.com">www.mysql.com</a>	GPL - open source and commercial
Microsoft SQL Server	Microsoft	<a href="https://www.microsoft.com/en-us/sql-server/sql-server-2019">https://www.microsoft.com/en-us/sql-server/sql-server-2019</a>	Commercial and restricted free version
PostgreSQL	PostgreSQL Global Development Group	<a href="https://www.postgresql.org">https://www.postgresql.org</a>	BSD – open source
IBM Db2	IBM	<a href="https://www.ibm.com/analytics/us/en/db2/">https://www.ibm.com/analytics/us/en/db2/</a>	Commercial and restricted free version

Storage systems grouped under the umbrella of the “NoSQL” term are in fact a set of very diverse approaches. They have reached different levels of maturity and have been designed to be applied in different application domains. For this reason, in the following sections, they are analysed by implementation approaches: key-value stores, wide column stores, document stores, graph databases and search engines.

In principle, all NoSQL databases can be schema agnostic thus allowing efficient storage of unstructured/semi-structured data. They are generally more horizontally scalable and fraud-tolerant than relational databases and can be easily distributed across different nodes. The downsides are weaker or eventual consistency instead of ACID, denormalized data, which usually means mass updates and being less widely adopted than relational database solutions.

Key-value stores work by storing data in, what is today more commonly known as dictionaries or hash tables. These contain a collection of objects/records and are stored and retrieved using a key that uniquely identifies one record. They work in a very different manner to relational databases, as they do not predefine the data structure, instead treating data as a single opaque collection, which may have different fields for every record. They are good for non-complex data where speed is important.

*Table 12: Shows the most popular Key-value stores in 2018 according to the DB-Engines ranking*

DBMS	Vendor	URL	License
------	--------	-----	---------

<sup>84</sup> <https://db-engines.com/en/ranking/relational+dbms>

Redis	Salavatore Sanfilippo	<a href="https://redis.io/">https://redis.io/</a>	BSD – open source and commercial
Amazon DynamoDB	Amazon	<a href="https://aws.amazon.com/dynamodb/">https://aws.amazon.com/dynamodb/</a>	Commercial

Wide column stores are essentially multi-dimensional key-value stores. They are well suited for scaling in massive distributed systems. Some of them use CQL, which is a variant of SQL, for data definition and manipulation.

*Table 13: Shows the most popular Wide column stores in 2018 according to the DB-Engines ranking*

DBMS	Vendor	URL	License
Cassandra	Apache Software	<a href="http://cassandra.apache">http://cassandra.apache</a>	Open Source
HBase	Apache Software	<a href="https://hbase.apache.org">https://hbase.apache.org</a>	Open Source
Datastax Enterprise	DataStax	<a href="https://www.datastax.com/products/datastax-enterprise">https://www.datastax.com/products/datastax-enterprise</a>	Commercial
Accumulo	Apache Software	<a href="https://accumulo.apache.org/">https://accumulo.apache.org/</a>	Open Source
Scylla	ScyllaDB	<a href="https://www.scylladb.com/">https://www.scylladb.com/</a>	Open Source

Document stores are schema-free where data is stored as JSON documents. They are similar to key-value stores in the way that they use document names as keys contents as values. These databases are well suited for managing semi-structured data in distributed systems.

*Table 14: Shows the most popular Document stores in 2018 according to the DB-Engines ranking*

DBMS	Vendor	URL	License
MongoDB	MongoDB, Inc	<a href="https://www.mongodb.com/">https://www.mongodb.com/</a>	Open Source
Couchbase	Couchbase, Inc.	<a href="https://www.couchbase.com/">https://www.couchbase.com/</a>	Open Source
CouchDB	Apache Software Foundation	<a href="http://couchdb.apache.org/">http://couchdb.apache.org/</a>	Open Source
MarkLogic	MarkLogic Corp.	<a href="https://www.marklogic.com/">https://www.marklogic.com/</a>	Open Source
OrientDB	OrientDB LTD; CallidusCloud	<a href="https://orientdb.com/">https://orientdb.com/</a>	Open Source

Graph databases represent data as a network of nodes/objects. These databases are used mostly when an in-depth analysis of relationships between data points is required.

*Table 15: Shows the most popular Graph databases in 2018 according to the DB-Engines ranking*

DBMS	Vendor	URL	License
Neo4J	Neo4j, Inc	<a href="https://neo4j.com/">https://neo4j.com/</a>	Open Source
OrientDB	OrientDB LTD; CallidusCloud	<a href="https://orientdb.com/">https://orientdb.com/</a>	Open Source
ArangoDB	ArangoDB GmbH / triagens GmbH	<a href="https://www.arangodb.com/">https://www.arangodb.com/</a>	Open Source

Virtuoso	OpenLink Software	<a href="https://virtuoso.openlinksw.com/">https://virtuoso.openlinksw.com/</a>	Open Source
----------	-------------------	---	-------------

Search engines are schema-free NO-SQL databases. Data is stored as JSON documents similar as in document stores, but with emphasis on making data easily accessible via text-based search.

