Deliverable D3.5

CPaaS.io System Architecture (v3)

Version V3.0

June 29, 2018

ABSTRACT

This fifth WP3 deliverable provides an update to the initial CPaaS.io architecture D3.2 and intermediate version D3.3. The architecture work follows the requirement collection phase (as already summarised in Deliverable D3.1) and improves the previous version of the architecture, capturing the work achieved by other technical Work Packages since D3.3 was released. This document provides some elements of the used methodology and then provides both the updated Functional and Information Views, together with some Perspectives (including security and interoperability in particular). The Functional View gives a second view about which functional components need to be implemented in order to fulfil functional requirements. Perspectives on the other hand focuses on how (through strategies and tactics) the non-functional requirements can be fulfilled.

As explained in the DoW, we are using the IoT ARM methodology, which means that this document features a technology-agnostic logical architecture - which is agreed upon by both parties (Japan and EU) - in addition to 1/ two concrete instantiations of this logical architecture (later referred as u2-based and FIWARE-based concrete architectures) which are formally captured within two so-called Instantiation Views and 2/ a set Deployment and Operation Views which give -on the one hand- detail about physical deployment of the platform and scenarios and -on the other hand- information about relation occurring between CPaaS.io users, scenario infrastructure and the platform.

Finally, the second version of the accompanying Volere sheet, keeps track of the functional and non-functional requirement coverage, take into account the latest requirement updates from D2.5 and software developments, through respectively the architecture Views and Perspectives. This Volere document is the result of the iterative requirement process, part of the IoT ARM methodology.
Disclaimer

This document has been produced in the context of the CPaaS.io project which is jointly funded by the European Commission (grant agreement n° 723076) and NICT from Japan (management number 18302). All information provided in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission and NICT have no liability in respect of this document, which is merely representing the view of the project consortium. This document is subject to change without notice.

Document Information

<table>
<thead>
<tr>
<th>Editor</th>
<th>François Carrez (UoS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors</td>
<td>Editor +</td>
</tr>
<tr>
<td></td>
<td>A. Skarmeta, J.-A. Martinez, M.-A. Zamora, A. Canovas (OdinS);</td>
</tr>
<tr>
<td></td>
<td>M. Strohbach (AGT),</td>
</tr>
<tr>
<td></td>
<td>S. Prost (TTN)</td>
</tr>
<tr>
<td></td>
<td>S. Haller, A. Gschwend, M. Fraefel (BFH);</td>
</tr>
<tr>
<td></td>
<td>E. Benz, B. Cheng, G. Solmaz, M. Bauer (NEC);</td>
</tr>
<tr>
<td></td>
<td>N. Koshizuka (UoT)</td>
</tr>
<tr>
<td></td>
<td>K. Shindo (YRP)</td>
</tr>
<tr>
<td>Reviewers</td>
<td>Martin Strohbach (AGT);</td>
</tr>
<tr>
<td>Delivery Type</td>
<td>R</td>
</tr>
<tr>
<td>Dissemination Level</td>
<td>Public</td>
</tr>
<tr>
<td>Contractual Delivery Date</td>
<td>June 30, 2018</td>
</tr>
<tr>
<td>Actual Delivery Date</td>
<td>June 29, 2018</td>
</tr>
<tr>
<td>Keywords</td>
<td>Requirement analysis, functional decomposition, system use-cases, system architecture, concrete platform instantiations, IoT ARM</td>
</tr>
</tbody>
</table>

H2020 EUJ-02-2016 CPaaS.io Page 2 of 110
## Revision History

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0.1</td>
<td>05/12/2017</td>
<td>Initial contribution with new Instantiation view</td>
<td>F. Carrez (UoS)</td>
</tr>
<tr>
<td>v0.2</td>
<td>04/05/2018</td>
<td>Update in several sections following F2F in Amsterdam</td>
<td>F. Carrez (UoS)</td>
</tr>
<tr>
<td>v0.3</td>
<td>25/05/2018</td>
<td>Additions and updates to §6.2</td>
<td>E. Berz (NEC)</td>
</tr>
<tr>
<td>v0.4</td>
<td>25/05/2018</td>
<td>Addition and updates to §6.2</td>
<td>B. Cheng (NEC)</td>
</tr>
<tr>
<td>v0.5</td>
<td>12/06/2018</td>
<td>Additions and updates to §5 (inter-p/f interoperability)</td>
<td>J-A. Martinez (OdinS)</td>
</tr>
<tr>
<td>v0.6</td>
<td>15/06/2018</td>
<td>New §6.2.4.3 (ColourRun Deployment View)</td>
<td>M. Strohbach (AGT)</td>
</tr>
<tr>
<td>v0.7</td>
<td>25/06/2018</td>
<td>Update to §5.2 (semantic interoperability)</td>
<td>A. Gschwend (BFH)</td>
</tr>
<tr>
<td>v0.8</td>
<td>26/06/2018</td>
<td>New §6.2.4.2 (Waterproof Am’dam Deployment View)</td>
<td>S. Prost (TTN)</td>
</tr>
<tr>
<td>v0.9</td>
<td>26/06/2018</td>
<td>Update to §4.3.2 (ontology section)</td>
<td>A. Gschwend (BFH)</td>
</tr>
<tr>
<td>v0.10</td>
<td>26/06/2018</td>
<td>Finalization</td>
<td>F. Carrez (UoS)</td>
</tr>
<tr>
<td>v0.11</td>
<td>27/06/2018</td>
<td>Review comments</td>
<td>M. Strohbach (AGT)</td>
</tr>
<tr>
<td>v0.12</td>
<td>27/06/2018</td>
<td>Addings to Ontology section §4.3.2</td>
<td>A. Gschwend (BFH)</td>
</tr>
<tr>
<td>v0.13</td>
<td>27/06/2018</td>
<td>Contribution to §6.1</td>
<td>N. Koshizuka (UoT)</td>
</tr>
<tr>
<td>v0.14</td>
<td>29/06/2018</td>
<td>Take into account Review comments + review of Japanese contribution + finalization for delivery</td>
<td>F. Carrez (UoS)</td>
</tr>
</tbody>
</table>
Table of Contents

1 Introduction .............................................................................................................................................. 11
  1.1 Delta with previous version D3.3 ....................................................................................................... 12
2 Methodology ........................................................................................................................................... 13
  2.1 Introduction to the IoT ARM ............................................................................................................ 13
  2.2 Using the IoT ARM in the context of CPaaS.io .................................................................................. 25
3 CPaaS.io Requirements Analysis and Mapping ..................................................................................... 26
  3.1 Summary of requirement collection phase ....................................................................................... 26
  3.2 Requirement Analysis ...................................................................................................................... 32
4 CPaaS.io Views .................................................................................................................................... 33
  4.1 Actors involved in data production .................................................................................................. 33
  4.2 CPaaS.io Functional View ............................................................................................................... 34
  4.3 CPaaS.io Information View ............................................................................................................ 44
5 Perspectives ........................................................................................................................................... 54
  5.1 Security ............................................................................................................................................ 54
  5.2 Semantic Interoperability ............................................................................................................... 58
  5.3 Performance ..................................................................................................................................... 59
  5.4 Scalability ......................................................................................................................................... 59
6 Instantiation views ............................................................................................................................... 59
  6.1 U2-based platform Instantiation View .............................................................................................. 59
  6.2 FIWARE-based platform Instantiation View ..................................................................................... 74
7 References ............................................................................................................................................. 103
Appendix : Requirements (Volere Template) .......................................................................................... 106

List of Figures

Figure 1: Simplified process leading to generating a concrete architecture ............................................ 13
Figure 2: Simplified IoT Domain Model (feat. most important concepts) ................................................ 14
Figure 3: IoT Domain Model (as in IoT-A D1.5 [1]) .............................................................................. 16
Figure 4: IoT Information Model (as in IoT-A D1.5 [1]) ......................................................................... 17
Figure 5: IoT Functional Model (as in IoT-A D1.5 [1]) .......................................................................... 19
Figure 6: IoT Functional View (as in IoT-A D1.5 [1]) ............................................................................ 20
Figure 7: DIKW model ........................................................................................................................... 22
Figure 8: Viewpoint layout for the Instantiation View ............................................................................. 23
Figure 9: CPaaS.io logical functional architecture ................................................................. 35
Figure 10: Core Concepts for IoT Service Orchestration .......................................................... 37
Figure 11: UML UC for Class-I RDP pre-configuration case ...................................................... 45
Figure 12: UML UC for Class-II RDP pre-configuration case ...................................................... 46
Figure 13: UML UC for Class-III RDP pre-configuration phase .................................................... 47
Figure 14: Data production for Class-I RDPs ............................................................................... 49
Figure 15: Data production for Class-II RDP .............................................................................. 50
Figure 16: EU-JP joint security scenario ....................................................................................... 57
Figure 17. Access Control System for Smart Building. Floors representation .............................. 57
Figure 18. XACML framework instantiation ................................................................................. 58
Figure 19: ucode structure ........................................................................................................... 60
Figure 20: ucode field names and their widths ............................................................................. 60
Figure 21: Predefined cc values and but boundary between SLDc and ic ..................................... 61
Figure 22: u2 Architecture (ucR focus) ........................................................................................ 62
Figure 23: u2 architecture (IoT Aggregator focus) ..................................................................... 64
Figure 24: u2 architecture ("Omotenashi" Cloud focus) ................................................................. 65
Figure 25: IoT Engine with IoT Engine Open Platform ................................................................. 67
Figure 26: uID architecture ........................................................................................................ 68
Figure 27: Matching of the u2 components to the CPaaS.io Functional View ............................... 69
Figure 28: uID architecture in deployment and operation view .................................................... 71
Figure 29: Public Transportation Open Data Centre providing both static and real-time location data of trains and buses in Tokyo (Red dots show train positions and blue dots show bus positions) .................. 72
Figure 30: Sapporo Open Data for Tourism ............................................................................... 73
Figure 31: Smart Ambulance using IoT Technologies .................................................................. 74
Figure 32: FogFlow Service Orchestrator .................................................................................. 79
Figure 33: Task Registration ....................................................................................................... 80
Figure 34: Graphical editor of service topology .......................................................................... 80
Figure 35: Defining a requirement with a specified geo-scope ..................................................... 81
Figure 36: FIWARE components mapped onto the Instantiation View ........................................ 85
Figure 37: Matching of the FIWARE components to the CPaaS.io Functional View .................... 87
Figure 38: Interactions of the FIWARE FCs ................................................................................. 88
Figure 39: FIWARE-based platform deployment and operation view ........................................... 92
Figure 40: Waterproof architecture and usage ............................................................................ 94
Figure 41: Waterproof deployment view ...................................................................................... 95
Figure 42: My Events Architecture .................................................................................................................................................. 96
Figure 43: Use Case Diagram My Events Application .................................................................................................................. 97
Figure 44. PAP Web configuration interface .................................................................................................................................... 99
List of Tables

Table 1: List of Non-Functional Requirements .................................................................................. 27
Table 2: List of Functional Requirements ....................................................................................... 29
Table 3: Mapping table for the u2-based architecture .................................................................... 70
Table 4: Mapping table for the FIWARE-based architecture ............................................................... 90

List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABE</td>
<td>Attribute Based Encryption</td>
</tr>
<tr>
<td>ACP</td>
<td>Access Control Policy</td>
</tr>
<tr>
<td>AE</td>
<td>Augmented Entity</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AuthN</td>
<td>Authentication</td>
</tr>
<tr>
<td>AuthZ</td>
<td>Authorization</td>
</tr>
<tr>
<td>BC</td>
<td>Business Case</td>
</tr>
<tr>
<td>BLE</td>
<td>Bluetooth Low-Energy</td>
</tr>
<tr>
<td>IoT ARM</td>
<td>IoT Architectural Reference Model</td>
</tr>
<tr>
<td>CapBAC</td>
<td>Capability-Based Access Control model</td>
</tr>
<tr>
<td>CAACS</td>
<td>Access Control System for Physical Space</td>
</tr>
<tr>
<td>CEP</td>
<td>Complex Event Processing</td>
</tr>
<tr>
<td>CP-ABE</td>
<td>Ciphertext-Policy Attribute-Base Encryption</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Process Unit</td>
</tr>
<tr>
<td>CRM</td>
<td>Customer Relationship Management</td>
</tr>
<tr>
<td>CSF</td>
<td>Common Service Function</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma Separated Values</td>
</tr>
<tr>
<td>CT</td>
<td>Capability Token</td>
</tr>
<tr>
<td>DC</td>
<td>Design Constraints</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>DCAT</td>
<td>Data Catalo Vocabulary</td>
</tr>
<tr>
<td>DoW</td>
<td>Description of Work</td>
</tr>
<tr>
<td>FC</td>
<td>Functional Component</td>
</tr>
<tr>
<td>FG</td>
<td>Functional Group</td>
</tr>
<tr>
<td>FREQ</td>
<td>Functional REQuirement</td>
</tr>
<tr>
<td>FV</td>
<td>Functional View</td>
</tr>
<tr>
<td>GDLI</td>
<td>Government Data Listener &amp; Ingestion</td>
</tr>
<tr>
<td>GE</td>
<td>General Enabler</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HFDS</td>
<td>Highly Functionally Distributed System</td>
</tr>
<tr>
<td>HSM</td>
<td>Hardware Security Module</td>
</tr>
<tr>
<td>IdM</td>
<td>Identity Management</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>IoT-A</td>
<td>Internet of Things – Architecture</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>KAT</td>
<td>Knowledge Acquisition Toolkit</td>
</tr>
<tr>
<td>IdM</td>
<td>Identity Management</td>
</tr>
<tr>
<td>IV</td>
<td>Information View</td>
</tr>
<tr>
<td>LDLI</td>
<td>Live Data Listener &amp; Ingestion</td>
</tr>
<tr>
<td>LOD</td>
<td>Linked Open Data</td>
</tr>
<tr>
<td>M2M</td>
<td>Machine to Machine</td>
</tr>
<tr>
<td>MPU</td>
<td>Micro Processor Unit</td>
</tr>
<tr>
<td>NFREQ</td>
<td>Non Functional REQuirement</td>
</tr>
<tr>
<td>NGO</td>
<td>Non-Governmental Organisation</td>
</tr>
<tr>
<td>NGSI</td>
<td>Next Generation Service Interface</td>
</tr>
<tr>
<td>OMA</td>
<td>Open Mobile Alliance</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>PAP</td>
<td>Policy Administration Point</td>
</tr>
<tr>
<td>PDP</td>
<td>Policy Decision Point</td>
</tr>
<tr>
<td>PDS</td>
<td>Personal Data Store</td>
</tr>
<tr>
<td>PEP</td>
<td>Policy Enforcement Point</td>
</tr>
<tr>
<td>PE</td>
<td>Physical Entity</td>
</tr>
<tr>
<td>PKC</td>
<td>Public Key Cryptography</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDFS</td>
<td>RDF Schema</td>
</tr>
<tr>
<td>RDP</td>
<td>Raw Data Producer (or Provider)</td>
</tr>
<tr>
<td>RTOS</td>
<td>Real-Time Operating System</td>
</tr>
<tr>
<td>SCIM</td>
<td>System for Cross-domain Identity Management</td>
</tr>
<tr>
<td>SCP</td>
<td>(ETSI) Smart Card Platform</td>
</tr>
<tr>
<td>SKC</td>
<td>Symmetric Key Cryptography</td>
</tr>
<tr>
<td>SotA</td>
<td>State of the Art</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL Protocol and RDF Query Language</td>
</tr>
<tr>
<td>SSN</td>
<td>Semantic Sensor Network</td>
</tr>
<tr>
<td>UC</td>
<td>Use-Case</td>
</tr>
<tr>
<td>VE</td>
<td>Virtual Entity</td>
</tr>
<tr>
<td>VRM</td>
<td>Vendor Relationship Management</td>
</tr>
<tr>
<td>XACML</td>
<td>eXtensible Access Control Markup Language</td>
</tr>
</tbody>
</table>
### List of Editing Shortcuts

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g.</td>
<td><em>exempli gratia</em> – for example</td>
</tr>
<tr>
<td>etc</td>
<td><em>et cetera</em></td>
</tr>
<tr>
<td>feat.</td>
<td>featuring</td>
</tr>
<tr>
<td>Gov</td>
<td>Government</td>
</tr>
<tr>
<td>h/w</td>
<td><em>hardware</em></td>
</tr>
<tr>
<td>i.e.</td>
<td><em>id est</em> – that is</td>
</tr>
<tr>
<td>incl.</td>
<td>including</td>
</tr>
<tr>
<td>p/f</td>
<td>platform</td>
</tr>
<tr>
<td>resp.</td>
<td>respectively</td>
</tr>
<tr>
<td>s/w</td>
<td><em>software</em></td>
</tr>
<tr>
<td>tbd</td>
<td>to be discussed/decided</td>
</tr>
<tr>
<td>vs.</td>
<td><em>versus</em></td>
</tr>
<tr>
<td>w.r.t.</td>
<td>with respect to</td>
</tr>
</tbody>
</table>
1 Introduction

This Deliverable D3.5 is the next to last version of the incremental architecture document that was initiated with D3.2 and improved with D3.3. It will be completed and consolidated by D3.7 due M30.

Leveraging on D3.3, this version goes deeper in the CPaaS.io architecture, providing an update requirement analysis, introducing more views (that illustrate 1/ the relation between logical and concrete architectures and 2/ the relation between the scenarios and the platform), providing updated content to the Functional and Information View and providing an updated Perspective section. Please refer to Section 1.1 which goes in the full detail of the delta between D3.3 and D3.5.

In the CPaaS.io project, we are reusing the architectural approach promoted by the IoT-A FP7 project called the Internet of Things Architectural Reference Model (IoT ARM). As we explain in Section 2, the IoT ARM is not an IoT architecture as such; instead it provides a whole methodology and framework for creating a logical IoT Architecture with potential multiple instantiations (or implementations) as we can see below.

While an architecture team usually uses the IoT ARM for creating one single concrete architecture considering one single use case (usually a targeted product with clear technical perimeter), CPaaS.io has to face an additional difficulty as the project will produce not one, but two different concrete platform architectures out of the logical architecture: one architecture for the Japanese side, based on the u2 technology and a second one for the European side based on FIWARE. The IoT ARM is therefore a tool that helps the two involved communities for reaching common understanding and agreeing on the main architecture principles. The scope of this agreement includes:

- A taxonomy of concepts (Domain Model);
- A logical structure for dealing with information (Information Model);
- A functional decomposition into logical components (Functional View);
- System use-cases that illustrate interactions taking place between functional components for some typical platform usage (Information View);
- A strategy for ensuring interoperability, since the two concrete instantiations of a common “abstract & logical” platform architecture must eventually talk with each other and understand each other in order to implement the interoperability and federation principles described as one of the main technical objectives of the CPaaS.io project;
- Strategies for ensuring additional system qualities (Perspectives).

The two later aspects – further developed in this third iteration of the architecture- are usually dealt with, with the adoption of common or compatible design and technology choices (e.g. adopting RDF as a semantic data internal format and JSON-LD for serialisation) and the definition of common interfaces (e.g. adopting REST, SPARQL etc.).

Finally this document is accompanied with a technical annex (Volere MicroSoft Excel document) that summarises the functional and non-functional requirements resulting from earlier project activities (see D2.1 [2], D2.5 [4] and D3.1 [5] for more detail about Requirement collection and analysis).

This deliverable is then structured as follows.

Section 2 gives an introduction to the IoT ARM and in particular details the process part of the IoT ARM methodology, focussing on requirement engineering, views and perspective definition.
Section 3 focuses on requirements and provides some insights about the analysis of requirements collected earlier from the scenario holders on the one hand and on the other hand on functional/non-functional requirements as seen from the platform perspective. It also shows the current coverage of the set of requirements by the functional decomposition discussed in Section 4.2.

Section 4 provides the revised views of the CPaaS.io architecture. The functional view depicts and describes a preliminary functional decomposition following the IoT ARM layered Functional Model, while the Information View, at this stage, mainly consists of a collection of system use-cases that describe inter-FCs interactions for a selected set of basic activities at platform level, involving not only the platform, but additional actors like platform clients and data producers as well.

Section 5 consists of the architecture Perspectives and gives insights about how the different non-functional properties of the system are dealt with.

Section 6 is an important section that bridges the common logical and abstract view upon the CPaaS.io architecture to two distinct – still compatible – instantiations of it. It describes in particular the various reused components (u2- and FIWARE-based technologies) and spots some new ones, which will be in the centre of CPaaS.io specific software developments. It also shows how the concrete sets of components maps to the abstract and logical functional decomposition.

Finally Section 7 provides a conclusion and in particular gives some hints about the few limitations and gaps of this preliminary version. It also details next steps towards CPaaS.io architecture v3.