*Table 16: Shows the most popular Search engines in 2018 according to the DB-Engines ranking*

DBMS	Vendor	URL	License
Elasticsearch	Elastic	<a href="https://www.elastic.co/products/elasticsearch">https://www.elastic.co/products/elasticsearch</a>	Open Source
Splunk	Splunk, Inc.	<a href="https://www.splunk.com/">https://www.splunk.com/</a>	Commercial
Solr	Apache Software Foundation	<a href="http://lucene.apache.org/solr/">http://lucene.apache.org/solr/</a>	Open Source
MarkLogic	MarkLogic, Inc.	<a href="https://www.marklogic.com/">https://www.marklogic.com/</a>	Commercial
Sphinx	Sphinx Technologies, Inc.	<a href="http://sphinxsearch.com/">http://sphinxsearch.com/</a>	Open Source

Time series database management systems are optimized for handling time series data, which are most commonly arrays of numbers indexed by time. These fields are sometimes referred to as profiles, curves or traces. These database management systems allow users to create, enumerate, update and destroy various time series and organize them. They allow certain basic calculations that work on a series as a whole; multiplication, addition, or otherwise combining data from various time series into a new time series. Ideally, repositories of time series database management systems are natively implemented using specialized database algorithms, but it is possible to save time series as a BLOB in a conventional relational database albeit at a cost of performance.

85

*Table 17: Shows the most popular time series database management systems in 2018 according to the DBEngines ranking*

DBMS	Vendor	URL	License
InfluxDB	InfluxData	<a href="https://www.influxdata.com/time-series-platform/influxdb/">https://www.influxdata.com/time-series-platform/influxdb/</a>	Open Source
KDB+	Kx Systems	<a href="https://kx.com/">https://kx.com/</a>	Commercial
Graphite	Chris Davis	<a href="https://github.com/graphite-project/graphite-web">https://github.com/graphite-project/graphite-web</a>	Open Source
RDDtool	Tobias Oetiker	<a href="https://oss.oetiker.ch/rrdtool/">https://oss.oetiker.ch/rrdtool/</a>	Open Source
Prometheus	Prometheus.io	<a href="https://prometheus.io/">https://prometheus.io/</a>	Open Source

For the implementation of the PIXEL Information Hub STS and LTS storages, Elasticsearch has been identified as a most suitable candidate. If a requirement for a different type of database storage is identified, one more suited for data structures provided by the PIXEL Data Acquisition layer, an implementation of alternate database sources will be assessed.

Elasticsearch is a mature open-source document-store, which is distributable and scalable to support different PIXEL scenarios. As a JSON-based storage with extensive REST API support, it will enable easy integration with downstream PIXEL components. It can also act as a key-value store for fast data access. Although several

<sup>85</sup> [https://en.wikipedia.org/wiki/Time\\_series\\_database](https://en.wikipedia.org/wiki/Time_series_database)

other analysed solutions may fit most of the above features, its large user community and number of support and hosting options will help lower operational costs in production scenarios.

## 7.3. PIXEL Operational Tools

### 7.3.1. Operational Tools SotA

Operational Tools comprise a set of tools that facilitate management operations (in various fields) for workers, helping a company staying in business and being more cost-effective. The scope of this definition is quite wide and varies from company to company, even from department to department. The main idea is to automate actions or operations that can be easily configured from (skilled) workers in a uniform way. This means:

- Identifying current available tools able to provide a specific service useful for a company.
- Providing an entry point to access all of these services, for configuration, execution and monitoring purposes. If possible, and probably with some limitations:
  - Provide a uniform interface (e.g. similar way for entering data).
  - Provide increased functionalities such as scheduling of activities and catenation of services (the output of one service serves as input for the other).
- Providing a way of enriching the tool set by adding new services
- Documenting all functionalities from a high-level perspective and the required skills from the users

As can be seen, the approach is pretty much similar to an operating system, but only intended for a specific business (operational) scope. Therefore, it does not make too much sense to provide a state-of-the-art section about operational tools, as possibly none of them will be useful for PIXEL.

Operational Tools are typically custom developments highly coupled with the embedded services to be offered, which often serve as the starting point. Considering the situation in PIXEL, the most important services that are being developed within the project are:

- Models: they act as simulators for energy, transport and environment (noise and air pollution), which are being developed within WP4. Even if not initially released as services in WP4, they will be converted into service in WP6 before being consumed by the Operational Tools.
- Predictive algorithms: able to predict future maritime or road traffic and, similar as for models, they will be treated as services in WP6.
- Custom algorithm(s) or process(es) for calculating the PEI (Port Environmental Index): which is a quantitative value; it is possible that either the PEI or part of it can be automatically calculated within a Complex Event Processor (CEP), which continuously consumes data from the PIXEL hub.

On top of such services the OT should be developed, and there is no current tool (to our knowledge) able to cope with this integration. Therefore, we propose a custom development with the following considerations for the components identified and described in section 4.3.3:

- The OT UI should be integrated within the Dashboard as an added functionality to configure, schedule or execute the different models and predictive algorithms.
- The CEP component can be selected from available (open) implementations in the market, no need to develop one from scratch. This will be commented in the next subsection (technological choices).
- The OT data model will have to be designed and accommodated to existing databases (MySQL and/or NoSQL).
- The execution environment is not yet completely defined at the time of writing this deliverable (v1), but will be clearly described in the final version (v2). This is caused because models and predictive algorithms are not yet implemented, and we do not completely know all execution requirements. For example, a service (model) can provide an answer in real time or it make take some time to provide an answer. In the latter case, and considering multiple instances, this may require some containerized

management; furthermore, if models are to be connected, some container orchestration environment might be required.

### 7.3.2. Technological choices

In this section we will focus here only on those components that can be used and adapted for PIXEL purposes: CEP, Database, Execution environment.

Regarding **Complex Event Processors** (CEP), it is important to note that this term sometimes gets mixed in the current literature with stream processing or stream analytics platforms. Technically there is little difference among them as (most of) all are able to process in real time large amounts of data and detect different events. Some years ago, the term *complex event* was more used and currently the term of *insights from stream analytics* is more utilized, primarily in the area of business intelligence. Therefore, the latter ones (stream analytics platform) typically include dashboards and anywhere access whereas the former ones (CEP platforms) were primarily focused on event detection.