1.1 Delta with previous version D3.3

Deliverables describing the CPaaS.io architecture, namely D3.2, D3.3, D3.5 and D3.7, are incremental documents. This section aims at describing updates brought within D3.5:

- Updated Section 3.1.1 taking into account latest scenario requirements from D2.5;
- Updated descriptions of the “logical” FCs in Section 4.2 (notably Section 4.2.2.1 and Section 4.2.2.2);
- IoT ARM Views in Section 2.1.4:
  - New Instantiation View (Section 2.1.4.5): Introduction of a new Layered Model which is used to describe the u2-based and FIWARE-based “concrete” platform architectures. This answers a remark made during the Year 1 project review in Tokyo;
  - New Deployment and Operation View (Section 2.1.4.6) and consequently, introduction of new Deployment and Operation Views in Section 6.1 (u2-based platform) and Section 6.2 (FIWARE-based platform);
- Added content about ontologies and various data formats used in the platform (FIWARE only) in Section 4.3.2;
- Added information about data stores and data management in Section 4.3.1.6;
- Perspective section: update on the platform interoperability in Section 5.1.5 and Semantic Interoperability in Section 5.2;
- Introduction of the three Deployment and Operation Views corresponding to the FIWARE-based platform in addition to two European scenarios (Enhanced User Experience in Section 6.2.4.3 and Waterproof Amsterdam in Section 6.2.4.2);
- Introduction of four Deployment and Operation Views corresponding to the u2-based platform in addition to three Japanese Scenarios (Transportation Scenario in Section 6.1.4.2, Sapporo scenario in Section 6.1.4.3 and Health-Care scenario in Section 6.1.4.4);
- Updated Volere Excel sheet (accompanying document).
2 Methodology

2.1 Introduction to the IoT ARM

The IoT ARM methodology (see an overview of the whole process in Figure 1 below) covers the full requirement process as introduced and described already in CPaaS.io deliverable D3.1 [5], but also covers everything about elaborating architectural Views and Perspectives. In the following sections we remind some aspects of the requirement process (the ones covered in D3.2) and introduce the Reference Model and Reference Architecture (which ultimately makes the system architecture).

![Figure 1: Simplified process leading to generating a concrete architecture](image)

In this picture we have preliminary activities which start with the elaboration of architectural views (Physical Entity and Context Views – see Sections 2.1.4.3 & 2.1.4.4 for more details about those views) that are depending on the scenarios and which basically describe in detail the aspects relating to IoT (what are the objects, what are the properties of interests, what is the hardware needed for capturing those properties, how do we virtualise those objects, what are the actors outside the system and how do they interact with the system etc...). Then based on those scenario specific information and business considerations, we start a Requirement Process. This process is described in former deliverables D2.1 and D3.1 [2] [5] and consists of 3 steps: requirement collection (tackled in the two deliverables afore-mentioned), analysis and mapping (see sub-sections below). Finally starts the elaboration of remaining architectural Views (Functional, Information, Instantiation and Deployment & Operation Views).

2.1.1 Requirement Analysis

As soon as the raw material has been collected (see D2.1 and D3.1) there is a need for reviewing those requirements, removing duplicates, unifying/factorizing similar ones and translating/rewriting all requirements using unified terms and concepts as introduced within the IoT ARM (Domain Model in particular). The result of this phase is a collection of unified system requirements collected and summarised within the Volere template (which is a technical annex to this document).

\[1\] Deployment & Operation will be called Deployment View in the rest of this document
2.1.2 Requirement Mapping

Functional unified requirements need to be mapped to the *Functional View* (FV) where one or more *Functional Groups* (FG) and *Functional Components* (FC) can be identified. This will result into an updated functional decomposition of the targeted system. Some of the non-functional requirements can be mapped to the Information and Deployment Views. The updated CPaaS.io requirement analysis and mapping are covered in Section 3.

2.1.3 IoT Reference Model

In the IoT ARM methodology, the IoT *Reference Model* (RM) consists of a set of models which are used to describe aspects of the IoT field which are meant to be agreed upon by architecture team members. In the context of CPaaS.io we consider – for sake of simplicity – only three of them, namely the IoT Domain, Information and Functional Models.

2.1.3.1 IoT Domain Model

The purpose of the IoT *Domain Model* (DM) as proposed by IoT-A [1] [11] is to introduce the concepts pertaining to the IoT domain (see Figure 3 below) and the different relationships between those concepts. Among the different concepts introduced by the DM, it is important to remind of the following main ones (see a simplified view of the Domain Model in Figure 2 below):

![Figure 2: Simplified IoT Domain Model (feat. most important concepts)](image)

*Physical Entities (PEs):* Physical Entities are the objects from the real world that can be sensed and for which different aspects can be measured. They are virtualized in cyber-space using Virtual Entities. Examples of PEs from the CPaaS.io project include people and geographical areas.

*Virtual Entities (VEs):* VEs are at the heart of an IoT system. They represent the PEs in the virtual world. Aspects of the PE are captured by VE properties, and using sensors and actuators allows one to bridge the physical and logical worlds and then to act on (or read about) properties. At the level of the VE, we will consider a special kind of service, called a VE Service, which is used to manipulate or access those properties. It is important to mention here that it is not compulsory that data providers provide modelling of PEs into VEs and manage the associated VE Services. However it is highly important that CPaaS.io provides the means for doing so. Actually the activity of defining VEs and their associated properties and then binding these properties to sensor readings for instance can be endorsed by additional CPaaS.io
actors bringing higher level of abstraction and added value services to the end-users of the CPaaS.io platforms.

**IoT Devices:** In CPaaS.io, IoT devices are the hardware supporting the sensing and actuation functions. Micro-controllers, batteries, ROM memory etc. are also devices (but without the IoT prefix).

**IoT Resources:** IoT Resources are the software embedded in IoT Devices that provides the raw readings (for sensors) and actuations. The IoT Domain Model advises not accessing directly resources, but on the contrary to access corresponding Resource-centric IoT Services (see below).

**IoT Services:** We can consider different kinds of IoT services depending on their level of abstraction:

- Resource-centric IoT services (r-IoT Service) are exposing the IoT Resources using standardized interfaces and possibly adding metadata to the raw reading available at the resource level. They all connect to a sole resource (sensor or actuator). For instance getting the reading of a temperature sensor (e.g. via a REST interface) is accomplished through an r-IoT Service;
- VE-centric IoT Services (ve-IoT Service) are associated to the VEs and are used for accessing VEs attributes/status or to access VE-level services not directly connected to VEs attribute or situation. In the Functional View the VE Service FC deals with such accesses. Getting the value of the “hasTemperature” property of a room VE is an example of a ve-IoT Service.

**Note:** In the rest of this document IoT Services and VE Services are to be understood as respectively r-IoT Services and ve-IoT Services.

Both kinds of IoT Services described above should be associated with service descriptions that can be used to discover particular sensing/actuation capabilities (as recommended by the Information Model).

**Services:** Services (without IoT prefix) are associated to VEs but do not relate to specific properties as illustrated in the example above. Services are not part of the IoT Domain Model but could be added to the global picture for the sake of clarity. For instance autonomous objects (with cognitive capabilities) may expose services that do not relate de facto to any of their VE properties.

**User:** Different kinds of users are expected to interact with the CPaaS.io platform.
2.1.3.2 IoT Information Model

The Information Model (IM) (see Figure 4 below) focuses on the description of the structure of Virtual Entities as a representation in the cyber space of Physical Entities. The representation of the information (either it is encoded in eXtensible Markup Language - XML, RDF, binary or any other format) is kept away from the Information Model and left to the architect’s choice, as part of the semantic interoperability perspective.

The central part of the IM (referring to Figure 4 below) consists of the structure of the Virtual Entity which is modelled using a set of Attributes and which are associated (via the Association relationship) to the so-called Service Description. These associations are essential in the IoT IM as they make the binding between a VE property – which as the name suggests is at the VE level – and a corresponding IoT Service (meaning...
a service exposing an IoT Resource), which is at the resource level. This association must be managed in a
dynamic way so that the binding between a VE property (attribute) and an IoT Resource (via an IoT Service)
can vary in time.

The Attribute is the aggregation of one-to-many ValueContainers. Each of those containers contains one
single Value and one-to-many MetaData (e.g. time stamp, location, accuracy, etc.).

VEs are described using a ServiceDescriptions where each Service would be characterised (e.g. by its
interface) or any useful information that a look-up service can exploit.

As an IoT Service is exposing IoT Resources, which are themselves hosted by IoT Devices, the IM authorises
ServiceDescription to contain 0 to many ResourceDescription(s) and Resource Description to contain 0-to-
many DeviceDescription(s). The structure of descriptions is not constrained by the IM and therefore left to
the architect’s own choice.

![IoT Information Model](image)

**Figure 4: IoT Information Model (as in IoT-A D1.5 [1])**

### 2.1.3.3 IoT Functional Model

The *Functional Model* (FM) (see Figure 5 below) proposed by IoT-A corresponds to a service-oriented
approach of IoT. It identifies 7 main *Functional Groups* (FGs) and 2 additional ones that are kept outside
the scope of the IoT ARM. The purpose of the section is to introduce the different layers (FG) of the FM as
they are central to the functional decomposition achieved later on in the Functional View (see Section 4.2).

The Functional Groups are defined as follows [1]:

---

*VirtualEntity*
- entityType
- identifier

*Attribute*
- attributeName
- attributeType

*ValueContainer*
- 0..*

*Service Description*
- 0..*

*Resource Description*
- 0..1

*Device Description*
- 0..*

*MetaData*
- metadataName
- metadataType
- metadataValue
- metadataMetadata

*Association*
- serviceType

---
- **IoT Process Management FG**: The purpose of this FG is to allow the integration of process management systems with the IoT platform. For example, the formal definition of a task-based application (supported by CPaaS.io platform) would fall into this category;

- **Service Organisation FG**: This FG is responsible for composing and orchestrating services, acting as a communication hub between other FGs. The execution of an application described within the IoT Process Management FG would take place in this FG, like any other kind of choreography/orchestration engine;

- **Virtual Entity FG**: This FG relates to VEs as defined in the IoT Domain model, and contains functionalities such as discovering VEs and their associations with IoT Services. This FG also allows access to the VE Service offered (formally “associated with”) by a Virtual Entity. In CPaaS.io those VE Services can be accessed via a VE endpoint;

- **IoT Service FG**: The IoT Service FG contains functions relating to Resource-centric IoT Services. Those services expose the resources like sensors and actuators and provide the means for reading sensor values or setting actuation. It also contains storage capability functionality. More specifically the IoT ARM states that: “A particular type of IoT Service can be the Resource history storage that provides storage capabilities for the measurements generated by resources”;

- **Communication FG**: The Communication FG is used to abstract the communication mechanisms used by the IoT Devices. Communication technologies used between applications and other FGs is out of scope for this FG as these are considered to be typical Internet technologies. A central message bus offering publish/subscribe functionalities would also be part of this FG as we will see when describing the CPaaS.io Functional View;

- **Security FG**: The Security “transversal” FG is responsible for ensuring the security and privacy of IoT-compliant systems. The management of security itself is also part of this FG;

- **Management FG**: The Management “transversal” FG contains components dealing with configuration, faults, reporting, membership and state. It should be mentioned here that this FG works in tight cooperation with the Security FG.
2.1.4 IoT ARM Views

The IoT ARM comes with a comprehensive list of architectural Views (see Figure 1). However not all Views will be touched in this document; the description of the CPaaS.io-related Physical Entity and Context Views, which are scenario specific will fall in the scope of deliverables relating to scenario implementation.

2.1.4.1 Functional View

In this section we provide a reminder (see Figure 6 below) of the “native” IoT-A Functional View as it shows an illustrative example of functional decomposition resulting from a thoroughly conducted requirement analysis (in that particular case it was the analysis of the IoT-A requirements coming from the different stakeholders resulting into the final list of UNIs [24]). We recommend referring to the IoT ARM final architecture deliverable D1.5 [1] in order to get more detail about the purpose of each of the Functional Components shown in that Functional View.
Along with a description of the FCs within FGs it is equally important to get a concise – still precise – description of the different FCs implemented in the various data producers (which step outside the CPaaS.io platform) and to understand also very clearly how they interact with and position w.r.t. other components. At this point in time we concentrate on a clear textual description, but in the architecture document System Use-Cases (UC) (see Section 4.3.1) we will formally elucidate those inter-component interactions.

The CPaaS.io Functional View will be based on the native IoT (native) Functional View but will also introduce those additional FCs which aim at capturing the CPaaS.io specifics.

### 2.1.4.2 Information View

We use the following various viewpoints for describing the information view:

- **Information flow**: shows how information flows between FCs;
- **System use-cases**: elucidate usage patterns and explicitly shows interactions between FCs;
- **Structure of information**: in the case of CPaaS.io we will describe the used ontologies and strategies for storing of information;
- **Sequence diagrams**: temporal sequences of interactions between FCs using e.g. the UML notation.

### 2.1.4.3 Physical Entity View

Considering a specific scenario, the PE View provides:

- The description of the Objects (hence so-called Physical Entities), their physical properties of interest and how those objects are virtualized into the cyber-spaces, explaining how the PEs are “translated” into VEs and how their physical properties can be modelled through VE properties;

---

**Figure 6: IoT Functional View (as in IoT-A D1.5 [1])**
• The physical association between objects and hardware devices, e.g. if a sensor is attached (touching, fixed) to the physical object or if the physical object is in the scope of a detached sensor (e.g. a camera);
• A clear description of the information captured by devices, including meta-data.

There is no CPaaS.io Context View described in this document. Such views will be provided in deliverables relating to the scenario implementation.

2.1.4.4 Context View
According to the IoT ARM [1] and Rozanki & Woods [21] the Context View describes “the relationships, dependencies and interactions between the system and its environment (the people, systems, external entities, with which it interacts)” either using plain text or UML-like notations.

The concerns addressed by the Context View cover the following aspects:
• System scope and responsibilities;
• Identity of external entities and services and data used;
• Nature and characteristics of external entities;
• Nature and characteristics of external interfaces;
• Other external dependencies;
• Impact of the system upon its environment.

In the context of CPaaS.io, a dedicated Context View will be associated with each Scenario implementation and therefore will be documented within deliverables relating to scenario implementation (like for PE Views).

2.1.4.5 Instantiation View
This additional view is not part of the IoT ARM native views and has been introduced in order to cover a special aspect of the CPaaS.io platform, which is that two different concrete instantiations of the platform are derived. The aim of this new view is therefore:

1. To provide a viewpoint where the instantiation of the concrete components can be shown;
2. To provide a instrument for comparing the different technical approaches leading to different implementations;
3. To provide a simplified layout (less abstract than the Functional Model) that fit better the nature of the CPaaS.io technical scope (which is not fully Service Oriented).

The instantiation view also provides a mapping between the “logical” FCs and their concrete counterpart (implemented FCs) as already presented in previous versions of this architecture document (this mapping is also summarised in a table).

It is worth reminding here that this correspondence between logical and concrete FCs is n-to-p, meaning that one (or more) concrete FC(s) can map onto one or more logical FC(s).

We introduce now the viewpoint discussed at the beginning of this section and the new layered model it uses (see Figure 8 below).
Since the CPaaS.io project is clearly more data-centric than service-centric, we provide a layered layout that basically follows the well-known *Data-Information-Knowledge-Wisdom* (DIKW) pyramid model introduced in the literature years ago\(^2\) (see Figure 7 hereafter).

![DIKW model](http://example.com/dikw.png)

**Figure 7: DIKW model\(^3\)**

Figure 8 below shows this Instantiation View layout and DIKW model correspondence. The far left part of this figure shows the correspondence with the DIKW layers. That left-most part of the model is NOT part of the Instantiation View and should not be kept when deriving the two u2-based and FIWARE-based instantiation views.

- **IoT Resource Layer:** this layer corresponds to the IoT Resource and supporting devices. IoT Services exposing those IoT Resource would also be part of this layer ONLY if they are not hosted within the platform; i.e. if they are part of a 3\(^{rd}\) party data provider. The IoT Resource Layer is not in the scope of the CPaaS.io platform since they are controlled by entities outside the platform itself (referred as Data-owners or data providers);
- **IoT Data & Ingestion Layer:** this layer is the first layer of the CPaaS.io platform and is concerned with ingestion and annotation of data released by the IoT Resources (hence belonging to the Information Layer); in this layer data still refers to IoT Resource, and therefore consists of e.g. sensor readings like a Water Levels. This means also that there is no connection whatsoever to a real-world object e.g. a water tank.
- **Virtual Entity Layer:** this second layer is also an Information layer in the DIKW taxonomy. It is the layer where real-world objects are virtualized into Virtual Entities associated with attributes/properties. Despite the level abstraction is higher than in the IoT Layer, one still talks about information; As an illustration, in this layer a real-world object water tank would be existing with a set of properties, including (but not only) a Filling% that would be derived from the water tank maximum capacity and the water level provided by the IoT Resource (sensor);
- **Semantic Data and Integration Layer:** in this layer high-order information, hence called knowledge, is built with alignment to various ontologies and published as linked data;
- **Knowledge layer:** is the higher degree of abstraction where utilizing various AI techniques like for instance machine learning allows to “transform” different pieces of knowledge into knowledge

---

\(^2\) Its origin is not clear but most likely originate from an American organizational theorist named *Russell Ackoff*, back to the late eighties.

which make sense for particular domain of applications, e.g. smart energy, transport, logistics, environmental, etc... this corresponds to the ultimate analysis of the row data received in layer1;

- SMART CITY Service Layer: is where knowledge is eventually exploited with some higher –often non technical, still wise- business, economic, managerial, planning objectives in mind, resulting in wisdom. All SMART CITY applications uses and correlates this knowledge for the sake of reaching specific objectives, and may also use similar knowledge from additional platform instances, if their scope is indeed more than one city. This layer is not part of the CPaaS.io platform but relies on platform enablers in order to reach it own aims.

![Diagram](image.png)

Figure 8: Viewpoint layout for the Instantiation View

### 2.1.4.6 Deployment & Operation View

The Deployment & Operation View (or Deployment View for short) main purpose is to describe how the different FCs and hardware (including gateways, sensors, actuators etc.) are deployed in “real life” according to specific scenarios. The two sets of Deployment Views (one for each of the u2-based and FIWARE-based architectures) consider concrete FCs, pieces of hardware/software (e.g. databases, servers, web servers etc...) supporting those FCs and individual scenarios. They provide the respective descriptions of the two concrete platform deployments (u2- and FIWARE-based):

1. The respective u2-based and FIWARE-based Instantiation Views (see Sections 6.1 and 6.2);
2. any additional deployment of components needed for implementing the scenarios (see the cases relating to Class-I to Class-VI data providers, where some existing infrastructure does exist outside the CPaaS.io platform).
A Deployment View is partly scenario specific. Each CPaaS.io scenario shall describe, using a Deployment View, how it is physically deployed in the field (with information about infrastructure at the RDP side, network, storage and computing resources, including FCs that are part of that Scenario-specific infrastructure (e.g. Semantic Annotator, Resource Manager, Data and Service endpoints...)).

2.1.5 IoT ARM Perspectives

According to Rozanski & Woods [21] an architectural Perspective “is a collection of activities, tactics and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the system’s architectural views”.

In this definition, a quality property is meant to be “an externally visible, non-functional property of a system such as performance, security or scalability” [21].

As we can see architectural Perspectives are orthogonal to architectural Views; therefore any architecture or design decision pertaining to non-functional or quality requirements often spans more than one architectural View, if not all. Following the methodology from Rozanski & Woods [21], the IoT ARM leverages this methodology and proposes to approach and structure those transversal aspects of architecture using a comprehensive list of perspectives that focus on specific non-functional requirements or desired quality properties of the architecture, with special focus on the IoT domain. The IoT ARM proposes the following structure as far as “aspects” of an architecture are concerned:

- Evolution (or Evolvability): is a quality of a system that has been designed in such a way it can easily be adapted to new technologies;
- Interoperability: ability of a system to easily interoperate with other systems at various levels like technical, syntactical, semantic and organisational [13]
- Availability: ability of a system to be fully (or partly) available when required
- Resilience: ability of a system to effectively handle failure or attacks that could affect the system availability
- Trust, Security and Privacy:
  - Security: ability of the system to reliably control, monitor and audit who can perform what actions on what resources, to detect and recover from failures insecurity mechanisms and to resist to cyber attacks
  - Trust: ability of a system to establish and enforce trusted relation between the different parties involved in a system (end-users, component, data) in such a way system operation and behaviours comply to expected ones
  - Privacy: ability of a system to deal with all kind of personal data and in particular to implement reliably privacy policies about accessing, sharing that data or hiding people’s identity
- Performance: ability of a system to predictably perform its operations within its mandated performance requirements and profile
- Scalability: ability of the system to cope with increasing demand in computing, networking, storage resulting from increasing volume of system usage
- Usability: quality that illustrate how easy a system can be used, how easy data can be apprehended by the end users, how easy the GUI is understandable and ergonomic while maintaining efficient work

---

4 adapted from [31]
Of course the list of qualities can be updated or adapted according to architects’ needs.

Each desired quality will be then associated with a set of activities (for instance activities associated with Trust, Security and Privacy are the collection of trust requirements, the conduction of risk and threat analysis, the definition of a trust model, etc.

Then defining a certain number of tactics allows showing how the desired system quality can be eventually reached. Because a tactic can span more than one view, the implementation of a tactic through Design Choices (DC) can lead to more than one of those DCs (e.g. a tactic for realizing semantic interoperability can lead to a collection of DCs relating to Data Structure for the Information View and DCs relating to interfaces, storage and protocols for the Functional View).

2.2 Using the IoT ARM in the context of CPaaS.io

In CPaaS.io we are using the IoT ARM in order to agree, between EU and Japanese colleagues, on the one hand about the logical functionalities (i.e. FCs) that a CPaaS.io concrete platform must provide, and on the other hand about the nature and sequences of interactions that must take place between those logical functional components in order to accomplish typical actions like producing data, consuming data etc. (i.e. system UCs).

The following Section 4 is therefore about identifying and defining all needed FCs, based partly on functional requirements (see Section 4.2), taking as a starting point the functional decomposition provided by IoT-A [1] and describing such sequence of interactions (see Information View/system UCs in Section 4.3.1).

2.2.1 Views

Since Japanese and European consortia are taking different technical approaches when it comes to implementing a concrete platform (respectively u2- and FIWARE-based), we need to provide two additional architectural views (called Instantiation Views), the role of which is:

1. to identify available and still-to-be-implemented u2-based (resp. FIWARE-based) concrete components;
2. to show how that list of concrete components map to the list of logical FCs (so that we can identifying coverage and gaps);
3. to provide an architectural view of the whole u2-based (resp. FIWARE-based) concrete architecture, i.e. instantiation of the common ARM-driven logical system architecture;
4. to show concrete instantiations of the system UCs featured in Section 4.3.1, using the u2-based (resp. FIWARE-based) concrete components.

2.2.2 Perspectives

As far as CPaaS.io is concerned, we will need to synchronise and align, between Japanese and EU partners, about 1/ the desired system properties or qualities and 2/ tactics to be followed, taking also into account that two platforms based upon two different technologies (u2 and FIWARE), need to offer the highest possible level of interoperability (interoperability being itself a perspective). The Volere template shows an initial mapping of the non-functional requirements to the list of Perspectives listed in Section 2.1.5.
3 CPaaS.io Requirements Analysis and Mapping

3.1 Summary of requirement collection phase

In the three following sub-sections we come back on the results of 1/ the two previous initial WP2 and WP3 phases (achieved between M1 and M3 and concluded with respectively D2.1 and D3.1) and 2/ the updated Requirement Specification document D2.5 (provided in M18) and provide updated lists of requirements:

1. Scenario requirements: requirements from the CPaaS.io scenario point of view (Section 3.1.1) as introduced in D2.1
2. Platform requirements: functional requirements (Section 3.1.3) and non-functional requirements (Section 3.1.2) as introduced in D3.1

The purpose of this section is then the analysis and unification of those requirements into a single list of unified requirements (called UNIs in the IoT-A terminology). The following step will be the mapping to views and perspectives, as explained in the previous introductory sections.

3.1.1 Scenario requirements

This section gives a summary of new findings in term of scenario requirements, which need further translation into platform functional and non-functional requirements (to be unified with the existing ones shown in Section 3.1.2 and Section 3.1.3).

This update is based on the final version of the Requirement Specification D2.5 [4] that elucidates requirements from the scenarios perspective. It is used as main input for the new iteration of the Volere sheet.

3.1.1.1 Event Management – Enhanced User Experience

1 new requirement is added (EUE-13): need to access more than one instance platforms, including u2 and FIWARE-based platforms (already covered in the initial version of the Volere sheet).

3.1.1.2 Event Management – Sapporo Visitor Experience

All requirements are new and covered by the existing Volere sheet

3.1.1.3 Event Management – Tokyo Public Transportation

9 new requirements: need for Clock Synchronisation, Data Analytics support, low query latency, Visualisation (which is outside the scope of the platform architecture, but part of dashboards), data sharing (covered already by the platform with the security components and through the dashboards), semantic integration (already covered in the Volere sheet with the Semantic Data integration layer FC).

3.1.1.4 Event Management – Tokyo Management of Service Vehicle

No additional requirement is provided. The requirements pertaining to this scenario are identical to the ones provided for the Tokyo Public Transportation scenario (see above)

3.1.1.5 Waterproof Amsterdam

A mix of updated and new requirements calling for user group/role definition and related control mechanisms, automated control delegation(new), dashboard features (covered) and some additional requirements which have to be covered outside the platform by the WaterNET infrastructure.
3.1.1.6 Yokosuka Emergency Medical Care

6 new requirements calling for analytic services, visualisation (covered by Japanese dashboard), user empowerment (incl. access control which are already covered, clock synchronisation and low data-retrieval latency.

3.1.1.7 Smart Parking

This is a new scenario with new requirements. Those requirements are already part of the list of platform requirements.

3.1.2 Non-functional platform requirements

The following two tables focus on Non-Functional REQUIREments (NFREQ) and Functional REQUIREments (FREQ) from the platform point of view (based on a technical analysis of the "DoW ‘generic’ technical objectives and challenges pertaining to Smart City domain"). In the two following Table 1 and Table 2, we have assigned a priority to the Non-Functional Requirements (NFReq) and Functional Requirements (FReq) where MUST, SHOULD, COULD correspond respectively to HIGH, MEDIUM and LOW priority. In its iterative development, the project main focus is on HIGH priority requirements, still trying to fulfil as many as possible of the lower priority ones.

Table 1: List of Non-Functional Requirements

<table>
<thead>
<tr>
<th>ID #</th>
<th>Description</th>
<th>Perspective</th>
<th>Priority</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFREQ.1</td>
<td>Support different service level agreements (SLA)</td>
<td>Scalability</td>
<td>SHOULD</td>
<td>Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.2</td>
<td>Process services and events on a set of distributed nodes</td>
<td>Scalability</td>
<td>SHOULD</td>
<td>Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.3</td>
<td>Continuously monitor quality of service at runtime</td>
<td>Scalability</td>
<td>SHOULD</td>
<td>Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.4</td>
<td>Balance its load at runtime</td>
<td>Scalability</td>
<td>SHOULD</td>
<td>Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.5</td>
<td>Provide high availability</td>
<td>Availability</td>
<td>SHOULD</td>
<td>Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.6</td>
<td>Guarantee infrastructure availability</td>
<td>Availability</td>
<td>SHOULD</td>
<td>Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.7</td>
<td>Ensure network availability</td>
<td>Availability</td>
<td>SHOULD</td>
<td>Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.8</td>
<td>Be able to perform self-healing</td>
<td>Availability</td>
<td>SHOULD</td>
<td>Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.9</td>
<td>Expose data and services to authorized users</td>
<td>Security</td>
<td>MUST</td>
<td>City Data, Platform</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------</td>
<td>----------</td>
<td>------</td>
<td>---------------------</td>
</tr>
<tr>
<td>NFREQ.10</td>
<td>Ensure services are always accessible to entitled users</td>
<td>Security</td>
<td>MUST</td>
<td>City Data, Platform</td>
</tr>
<tr>
<td>NFREQ.11</td>
<td>Ensure data freshness</td>
<td>Trust, Performance</td>
<td>MUST</td>
<td>Platform</td>
</tr>
<tr>
<td>NFREQ.12</td>
<td>Support access control mechanisms</td>
<td>Security</td>
<td>MUST</td>
<td>Platform</td>
</tr>
<tr>
<td>NFREQ.13</td>
<td>Have security mechanisms to protect data transmission</td>
<td>Security</td>
<td>SHOULD</td>
<td>Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.14</td>
<td>Make it difficult to spy on communicated messages</td>
<td>Security</td>
<td>SHOULD</td>
<td>Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.15</td>
<td>Be able to perform to detect threats at runtime</td>
<td>Security</td>
<td>SHOULD</td>
<td>Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.16</td>
<td>Provide trusted and secure communication and information management</td>
<td>Trust</td>
<td>SHOULD</td>
<td>Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.17</td>
<td>The platform infrastructure and services shall be trustable</td>
<td>Trust</td>
<td>SHOULD</td>
<td>Infrastructure</td>
</tr>
<tr>
<td>NFREQ.18</td>
<td>Allow users to use free services anonymously</td>
<td>Privacy</td>
<td>SHOULD</td>
<td>Societal Needs, Platform</td>
</tr>
<tr>
<td>NFREQ.19</td>
<td>Allow people to use free services anonymously</td>
<td>Privacy</td>
<td>SHOULD</td>
<td>Societal Needs, Platform</td>
</tr>
<tr>
<td>NFREQ.20</td>
<td>Allow users to control which data they are willing to provide and how their data should be used</td>
<td>Privacy</td>
<td>SHOULD</td>
<td>Societal Needs, Platform</td>
</tr>
<tr>
<td>NFREQ.21</td>
<td>Keep users access-control rights/policies secured.</td>
<td>Privacy</td>
<td>SHOULD</td>
<td>Platform</td>
</tr>
<tr>
<td>NFREQ.22</td>
<td>Provide privacy protection for users interacting with the platform</td>
<td>Privacy</td>
<td>SHOULD</td>
<td>Societal Needs, Platform</td>
</tr>
<tr>
<td>NFREQ.23</td>
<td>Provide communication confidentiality</td>
<td>Privacy</td>
<td>SHOULD</td>
<td>Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.24</td>
<td>Be extensible for future technologies.</td>
<td>Evolvability</td>
<td>MUST</td>
<td>Societal Needs, City Data, Platform, Infrastructure</td>
</tr>
<tr>
<td>NFREQ.25</td>
<td>Provide standard interfaces for service providers</td>
<td>Extensibility</td>
<td>SHOULD</td>
<td>Platform, Business Needs</td>
</tr>
<tr>
<td>NFREQ.26</td>
<td>Be able to provide services in an interoperable manner</td>
<td>Extensibility</td>
<td>SHOULD</td>
<td>Platform</td>
</tr>
<tr>
<td>NFREQ.27</td>
<td>Data must be interoperable across the different parties (data / service producers)</td>
<td>Interoperability</td>
<td>MUST</td>
<td>Platform</td>
</tr>
<tr>
<td>NFREQ.28</td>
<td>Provide data quality constraints for data sets</td>
<td>Trust</td>
<td>SHOULD</td>
<td>Business Needs</td>
</tr>
<tr>
<td>NFREQ.29</td>
<td>Enable the linking of data sources across platforms</td>
<td>Evolvability</td>
<td>MUST</td>
<td>Business Needs</td>
</tr>
</tbody>
</table>

### 3.1.3 Functional platform requirements

#### Table 2: List of Functional Requirements

<table>
<thead>
<tr>
<th>ID #</th>
<th>Description</th>
<th>Priority</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREQ.1 [REQ25]</td>
<td>Discovery and lookup service of IoT systems should allow locating the physical entities based on geographical parameters.</td>
<td>SHOULD</td>
<td>Discovery Lookup</td>
</tr>
<tr>
<td>FREQ.2 [REQ31]</td>
<td>The look-up service of CPaaS.io shall withhold or grant information depending on context such as application involved, requesting entity, and security permissions.</td>
<td>MUST</td>
<td>Discovery Lookup</td>
</tr>
<tr>
<td>FREQ.3</td>
<td>The IoT system shall enable the dynamic discovery of relevant virtual entities and their related services based on respective specifications.</td>
<td>MUST</td>
<td>Discovery Look-up</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------</td>
<td>------------------</td>
</tr>
<tr>
<td>[REQ36]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FREQ.4</td>
<td>The IoT system shall enable the dynamic discovery of relevant VEs and their related services based on a geographical location scope.</td>
<td>MUST</td>
<td>Discovery Look-up</td>
</tr>
<tr>
<td>[REQ37]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FREQ.5</td>
<td>The IoT system shall enable the lookup of service descriptions of specified services for Virtual Entities with the VE identifier as key for the lookup.</td>
<td>MUST</td>
<td>Discovery Look-up</td>
</tr>
<tr>
<td>[REQ38]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FREQ.09</td>
<td>Object (Virtual Entities), services and resources shall be semantically described</td>
<td>MUST</td>
<td>Interoperability</td>
</tr>
<tr>
<td>FREQ.10</td>
<td>Any data at the platform level shall have data quality annotations</td>
<td>MUST</td>
<td>Trust</td>
</tr>
<tr>
<td>FREQ.11</td>
<td>Data search and retrieval shall be achieved upon a collection of criteria that includes location (of the Physical entity concerned), characteristics of the underlying IoT Resource reputation and the quality of information.</td>
<td>MUST</td>
<td>Interoperability</td>
</tr>
<tr>
<td>FREQ.12</td>
<td>The platform shall provide a SPARQL end-point for accessing semantic information (either annotated data or semantic descriptions)</td>
<td>MUST</td>
<td>Interoperability</td>
</tr>
<tr>
<td>FREQ.13</td>
<td>Any data at the platform level shall be following an accepted set of standard schemas (ontologies)</td>
<td>MUST</td>
<td>Interoperability</td>
</tr>
<tr>
<td>FREQ.14</td>
<td>The platform shall be able to aggregate data from various sources using LOD principles</td>
<td>MUST</td>
<td>Interoperability</td>
</tr>
<tr>
<td>FREQ.20</td>
<td>User anonymity shall be provided in order to enforce privacy (e.g. at communication level)</td>
<td>MUST</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>[REQ19]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FREQ.21</td>
<td>Personal data in servers should be ciphered by a private key.</td>
<td>SHOULD</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>[REQ81]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FREQ.22</td>
<td>Secure storage of data should be ensured.</td>
<td>SHOULD</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>[REQ84]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FREQ.23</td>
<td>Data owners shall be able to set access-control rights/policies (set up by data owners) to their data stored on resources.</td>
<td>MUST</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>[REQ34]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FREQ.24</td>
<td>Access-control rights/policies (set up by data owners) should not be published publicly.</td>
<td>SHOULD</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------------------------------------------</td>
<td>--------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>FREQ.25</td>
<td>Communicated data shall remain confidential.</td>
<td>MUST</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>FREQ.26</td>
<td>Service providers shall be able to set access-control rights/policies (set up by service owners) to their services</td>
<td>MUST</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>FREQ.30</td>
<td>CPaaS.IO shall be able to evaluate access request depending on access control policies.</td>
<td>MUST</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>FREQ.31</td>
<td>CPaaS.IO shall provide an authorization policy to data, object and service for the different users or devices.</td>
<td>MUST</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>FREQ.32</td>
<td>User or device as part of the platform deployment shall be registered before using any services provided by the platform.</td>
<td>MUST</td>
<td>Management Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>FREQ.33</td>
<td>User or device shall be identified.</td>
<td>MUST</td>
<td>Management</td>
</tr>
<tr>
<td>FREQ.34</td>
<td>User or device shall be authorized to use a service.</td>
<td>MUST</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>FREQ.35</td>
<td>Objects should authenticate a person or object that try to access its data.</td>
<td>SHOULD</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>FREQ.36</td>
<td>Authentication of user or service shall be carried out by devices before delivering data.</td>
<td>MUST</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>FREQ.37</td>
<td>Authentication of user or service shall be carried out to access a service.</td>
<td>MUST</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>FREQ.38</td>
<td>User’s data shall be processed only with the user's consent.</td>
<td>MUST</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
<tr>
<td>FREQ.39</td>
<td>CPaaS.io should support context-aware access policies.</td>
<td>MUST</td>
<td>Security/Privacy &amp; Trust</td>
</tr>
</tbody>
</table>
### 3.2 Requirement Analysis

The starting point of the analysis is the set of consolidated functional and non-functional requirements introduced in the earlier sections 3.1.2 & 3.1.3 taking into account D2.5 updates as explained earlier. Those requirements were consolidated, then duplicates were removed and similar ones were merged in order to minimize their number. Some of scenario-originated requirements were also rewritten in order to remove ambiguities and/or to extend their scope.
Then all scenario-originated requirements were examined and compared to existing NFREQs and FREQs and then were – depending on the result – linked to existing requirements or considered as new requirements (classified then in FREQs, NFREQs or Design Choices). Existing requirements may also be updated in order to cover the scope of the scenario-originated requirements.

The result of this second analysis consists therefore of a list of consolidated UNIs which are fully captured in the second official version of Volere Excel sheet (technical Annex to this document).

A second requirement mapping has been also achieved, with mapping to Views, Perspectives, Functional Groups and Functional Components.

Many comments are also associated to those requirements and will be used to feed forthcoming technical discussions (towards the next architecture release e.g.).

## 4 CPaaS.io Views

### 4.1 Actors involved in data production

When defining the CPaaS.io logical architecture we needed to define various kinds of situations for data provision, resulting in various classes of *Raw Data Providers* (RDP) numbered along decreasing capabilities; Class-I being fully semantics-ready while Class-IV does not own/manage any local infrastructure therefore relying fully on CPaaS.io platform for basically everything. The various RDP classes are described below:

#### 4.1.1 CLASSIFICATION OF RDPs

In CPaaS.io, RDPs are classified as passive RDPs or active RDPs. Active RDPs as feature in the CPaaS.io architecture, correspond to data producers which act as a data sources and data providers, who proactively push data to the CPaaS.io platform (see Class-I and Class-II in Figure 9) while passive RDPs correspond to data sources that the CPaaS.io platform need to query or poll data from (see Class-III, Class-IV and Class-V RDPs in Figure 9).

- **Class-I RDP:** This class of RDP produces data in semantic form and stores locally that data in its own triple store (it may decide to push all or part of that data to the CPaaS.io platform as well). It offers a SPARQL end-point so that semantic information can be queried. In addition a Class-I-RDP advertises its dataset to the CPaaS.io platform. Optionally, a class-I RDP may offer an IoT Service layer that allows either to retrieve directly observations from a sensor or to trigger actuation by an actuator. If so, a Class-I RDP must describe semantically those IoT Services and store their descriptions either locally or at the CPaaS.io side. In the former case, an IoT Service endpoint must be provided in order to allow on the one hand discovery of those IoT Services at the CPaaS.io level as part of the overall federated discovery feature and on the other hand, the handling of IoT Service invocation requests;

- **Class-II RDP:** This class of RDP produces data in a non-semantic form (e.g. Json) and pushes it to the CPaaS.io platform for storage. Data may be stored locally in a non-semantic format and such data sets may consequently have to be advertised to the platform, so that they can be queried by CPaaS.io users. Optionally, a class-II RDP may offer an IoT Service layer that allows either to retrieve directly observation (in native non-semantic form) from a sensor or to trigger actuation from an actuator. If so, Class-II RDP must describe semantically those IoT Services and store those descriptions at the CPaaS.io side. An IoT Service endpoint must be provided in order to allow on the one hand
discovery of those IoT Services at the CPaaS.io level as part of the overall federated discovery feature and on the other hand, the handling of IoT Service invocation requests;

- **Class-III RDP:** In the fourth case, the RDP only manages its IoT Resources locally and relies on CPaaS.io for everything else. In an initial pre-configuration phase it needs to upload its production policy as well as the IoT Services exposing its IoT Resources to the CPaaS.io platform (resp. to the Resource Manager FC and IoT Service FC). It does not provide locally any direct data endpoint, however IoT Services can be of course invoked at the platform level;

- **Class-IV RDP:** they refer in the CPaaS.io architecture to Web third parties, like social media platforms producing live data e.g. twitter, news feeds, RSS …;

- **Class-V RDP:** they refer to open Government data, e.g., data stored in CKAN repositories.

**NOTE:** The various use-cases relating to Class-IV and –V will be dealt with in the third release of the CPaaS.io architecture.

### 4.2 CPaaS.io Functional View

Each sub-section below provides a textual description for each FC and a list of functionalities provided by the FC.

Below is a second version of the logical functional decomposition featuring the current list of FCs which will be confirmed, updated and extended during the forthcoming architecture iterations. The legend is as follows:

- **In Green:** FCs which were already identified in D3.2;
- **In Red:** the new FCS introduced in this iteration;
- **In Dashed lines:** FCs which could be deployed at the edge, closer to the data.
Figure 9: CPaaS.io logical functional architecture
4.2.1 MANAGEMENT FG
The management FG groups a set of FCs offering management functionalities in order to manage platforms and platform federation.

4.2.1.1 Web Front-End FC
The Web Front-End FC is a component that offers a web interface to many of the FCs implemented across the various FGs. Those components include, but are not limited to:

- All management functionalities;
- Discovery and browsing of VE, VE properties, IoT service, IoT Resources, data sets etc.;
- Look-up and resolution;
- Definition of access and privacy policies;
- Management of credentials.

4.2.1.2 Platform Management FC
Platform Management FC is responsible for managing configurations and platform-related aspects of CPaaS.io IoT architecture.