Among the top open source stream analytics platforms, we can find:

- Apache Flink (<https://flink.apache.org/>) is an open source platform for distributed stream and batch data processing.
- Apache Spark (<https://spark.apache.org/>) is a unified analytics engine for large-scale data processing. It runs on Hadoop, Apache Mesos, Kubernetes, standalone or in the cloud, and can access diverse data sources.
- Apache Storm (<http://storm.apache.org/>) is a distributed real-time computation system. Storm reliably processes unbounded streams of data, doing for real-time processing what Hadoop did for batch processing.
- WSO2 Complex Event Processor (<https://wso2.com/products/complex-event-processor/>) is able to detect meaningful events and patterns from different data sources, analyse their impacts, and react based on predefined configuration in real time.
- The Perseo (<https://github.com/telefonicaid/perseo-fe/tree/master/documentation>) Generic Enabler (GE) introduces Complex Event Processing (CEP) defined using a rules-based system, enabling you to fire events which send HTTP requests, emails, tweets, SMS messages etc. The GE relates to the series of FIWARE enablers.

The Perseo enabler, even if it is a FIWARE enabler that easily integrates with the Orion Context Broker (most likely to be used in the PIXEL Data Acquisition Layer), it is not of common use, has reduced examples and support and it is not clear if it supports distributed storage system as provided by the PIXEL information hub. Therefore, after an initial analysis it seems more feasible to use WSO2 CEP or any of the other stream analytics platforms from the Apache foundation.

Related to **databases** we have different alternatives: MySQL, NoSQL, Graph-based. They have already been described in section 7.2.3.3. As the amount of information to be stored in the OT database is not huge (not as for the case of the information hub), there is no strong requirement for this database and, after an initial analysis, mostly all existing databases can be used for our purpose. However, it makes sense to use a similar one that is being used in other components to minimize the parsing of data and facilitate the integration. For example, if models are described and providing their outputs in JSON format, it makes sense to use a no-SQL database (e.g. MongoDB).

## 7.4. Dashboard and notifications

### 7.4.1. Dashboard SotA

An IoT platform must not only manage IoT devices and store data but must also provide us with an interface where we can **view** and **analyse** the data from these devices. This is what is known as **Dashboard**.

In PIXEL it is the component that we have called **PIXEL Integrated Dashboard and Notifications**.

There are many IoT platforms on the market, whose functionalities vary enormously.

Although most IoT platforms have a dashboard to display data, some of these platforms are basically just a dashboard where to show device data. So, it's important to distinguish between two very similar concepts: Dashboard and Platforms<sup>86</sup>.

An IoT Dashboard can be considered as a basic IoT platform. The general characteristics of a dashboard are summarized as follows:

- Display data.
- Control devices.

While an IoT platform can:

- Collect data from various sources.
- Store data.
- Control devices.
- Display data.
- Run tests.
- Deploy device updates.
- Manage device inventory.

The above characteristics allow us to draw a first conclusion: **PIXEL is an IoT platform** with all the modules for their own characteristics.

Having seen the difference between IoT platform and IoT Dashboard, identifying that PIXEL as IoT platform should be able to perform all the above tasks. The next step would be to identify possible 'out of box' IoT platforms.

## 7.4.2. Technological choices

There are a variety of free/open IoT platforms on the market that can perform partially or completely functions described in PIXEL use cases. In the following list we will see some of the available ones:

1. **Thingsboard**<sup>87</sup>: An open-source IoT platform whose features cover each layer of the end-to-end platform needs.
2. **ThingWorx**: ThingWork delivers IIoT applications and solutions built on its market-leading technology platform, all designed to accelerate digital transformation<sup>88</sup>.
3. **Thingspeak**<sup>89</sup>: The open IoT platform with MATLAB analytics.
4. **Wolkabout**<sup>90</sup>: An IoT platform to rapidly develop powerful IoT applications and control your business ecosystem. Integrates any device, transforms real-time readings into meaningful data and combines different devices and services into a complete IoT solution.
5. **IOTgo**<sup>91</sup>: An open source IoT platform. It's easy to build our own IoT system by using the platform and open source device firmware.

---

<sup>86</sup> [http:// www.steves-internet-guide.com/iot-mqtt-dashboards/](http://www.steves-internet-guide.com/iot-mqtt-dashboards/)

<sup>87</sup> [https:// thingsboard.io/](https://thingsboard.io/)

<sup>88</sup> [https:// www.ptc.com/en/products/iot/thingworx-platform](https://www.ptc.com/en/products/iot/thingworx-platform)

<sup>89</sup> [https:// thingspeak.com/](https://thingspeak.com/)

<sup>90</sup> [https:// wolkabout.com/](https://wolkabout.com/)

<sup>91</sup> <http://iotgo.iteadstudio.com/>



6. **kaa**<sup>92</sup>: An Enterprise-grade IoT platform built on a modern cloud-native architecture and a fully customizable feature set. Based on microservices.

On the other hand, there are extensive industrial and paid IoT platforms that also cover a variety of requirements of PIXEL:

1. **Azure**<sup>93</sup>: Is a complete cloud platform which facilitates the development of applications as well as allowing to be hosted. Azure integrates in its platform the necessary services to: develop, test and implement applications. In addition to having the advantages of cloud computing. Azure has a portal from which it is possible to manage everything.

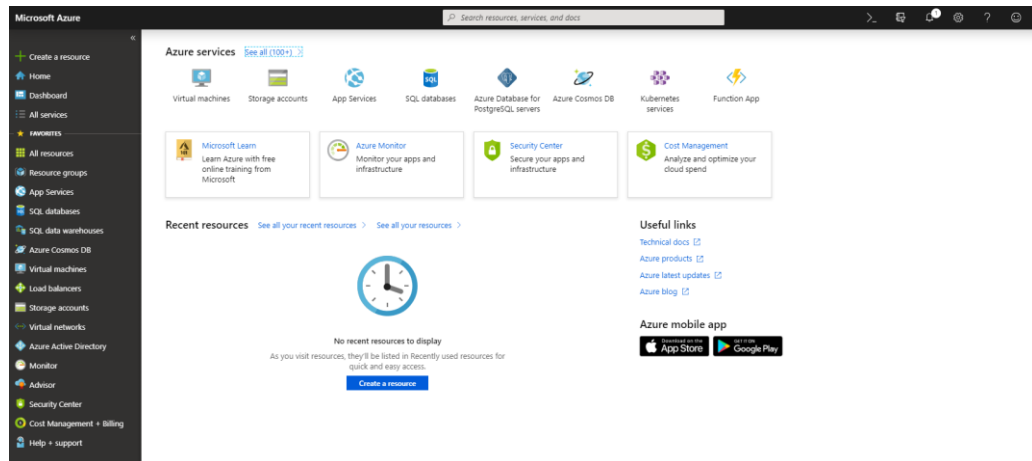


Figure 60: Azure portal

Dashboards that can be mounted on Azure allow a very complete representation of the data to be visualized.

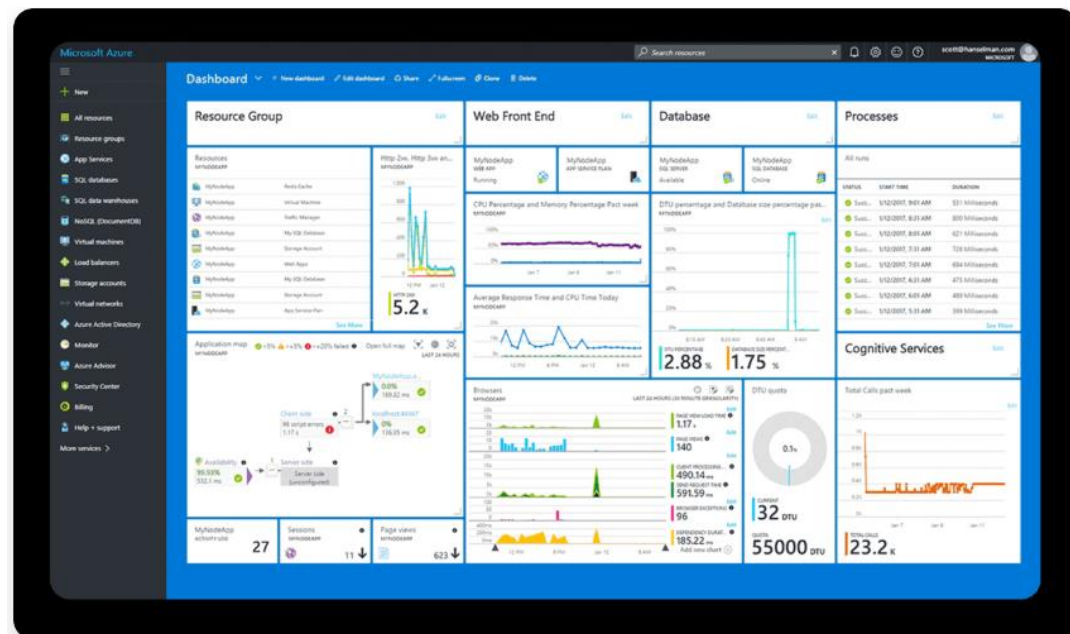


Figure 61: Azure Dashboard

2. **IBM**<sup>94</sup>: IBM has an IoT platform called Watson IoT platform. Its platform has been designated as a leader in *The Forrester Wave*<sup>TM</sup> in the third quarter of 2018.

<sup>92</sup> <https://www.kaaproject.org/>

<sup>93</sup> <https://azure.microsoft.com/en-us/>

<sup>94</sup> <https://www.ibm.com/internet-of-things>



IBM Watson IoT platform is a platform hosted in the cloud. Designed to provide all the functionalities necessary in the life cycle of an IoT platform: device registration, connectivity, data storage, visualization. The following figure illustrates the appearance of the platform:

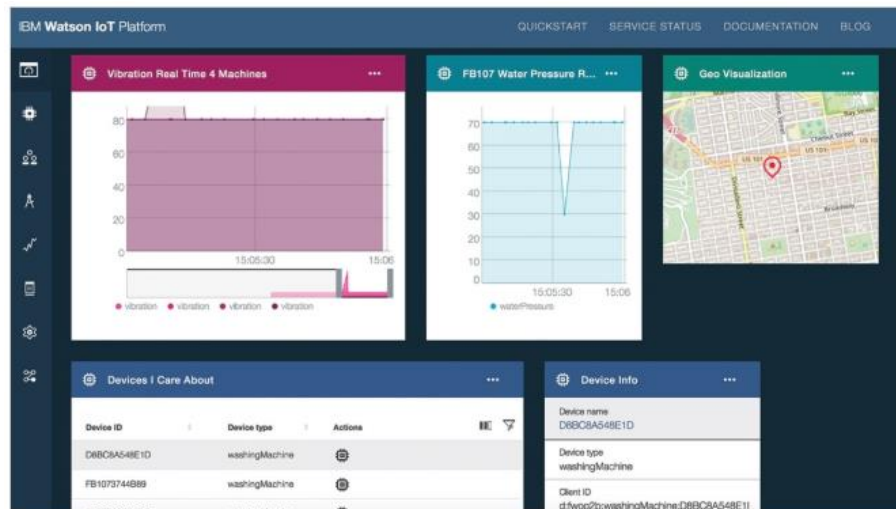


Figure 62: IBM Watson IoT platform

3. **AWS IoT platform**<sup>95</sup>. AWS IoT is Amazon's IoT platform. It allows communication between devices connected to the internet (sensors, actuators, microcontrollers, smart devices, etc) and the AWS cloud. This communication is bi-directional.
4. **MindSphere**<sup>96</sup>. Is the cloud-based IoT open operating system from Siemens. MindSphere provides a web development platform that allows to connect devices, systems, machines and analyse its data for later visualization on a dashboard (as depicted in following picture).



Figure 63: MindSphere. IoT platform from Siemens

5. **Carriots**<sup>97</sup>. Is the IoT platform designed by Altair Engineering. Carriots is now integrated into Altair's SmartWorks suite. It has the following products that are framed within the different components of an IoT platform.

<sup>95</sup> <https://www.amazonaws.cn/en/iot-platform/>

<sup>96</sup> <https://siemens.mindsphere.io/>

<sup>97</sup> <https://www.altairsmartworks.com/index.php/>



Figure 64: Altair SmartWorks suite

6. **FIWARE**<sup>98</sup>. Is an open source platform that contributes to the development of smart solutions. Fiware provides a set of APIs that facilitate the development of intelligent applications.



Figure 65: FIWARE Lab

The free platforms have generally a very narrow scope so that they are not valid for the PIXEL use cases. The problem with paid platforms is that their cost is usually not affordable for small and medium ports as in the case of PIXEL and they are very commonly oriented to general industry without implementing specific rules or algorithms, what difficult the adoption in sectors as maritime.

From the platforms listed above: *Thingsboard*, *ThingWorx*, *Wolkabout*, *kaa*, *Azure*, *IBM Watson*, *MindSphere* and *Carriots* (Altair SmartWorks) have a built-in dashboard solution, while the rest have to represent the data in external services, this is another advantage of the PIXEL architecture.

The next step would be to identify possible solutions that we can use for the PIXEL dashboard.

Among these solutions we can find:

<sup>98</sup> <https://www.fiware.org/>

- **Kibana**<sup>99</sup>: Is an open source analytics and visualization platform component of the Elastic stack. It allows to work with huge and complex data sets easily understandable through its graphs. It allows to generate reports in PDF. Your data can be represented by bar charts, scatter plots or pie charts. Kibana has multiple plugins that can be installed. Among them is ElastAlert that allows creating alerts and if met generating notifications. This plugin is ElastAlert. For its use it is necessary to deploy an independent Kibana service, responsible for analyzing the data and launching notifications when they are met.

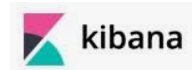


Figure 66: Kibana Logo

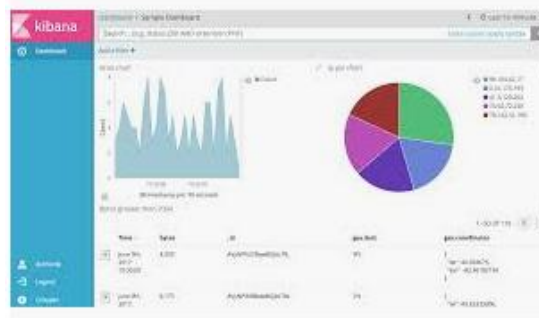


Figure 67: Kibana Dashboard

- **Grafana**<sup>100</sup>: Open source software for data analysis and monitoring. Grafana doesn't have its own database, as was the case with Kibana. However, it can work with multiple databases such as:

- MySQL.
- PostgreSQL.
- Graphite.
- Elasticsearch (as Kibana).
- OpenTSDB.
- Prometheus.
- InfluxDB.



Figure 68: Grafana Logo

Among its features are the following:

- Large amount of graphics for data visualization.
- Dynamic and reusable panels.
- Provides the user with authentication through LDAP, Google Auth, Grafana.com and GitHub.
- Allows exchange of data and dashboards between teams.

Grafana includes its own alert system in the standalone version. It allows to add visual warnings to the user in the different dashboards. Grafana's own service is in charge of analysing the data in real time and launching the alerts.

- **Tableau**<sup>101</sup>: A BI platform for visualization and data analysis. It has a very simple interface, allows users to connect to most DB and spreadsheets, create and share their dashboards.

Among its features we highlight:

- **Connection to multiple data sources.**
- **Share.** It is possible to share your workbooks with other users (**Table Desktop**) or online (**Table Public**).