This component deals with both intra- and inter- platform management. Different aspects are covered like:

- Platform federation: Registration of peers and inter-peer links, definition of nature of links between the different platform instances and the functionalities they are supporting: federated data query, discovery, task composing spanning multiple instances, etc.;
- Task deployment: in order to enable smart and flexible deployment of tasks within the platform (see the Task Deployment FC) and in particular towards the platform edge, information about the platform components like network capacity, CPU and memory capacities need to be known at the platform level;
- Management of security across the various platform instances and domains;
- Publishing of new data/knowledge or replication of data from one platform instance to another (e.g. considering a hierarchical model from leaves towards the root) in order to allow multiple input CEP or data processing.

Platform management is an essential requirement for realization of the CPaaS.io platform due to the distributed nature of IoT. In the CPaaS.io platform, there will be vast number of IoT Devices (e.g. sensor nodes) and various hardware and software components. Moreover, different deployments from Japan and Europe may be connected to each other and exchange information based on the federation concept.

4.2.1.3 Resource Deployment Tool FC
The Resource Deployment component is a web-based front end for managing the deployment of IoT Devices. It covers an inventory for managing IoT Devices and sensors and allows IoT deployment managers to plan a deployment of physical devices including a description (location, time, floor plan) of deployments. The tool supports both the deployment of static and mobile devices. The management of mobile devices is realized through creating associations with mobile entities such as people. The data generated by this tool is used for metadata enrichment of raw data and analytic results as described in deliverable D2.2 [3].

Deployment metadata created by the tool instantiates a deployment ontology extending the SSN ontology (see D6.1 for further details about the ontology) and is stored as RDF triples in a triple store (deployment knowledge base) within the platform. An initial version of the deployment tool is available to partners in the project. The deployment knowledge base currently contains inventory and deployment metadata for the “Color Run” event.
4.2.2 IOT PROCESS MANAGEMENT FG

4.2.2.1 Task Composer FC

Task Composer FC is a tool for service designers to compose multiple operators to form the high-level data processing logic of an IoT service. During the design phase, an IoT service is represented by a service topology, which includes a set of linked operators and their annotations. Each operator takes input from IoT devices or from earlier parts of the processing flow. It performs the business logic of the IoT service and passes the intermediate result to the next operator. In terms of implementation, an operator can be realized as a docker-ized application. As illustrated by Figure 10-a, three operators A, B, and C, are used to construct the service topology. Task Composer FC provides a design tool to form such a service topology and also configure and annotate the properties of each operator in the service topology. For example, one of the important properties of an operator is its granularity, which will be taken into account by Service Orchestrator FC to decide how many task instances of such an operator should be instantiated based on the available data. During the runtime, operators are instantiated by Task Deployment FC as tasks based on the orchestration decisions made by Service Orchestrator FC. In the end, the tasks are executed by Task Execution Engine FC and linked with each other during runtime based on the data dependency of their inputs and outputs. Figure 10-b shows an example of how tasks are constructed to form a data processing flow based on the available data.

Figure 10: Core Concepts for IoT Service Orchestration

4.2.2.2 Process Optimization FC

Process Optimization FC is a component that decides on how to execute the processes that are required for providing services. This optimization includes or relies on aspects such as the order of execution of the tasks, the necessary inputs and needed configurations for optimized performance, as well as the optimization algorithms for the processes. Process Optimization FC is utilized by Service Orchestrator to make decisions based on improving the performance of the processes.

4.2.3 SERVICE ORGANISATION FG

4.2.3.1 IoT Resource Manager FC

The Resource Manager FC is an agent/component which can be configured and used by infrastructure-less owners of IoT Resources (see the RDP Class-III case) in order to manage the reading of their IoT Resources and its periodicity.
This FC allows RDPs for defining publishing policies and relies on CPaaS.io-hosted IoT Services which are deployed by the RDPs Class-III and which are exposing the IoT Resources under the RDPs' responsibility.

4.2.3.2 Service Orchestrator FC

Service orchestrator FC is the component which generates task instances and manages execution of these task instances based on the available resources. In order to perform this role, the Service Orchestrator FC must know about the availability of the resources such as the cloud or edge resources (e.g., servers). It monitors the usage of resources and the status of all running tasks on the nodes.

This FC manages all the tasks and the execution of task instances over the edge nodes or cloud nodes. Hence, the assignment of task instances to the available resources is handled by Service Orchestrator FC.

4.2.3.3 Task Deployment FC

Task Deployment FC takes the assignment from Service Orchestrator FC as an input and performs the necessary deployment of task instances. It aims to minimize the cross-edge node and cross-site network traffic while doing these deployments by generating a deployment plan and performs the task deployment on the available cloud or edge resources based on a deployment plan.

4.2.3.4 Task Execution Engine FC

Task Execution Engine FC is the component that executes the tasks that are defined by the Task Composer FC and deployed by the Task Deployment FC based on the available resources. The instantiation of this FC is a worker that executes the necessary tasks.

4.2.4 VIRTUAL ENTITY FG

As explained earlier in Section 2.1.3.3 this FG deals with all aspects pertaining to Virtual Entities like storage of VE-related observations and VE/VE Service Descriptions, discovery, look-up and resolution of VEs and VE Services, discovery, look-up and resolution of Associations and VE Service endpoint. Functionalities shared with the IoT Service FG are handled in Section 4.2.6.

4.2.4.1 Semantic VE Data repository FC

This FC provides a database for storing VE-related data (i.e. historical data) as well as a SPARQL endpoint for accessing that VE-related data.

4.2.4.2 Non-Semantic VE Data repository FC

This FC provides a database for storing VE data in a non-semantic form. The semantic Integration layer can later on ingest it before storing it in a semantic data store. Typical example would be –for the FIWARE based instantiation of it- VE context data expressed in the FIWARE internal NGSI format.

4.2.4.3 VE Registry FC

VE Registry FC is the component that stores, queries and retrieves VE semantic descriptions. This FC could be merged with the VE Resolution FC or can be implemented by a stand-alone component that is integrated with the VE Resolution component.

4.2.4.4 VE Resolution FC

This FC is an extension of the IoT-A native component that can be used to discover and retrieve associations existing between VEs and IoT Services. Using Associations (see Figure 4) the entity managing VE Services can know about which IoT Services should be used in order to return the value of a VE Property.
Deliverable D3.5

H2020 EUJ-02-2016

Page 39 of 110

(as associations between VEs and IoT Services are dynamic in order to deal with mobility). In addition to providing means for retrieving those associations, this component can be used to by VEs to register their availabilities. The existing VEs that are available and registered can be discovered based on fitting the searching criteria. This search can typically be done by the VE Service endpoint FC for identifying the IoT Service to be triggered for answering a READ/UPDATE request upon a specific VE property.

4.2.4.5 VE & IoT Service Monitoring FC
This component is an IoT-A native FC responsible for automatically finding new associations, which are in turn stored by the VE Resolution FC.

4.2.4.6 VE Service End-Point FC
This FC is an extension of the IoT-A native FC that provides a unique endpoint to accessing (Read/Set) Virtual Entities and their properties. This FC is responsible for:

- Maintaining and publishing a snapshot of VE state at regular interval (sampling rate to be configured) as VE observations which are therefore also stored within the Semantic VE Data Repository FC;
- Handling and answering requests for accessing VE properties. As explained in case 2/ in Section 4.3.1.4, answering such requests requires identifying the appropriate corresponding IoT Service via existing associations between device-level and thing-level information.

4.2.5 IOT SERVICE FG
This IoT Service FG deals with all aspects pertaining to IoT data like storage of IoT Data, discovery, look-up and Resolution of IoT Services, invocation of IoT Services, storage, discovery, look-up and resolution of IoT Service descriptions and IoT Resource descriptions.

Functionalities shared with the Virtual Entity FG are handled in Section 4.2.6.

4.2.5.1 Semantic Data Repository FC
Semantic Data Repository FC is used for storing semantic data managed at CPaaS.io level. Semantic information can be annotated sensor readings (annotated raw data coming from RDP) in the form of observation or any kind of semantic data resulting from applying analytics or from processing within the Semantic Integration FC. This FC provides a SPARQL endpoint that can be used for searching and retrieving the data from the database (either it is a full-fledge triple store or relational database with SPARQL endpoint).

4.2.5.2 Non-Semantic Data Repository FC
The Non-Semantic Data Repository is used to store non-semantic data coming e.g. from Raw Data Producers. It can be then used by other components for various tasks like CEP, analytics etc... This data can also be queried by the Semantic Integration FC in order to produce and store semantically annotated data.

4.2.5.3 IoT Service & Resource Registry FC
The IoT Service/Resource Registry FC provides an API for registering a Resource and the associated IoT Services within a registry with associated metadata. This particular API can be used either by CPaaS.io users

---

5 This FC correspond to the IoT Service Resolution FC in Figure 6
Deliverable D3.5

for registering virtual resources and associated IoT services or by RDPs which do not handle locally the definition of the IoT Services that expose their resources (see definition of Class-II and -III).

This registry allows also to look-up IoT Services exposing IoT Resources based on various criteria (metadata).

4.2.5.4 Live Data Listener & Ingestion FC

This component is responsible for pulling data from external data sources (like e.g. Twitter RSS or news feed-which correspond to RDPs Class-IV) and putting it in an appropriate form with needed meta-data before pushing it to the appropriate repository. This FC therefore may use as well the Semantic Integration FC and may publish related knowledge to the communication FG for other components/users to use. Finally this component can also push the data to the Analytics FC or/and Machine Learning FC for further analysis (e.g. classification) before 1/ some higher-level data is published or 2/ the incoming data is enriched its meta-data before being published.

A first version of this component allows collecting tweets according to a given set of #-tags and providing a sentiment analysis over the collected matching tweets (using the Machine Learning component described later on), in the form of a percentile of perceived “positive” tweets vs. “negative” tweets. It is worth noting that this component can deal with historical data and also with feeds (real-time data).

As twitter does not allow the retrieval of all tweets matching given #-tags, only a small –still reasonable- percentile is returned (around 20% as far as we could notice).

An API allows to provide the FC with information such as: list of #-tags, ingestion mode (feed / historical Data), duration of ingestion (in case of mode = Feed), etc.

Data pushed by this component towards FIWARE is Json structured data and may feature various structures depending of the user’s choice when invoking the component.

4.2.5.5 Government (Gov) Data Listener & Ingestion FC

This component behaves in a similar way to the Live Data one, but applies to open-data data sources like government, institutions or NGOs for instance (RDPs Class-V). Such data is usually accessible via CKAN repositories. This FC will be further developed in a next iteration of the architecture in accordance with project scenario needs which as we could see include for instance weather forecast data (SmartParking and Waterproof Amsterdam scenarios).

4.2.5.6 IoT Service FC

This FC provides a REST endpoint for accessing resource data (resources are exposed to CPaaS.io users by IoT Services).

4.2.6 COMMON TO VE AND IOT SERVICE FGs

4.2.6.1 Semantic Integration FC

As described in D6.1, CPaaS.io is using Linked Data and RDF to facilitate integration of data from platforms like FIWARE and u2. Neither of the two base platforms is currently supporting RDF and Linked Data out of the box. To address this, CPaaS.io is providing and integrating a semantic layer that enables mapping existing data to RDF.

This semantic integration is implemented in different phases and levels. In its first form, the semantic layer is simply exposing metadata as Linked Data, using common vocabularies and best practices as described in D6.1 [8]. This enables users to query information about available data within the FIWARE and u2 platform.
as Linked Data. To access this metadata layer the current implementation provides a SPARQL endpoint that can be queried.

In a second step, data residing in FIWARE or u2 can be mapped to RDF or JSON-LD. For that, one needs to extend the appropriate data model used and introduced in D6.1, Section 3.2 [8]. This is done by a semantic middleware that can be configured accordingly. By providing appropriate tools and user interfaces, FIWARE users can thus map existing data to RDF representations. The u-code data model of u2 is close to a RDF as it stores information in a triple-like data model. The semantic integration layer needs to map internal u-code IDs to publicly used and de-referencable URIs, preferably as HTTP URIs to allow Linked Data usage. In some use-cases like the Tokyo Metro real-time data system, this is already being done and can be used as an example implementation.

To be able to query this kind of data, a SPARQL endpoint will proxy requests to the platform⁶. Users should be able to run SPARQL queries on data residing in FIWARE- or u2-based platforms.

4.2.6.2 Pre-processing FC

Pre-processing groups all kinds of algorithms that could apply straight away to the raw data before it is annotated and published. More precise description about needed algorithms will be provided in the next iteration of the architecture as soon as we get more detail about the CPaaS.io scenario developments.

4.2.6.3 Analytics FC

The Analytics FC provides a large variety of algorithms that can take as input a broad range of structured data (e.g. video streaming, time series, t-uple series) in order to perform for instance statistical analysis, machine learning, prediction, anomaly/event detection before publishing their result through the communication channel. Example of analytics (input formats, algorithms, output format) can be found in the two Instantiation Views.

4.2.6.4 Machine Learning FC

This component is still under discussion, however it already provides support for supervised classification tasks, more specifically for determining if social data (i.e. tweets) stemming from social media platform (in that case Twitter) are to be considered as Positive, Negative or Neutral (which we refer to as Sentiment Analysis). The ultimate aim is to publish a semantically enriched digest of the outcome of Sentiment Analysis towards the FIWARE platform. See Section 6.2.1 for more detailed description of the current version of that component.

4.2.7 COMMUNICATION FG

4.2.7.1 Communication Channel FC

The communication channel is a FC that provides a message bus with support for publish/subscribe and publishing/consuming along topics. In the architecture it is used as a way 1/ to provide subscribers with “live data” 2/ to store data using a pre-determined topic (see UCs in Section 4.3.1)

4.2.8 SECURITY FG

The Security FG follows closely the recommendation for IoT-A native FCs as shown in Figure 6. This FG is made of several components already identified in the IoT-A Native Functional View:

---

⁶ See for example R2RML implementation in Stardog: http://docs.stardog.com/#Virtual%20Graphs
• AuthN: Authentication of CPaaS.io users;
• AuthZ: Access-Control policies, decision and enforcement;
• KEM: Key Exchange and management (including as well Group Management);
• Identity Management (including also The key/X509 certificate generation).

Those different FCs are described here after in more detail.

4.2.8.1 Authentication (AuthN) FC

This AuthN\(^7\) component is responsible for enforcing the authentication of registered CPaaS.io users. Based upon a user credential passed to the AuthN FC we can make an assertion about the identity of the user i.e. that they are a known CPaaS.io users.

The AuthN component interacts with the User Management FC when the user registers in order to produce the user credentials (such credentials can take multiple forms e.g. both username/password and an X509 certificate).

When access to a resource or service is requested by a user, the request is captured by a decision point as whether to grant or deny the request based upon whether the requester is authorized to do so. At this point, the AuthN FC can be contacted by the decision point to assert the authenticity of the requester.

4.2.8.2 Authorization (AuthZ) FC

The AuthZ\(^8\) component makes decisions about access control requests (intercepted at access decision points) based upon Access Control Policies (ACPs) which are based on two different technologies: XACML [39] and CapBAC [40] which are based on the concept of a Capability Token (CT) which has been explained in D5.1 and D5.2. The first technology is used for validating the privileges of a resource access request issued by a service or application. The second technology is applied in order to provide the CT which is presented as a proof of possession to be validated by the service which contains the requested resource.

Access control can be applied at the level of look up, e.g.:

- for a request to search for a list of VEs/resources in a specific domain or part of a domain meeting some criteria; or
- for direct access to resources via their interfaces.

Access is denied if the assertions made about the request and requester do not comply with the Access Control Policy. Usually the three parameters of an access request include an assertion (data guaranteeing for the occurrence of an authentication of a user client at a particular time using a particular method of authentication), the targeted resource and requested operation (read or write for instance).

In the same sense, ACPs can also be applied with a different approach. Specifically, the AuthZ component can also make decision about the access to certain information which is being broadcasted. That is, it can implement ACPs based on the attributes of the users’ profiles to the information itself, allowing only the consumers with the right permissions to access to it.

\(^7\) AuthN correspond to the one proposed in Figure 6 as part of the Security FG
\(^8\) AuthZ correspond to the one proposed in Figure 6 as part of the Security FG
4.2.8.3 Access Policy Administration FC

This sub-component of AuthZ FC provides a Policy Administration Point to the AuthZ FC where access policies are defined and managed. A GUI is provided to the owner of resources who wish to protect access; using the GUI the owner can create new access policies, attach them to resources, update access policies and delete access policies.

The Access Policy Administration FC will typically be used by Raw-data producers signing up to CPaaS.io and registering their resources. When they register a resource, they can assign an access policy (applying a default rule e.g. all CPaaS.io users can perform a read operation on the API, or define their own policy using the GUI).

Access Policies defined in this FC will be followed by the AuthZ component when making access control decisions.

4.2.8.4 Key Exchange and Management FC

The Key Exchange and Management (KEM) component manages the exchange of security information between two parties. In particular, it ensures that the keys required to construct a secure and trusted communication channel is carried out in a secure manner.

4.2.8.5 Identity Management

The Identity Management (IdM) component takes into account the management of the identity lifecycle and also the interoperability issues with components like Authentication, Authorization and KEM. While the Authentication component is responsible for asserting the identity of subjects using different IoT frameworks (different identifiers, cryptographic algorithms, key strength), the IdM component must support pseudonym creation across different identity frameworks.

The IdM component needs to interact with the Authentication, Authorization and IoT Service Resolution component in order to provide its functionalities.

- Relationship with the Authentication component;
- Authenticate the requesting subject;
- Authenticate the subject for which a pseudonym is requested.

Relationship with the Authorization component:

- Get the access rights of the requesting subject and of the subject for which a pseudonym is requested;
- Create a new access policy associated to the pseudonym.

One of the tasks of the IdM FC is to enable security features between different IoT frameworks by creating pseudonyms for identities pertaining to other IoT frameworks. The IdM component addresses this kind of problems by issuing pseudonyms and accessory information to trusted subjects so that they can operate (use or provide Services) anonymously. Pseudonyms are temporary identities of fictional subjects (or groups of subjects) whose credentials can be used by a subject or a set of subjects when interacting with other subjects instead of using its own credentials. Access rights associated to pseudonyms depend on those of the identities of the request issuing subject(s). As pseudonyms are identities, they can be used in turn to request other pseudonyms.
4.3 CPaaS.io Information View

The purpose of the Information View is to provide system UCs and information about the information structure. This second aspect of the IV will be dealt with in the next iteration of the architecture, in the form of 1/ CPaaS.io core ontology for data & VE observation, VE/VE Service/IoT Resource/IoT Service Descriptions and 2/ scenario related domain ontologies.

4.3.1 System use-cases

The general idea of having System Use-Cases described in this document is two-fold:

- Having a common understanding on how the platform behaves (meaning how FCs interacts with each other) at both logical and concrete levels (that's why system UCs appears in two different sections, respectively section 4.2 and 6);
- Proposing to CPaaS.io users, a catalogue of usage patterns (or reference implementations) that can be referred to when implementing new scenarios.

This section provides a non-exhaustive list of system use-cases which are used to illustrate the accepted platform behaviours and to reach common understanding between the various partners:

- Configuration phase (pre-operational phase);
- Production of data (depending of various cases) and subsequent data flows between components;
- Consumption of data (ranging to simple consumption to usage of reasoning and analytics);
- Any other typical usage.

When a common understanding is reached each single use-case needs to be "instantiated" within the Instantiation View using concrete components (for both u2- and FIWARE-based platforms).

4.3.1.1 Management phases

Note: The scope of management FC is still under discussion, the following list gives a non-exhaustive overview of the topics to be considered:

- To assign the platform with a unique identifier;
- To define CPaaS.io clients (and actors in general);
- To create and configure a set of roles that the platform users can endorse i.e. actors producing data, accessing data to/from the platform or creating/running application using CPaaS.io capabilities;
- To set-up access rights for roles and actors;
- To register platform components capabilities (as an enabler to Edge Computing);
- To link platform instances with each other (for federation purpose).

4.3.1.1.1 Platform manager creates a new platform user

Creating a new user in the platform involves different activities like:

- Gathering and storing “administrative” data like name, organisation, department, ...
- Creating / Allocating and storing security credentials;
- Setting access policy up.

Some of those actions involve the IdM Functional Components. Please refer to D5.2 [7] for detailed System Use-Cases.
4.3.1.2 RDP pre-configuration phases

We describe in this section UCs (in the form of UML Use-Cases) that should occur at the RDP level before operation phase. They include actions such as:

- Declaring IoT services, VE services and Data endpoints to the platform;
- Describing and registering IoT Services, IoT Resources, VE services semantic descriptions;
- Advertising data sets, ...

4.3.1.2.1 Data producer Class-I pre-configuration

In order to be either forwarded SPARQL data requests by CPaaS.io platform instances or be invoked an IoT Service, the different endpoints (Semantic data endpoints and IoT Service endpoints) must be declared to the platform instance, the RDP is linked to (See Figure 11 below). This also allows discovery requests to reach Class-I RDP as they locally manage IoT Service and IoT Resource semantic descriptions. Optionally, data sets can also be described and advertised so that data can be accessed directly at the RDP level by the CPaaS.io client without requesting that data at the platform level and relying on federation. Finally access-rights must be defined so that discovery, look-up and access can be eventually granted (or denied) to the pre-registered CPaaS.io clients.