Figure 69: Tableau Logo

<sup>99</sup> <https://www.elastic.co/products/kibana>

<sup>100</sup> <https://grafana.com>

<sup>101</sup> <https://www.tableau.com/>

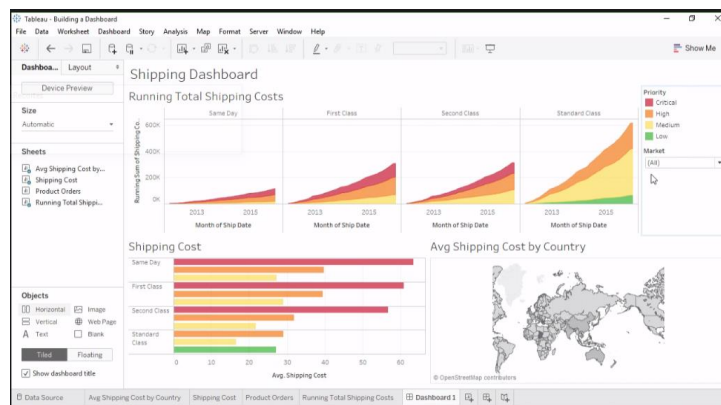


Figure 70: Tableau Dashboard

- **Apache Superset**<sup>102</sup>. Is an open source data visualization platform as is depicted in the figure follows this description. It was originally developed at Airbnb. Apache Superset allows create interactive dashboards very easy. Only works over SQL databases.



Figure 71: Apache Superset Dashboard

- **Graphite**. Is an open source tool to create dashboards. It focuses on the representation of time data series and renders graphs of this data on demand.



Figure 72: Graphite Dashboard

<sup>102</sup> <https://superset.incubator.apache.org/>

Among the functionalities of the dashboard is the sending of notifications to certain situations (bottlenecks, e.g.) or rules created. The following options stand out:

- **Elastalert**. Is a framework for alerting about anomalies, spikes or other patterns of interest in Elasticsearch data. It is a tool that complements Kibana to alert about inconsistencies in the data. 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' (*flatline 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*).
- **Grafana**<sup>103</sup>. It has a native component for alerts. The alerts are analysed by Grafana, the alerts are launched internally and distributed by infinity of channels (email, slack, etc). It has a type of visualization where it is possible to see the state of the alerts (if data have been received that have caused the sending of the alert or not) as depicted the following figure.

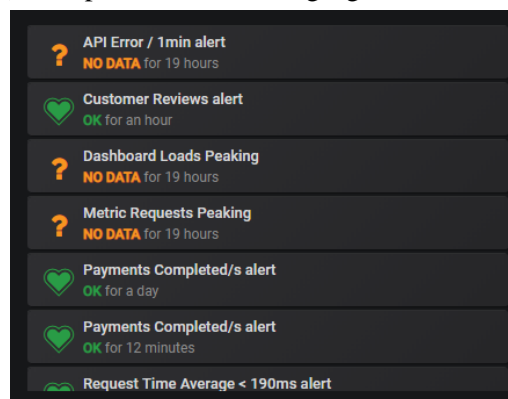


Figure 73: Visualization of Grafana to see the state of the alerts

There are also payment options such as:

- **Skedler**<sup>104</sup>. Payment tool that allows to automate the monitoring of Elasticsearch data. Skedler works with Elasticsearch, Kibana and Grafana. It allows to be automatically warned when an anomaly occurs and to execute the corresponding action.

The way to generate alerts and actions to undertake is simpler than in the case of ElastAlert or other applications because everything is done from the interface without generating any script.

<sup>103</sup> <https://play.grafana.org/d/000000074/alerting?orgId=1>

<sup>104</sup> <https://www.skedler.com/alerts>

## 7.5.Security

### 7.5.1. IoT Security

#### 7.5.1.1. Global aspect<sup>105</sup>

In an environment where digital threats continue to grow, trust is an essential prerequisite for the societal and commercial success of the Internet of Things. This is achieved by designing secure IoT solutions that respect user's data and privacy.

While cybersecurity has become a prerequisite for the proper functioning of digital networks and services, IoT systems still present specific weaknesses today:

1. The absence of a strong security culture which leaves some solutions vulnerable and leads to implementation failures. Security analyses and audits regularly found objects inadequately protected by a single or simple password, open ports, unprotected radio interfaces, obsolete kernels in firmware or cleartext keys.
2. The positioning of objects as an entry point to Internet networks and users' personal (home LAN) and professional (business LAN) information systems. Indeed, the spread of objects in various locations, and the ability to access them both locally and remotely, extends the attack range of networks and systems, thereby amplifying the risks involved.
3. The massive distribution of objects built on the same foundation transforms any vulnerability into a large-scale threat.
4. The creation of a vast amount of personal data that must be rigorously protected as part of users' rights to privacy and control of their data, particularly since the implementation of the GDPR.
5. The opportunity for hackers to act on the real world, which generates new malicious behaviours such as spying, deactivation or remote control of certain objects and systems.



Figure 74: Device interacting with several environment / applications

<sup>105</sup> [https://hellofuture.orange.com/app/uploads/2019/01/181220\\_OrangeTGI\\_LivreBlanc\\_Ill367\\_VA.pdf](https://hellofuture.orange.com/app/uploads/2019/01/181220_OrangeTGI_LivreBlanc_Ill367_VA.pdf)



The exploitation of these weaknesses represents a real threat to related IoT services; examples include disrupting a factory, spying on a home, opening a door, hijacking a car or even stopping a pacemaker. Objects can also be hacked in order to penetrate information systems, or simply for their computational and communication capabilities, like the networks of objects infected with Mirai malware in 2016.

To ensure the security of a system it's important to rely of standards and to adapt the architecture and the design to the threat of losing or altering data. There is no perfect solution applicable to all needs, but we have to be sure that a low criticality device could be used to corrupt a high one.

### 7.5.1.2. The PIXEL perimeter

The PIXEL project is designed to measure the environmental impact of the port for the ecosystem. It's a low criticality system as this solution as no impact on the port activity, but to calculate this PEI, PIXEL use several data sources that are involved on several level of port security.

As for other digital services, it's important to separate and isolate the different security level. For example, we have to be careful that a sensor that measure air quality can't be used to attack a device that is involved on the navigation system of the ships.

To prevent the impact of security issue, it is better that the data sources with higher criticality level push their data to the PIXEL data hub.

For each use case it will be important to analyse and audit the existing solutions to propose the right way to extend it with new device without creating new security threat for the port infrastructure. If we add new environmental sensors in the port, it will probably be better to relay on a separate IoT platform from the one use to manage port activity.

### 7.5.2. PIXEL Security

The PIXEL project needs a solution to properly address the security needs of its platform:

- Provide a solution for Identity Management that could be connected eventually to the port solution. This solution must provide a way to manage users, organizations, roles, applications.
- Provide a solution to manage API access to protect the several API that will be exposed by PIXEL.
- Provide a solution to ensure fine right management on an API to prevent for example a Data Source to write data that it does not manages.

These are standard needs, so the better way is to relay on standard solutions. This way it will be possible to change the solution provider with a low impact on the global solution.

For Identity Management and API access a strong solution used on the market is Oauth2 that provide most of the mechanisms needed by PIXEL. It provides a framework to authenticate users and devices and to grant limited access for a third-party application to an API.

To manage fine right access a good standard is XACML "eXtensible Access Control Markup Language". The standard defines a declarative fine-grained, attribute-based access policy language, an architecture, and a processing model describing how to evaluate access requests according to the rules defined in policies.

The FIWARE Security solution provides a component that implement all of these mechanism:



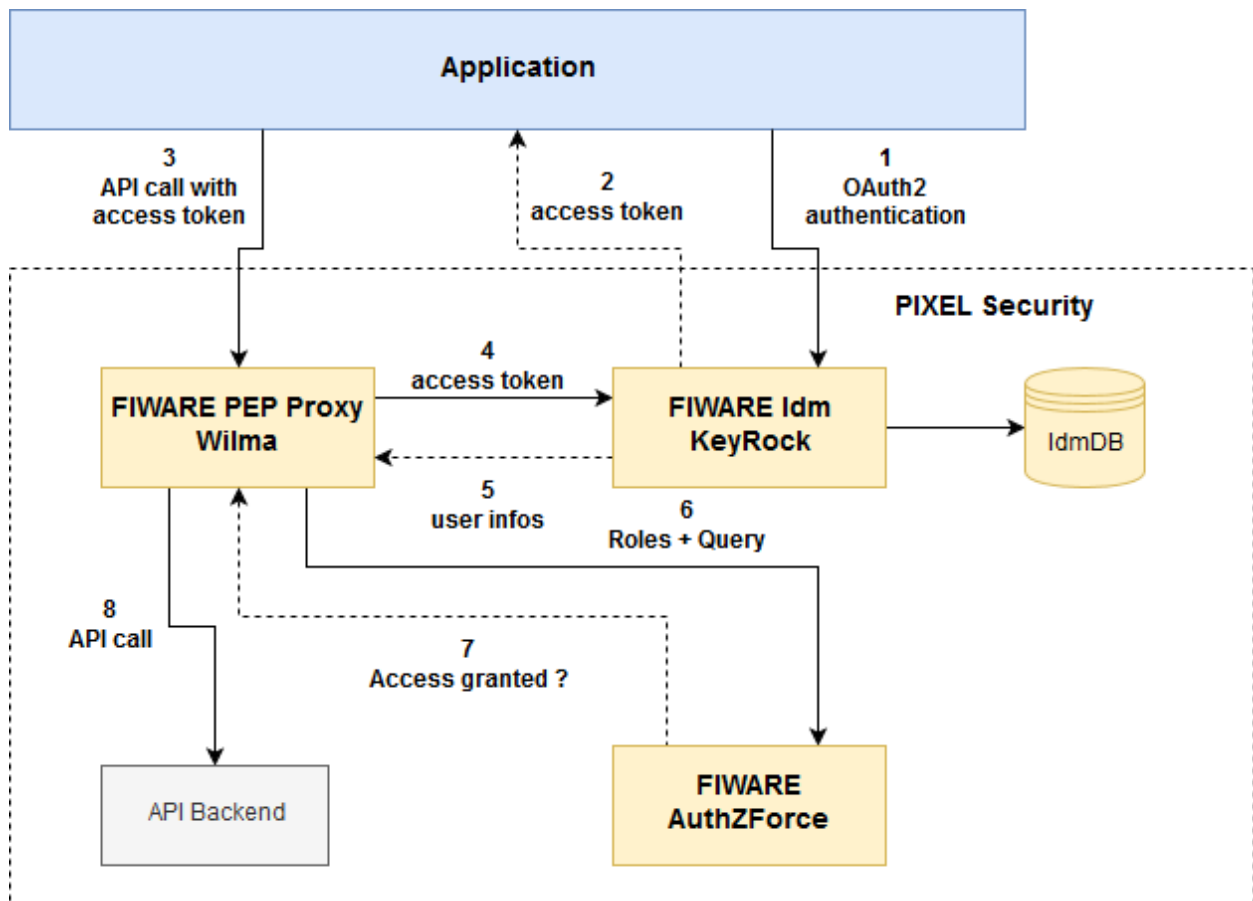


Figure 75: FIWARE Security solution

- The Keyrock<sup>106</sup> Identity Management Generic Enabler brings support to secure and private OAuth2-based authentication of users and devices, user profile management, privacy-preserving disposition of personal data, Single Sign-On (SSO) and Identity Federation across multiple administration domains.
- The Wilma<sup>107</sup> proxy Generic Enabler brings support of proxy functions within OAuth2-based authentication schemas. It also implements PEP functions within an XACML-based access control schema.
- The AuthZForce PDP/PAP<sup>108</sup> Generic Enabler brings support to PDP/PAP functions within an access control schema based on the XACML standard.