![UML UC for Class-I RDP pre-configuration case](image-url)
4.3.1.2.2 Data producer Class-II pre-configuration

The main difference with the Class-I case is that a Class-II RDP needs to provision a Semantic Annotator that will be used to annotate raw data so that it complies with the CPaaS.io ontology (see Figure 12 below). As a Class-II RDP is not semantic-ready, only a non-semantic data endpoint is provided in order to accommodate direct access by a CPaaS.io client to the locally stored raw data (following e.g. prior advertisement of non-semantic data sets as illustrated in Section 4.3.1.4.2).

![Figure 12: UML UC for Class-II RDP pre-configuration case](image)

4.3.1.2.3 Data producer Class-III pre-configuration

As explained in Section 4.1 RDPs Class-III are infrastructure-less parties that only own IoT Resources. They rely on the CPaaS.io platform for:

- Registering their IoT Resources so that they can be discovered or looked-up;
- Serving, registering and managing IoT Services that expose their IoT Service so that they can be discovered, looked up and invoked (for synchronous live data consumption);
- Pulling data from the IoT Resources;
- Annotating semantically the data;
- Producing the data towards the Semantic Data Repository.
The two first items of this list pertain to a configuration phase that occurs prior to any data production (which is elucidated in Section 4.3.1.4). The UML use-case below (Figure 12) explains in details those actions.

RDP Class–III have the responsibility of providing IoT Services exposing their IoT Resources, they also provide IoT Service and IoT Resource semantic descriptions. The sequence of actions is therefore to 1/ to deploy IoT Service to the CPaaS.io (the needed IoT Service server is not shown in the architecture but will be part of the Deployment and Instantiation Views) 2/ to register the descriptions to the IoT Service and Resource registry FC 3/ to provide a polling policy to the Resource Manager FC.

**4.3.1.3 Declaring a new platform to the CPaaS.io eco-system**

For the sake of platform federation, a new platform instance needs to be known from existing CPaaS.io platform instances. This UML use-case will be fully described as soon as aspects pertaining to federation have been fully dealt with (architecture next release) from both EU and Japanese sides.

**4.3.1.4 Production of data**

There are basically two main ways to producing data:

- Asynchronously: it is meant here that the data is produced on its own and then fed into the platform (published, stored) at the initiative of the data producers, following its own production policy (including sampling rate). In this scenario producers can either fully rely on platform capabilities (we will see later on what this exactly means) or rely on their own infrastructure. In this later case they will decide which part of the data needs to be published to the CPaaS.io platform;
Synchronously (or on-demand): it is meant in this second mode, that the data will be pulled from the IoT Resource (i.e. sensor) only when there is a request from “the top” to receive or consume such data, and this is done basically by triggering an IoT Service exposing that IoT Resource. Again this case refines into two sub-cases:

- The IoT Service is triggered explicitly, i.e. through a direct call like for instance a REST GET;
- The IoT Service is triggered implicitly whenever a platform user wants to access a VE property (as VE properties are bound to IoT Service through the use of Associations). This second case makes the assumption that the VE endpoint implements this mechanism.

If the asynchronous mode is used, accessing a VE property results in the VE endpoint retrieving the latest stored value for the IoT Resource bound to the property via the Association. Alternatively it is possible that the direct invocation to the IoT Service is made in the case the RDP provides an IoT Service endpoint as well (even if working mostly in the asynchronous mode).

Publishing of data asynchronously is made through the Communication Channel which offers publish/subscribe functionalities. Implicit storage can be achieved when publishing along specific topic (see Section 4.3.1.6).

This section gives illustrations to those different possibilities, while section 4.3.1.4.3 will focus on data consumption.

### 4.3.1.4.1 Data Producer Class-I produces data to the platform

We develop here three different cases:

- Case I.a/: corresponds to classical publishing of semantic data (without federation capability);
- Case I.b/: corresponds to classical publishing of semantic data (with federation capability);
- Case I.c/: corresponds to the publishing of data/data set descriptions.

The detail of those 3 cases is discussed hereafter:

- Case I.a/: The general idea (Case I.a/) explained in Figure 14 (shown with the plain red arrows) is that a Class-I RDP follows its own publishing policy and therefore pulls and stores data at a pre-determined rate and then publishes its semantic data (or part of it) to the CPaaS.io semantic repository using the Communication Channel FC (which can be seen as a message bus with publish/subscribe capability (see the Asynchronous case in 4.3.1.4 1/)). Relevant interactions in Figure 14 are 1/, 2/, 3/, 4/ & 5/. Corresponding data consumption is illustrated in section 4.3.1.5.1 Case II.a/.
- Case I.b/: This case is very close to Case I.a/. Main difference is that, since the semantic data is already stored locally there is no additional publishing towards CPaaS.io platform. Data stays local and any attempt to consume data using a SPARQL request at the CPaaS.io level is forwarded to the data endpoint provided by the CPaaS.io-registered Data Providers. Relevant interactions in Figure 14 thus are only 1/, 2/ and 3/.
- Case I.c/: This last case illustrates a quite different scenario, where the RDP stores the semantic data within his own repository (therefore locally outside the CPaaS.io) and only publishes to CPaaS.io semantic repository a semantic description of the data sets that can be therefore accessed only locally. This scenario is quite different from 1.a/ but is still very relevant in practice. Following this strategy, a data consumer will not consume the semantic data from the CPaaS.io repository like in case II.a/ (see Section 4.3.1.4.1) but indeed will discover which data is available and how to access it. The final part of the consumption process involves a classical SPARQL request sent out to the RDP data endpoint in order to retrieve the semantic data from the local semantic data store. Relevant interactions are 1/, 2/, 3/, 4bis/ and 5bis/.
4.3.1.4.2 Data Producer Class-II produces data towards the platform

In the Class-II case there is no local support to semantic so only non-semantic data is stored locally. The RDP then needs to annotate the raw data (aligning it with the CPaaS.io ontology) before publishing to the platform. Only a non-semantic data endpoint is provided by such RDPs.

- **Case II.a/:** This case follows the general idea already presented in section 4.3.1.4.1 Case 1/. Data is retrieved by the Resource Manager FC and stored locally, then annotated in order to comply with CPaaS.io ontology and published towards CPaaS.io Semantic repository via the Communication Channel FC. Relevant interactions in Figure 15 are 1/, 2/, 3/, 4/, 5/ & 6/;

- **Case II.b/:** This second case is similar to Case I.b/ described in section 4.3.1.4.1. Instead of systematically annotating and publishing data towards CPaaS.io, the RDP publishes only descriptions of available data or data set (incl. information about how to access that data). Relevant interactions in Figure 15 are 1/, 2/, 3/, 4bis/ and 5bis/.

![Figure 14: Data production for Class-I RDPs](image-url)
### 4.3.1.4.3 Data Producer Class-III produces data towards the platform

Since a RDP Class-III is passive, it does not *per se* push data towards the platform. On the contrary an agent (namely IoT Resource Manager FC) located at the platform side is responsible for polling data from the resources owned but the RDP and pushing it to the platform.

A WEB front-end allows the owner of the IoT Resources to specify a polling policy like for instance a time schedule, sampling rate and topic to be used, would the data be forwarded to the Communication Channel for real-time consumption. Instead of using this channel, the RDP may decide to store it directly within the non-semantic data repository.

### 4.3.1.4.4 Data Producer Class-IV & V produce data towards the platform

Class-IV and –V RDP are yet another two similar classes of passive RDP which are relying on platform agents for polling data and passing it on towards the platform. Like for Class-III RDPs, they rely respectively on the two Live Data Listener & Ingestion (LDLI) and Government Data Listener & Ingestion (GDLI) FCs part of the IoT Service FG. Both FCs are instrumented via either a web front-end page where the nature of data to be retrieved / subscribe to is specified or directly via the FC API (this is the mode used by the ML component).

The LDLI is focussed only on Twitter in its preliminary version. It can retrieve data in two different modes:

- Streaming data: it then subscribes to a tweet feed according to a given list of hash tags and strings and sends to the Communication Channel;
- Historical data: it retrieves in one-go a bunch of social media data and sends it to the Communication Channel or back to the invoking party (e.g. the ML component).

### 4.3.1.5 Consumption of data

Consumption of data can be done as follows:

- Live data
• Invoking an IoT Service, in order to either retrieve annotated current readings from a sensor resource, or setting a new actuator value;
• Invoking a VE-service in order to get directly current value of VE properties;
• Historical data (stored data)
• Retrieving VE-level observations or annotated sensor historical data using SPARQL queries.

4.3.1.5.1 CPaaS.io client consumes data originating from RDP Class-I

Data consumption from Class-I RDP offers 5 alternative methods:

• Case I.a/: Data previously published by a Class-I RDP can be consumed from the CPaaS.io semantic repository via a SPARQL request (this corresponds to the case where the RDP either duplicates all its semantic data or publishes surveys or part of its semantic data within CPaaS.io repository) to the CPaaS.io semantic data repository;
• Case I.b/: Alternatively if the RDP local platform has registered its data endpoint to the CPaaS.io, the federation capabilities of the platform, makes sure that any incoming SPARQL request is forwarded to the RDP. The request is therefore answered by the RDP itself;
• Case I.c/: In this last case, the data endpoint of the RDP is accessed directly by the CPaaS.io client as means to access the data/data set and data/data set descriptions were previously advertised to the platform and therefore could be discovered by the client prior to data consumption (See Case I.c/ in Section 4.3.1.4.1);
• Case I.d/: As far as live data is concerned, the CPaaS.io client can invoke directly an IoT Service (following discovery and resolution) via the IoT Service endpoint and therefore being returned the current annotated value;
• Case I.e/: Subscribing to the Communication Channel in order to receive automatically the flow of produced data.

4.3.1.5.2 CPaaS.io client consumes data originating from RDP Class-II

Data consumption from Class-II RDP offers 4 alternative methods:

• Case II.a/: like in Case I.a/ above the historical data can be accessed from the CPaaS.io Semantic data Repository FC via a SPARQL request sent to the Semantic Data Repository SPARQL data endpoint;
• Case II.b/: Corresponding to Case I.b/ above, {historical data/data set} has been previously advertised and therefore can be accessed directly by the CPaaS.io client from the RDP SPARQL data endpoint, without involving the CPaaS.io platform (apart for the discovery phase);
• Case II.c/: Subscribing to the Communication Channel in order to receive automatically the flow of produced data;
• Case II.d/: As far as live data is concerned, if the CPaaS.io client is not satisfied with the sampling rate implemented by the RDP, it can also invoke directly an IoT Service (following discovery and resolution) via the IoT Service endpoint and therefore being returned the current annotated value in near real time.

4.3.1.5.3 CPaaS.io client consumes data originating from RDP Class-III

Data consumption from Class-III RDP is achieved in 3 ways:

• Using SPARQL request at the CPaaS.io platform level in order to query historical data from the Semantic Data Repository;
• Invoking IoT Services exposing the RDP resources via the IoT Service server at the platform level;
• Subscribing to the Communication Channel for live data consumption.
4.3.1.5.4 CPaaS.io client consumes data originating from RDP Class-IV or Class-V

Data consumption from Class-IV or Class-V RDPs is achieved in 3 ways:

- Using SPARQL request at the CPaaS.io platform level in order to query historical data from the Semantic Data Repository;
- Invoking the API from (respectively) the LDLI and GDLI FCs;
- Subscribing to the Communication Channel for live data consumption.

4.3.1.6 Storage of Data

Storage of IoT data or VE data can be achieved explicitly using the two Semantic/Non-semantic Data Repository FC and Semantic/Non-semantic VE Data Repository FC APIs or implicitly when publishing the data to the Communication Channel along a pre-determined topic.

By default the four data repositories (VE-level and IoT-level) are listening the Communication Channel and storing everything coming along pre-determined topics they are assigned to.

On the one hand, currently the brokers deployed for these platforms (IoTBroker and Orion Context Broker) make use of NoSQL databases such as MongoDB or CouchDB because of its versatility at representing the information. Compared with traditional SQL databases where the user needs to pre-define the table where the information is going to be stored by means of a set of fields, NoSQL databases are based on JSON (JavaScript Object Notation) or BSON (Binary JSON) documents without predefining the schemes into the corresponding tables.

Thanks to this representation format, heterogeneous information sources can be quickly stored providing also a good trade-off regarding the performance in accessing the stored information.

On the other hand, a lot of efforts have been put to provide a richer representation format using Linked Data. One of those efforts is the definition of a standard API for Context Information Management (NGSI-LD API) [44], enabling close to real-time access to information coming from many different sources. In this standard, an NGSI-LD Entity shall be represented by an object encoded using JSON-LD, a JSON-based format to serialize Linked Data. The NGSI-LD API incorporates the latest advances from Linked Data and brings the possibility of publishing data to NoSQL, RDF or graph databases, such as Apache Jena, Neo4J, OrientDB, etc.

4.3.2 CPaaS.io ontologies and internal formats

FIWARE and u2 platforms are using different data formats internally and expose different public APIs. For data users this is inconvenient and increases the amount of work needed to consume CPaaS.io-based data in these platforms. To address this issue and increase data and information re-use, the RDF data model and according ontologies are used.

Using ontologies implies exposing metadata and/or data in RDF. This can either be done by implementing an additional RDF data store or by mapping existing data to RDF, see Section 4.2.6.1 for more detail.

There are two main areas where ontologies are used: Exposing metadata, i.e. information about the sensor or exposing data, i.e. observations of a sensor. Ontologies will be used for describing PEs (via the modelling of VE properties), for metadata like quality information, and for linking of data sets from different providers. In particular for the federation of implementation platforms between Europe and Japan, as the implementation platforms in the two regions provide the data in different forms (NGSI in Europe, ucR in Japan).
As mentioned in Deliverable D6.1 [8], the project is following the W3C recommendation on “Data on the Web Best Practices”\(^9\) and makes use of existing ontologies and evolving standards. A subset of used ontologies is mentioned below, more details and links to the specifications of these ontologies can be found in the Data on the Web Best Practices document. The application and, where necessary, extension of these ontologies acts as a CPaaS.io core ontology. For more specific details about ontologies mentioned in this document, please refer to Deliverables D6.1 [8], D6.3 [9] and forthcoming D6.4.

### 4.3.2.1 Metadata ontologies

The goal of metadata ontologies is to provide information about the data available in the platform. The most basic and useful vocabulary is Data Catalog Vocabulary (DCAT). For every dataset some basic information in DCAT needs to be published, including some descriptive text about what the dataset is about. To add trust about the data set, Data Quality Vocabulary is used. This vocabulary encourages specific definitions of so called quality dimensions. These dimensions should relate to the specific data and use cases. Within forthcoming Deliverable D6.4, specific CPaaS.io dimensions are developed.

In the context of CPaaS.io and Smart Cities, data is often captured by IoT devices. Providing metadata about the sensor and sensor network is essential to understand and interpret the data properly. Initially CPaaS.io was focusing on multiple sensor related ontologies. In a second round of discussions the previously favoured FIESTA-IoT ontology was discarded due to some doubt about its sustainability. In the meanwhile a completely revised version of the Semantic Sensor Network Ontology (SSN) was released. This version includes actuation and has been released by the W3C consortium (Spring 2017). It is still lacking some concepts and relations but we are confident it could be extended in order to cover those missing ones, using for instance fragments of OWL-S or IoT-A ontologies (for covering the descriptions of IoT-services and others like e.g. analytics-as-a-service).

We envision that the main usages for the CPaaS.io core ontology will be:

- To model/describe entities of interest (PEs) and their virtual counterparts, the VEs (classes and instances);
- To model/describe IoT Resources (i.e. Sensors and Actuators);
- To model/describe Sensor and VE observations (only sensor observations are covered by SSN2);
- To model/describe Actuation actions and VE actions;
- To model/describe Services profiles (IoT Services and general purpose services e.g. analytics);
- To describe data-sets (either stored locally or within the platform);
- To model Domain-related concepts and relations.

The full specification of the CPaaS.io ontology will be provided during Year 2 and Year 3 of the project, and documented partly in D3.5 and fully in WP6 deliverables D6.1 [8], D6.3 [9] and forthcoming D6.4 and D3.7 (Year 3 deliverables).

Additional metadata can and should be provided according to Data on the Web Best Practices. Adding information like provenance and license information are recommended to encourage data re-use.

### 4.3.2.2 Data ontologies

SSN2 not only provides an ontology for metadata about sensors and captured data but also the concept of Observation. Observation can be used to describe a specific measurement of a sensor in RDF. While the SSN concept of Observation is rather simple it is close to the definition of an Observation in the RDF Data

---

\(^9\) https://www.w3.org/TR/dwbp/
Cube Vocabulary. This vocabulary seems to become the de-facto standard for publishing multidimensional data in RDF and is actively used in many different domains, from publishing statistical data to environmental, geospatial or earth observation data. In the RDF Data Cube Vocabulary an observation is built around three different component types called dimensions, attributes and measures. Any of these components is defined within the specific use case, which makes applying it in a specific context straightforward. SSN Observation can be seen as a simplified version of that which often can be enough in the context of IoT devices.

For many CPaaS.io related use cases publishing sensor data in any of the two Observation concepts should be sufficient.

5 Perspectives

This section provides a preliminary description of the CPaaS.io perspectives and starts sketching strategies for reaching the identified targeted properties/qualities of the CPaaS.io platform. The objective here is to refine strategies and find synchronisation points between the Japanese and European approaches so that interoperability can be eventually ensured. See Section 2.1.5 for an introduction to Perspectives.

5.1 Security

5.1.1 Policy Based Authorization for Access Control

Description: The Access Control module within CPaaS.io is responsible for making authorization decisions based on access control policies. These polices define the permissions that a subject (smart object or user) has over certain target resources (e.g. IoT service). Thus, the policies specify which particular actions that subjects or groups are allowed to perform over a target resource under certain conditions. These conditions usually refer to the context information provided by the context manager module. In this regard, the eXtensible Access Control Markup Language (XACML) [26] policy language will be adopted in CPaaS.io to deal with authorization policies.

Additionally, it is desirable to employ access control solutions specially designed for IoT scenarios. Recently, the capability-based access control model (CapBAC) has been postulated as a realistic and promising approach to be deployed on IoT scenarios. In this sense, the authorization component is able to perform capability-based access control based on the mechanism presented in [27].

In a typical CapBAC scenario, an entity (subject) tries to access a resource of another entity (target). Usually, a third party (issuer) generates a token for the subject specifying which privileges it has. Thus, when the subject attempts to access a resource hosted in the target, it attaches the token which was generated by the issuer. Then, the target evaluates the token granting or denying access to the resource. Therefore, a subject which wishes to get access to certain information from a target, requires sending the token together with the request. Thus, the target that receives such token already knows the privileges that the subject has and a local Policy Decision Point (PDP) only needs to verify the capability token is valid. This simplifies the access control mechanism, and it is a relevant feature in scenarios with resource-constrained devices since complex access control policies are not required to be deployed on constrained end-devices.

Additionally, the token must be tamper-proof and unequivocally identified in order to be considered in a real environment. Therefore, it is necessary to consider suitable cryptographic mechanisms to be used even on resource-constrained devices which enable an end-to-end secure access control mechanism. Moreover, given its high level of flexibility, CapBAC could be also extended to consider contextual conditions related...
to parameters which are locally sensed by end-devices. Also, this model could be complemented by other access control models by providing automated tools to infer the privileges to be embedded into the token.

On the other hand, this component is also responsible for providing access control to the information broadcast in a shared media allowing only the consumers with the right credentials to gain access to such information.

To do so, this component uses CP-ABE [35], an encryption schema that ciphers information under a certain policy of attributes, and the keys of the users are associated with sets of attributes. This enables dissemination of information using different information sharing models where the producer of the information is always in control of how the information is accessed.

CP-ABE is considered, because IoT Systems need not only effective ways of coping with security issues but also efficient ones. In this sense, although the classic security schemes such as Secret Key Cryptography (SKC) and Private Key Cryptography (PKC) provide proven security primitives, they may not be suitable for distributed systems as IoT. The use of CP-ABE is an interesting improvement to the other ciphering schemes. With CP-ABE, the information can only be ciphered with the policies previously defined, and all users whose keys satisfy the policy are able to decipher the information. This not only saves resources, but might also be well related with the attribute-based nature of XACML, giving consistency to the security functionality.

5.1.2 Authentication

**Description:** Authentication ensures that an identity of a subject (user or smart object) is valid, i.e., that the subject is indeed who it or what it claims to be. The authentication module enables authenticating users and smart objects based on the provided credentials. The credential can be in form of login/password, shared key, digital certificate.

Traditional authentication mechanisms based on for instance login-password or electronic IDs have been already solved in by other projects and solutions. CPaaS.io focuses on alternative and more sophisticated ways of performing authentication ensuring, at the same time, privacy and minimal disclosure of attributes. Thus, this kind of alternative privacy-preserving way of authentication using anonymous credential system is handled in CPaaS.io framework by the Identity Management Component.