It is possible to deploy several PEP Proxy connected to the same Keyrock server. This solution provides a simple solution based on common standards to protect access to API. It could be enhanced with other components:

- Idra<sup>109</sup> is able to federate existing Open Data Management Systems based on heterogeneous technologies (e.g. CKAN, SOCRATA, DKAN etc.) providing a single API and a standard metadata format (DCAT-AP) to discover open datasets.
- APIInf<sup>110</sup> API Management Framework is a tool for API owners to manage their APIs. It provides all the necessary features to run business with APIs and makes it easy for API consumers to find and start using the standard APIs.

<sup>106</sup> <https://fiware-idm.readthedocs.io/en/latest/>

<sup>107</sup> <https://fiware-pep-proxy.readthedocs.io/en/latest/>

<sup>108</sup> <https://authzforce-ce-fiware.readthedocs.io/en/latest/>

<sup>109</sup> <https://idra.readthedocs.io/en/latest/>

<sup>110</sup> <https://github.com/apinf/platform>

## 8. Component interfaces

### 8.1. Data Acquisition Layer

#### 8.1.1. Programming interface

The PIXEL Data Acquisition is composed of several components:

- **FIWARE Context Broker Orion.** This component exposes several APIs.
  - NGSI v2 API<sup>111</sup>. Used to manipulate the data. It is a HTTP REST API used to create, update, delete and request the entities and their attributes.
  - Subscription API<sup>112</sup>. 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<sup>113</sup> and aggregate<sup>114</sup> data

### 8.2. Information Hub

#### 8.2.1. Programming interface

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

---

<sup>111</sup> <http://fiware.github.io/specifications/ngsiv2/stable/>

<sup>112</sup> [https://fiware-orion.readthedocs.io/en/master/user/walkthrough\\_apiv2/index.html#subscriptions](https://fiware-orion.readthedocs.io/en/master/user/walkthrough_apiv2/index.html#subscriptions)

<sup>113</sup> <https://fiware-sth-comet.readthedocs.io/en/latest/raw-data-retrieval/index.html>

<sup>114</sup> <https://fiware-sth-comet.readthedocs.io/en/latest/aggregated-data-retrieval/index.html>

memory, execute simple transformations and enrich the extracted data and pass it to the connected client application through the API Gateway.

- **Data reductor.** 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 reducers. This allows implementation of custom reduction, aggregation or other data transformation algorithms provided by customers.

## 8.3. Operational Tools

### 8.3.1. Programming interface

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 it. 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:

- Publishing models and predictive algorithms in the PIXEL marketplace. Even if this can be a service by its own, it should be somehow assisted by or synchronized with the Operational Tools.
- Importing models and predictive algorithms from the PIXEL marketplace. The PIXEL marketplace tackles a business concept whereas the OT database tries to reproduce an operational environment.
- Creation, edition and execution of models and predictive algorithms, both in runtime or as cron jobs.
- Access to the OT data model, at least to part of it (e.g. model configuration, model result, etc.).
- Creation and edition of CEP 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.

Support for security and integration with other PIXEL APIs

## 8.4. Dashboard

### 8.4.1. Programming interface

At this time it is considered that the development of any API in this module will not be necessary. If necessary, it will be documented in deliverable D6.2.

## 8.5. Security

### 8.5.1. Programming interface

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<sup>115</sup>.
- **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.

---

<sup>115</sup> <https://keyrock.docs.apiary.io/#reference/keyrock-api>

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<sup>116</sup>.

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 allowed to execute this API call.

---

<sup>116</sup> <https://github.com/authzforce/rest-api-model/tree/release-5.3.1/src/main/resources>

## 9. Conclusion and future work

### 9.1. Conclusion

This document includes the specification of the architecture proposed in the PIXEL project to base the IoT platform that will enable operations analysis and optimization of ports. This architecture has been extracted from the requirements and use cases described during the first months of work of the project, as well as from the expertise of the technical partners participating in the project and other stakeholders involved.

The present architecture contains, thus, a strong guideline to build the different functional blocks of the ICT part of the project. To perform this work, it has been needed an important work of designing a system from scratch, analysis of the current state of the art and the alternative technologies and finally a complete exercise of communication among all the differently skilled partners participating in the project. The result is a document that can be interpreted as a technical manual to implement the PIXEL IoT platform, as well as a registry of the different design decisions that have been made during the period of work reported.

In a context where the information is a major asset and the systems, specially IT based, change and evolve rapidly, it is important to find a proper balance between detailed technical specifications and enough flexibility for future changes, adapting to new conditions in the markets, the emergence of technologies or variations in the approach of the ports that could be faced in the near future. Consequently, the document is technically precise from the point of view of the engineering but at the same time provides a degree of freedom to do some important architectural choices, such as the deployment strategy.

### 9.2. Future work

As the document has been written based on 8 months of work (4-12), it is focused on the back-end mechanism that will allow most of the PIXEL features. However, front-end designs and interfaces are still to be completely defined, since they are derived from the former. These are going to be provided in the second version of the document (D6.2).

In that second and final version, it will be included other architectural details such as the standard deployment schemas proposed for the different ports, the detail of the APIs of the different functional modules or any change that could be made during the period of months (12-18) derived from the feedback of the technical partners during the implementation phase of the project.

## 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](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](https://hellofuture.orange.com/app/uploads/2019/01/181220_OrangeTGI_LivreBlanc_Ill367_VA.pdf)