The IdM is able to verify anonymous credential and then, in case the identity is proved, the IdM interacts with the authentication module which is the one that actually delivers authentication assertion to be used during a transaction.

5.1.3 Privacy Preserving Identity Management

**Description:** The Identity Management (IdM) module is responsible for managing the identities of smart objects and users. The module is able to take into account privacy concerns to manage subjects’ credentials (from users or smart objects), in a privacy-preserving way relying on anonymous credential systems. It is able to endow subjects with mechanisms to ensure anonymity, which is done mainly issuing pseudonyms and proof of credentials to minimal personal data disclosure.

The IdM system is responsible for managing and storing the credentials, which are used by subjects to request information from an IoT services. The IdM allows users to obtain the proper credentials that can be use afterwards to derive proofs of identity to be presented to a Verifier (e.g., IoT Service provider) in order to prove the identity condition while disclosing the minimum amount of private information.

The component is envisioned as endowed with a cryptographic engine like Idemix [28] responsible for low level cryptographic operations required by the anonymous credential system.
5.1.4 Extension to Key Exchange and Management

**Description:** The Key Exchange and Management (KEM) component assists peers involved in a communication in the process of establishing a security context, like for instance, setting up tunnels for a security communication. It involves cryptographic key exchange and provides interoperability between the peers in order to reach an agreement regarding the security functions to use for the communication. This component is also in charge of storing and generating the cryptographic keys.

The Group Manager module as an extension of the KEM, enables sharing information in a secure and private way, with those groups of entities that satisfy certain particular sets of identity attributes values. These particular sets of attributes are represented by attribute sharing policies, which are influenced by context information where the data sharing is being performed.

This module is envisioned to manage dynamic sharing groups. To this aim, the group manager makes use of attribute based encryption mechanism, namely the Ciphertext-Policy Attribute-Based Encryption (CP-ABE) scheme [29]. With this scheme, data to be shared within a group is encrypted under a policy of attributes, while keys of participants are associated with sets of attributes. Only those target users that satisfy particular identity attributes (those given in the policy) possess the cipher keys to decrypt the data. In this way, a data producer can exert full control over how the information is disseminated to other entities, while a consumer’s identity can be intuitively reflected by a certain private key.

5.1.5 Interoperability between platforms from EU and Japan

**Description:** Interoperability is a fundamental aspect in the scope of CPaaS.io project. For this reason, we have considered two different scenarios where such security considerations take place: Integration of Personal Data Stores (PDS) from both platforms, and the integration of the access control management in smart buildings from them too.

For the first scenario, where PDSs are managed by a user in different countries (Figure 16), an Open ID interoperability must be granted. To do so, our proposal comprises the integration of this security mechanism, as well as a development of a PDS from the EU-side which will comply with the API already defined by the Japanese platform. This way, through the Open ID technology interoperability between EU and Japan will be granted.
Regarding the second scenario, building access control management, a Context-Aware Access Control System for Physical Space (CAACS) has been adopted by our partners from Japan, Figure 17. This technique makes use of the XACML architecture where the well known PEP, PDP, PAP and PIP entities for administration, decision, enforcement and information policies have been instantiated as depicted in Figure 18. For this reason, we plan to integrate this security mechanism with our Capability Access Control Mechanism which also makes use of PAP, PDP and PEP for managing, deciding and enforcing XACML policies by using Capability Tokens.

Figure 16. EU-JP joint security scenario

Figure 17. Access Control System for Smart Building. Floors representation
5.2 Semantic Interoperability

Information Model

**Description:** The FIWARE focuses on a common data model and powerful interfaces for searching and finding information. FIWARE is using the OMA Next Generation Service Interface (NGSI) data model as the common information model of IoT-based systems and the protocol for communication. We describe the NGSI9 and NGSI10 below.

**NGSI9** is used to manage the availability of context entity. A system component can register the availability of context information, and later on the other system component can issue either discover or subscribe messages to find out the registered new context information.

**NGSI10** is used to enable the context data transfer between data producers and data consumers. NGSI10 has query, update, subscribe and notify context operations for providing context values. A context broker is necessary for establishing data flow between different resources as well as consumers or providers.

As a semantic data consumer accessing data in either a u2 or FIWARE platform should be as transparent as possible. This requires semantic interoperability on different levels: First one needs to proxy a SPARQL query to the correct platform and second the platform should be able to handle different schema and vocabularies and return unified results over both platforms.

The first requirement can be met by providing transparent federation for different SPARQL endpoints. From a users perspective only one SPARQL endpoint is accessed. This SPARQL endpoint acts as a proxy for two or more SPARQL endpoints in the FIWARE & u2 platform. A possible solution for this approach is CostFed [41], an index-assisted federation engine for federated SPARQL query processing over multiple SPARQL endpoints.

Semantic interoperability on schema and vocabulary level can be provided by supporting SPARQL 1.1 Entailment Regimes [42]. RDF and OWL provide semantic interpretations for RDF graphs that allow additional RDF statements to be inferred from explicitly given assertions. In SPARQL endpoint
implementations this inference is called reasoning, multiple open source and commercial endpoints support various levels of reasoning on top of RDF, RDFS and OWL. Reasoning can only work if relationships between multiple schema and vocabularies are clearly described [43]. In the case of FIWARE and u2, this mapping needs to be described accordingly and reasoning needs to be supported on SPARQL endpoint level to provide transparency in data access for the user.

5.3 Performance

Service Orchestrator

Description: The Service Orchestrator module is responsible for generating processing task instances and assigning them to proper resources (e.g., edge or cloud nodes) for task deployment, based on the topology defined by developers, the current system status, and also some parameters (e.g., the scope of data sources) given by the operator on the fly. Meanwhile, the Service Orchestrator is managing all task instances over the resources at the global level.

Service orchestrator allows leveraging edge for more efficiency and higher performance in computing and communication. Leveraging edge computing with Service Orchestrator allows significant bandwidth reductions, especially for scenarios such as input streams (e.g., video streams captured by cameras) are filtered or even analysed on the edge and only the filtered values or analytical results are sent to cloud as opposed to sending the raw data as it is.

5.4 Scalability

Federation Agent

Description: Federation agent of the IoT Broker provides capability of forwarding data from one IoT Broker to another with hierarchical or mash-up topologies. For instance, one deployment in a certain area using an IoT Broker can be accessed by another deployment with another IoT Broker. Hence, Federation Agent separates IoT data in different domains and enhances the privacy by giving control of the data access to the IoT domain administrators.

IoT Broker federation allows local deployments to be handled by a local IoT Broker and multiple IoT Brokers can access data from each other when necessary. Moreover, it may also allow deployments from different domains in the same region (e.g., a smart city with multiple business domains) having different IoT Brokers. Compared to the centralized context brokering approach where there is a context broker in the cloud which collects and manages all information, using Federation Agent is more scalable and distributed approach which is suitable for IoT deployments.

6 Instantiation views

6.1 U2-based platform Instantiation View

The TRON Project, since its inception in 1984, has been aiming at the realization of Highly Functionally Distributed System (HFDS) where any objects worth connected in our surroundings will be embedded with computers and work in a collaborative manner to offer better service to human users. The concept of HFDS has been realized partially by the concept of the IoT and/or ubiquitous computing. In a sense, the world has finally caught up with the project. So far the project has attempted to realize HFDS by developing
technological components that would be necessary such as node devices, identification infrastructure, information distribution infrastructure, etc. We are now at the stage when we can combine the components to build a system as a whole and fill in the details since the component technologies have matured.

6.1.1 Description of u2 Functional Components

6.1.1.1 Pre-requisites

In this short section we introduce some concepts which are at the centre of the u2 technology.

6.1.1.1.1 ucode

To identify individual objects, spaces, and concepts in the real world, unique identifiers are assigned to respective objects, spaces and concepts that we wish to identify. This model is called a ucode model. The target in the real world to be identified in the ucode model (see Figure 19 and Figure 20) is called an "entity," and the unique identifier assigned for identifying the entity is called a ucode.

The entity consists of objects, places, and concepts. "Objects" in the ucode model include tangible objects such as industrial products and agricultural crops, and intangible objects such as content and programs. "Places" include features in the real world such as roads and buildings and more detailed smaller components in the real world such as rooms and corridors. "Concepts" include the relationships between "objects" and "places" and the information which can be the real-world context. In a nutshell, the entity is a target to be identified among objects, places, or concepts in the real world, and the identifier to identify the entity is a ucode.

\[
\text{ucode} = \text{version} + \text{TLD Code (TLDc)} + \text{Class Code (cc)} + \text{SLD Code (SLDc)} + \text{Identification Code (ic)}
\]

Figure 19: ucode structure

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>4 bits</td>
</tr>
<tr>
<td>Top Level Domain Code: TLDc</td>
<td>16 bits</td>
</tr>
<tr>
<td>Class Code: cc</td>
<td>4 bits</td>
</tr>
<tr>
<td>Second Level Domain Code: SLDc</td>
<td>Multiple types</td>
</tr>
<tr>
<td>Identification Code: ic</td>
<td>Multiple types</td>
</tr>
</tbody>
</table>

Figure 20: ucode field names and their widths
6.1.1.1.2 ucR: ucode Relation

**ucR**: u2 architecture represents the real-world context by modeling the relation on a real-world entity as a relational representation among ucodes allocated to entities or between the ucode and atom, which is a value such as a string, a number or a date. This representation model is called a ucode Relation model (ucR model). When a relationship is represented by ucR model, it is called a Relation.

**ucR Unit**: The basic unit of the ucR model consists of two ucodes or a ucode and atom, and their relationships. Moreover, the respective relationships between the two are provided with a logical ucode called a relation ucode. This basic unit consisting of the above triplet is called a ucode relation unit (ucR unit). When this triplet is set in a sentence where the relationships are represented as the predicate, the ucode as the subject is called a subject ucode, and the ucode as the predicate which is a relation part is called a relation ucode, and the ucode as the object or complement is called an object ucode. The atom can be substituted for the object ucode. For example, in a description, "R of S is O" or "there is a relation R between S and O," S refers to the subject ucode, R to the relation ucode, and O to the object ucode or atom. When it is not necessary to uniquely identify any of the three elements comprising a ucR unit, their relevant parts can be left blank.

**ucR Graph**: The information and context about an entity are represented by combining relations among multiple entities. For example, for information on an entity indicating a “stuffed cabbage,” pieces of information such as its recipe, a cabbage as ingredients, and a production area of the cabbage are combined. To represent such information, ucR units are combined, and a digraph where multiple ucodes and atoms are connecting by relation ucodes is built. The digraph generated based on the above method is called a ucode Relation graph (ucR graph). More specifically, the ucR graph is a massive graph structure, in which ucodes are connected by relations and pieces of information described as atoms are associated with each ucode.

### 6.1.1.2 ucode Resolution Component

The wide-area distributed database which manages ucR graphs is called a ucode Relation database. The ucR database comprehensively manages information on the relations among multiple ucodes in addition to the content such as information services associated with individual entities to which ucodes are associated.
assigned. The ucR database is basically an open database which anyone can use to reference or register information, but it can also implement an access control.

u2 architecture does not specify the internal configuration of the ucR database. However, the ucR database must provide an interface for ucode resolution explained in the following section.

When a user physically accesses an entity in the real world, u2 architecture identifies the information appropriate to the situation from the ucR database, based on the ucode assigned to the entity. This process is called ucode resolution. Moreover, the information associated with ucodes, namely, the ucR graph is registered in the ucR database. The protocol for accessing the ucR database in this manner is called ucode Resolution Protocol (ucodeRP).

![Figure 22: u2 Architecture (ucR focus)](image)

### 6.1.1.2.1 Functionalities

The main functionalities of the ucode Resolution Component are listed below:

- **Identification Resolution Functions**
  - Resolve ucode: Acquiring information about a certain ucode;
  - Define ucode-associated information: Defining relationship between a certain ucode and information
  - Update ucode-associated information: Updating information about a certain ucode;
  - Delete ucode-associated information: Deleting information about a certain ucode;

- **Contents Management Functions**
- Search data: searching ucR triple data from a certain ucode
- Register data: registering ucR triple data associate with a certain ucode
- Get data from ucode: getting ucR triple data from a certain ucode
- Get data from property parameters: getting ucR triple data by property-based query
- Update data from ucode: updating ucR triple data from a certain ucode
- Update data from property parameters: updating ucR triple data by property-based query
- Delete data: deleting ucR triple data from a certain ucode
- Delete data property: deleting ucR triple data by property-based query

### 6.1.1.2.2 Interfaces

The interfaces of the ucode Resolution Component are based on HTTP/1.1.

The ucode Resolution Component provides the Query ucode interface based on the RDF/SPARQL model and returns JSON-formatted response. The component provides the RESTful otherwise interface and returns JSON-formatted response. The following tables show major REST-based APIs concretely.

#### Identification Resolution Functions

<table>
<thead>
<tr>
<th>URL Path</th>
<th>HTTP Method</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/api/v2/rs/&lt;ucode&gt;</td>
<td>GET</td>
<td>Resolve ucode (simple API)</td>
</tr>
<tr>
<td>/api/v2/resolve/&lt;ucode&gt;</td>
<td>GET</td>
<td>Resolve ucode (general API)</td>
</tr>
<tr>
<td>/api/v2/resolve</td>
<td>POST</td>
<td>Define ucode-associated information</td>
</tr>
<tr>
<td>/api/v2/resolve/&lt;ucode&gt;</td>
<td>PUT</td>
<td>Update ucode-associated information</td>
</tr>
<tr>
<td>/api/v2/resolve/&lt;ucode&gt;</td>
<td>DELETE</td>
<td>Delete ucode-associated information</td>
</tr>
</tbody>
</table>

#### Contents Management Functions

<table>
<thead>
<tr>
<th>URL Path</th>
<th>HTTP Method</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/api/v2/datapoints</td>
<td>GET</td>
<td>Search data</td>
</tr>
<tr>
<td>/api/v2/datapoints</td>
<td>POST</td>
<td>Register data</td>
</tr>
<tr>
<td>/api/v2/datapoints/&lt;targets&gt;</td>
<td>GET</td>
<td>Get data from ucode</td>
</tr>
<tr>
<td>/api/v2/datapoints/&lt;targets&gt;/&lt;properties&gt;</td>
<td>GET</td>
<td>Get data from property parameters</td>
</tr>
<tr>
<td>/api/v2/datapoints/&lt;target&gt;</td>
<td>PUT</td>
<td>Update data from ucode</td>
</tr>
</tbody>
</table>
### 6.1.1.3 IoT Aggregator Component

Aggregate computing uses all devices, services and systems that are connected to the network to achieve desired services. Let us take an example of the optimum control of air conditioning in our living environment. Aggregate computing uses air conditioners, of course, but it also employs other objects such as windows, air vents, gas stoves, temperature sensors, humidity sensors, and any other devices that are related to the control of the air in our living space to achieve the desired optimum control of the environment beyond the confines of the original manufacturers. While we aim to achieve such computing environment, we also have to pay due attention to the governance of the usage of functions. Users should have the final say about who can control what and when instead of the vendors of the devices or service providers. Thus the users should guide the governance, and we need to provide flexible access control that can be dynamically changed according to the current context. However, note that the embedding of complex and comprehensive access control in each node will negate the general efforts to achieve the miniaturization and energy-efficiency of such devices. Yet, we can still achieve the miniaturization, low-cost, energy-saving by moving the complex advanced functions such as access control, which is auxiliary to the original inherent function of the device, to the cloud. Thus “Aggregate Computing” tries to offer a holistically optimized IoT service by mixing various devices and services as a whole.

The reference implementation for it is “IoT-Aggregator”. It has not been easy to connect devices from different manufacturers and make them collaborate with each other. IoT-Aggregator will act as lubricant and mediator among the devices, systems and services that would not have been capable of collaborating with each other easily.

![Figure 23: u2 architecture (IoT Aggregator focus)](image)

### 6.1.1.4 OPaaS.io “Omotenashi” Cloud Component

The TRON Project has been aiming to realize the vision of so-called HDFS environment in which many computers in our surroundings cooperate together to offer better services to improve the quality of everyday living. To realize such an environment, we need the mechanism to let objects to communicate with each other to collaborate. But we also need a mechanism to learn and collect the attributes of each
individual user such as the language spoken, age, many physical characteristics, preferences, etc. Such knowledge must be reflected to create an optimized environment. For this purpose, TRON Project is now building a general-purpose Personal Data Store (PDS) that uses the IoT technology. It is called "Omotenashi (hospitality) cloud" under its nickname. "Omotenashi" cloud is a project supported by industry-government-academia and now has officially started with the funding from the Ministry of Internal Affairs and Communications.

In the short time span, this "Omotenashi cloud" can be used to advance the hospitality service to the increasing number of overseas tourists who are expected to peak in 2020. Customers (consumers) store their individual characteristics such as preferred spoken language, food taboos, passport information for sales tax-exemption in the cloud. Once the data is stored, such information is easily retrieved and passed on the spot to service providers such as stores and public services by using the smartphone apps or transportation IC cards as retrieval key. OPaaS.io has been designed to collaborate with IoT-Aggregator. The collaboration will implement an HFDS environment.

The party who receives a service escrows the personal data in the PDS cloud, and lets it pass the data to the service vendors as the need arises. So it is the customer (consumer) that controls the vendors. This concept, Vendor Relationship Management (VRM) is the reverse of the conventional idea of Customer Relationship Management (CRM). OPaaS.io plays the important role of PDS in the scheme. This framework has been designed to be useful as legacy service platform in Japan after the year 2020 when it will be utilized very much for hospitality service to foreign tourists. Small regional service vendors who did not have the resources to build conventional CRM can join OPaaS.io service framework and can now provide better services suited for the new IoT age. Omotenashi Cloud will be useful for invigorating regional economy and also is expected to add to e-inclusion by accelerating the realization of Enableware.

![Figure 24: u2 architecture ("Omotenashi" Cloud focus)](image)

**6.1.1.4.1 Functionalities**

The main functionalities of "Omotenashi" Cloud Component are listed below:

- Authorization: Starting authentication, logging in by IDm, and acquiring an access token;
• User Management: Searching users, creating users, and updating users’ information;
• Access Authority Management: Searching authorized services, searching authorized user attributes, and updating authority;

6.1.1.4.2 Interfaces

The interfaces of the "Omotenashi" Cloud Component are based on HTTP/1.1. The "Omotenashi" Cloud Component provides RESTful interface and returns JSON-formatted response. The following tables show major REST-based APIs concretely.

● Authorization Functions

<table>
<thead>
<tr>
<th>URL Path</th>
<th>HTTP Method</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/api/v1/oauth2/authorize</td>
<td>GET</td>
<td>Authorization request</td>
</tr>
<tr>
<td>/api/v1/oauth2/token</td>
<td>GET</td>
<td>Token request</td>
</tr>
<tr>
<td>/api/v1/verify</td>
<td>GET</td>
<td>Token verification</td>
</tr>
</tbody>
</table>

● User Management Functions

<table>
<thead>
<tr>
<th>URL Path</th>
<th>HTTP Method</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/api/v1/user_attributes</td>
<td>GET</td>
<td>Getting user's attributes</td>
</tr>
<tr>
<td>/api/v1/user_attributes/{original_id}</td>
<td>POST</td>
<td>Registering user's attributes</td>
</tr>
<tr>
<td>/api/v1/user_attributes/{original_id}</td>
<td>GET</td>
<td>Getting user's attributes</td>
</tr>
<tr>
<td>/api/v1/user_attributes/{original_id}/history</td>
<td>GET</td>
<td>Getting history of user's attributes</td>
</tr>
<tr>
<td>/api/v1/original_id</td>
<td>POST</td>
<td>Issuing original ID</td>
</tr>
<tr>
<td>/api/v1/link_id</td>
<td>GET</td>
<td>Getting link ID</td>
</tr>
<tr>
<td>/api/v1/original_id</td>
<td>GET</td>
<td>Getting original ID</td>
</tr>
</tbody>
</table>

6.1.1.5 IoT-Engine Component

IoT-Engine is a standard development platform, standardized by TRON Forum, for Open IoT to realize "Aggregate Computing". IoT-Engine standard specifies the connector on the Micro Processor Unit (MPU) board, and the RTOS used by the MPU, and requires the function to connect to the cloud services on the Internet. The RTOS used on the IoT-Engine is µT-Kernel 2.0 which TRON Forum releases as open source. IoT-Engines collaborate with the cloud platform called IoT-Aggregator. By doing this, edge nodes can be connected via the cloud and they act as the components of the "Aggregate Computing", a vision of the
computing architecture framework in next generation. IoT-Engine project has the participation of seven semiconductor companies from six countries and regions of the world (as of December 2016): Toshiba Microelectronics, Renesas Electronics, Cypress, Imagination Technologies, Nuvoton Technology, NXP Semiconductors, and STMicroelectronics.

![Figure 25](image)

**Figure 25: IoT Engine with IoT Engine Open Platform**

### 6.1.2 Description of CPaaS.io u2-based platform architecture

Figure 26 shows the mapping of “concrete” components in u2 architecture onto the Instantiation View (Section 2.1.4.5). For historical reasons, u2 architecture was designed by a top-down manner driven by the needs of applications. Applications themselves have held many functions, which correspond to FGs within themselves. This is in contrast to today’s CPaaS.io architecture overview, so it is not easy to create concrete mapping since some functional modules in u2 architecture is still offered as part of applications.
6.1.3 Mapping between “Concrete” and “Logical” Functional Components

In this section we use the IoT ARM Functional Model and “logical” FCs as identified and elucidated in Section 4 and show how the “concrete” FCs described in Section 6.1.1 u2-based concrete architecture maps to those “logical” FCs.

In Figure 27 below, yellow round corner rectangles corresponds to the u2-based architecture functions (ucR DB, resolution, etc.) Blue and violet round corner rectangles correspond to functions offered in Omotenashi (Hospitality) Cloud. Red rounded corner rectangles correspond to modules from IoT-Aggregator. But do note that the u2 architecture itself is in flux, and many applications will offer additional modules in the future, the mapping today may be slightly changed in 6-12 months time.
Figure 27: Matching of the u2 components to the CPaaS.io Functional View
The following Table 3 summarizes the mapping between “logical” and “concrete” components as shown in Figure 27 above.

**Table 3: Mapping table for the u2-based architecture**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ucode resolution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ucR DB</td>
<td></td>
<td>x</td>
<td>x x x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IoT Aggregator Device Manager modules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IoT Aggregator p2p secure communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Omotenashi ID/Authentication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPaaS.io: Omotenashi VRM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apps</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.1.4 Deployment and Operation View

In this last architecture view of the Japanese platform we show how the u2 platform is deployed, giving hints about hardware & software used (computers, networking, databases, (web-) servers, etc...).

We also give information for selected scenarios (see sub-sections below), showing the architecture used at the data producer side (e.g. the infrastructure residing outside the platform for the Sapporo scenario).

6.1.4.1 U2-based platform deployment and operation view

![Diagram of u2 architecture](image)

**Figure 28: uID architecture in deployment and operation view**

6.1.4.2 Transportation Scenario

Public transportation is one of the most important and complex infrastructures and gives the most basic function in citizen. Especially, the public transportation network in Tokyo is the most complex in the world: e.g. hundreds of railway stations, thousands of bus stops and routes, and so on. Furthermore, these are operated many private companies: about 50 railway companies, more than 100 bus companies, and more than 1,000 taxi companies. Today, advanced public transportation operators in the world are providing information services such as transfer guidance services based on timetables, and operational information services such as delays and service disruptions. In case of mega-city with advanced public transportation network like Tokyo, providing open data is the best and only approach to providing integrated information.

Association for Open Data of Public Transportation (ODPT, odpt.org) is now making efforts to build "Public Transportation Open Data Centre" that offers information of railway, bus, airline and all other means of transportation in Tokyo (Figure 29). The demand for open data of public transportation has become very high, and the number of the association members has increased to 56 corporations (April 23, 2018) and 9 observers.

From the standpoint of technology, ODPT uses IoT and open data technologies for this activity. Static data such as timetable data and map data of stations can be easily distributed using only open data technologies, but dynamic real-time data such as location of trains and/or buses and their real-time operation status data are to be dealt with IoT technology such as sensor networks and geo-location systems.
6.1.4.3 Sapporo Scenario

International tourism is a big industry both in Japan and Europe. For example, in 2016, number of arrivals of international tourists of Japan is 24,040,000 while the one of EU is 482,480,000. So, tourism support is a good application for smart cities. Sapporo city is one of the most popular cities for tourism in Japan, providing ski resort, hot springs, and so on. We have a joint project with Sapporo City for promoting tourism using digital technologies and smart city platform.

First, we have established Sapporo Open Data Association in 2016 with 22 organizations (sapporo.odcity.org). This association conducts the activities such as the study of open data provision and its functions for Sapporo tourism and public transportation, holding events such as application contests, hackathons, ideathons, promotion of open data usage, and promotion of application usage on smartphones during feasibility study experiments. We have collected and integrated many data sets related with tourism such as sightseeing, hotel and restaurant information, public transportation information, and opened the data in the CC-BY license via open data catalog site (Figure 30).

Next, we have developed a mobile application called “Kokosil Sapporo” which gives information such as tourism spots, restaurants, hotels, public transportation and events for tourists (home.sapporo.kokosil.net/en/). It uses open data for Sapporo tourism and also location-aware technologies. With a geo-spatial position recognition system using GPS and Bluetooth Low-Energy (BLE) beacons, tourists can get information at the point where they need it in a timely manner.

Open data approach and IoT technology is a good combination for efficient information development of tourism from the view of governance of data and systems between governments and industries. In case of Sapporo city, local government provides open data for tourism and IoT infrastructure such as BLE beacons for tourism industry, and the tourism industry provides services and applications for tourists. We have already special tourism information services for temporal big events such as The Sapporo Snow Festival 2016, and 2017, and The Asian Winter Games in 2017.
6.1.4.4 Heath-Care Scenario

Emergency medical service is made possible by people of rescue crews on ambulances, at hospitals, and at dispatching headquarters, and by their teamwork. Information sharing among them is very important to provide smooth emergency medical care. Yokosuka City and YRP UNL in Japan have jointly been developing the system, "Ubiquitous Emergency Medical Support System" on that CPaaS.io platform, which collects the information from emergency hospitals and ambulances and shares it among them and headquarters.

Our IoT Emergency Medical Services is a very simple system technically, which transmits the video images of patients in ambulances to share them among the rescue crew and doctors at the hospitals in real-time via tablet devices and IP cameras placed in ambulances (Figure 31). This facilitates information sharing between the rescue crew and doctors. It provides another function which transmits the real-time location information of ambulances to emergency hospitals. This is because medical care professionals always race against time in emergency medical practice. So, when the ambulance arrives at the hospital is very important information for preparing for receiving patients smoothly. This system has been adopted for all the emergency medical care crew units since 2014 in Yokosuka City and 2017 in Miura City.
6.2 FIWARE-based platform Instantiation View

6.2.1 Description of FCs used in the FIWARE-based platform

In this short section we introduce components which are at the centre of the FIWARE IoT Chapter as well as some components from the European side coming from previous or ongoing EU projects (e.g. KAT Toolkit or FIESTA-IOT IoT Registry).

6.2.1.1 IoT Broker

The IoT Broker is specified as a lightweight and scalable middleware component that separates IoT applications from the underlying device installations. The IoT Broker implementation available through the FIWARE Catalogue is the reference implementation of this Generic Enabler. This implementation satisfies all properties described in the specification of the Generic Enabler. The main features of the IoT Broker are:

- Offering a single point of contact to the user, hiding the complexity of the multi-provider nature of the Internet of Things.
- Collecting and aggregating information about thousands of real-world objects on behalf of the user.
- Provide means to assemble lower-level device information (device-centric access) into higher-level Thing information (information-centric access).
IoT Broker is integrated with three other components: IoT Discovery (ConfMan), IoT Knowledge Server, and FIWARE (Orion) Context Broker.

6.2.1.1 Functionalities
The main functionalities of the IoT Broker are listed below. Detailed information regarding some of the standard functionalities can be accessed in the specification document of IoT Broker [30].

- Context query: Accessing context information (for data consumers);
- Context subscription: Subscribing for a change or update for a context;
- Context subscription update: Modifying an existing subscription;
- Context notification: IoT Broker sending notifications to the subscribers;
- Context data push: Context update from data providers.

Other than these basic context management functionalities, IoT Broker provides access to historical data (via its historical agent functionality), allows geo-scoped queries, federation (via federation agent), and scalability (having multiple server instances).

6.2.1.2 Interfaces
The main aspects of the IoT Broker are listed below. Detailed information regarding the interface can be found in the interface specification for NGSI-10 [32].

The IoT Broker is implemented based on the FIWARE OMA NGSI 10 interface and it provides a RESTful API via HTTP. The details about the NGSI-10 interface can be found in [33]. The purpose of the NGSI-10 interface is to exchange context information. There are three main interactions types:

- One-time queries for context information (Context query);
- Subscriptions for context information updates (Context subscription) / Notifications base on the subscriptions;
- Unsolicited updates (invoked by data providers).

The IoT Broker is a RESTful web service supporting HTTP/1.1 with JSON or XML data serialization formats. Some of the operations of NGSI 10 for exchanging context information are listed as follows: `queryContext`, `contextEntities`, `contextEntityTypes`, `subscribeContext`, `unsubscribeContext`, `contextSubscriptions`, `updateContext`, `updateContextSubscription`.

6.2.1.2 ConfMan
The NEC Configuration Management or NEC ConfMan is an implementation of the FIWARE IoT Discovery Generic Enabler. This implementation is specifically designed to interwork with the IoT Broker GE FIWARE reference implementation, serving as the registry of FIWARE NGSI context providers. The detailed information about the IoT Discovery GE can be found in [31].

ConfMan is responsible for discovering the availability of context. The availability of context must be through the registrations made by IoT Agents (i.e., data providers). By registering, data providers make their access endpoints available for the data consumers. Data providers offer information about context entities (i.e., virtual entities) and their attributes (e.g., measurement values, metadata). The role of IoT Agents can be taken by either the Data Handling GE in IoT Gateways or the Back-end Device Management GE (see Figure 36). Other IoT Backend systems may also provide context information.

Typically, context source availability information is forwarded to the FIWARE Context Broker GE (Figure 36). This allows context information (i.e., information generated by or coming from the “Things” i.e. the Physical
Entities) to be available and accessible by the applications. ConfMan is integrated with two other components: IoT Broker and IoT Knowledge Server.

### 6.2.1.2.1 Functionalities

The main functionalities of the ConfMan (IoT Discovery) are listed below. Detailed information regarding some of the standard functionalities can be accessed in the specification document of ConfMan [31]. ConfMan allows applications and services to register, discover, and subscribe to the context availability information. Context availability information can be one of the following.

- Information about IoT resources such as data sources or IoT devices and the variables they measure.
- Information about “Things” (i.e. PEs) and their attributes (i.e associated VEs attributes).
- Information about associations between entities. For instance, the attributes of a thing can be derived from attributes of other things.

ConfMan allows geographic discoveries of context availabilities based on the defined geo-scopes such as a location (point), circle or a polygon.

### 6.2.1.2.2 Interfaces

The main aspects of ConfMan are listed below. Detailed information regarding the interface can be found in the interface specification [32].

The NEC ConfMan API is fully based on FIWARE NGSI-9 which is a RESTful API via HTTP. ConfMan supports the following operations for exchanging context availability information:

- The registerContext operation is available to register information about data sources in the ConfMan’s database;
- The discoverContextAvailability operation can be used to query ConfMan for data sources.

Users can subscribe for data sources using the subscribeContextAvailability, updateContextAvailabilitySubscription and unsubscribeContextAvailability operations. Subscribed users need to expose a resource for receiving notifyContextAvailability messages from ConfMan.

### 6.2.1.3 IoT Knowledge Server

IoT Knowledge Server adds semantic information into NGSI messages and enhances these NGSI messages with semantic reasoning. IoT Knowledge Server component has an internal triple-store where NGSI (or other) ontologies are kept. It serves semantic knowledge such as entity subtypes or supertypes and provides high-level access to the semantic ontologies via query/subscription functionalities.

IoT Knowledge Server has the ability to serve high-level queries (e.g., get subtypes of a sensor). Moreover, it is a flexible component which can be extended with new features on the fly. IoT Knowledge Server is composed of two HTTP web servers and two databases along with their HTTP servers. One of the databases is HSQLDB (a SQL database) and the other is given by Apache Jena Fuseki server (a RDF 3-store).

IoT Knowledge Server is currently integrated with the ConfMan component.

### 6.2.1.3.1 Functionalities

The current functionalities of IoT Knowledge Server are listed as follows.

- Query functionalities: getSubTypes, getSuperTypes, getAttributes, getAllSubTypes, getAllSuperTypes (current list based on NGSI ontology, new features could be easily added)
• Add new queries: New queries with one or zero variables (e.g. Entity Type) can be added to a file and we can start using as a new functionality (other than the 5 above)
• Register new queries: Adding new queries by HTTP request on the fly (without restarting the server)
• Forward SPARQL queries: To provide single point of contact even for direct SPARQL queries along with the high level ones (getSubTypes)
• Subscribe functionality: Subscribing to queries and regular (fixed time) updates on change to the subscribers by the IoTKnowledgeServer.
• Caching mechanism: Caching mechanisms for fast respond (without asking SPARQL server) to queries (both for Queries and for Subscriptions)

6.2.1.3.2 Interfaces

IoT Knowledge Server offers service via its own REST API and sends response/updates with JSON format. Some example requests for IoT Knowledge Server are listed below. Other than those, IoT Knowledge Server also supports REST-based SPARQL interface via HTTP requests (similar to the Apache Jena Fuseki interface).

GET SUBTYPES: (HTTP.GET)
http://<ip_address+port>/query?request=getSubTypes&entityType=<entity_type>
Example Use: http://localhost:8080/query?request=getSubTypes&entityType=Node

GET SUPERTYPES: (HTTP.GET)
http://<ip_address+port>/query?request=getSuperTypes&entityType=<entity_type>
Example Use: http://localhost:8080/query?request=getSuperTypes&entityType=Sensor

GET ATTRIBUTES: (HTTP.GET)
http://<ip_address+port>/query?request=getAttributes&entityType=<entity_type>
Example Use: http://localhost:8080/query?request=getAttributes&entityType=Sensor

REGISTER NEW QUERY: (HTTP.POST)
http://<ip_address+port>/registerquery
Example Use: http://localhost:8080/registerquery

6.2.1.4 FIWARE Data Context Broker

FIWARE’s Data Context Broker (or simply Context Broker) is a specific middleware for brokering the communication between IoT data providers and IoT applications based on NGSI-standard as an open-source generic enabler and it can provide functionalities such as queries pertaining to historical information, federation, entity compositions (aggregating information), and so on. FIWARE Context Broker has functionality for discovery of information sources (for things/entities) and the information sources and what information they can provide are registered to this Context Broker. FIWARE Context Broker is used for discovering information sources and then accesses/forwards the information.

6.2.1.4.1 Functionalities

The functionalities of FIWARE Context Broker include standard context management operations such as:
• Update context;
• Query context;
• Subscribe context;
• Update context subscription;
• Unsubscribe context.

Similar to the IoT Broker, FIWARE Context Broker gives access to the information based on virtual entities, meaning that it provides an abstraction to the context, which makes easier application development and data analytics.

As opposed to having two stand-alone components such as IoT Broker and IoT Discovery, FIWARE Context Broker is a single component which also has context discovery functionalities (standard operations [34]) such as:

• Register context;
• Discover context availability;
• Subscribe context availability;
• Update context availability subscription;
• Unsubscribe context availability.

6.2.1.4.2 Interfaces

The communication between the Context Broker for context value data forwarding as well as registration of entity registrations are based on the NGSI protocol (NGSI-9 and NGSI-10). The API of the Context Broker is very similar to the APIs of the IoT Broker and ConfMan components. For detailed information and APIs, please refer to [34]. This document contains HTTP requests with JSON messages for both NGSI-9 and NGSI-10 interfaces.

While the FIWARE Context Broker (Orion) supports two NGSI versions (NGSI v.1 and v.2), the first version is considered as the standard to follow for both Orion and IoT Broker/ConfMan. The major difference of the FIWARE Context Broker for NGSI v.1 is the lack of “domainMetadata” in the JSON structure. Full-compatibility between the two brokers can be satisfied with minor modifications.

6.2.1.5 FogFlow Service Orchestrator

Nowadays more and more stream data are constantly generated by a huge number of IoT devices that have been widely deployed at different locations in our cities. By outsourcing more and more data processing logics from the central cloud down to the network edges, such as Base stations, IoT gateways, even some endpoint devices, we are able to largely reduce the data transmission cost and also enable real-time data analytics service with fast response time and better privacy-preserving capability. In such a cloud-edge environment, we need to decompose our IoT services into smaller tasks so that the small tasks could be dynamically migrated between cloud and edges. However, it is a big challenge to dynamically orchestrate a decomposable service and manage its tasks over cloud and edges, due to the high heterogeneity, openness, and dynamics of such system environments. The FogFlow service Orchestrator is designed and implemented to solve this problem. It provides efficient programming model and tools for service providers to easily program decomposable IoT services and help them manage the service orchestration during the runtime.

The FogFlow Service Orchestrator is implemented within FogFlow, which is going to be an open source FIWARE GE soon. The FogFlow Service Orchestrator is mainly to dynamically generate, configure, and deploy data processing flows over cloud and edges in an optimized manner. The data processing flows are generated from a service topology based on a service requirement. This service requirement defines which service topology to trigger, which type of output data is expected, and which type of scheduling algorithm is preferred.
The FogFlow Service Orchestrator consists of two major parts: a Global Orchestrator in the Cloud and a Local Orchestrator at edges. As illustrated in Figure 32, according to a service topology defined by service developers, FogFlow dynamically launches necessary data processing tasks from the provided service topology and the launched data processing tasks can process the raw NGSI context information from context producers like sensors and then generate real-time results required by context consumers like actuators. The distributed context management can be provided by other components in the virtual entity layer.

![Figure 32: FogFlow Service Orchestrator](image)

### 6.2.1.5.1 Functionalities

FogFlow Service Orchestration provides the following functionalities to service developers and service operators via its UI based task designer.

For service developers:
- Registering an implemented docker image as a data processing task;
- Editing and defining a service topology;
- Managing all registered service topologies;

For service operators:
- Triggering and generating the data processing logic from a specific service topology based on a requirement;
- Deploying the generated data processing tasks over cloud and edges;
- Optimizing the deployment plan based on the locality of producers and consumers and also device mobility;

### 6.2.1.5.2 Interfaces

Currently, the following based interfaces are provided via a UI task designer:
- Register a task with a docker image;
• Editing and submitting a service topology;

• Trigger a service topology with a geo-scope based requirement;
6.2.1.6 KAT Toolkit

The Knowledge Acquisition Toolkit (KAT) provides a set of methods for labelling data sets after initial pre-processing and data abstraction. In particular the tool provides an algorithm workflow that is implemented sequentially: pre-processing method, along with dimensionality reduction e.g. principle component analysis), data abstraction (e.g. clustering) and finally labelling/annotation of the resulting abstracted data is carried out. Input data format is time series in a tuple format or Comma Separated Values (CSV). Some newly developed machine learning techniques for data prediction will be integrated to the existing toolkit in the course of the project.

6.2.1.7 Analytics FC

The Analytics FC is provided as AGT background to CPaaS.io as an external service that enables access to analytics results in semantically rich abstraction level that is meaningful for applications. The service can be accessed via a GraphQL\textsuperscript{10} interface that provides access to a data represented in an underlying semantic data model. In CPaaS.io the service can be pre-configured to execute the full data ingestion and analytics pipeline as required for the My Experience application. For instance the service can deliver user activity and emotions of people as well as provide contextual information, e.g. location of colour stations in a Colour Run event as derived from video analytics pipeline.

6.2.1.8 Machine Learning

The Machine Learning (ML) component implements different algorithms fitting different purposes and focuses. In a first step, it is used to extract information from available Social Media (we chose Twitter for the first prototype), before publishing it to the CPaaS.io platform (this case corresponds to a RDP Class-IV case). The first version of the ML module is based on Naïve Bayes and provides a classification of tweets

\textsuperscript{10} http://graphql.org
according to the mood/sentiment they convey i.e. Positive, Negative or Neutral. This classification of event-related social media data has been used for the Snow Festival event as an additional source of event-related information and will be used during the next ColorRun event.

6.2.1.9 Authorization Component

The authorization component is able to perform capability-based access control based on the mechanism presented in [27]. It describes authorization tokens specified in JavaScript Object Notation (JSON) and optimizations to manage access control on constrained devices. The access control component it is controlled by XACML policies.

The CPaaS.io Authorization scenario will consists of the following main entities:

- Capability Client: performs requests to the Capability Manager to obtain capability tokens, which are used to perform actions over entities that are registered with the Context Manager;
- Capability Verifier: a server receiving access requests from Capability Clients. Such access requests contain a capability token, which is evaluated by the Capability Verifier in order to deny or grant the requesting action. This entity makes use of the Capability Evaluator functionality, which is intended to validate capability tokens;
- Capability Manager: a server accepting requests for capability tokens generation. Additionally, this entity acts as a client requesting authorization decisions to the Policy Decision Point;
- Policy Decision Point (PDP): It is a server that accepts XACML requests to make authorization decisions. The PDP is contacted by the Capability Manager before generating a capability token for the Capability Client;
- Policy Administration Point (PAP): It is a web application responsible for defining and managing the access control policies. It provides the functionality so users can define XACML policies in a user-friendly way. The PAP has a GUI to facilitate the generation of policies.

6.2.1.9.1 Functionalities

This component, as described will be responsible for making authorization decisions. In this sense, this component will have the following functionalities:

- Policy generation: In order to provide new access control rules, thanks to the PAP, administrators will be able to add new access control policies;
- Policy validation: Every receiving access request must be validated by the PDP;
- CT generation: If the access to a specific resource is granted, the authorization component provides a Capability Token in which such authorization is granted;
- Secure group communications: In this sense, the authorization component can encrypt the information to be provided to a group of consumers allowing only the ones with the right credentials to decrypt that information.

6.2.1.9.2 Interfaces

- According to the specific functionalities defined in the last section, each of this functionality will have the corresponding interface in order to grant authorization access to the resources to the appropriate consumers. In this sense, the following interfaces or end points should be provided by our authorization component:
  - Policy generation: A certificate must be provided in order to validate the profile and permissions of the user requesting such generation. In addition, the triplet (resource/s, subject/s, action/s) must be provided too;
• Policy validation: This interface should receive the request in a JSON format, the identity of the user and an authentication token which make possible to generate an access control decision. If the access is granted, a CT will be provided to the requester;
• Secure group communications: The authorization component must have an interface which allow requesters to provide the kind of policy (attribute-based encryption) must be applied.

6.2.1.10 Identity Management

The IdM module is responsible for managing the identities of the smart objects and users. The module is able to take into account privacy concerns to manage subjects’ credentials (from users or smart objects), in a privacy-preserving way relying on anonymous credential systems. It is able to endow subjects with mechanisms to ensure anonymity, which is done mainly issuing pseudonyms and proof of credentials to minimal personal data disclosure.

The Identity Management system is responsible for managing and storing the credentials, which are used by subjects to request information from an IoT services. The IdM allows users to obtain the proper credentials that can be use afterwards to derive proofs of identity to be presented to a Verifier (e.g., IoT Service provider) in order to prove the identity condition while disclosing the minimum amount of private information.

The component is endowed with the Idemix cryptographic engine that is responsible for low level cryptographic operations required by the anonymous credential system. The information provided by the Context Manager module is used by the IdM to handle the usage of partial identities under a specific context.

The CPaaS.io IdM has been integrated within the FIWARE IdM, i.e., the Keyrock IdM. The CPaaS.io IdM Issuer, which is in charge of generating the Idemix credentials, is able to communicate with the Keyrock to generate the credentials based on the attributes stored in the Keyrock for the user requesting the credential. To this end, the CPaaS.io IdM provides a Java library to interact with Keyrock by means of the SCIM standard [30]. This library provides methods such as add, remove or update users.

Because CPaaS.io IdM relies on Keyrock, users profile management is delegated to Keyrock, the rest of the components of the CPaaS.io framework interacts with Keyrock through the CPaaS.io IdM.

The IdM components that have been designed and implemented in the scope of CPaaS.io are enumerated below:

• CPaaS.io IdM Client: This is a client application that allows obtaining Idemix credentials from the Issuer Server. It also allows interacting with the Verifier server which can validate the partial identity derived from the credential;
• CPaaS.io-Issuer-Server: This is a web application which allows generating Idemix credentials for clients. The client must be authenticated against the Issuer using a valid certificate. The Issuer also supports the verification functionality;
• CPaaS.io-Verifier-Server: This is a web application, which is able to validate partial identities presented by the client application;
• CPaaS.io-IdM-Enabled-Capability Manager: This is a web application that allows users to obtain capability tokens using their partial identities. In other words, it allows authenticating and demonstrating their attributes by means of Idemix proofs of having a valid credential issued by the Issuer;
• CPaaS.io IdM KeyRock Client: This is a library that provides a basic API for identity management by implementing a client to interact with the FI-WARE KeyRock server.
6.2.1.10.1 Functionalities

The main functionalities provided by this component are related to registering and authentication operations. The first one, which should be done by the administrator providing for this purpose his credentials, and the second one which must be opened to all the users to allow them to log into the platforms.

6.2.1.10.2 Interfaces

In light to the previous functionalities two interfaces will be provided:

- Registering new users: The interface provided by the IdM will need the credentials of the administrator, his username and password, and a parameter indicating the new user and their corresponding attributes;
- Authentication: This interface will receive the username and password of the users in order to be validated.

6.2.11 Authentication

The authentication module verifies if a subject is truly who or what it claims to be. Authentication services are provided by the CPaaS.io IdM system. Besides the conventional password-based and key-based mechanisms, the CPaaS.io IdM also supports authentication based on anonymous credentials.

- Login password-based authentication: The CPaaS.io IdM relies on the Keyrock IdM to perform this kind of authentication. To this end, the CPaaS.io IdM library provides a method that given the username and password authenticates the user against the keyrock and generates an authentication token that can be used afterwards to perform other actions against the Keyrock IdM. The CPaaS.io Keyrock IdM client library provides two main methods for dealing with this kind of authentication. The first method is for authenticating a user registered in the Keyrock given the password and username and its domain. The second method, provided by the Keyrock IdM client, is for validating a given token ID;
- PKI-based authentication: CPaaS.io framework allows users to authenticate using their digital certificates. It requires a Public Key Infrastructure (PKI) where the user’s certificates are signed by the Certificate Authority (CA) and validated on the server side. The IdM server performs client authentication based on asymmetric-key cryptographic. This is done actually by the IdM web server that authenticates the client that must possess the private key associated with its public X509 certificate (the server trusts the CA);
- Anonymous credentials based authentication: This kind of authentication is done by the CPaaS.io privacy-preserving IdM, which offers a claim-based solution for authentication. The CPaaS.io IdM relies on an anonymous credential system, namely Idemix from IBM [28], to ensure a privacy and minimal disclosure of personal information.

6.2.1.11.1 Functionalities

Since the authentication operations are based on the identity attributes defined by the IdM all the functionalities have been defined in the previous section.

6.2.1.11.2 Interfaces

Again, the interfaces provided by IdM have been provided in the previous section.
6.2.2 Description of CPaaS.io FIWARE-based platform architecture

Figure 36 shows the mapping of “concrete” FCs identified in Section 6.2.1 onto the Instantiation View (Section 2.1.4.5). This figure has components and relationships from FIWARE IoT Services Enablement Architecture [31] and it also includes the IoT Knowledge Server, SPARQL agent, NGSI to RDF mapper, LoRaWAN bridge, and FogFlow Service Orchestration components. This view mainly consists of six layers and two pillars: IoT Resource Layer, IoT Data and Ingestion Layer, Virtual Entity Layer, Semantic Data and Integration Layer, Knowledge Layer, Smart City Service Layer, Security Pillar, and Platform and Management Pillar.

The IoT Resource Layer is made of all on-field IoT infrastructure elements needed to connect physical devices to FIWARE Apps or SPARQL Apps. Typically, it comprises: IoT end-nodes, IoT gateways and IoT networks (connectivity).

The IoT Data and Ingestion Layer comprises the set of functions, logical resources and services hosted in a Cloud data centre as well as on the Edge. Up north, it is connected to the data chapter Context Broker, so IoT resources are translated into NGSI Context Entities. South-wise the Ingestion Layer is connected to the IoT devices, that are all the physical IoT infrastructure. Platform operation pillar has the FogFlow Service Orchestrator, which enables processing optimization between the Cloud and Edge resources.

FIWARE’s IoT architecture follows the FIWARE OMA NGSI context management information model, which allows performing context operations such as query, update, subscribe, and notify. On the Virtual Entity Layer, the NGSI9 API is mainly used for discovering VEs while NGSI10 is mainly for accessing context information.
Considering the architecture from a bottom-up (south to north) perspective, in the south IoT devices are connected to the IoT Broker and IoT Discovery components via the IoT NGSI gateway. The IoT NGSI Gateway can be implemented with an adapter so that the devices with various APIs (shown as Device API) can be registered via NGSI9. Moreover, the context information can be sent to the IoT Broker via NGSI10. The IoT Broker then forwards the context to the data context broker and to the application developers. In FIWARE, each FC in this architecture is realized by Generic Enablers (GEs). Detailed information about specific GEs can be found in the FIWARE catalogue [33].

Currently, the FIWARE-based CPaaS.io platform provides the possibility for devices to push SPARQL or NGSI data through IoT or SPARQL Agents. The agents are responsible for converting device data (device API) to NGSI or SPARQL data. Moreover, the CPaaS.io platform provides app developers the flexibility to use either the NGSI or the SPARQL interface to access data. The SPARQL interface is provided by the IoT Knowledge Server on the Semantic Data Layer, whereas the NGSI interface is provided by the IoT Broker (AeronBroker) and the Data Context Broker (Orion Context Broker). In the next phase of the project, the conversion from NGSI to SPARQL may allow pushing NGSI data through IoT Agents and accessing this data by SPARQL app Developers.

We listed the components in Section 6.2.1 and showed them in the above figure mapped onto multiple layers and pillars. These components, namely AeronBroker, ConfMan, IoT Knowledge Server, Orion Context Broker, and FogFlow Service Orchestrator are supported by FIWARE. Now, in the next section we match these components to the CPaaS.io logical FCs based on where the already implemented FIWARE components possibly fit in this Functional View.

The Security and Privacy Pillar provides the necessary authentication and authorization mechanisms to the CPaaS.io platform. The listed components in the Security Pillar are KeyRock, XACML, Cap Manager, PEP Proxy, and CP-ABE. The pillar is responsible for providing secure access control to the data providers and consumers. Currently, these components are already integrated with the Orion Context Broker and they are being integrated with IoT Broker. FIWARE app developers (data consumer) as well as the providers (IoT Agents, IoT Broker) can access content through the PEP Proxy component.

### 6.2.3 Mapping between “Concrete” and “Logical” Functional Components

In this section we use the IoT ARM Functional Model and “logical” FCs as identified and elucidated in Section 4 and show how the “concrete” FCs described in Section 6.2.1 map to those “logical” FCs.
In the Figure 37 above, the FIWARE components are matched with the FCs of CPaaS.io. Note that the FIWARE components are placed in arbitrary places for visual purpose, while their places actually correspond to the places of the marked FCs in CPaaS.io functional view.
While CPaaS.io architecture is clearly more advanced, having each FC has various requirements, some requirements of the FCs are not completely met by the FIWARE components. For instance, considering the requirements that are listed for use-cases, Semantic VE Data Repository may require more advanced and novel features compared to the existing functionalities of the IoT Knowledge Server component. For this reason, some FIWARE components may be either extended or improved or new components should be developed during the CPaaS.io project when necessary. At this stage, the decisions have not been finalized for the implementation necessities of the proposed CPaaS.io architecture. On the other hand, some components can already meet the listed requirements of some of the FCs without further implementation effort. For instance, VE Resolution is provided by IoT Discovery and therefore IoT discovery component could be integrated without much effort.

We provide an interface view for the FIWARE components below. The Figure 38 below includes IoT Broker instances, IoT Knowledge Server, Context Broker, as well as IoT Discovery GEs including also their interactions (APIs).

![Figure 38: Interactions of the FIWARE FCs](image)

In the above Figure 38, the historical data is stored in CouchDB by IoT Broker. IoT Broker also forwards the context to the context broker or applications. Moreover, subscriptions are stored in and accessed from a shared SQL database by the distributed IoT Broker instances. Registrations are saved in the Registration Repository (ConfMan) via NGSI9. Lastly, IoT Agent is responsible for providing context of VEs in NGSI9/10 information model and therefore it can be considered as an NGSI device.
The following Table 4 summarizes the mapping between “logical” and “concrete” components as shown in Figure 37 above.
Table 4: Mapping table for the FIWARE-based architecture

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>KAT Toolkit (3rd party)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context Broker (FIWARE)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IoT Broker (FIWARE)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IoT Discovery (FIWARE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IoT Knowledge Server (FIWARE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FogFlow Service Orchestrator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SocIoTal FIWARE based Authentication</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SocIoTal FIWARE based Authorization</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SocIoTal FIWARE based Identity Management</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ML component</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

X: covers the FC currently
NR: covers the FC in Next Release of the concrete FC
PA: covers PArtly the FC
6.2.4 Deployment and Operation View

In this last architecture view of the European platform we show how the FIWARE-based platform is deployed, giving hints about hardware & software used (computers, networking, databases, (web-) servers, etc.).

We also give information for selected scenarios (see sub-sections below) showing the architecture used at the data producer side (e.g. the TTN infrastructure residing outside the platform for the Waterproof Amsterdam scenario).

6.2.4.1 FIWARE-based platform deployment and operation view

Figure 39: FIWARE-based platform deployment and operation view

Figure 39 shows a static view of the run-time configuration of processing nodes and components that run on those nodes in the FIWARE-based platform. On this deployment view, there is a central node (Application server 1) hosting IoT Broker (AeronBroker), IoT Discovery (ConfMan), IoT Knowledge Server and the NGSI to RDF mapping service. Additionally, CouchDB and PostgreSQL databases store data related to sensor values from IoT devices and NGSI-9 registrations, respectively. “Application server 2” hosts the Context Broker and security components that communicates to IoT Broker server. Another node hosts the LoRaWAN component and a proxy to communicate to “Application server 1”. Finally, the Deployment tool
is integrated to their storage components and the Semantic Data Repository also communicates to the central node. FogFlow integration is still in progress and its components are placed in a separate node.
6.2.4.2 Waterproof Amsterdam deployment and operation view

6.2.4.2.1 Architecture

![Diagram of Waterproof architecture and usage]

Figure 40: Waterproof architecture and usage
6.2.4.2.1.2 Deployment

The waterproof application is deployed via FogFlow and a service topology. The topology contains all the necessary information for the services to start and be configured for a designated city. CPaaS.io is in charge of maintaining the dashboard up.

6.2.4.3 MyExperience deployment and operation view

6.2.4.3.1.1 Architecture

Figure 42 depicts the My Events architecture using the European CPaaS.io platform instance. In later deliverables we will describe the interaction with the Japanese platform.

The MyEvents app can use three interfaces:

- GraphQL to semantically access data stored in AGT’s IoTA platform,
- NGSI 10 to access data stored in the IoTBroker (not shown in Figure 42), or
- SPARQL to access data in the Semantic Data Repository.
All of the raw data and analytics collected via the ColorRun is available in IoTA and can be accessed via its GraphQL interface. In addition the data is replicated in the IoTBroker to make it accessible to all CPaaS.io applications. In a previous deliverable we described that the MyEvents application can be configured to use the NGSI 10 interface or the native IoTA interface. In this updated version, we do not show the direct access to the IoT Broker anymore, as all data access is done via the semantic interfaces. For this purpose the MyEvents app uses the NGSI to RDF mapper described in D6.3. For instance Twitter data is inserted in the IoTBroker and made accessible into the Semantic Data repository. Then the MyEvents app can directly access the Twitter data via a SPARQL query to the Semantic Data repository in the Semantic Integration Layer.

6.2.4.3.1.2 Use Cases Diagrams

Figure 43 shows the use case diagram for accessing data. The end users can log in and out of the application. After log in the experience log, a dashboard showing cross event information including event highlights, a button for managing data related to the events (Manage Event Data use case), and the possibility to see event specific highlights by selecting an event (View Event Use Case). The View Event Use Case displays a map in which the user can retrieve additional information about detected highlights (View Running Highlights Use Case).

As indicated in the use case diagram each of the app use cases depends on the retrieve data use case of the CPaaS.io platform that allows to retrieve data from IoTA, the Semantic Data Repository or the Deployment Knowledge Base.
6.2.4.3.1.3 Deployment Information

The My Event app is deployed on a virtual machine on AGT’s data center. The virtual machine also includes MongoDB that is used for managing users data.

The IoTa platform is for commercial purposes usually deployed in the cloud such as AWS. But for testing and costs reasons we deployed IoTa on a local machine. The IoTa platform contains all the master data and provides selected information to the IoTBroker of the European CPaaS.io platform instance.

The NGSI mapping service is deployed on a virtual machine on AGT’s data centre and proxied by an NGINX instance deployed on an AWS EC2 instance that inserts the data in the CPaaS.io Semantic Data Repository. The NGSI mapping service is currently configured to use an Allegrograph Triple Store deployed on a virtual machine at AGT’s data centre.

The deployment knowledge base is a separate Allegrograph Triple Store instance that is also deployed on a separate VM in AGT’s data centre.

6.2.5 Concrete System Use-cases

In this section we revisit some of the System Use-cases that have been already described in Section 4.3.1 in order to illustrate typical operation usage patterns in the form of interactions taking place between “logical” function components and actors that stand outside the CPaaS.io platform. We focus here on describing how such interactions between logical FCs and external actors derives into interactions between those actors and the concrete FCs as introduced in this chapter. While section 4.3.1 refers to different classes of Raw Data Producers we will in this section refer more explicitly to the project scenarios (e.g. Waterproof Amsterdam or ColorRun) in order to 1/ give a concrete grounding of the architecture work within the project scenarios and actual platform development and 2/ give more support to the implementation and integration phase running in parallel to this architecture paperwork activity.
6.2.5.1 Configuration phases

This section revisits Section 4.3.1.2 and describes few concrete system UCs dealing with system configuration where the system consists of the CPaaS.io platform and platform clients (actors defined in the scenarios). Most of those UCs are relevant to all scenarios. This ‘catalogue’ of UCs will be extended within the next iterations of the CPaaS.io architecture document.

6.2.5.1.1 Register a user/actor to the CPaaS.io platform

As already expressed before in this document the architecture of our platform contains a specific element for managing the properties and attributes of its users. This component receives the name of Identity Management.

So, in order to register a new user/actor into the CpaaS.io platform, first we need to interact with this component in order to define the different parameters of the user’s profile. Specifically, we have defined the following attributes for it:

- NickName nickName: the casual way to address an entity user in real life
- Name name: the components of the user’s name
- ArrayList<Address> addresses: the physical mailing addresses of an entity user
- ArrayList<Email> emails: the e-mail addresses of an entity user
- Organization organization: it identifies the name of an organization
- Department department: it identifies the name of a department
- ArrayList<X509Certificate> x509Certificates: the X.509 certificates associated with the entity
- String password: the entity’s clear text password
- boolean active: a value indicating the entity’s administrative status

To do so, the Identity Management provides and endpoint to receive such registration operations using a RESTful API and the POST method. The URL that will handle such registrations is similar to the following one:


The body of this request should contain a JSON text that must meet the specification of SCIM core (RFC 7643 SCIM Core Schema). Since this operation must be done by the administrators of the platform, such operation requires also the addition of a certificate in order to securely validates the origin of such request.

6.2.5.1.2 Create and assign security credential

Although after applying the previous operation the user has been registered in the CPaaS.IO platform, he does not have any access rights over the information stored in it. So, we need to establish the security policies for such information. This is done thanks to the XACML component which comprises the following entities: Policy Administration Point (PAP), Policy Decision Point (PDP) and Policy Enforcement Point (PEP).

Thanks to an administration web page, the administrator can establish the different security policies that will be applied to the user and the resources stored in the platform.
After having logged in, the administrator will be able to generate new XACML policies. These are identified by a triplet (resource, subject, action) which means that for a specific resource or resources, the selected subjects or users will be able to perform the specified actions over them. Applied to our platform, the resources will be the different stored entities with their corresponding attributes, the subjects are the different users of the platform, and finally, the actions could be register, updating or requesting the information.

Thanks to the definition of these policies, the Capability Manager, after receiving an authorization request will forward it to the PDP which will carry out the validation operation over it. If the validation process is successful, it will answer the Capability Manager that will generate the Capability Token that will allow the requester of such information to access to it.

### 6.2.5.1.3 A CPaaS.io user / RDP creates a privacy/security policy

Another kind of authorization that the platform can perform is the authorization over the broadcast information. In this sense, a specific authorization policy can be applied over the transmitted information in such a way that only the receivers with the appropriate credentials will be able to reveal it.

Such authorization or privacy policy is based on Attribute Based Encryption (ABE) so, the users and RDPs which are intended to apply this authorization policy must specify the combination of attributes that will be able to decrypt such information, i.e. organization = OdinS and department = R&D. Such combination implies that only the users which are employed by OdinS and are working on R&D will be able to decrypt the information. If there are any users from OdinS which working in other areas, they will not be able to decrypt it.

The following example describes how the information must be registered in our CPaaS.IO platform.
According to it, the information to be stored in our platform is an entity whose id is Test:1 of the type Test. It only contains one attribute whose name is cipheredAttribute of the type ciphertext. The security information is included in the metadata field where we specify that this attribute should be encrypted using a CP-ABE policy using the combination of aforementioned attributes.

This information is included in the NGSI request to register information into our platform. It will be received by our PEP_Proxy which will make the validation tests first, and afterwards it will proceed to encrypt the information using the appropriate CP-ABE key.

On the other hand, the registered users will have received a specific CP-ABE key which is based on the attributes of their identities stored in the Identity Management component. This way, after receiving this information, only the one with the appropriate attributes will be able to decrypt it.

### 6.2.5.2 Data production

#### 6.2.5.2.1 An RDP Class-I pushes semantic data to the CPaaS.io platform

Semantic data can be pushed to CPaaS.io in two ways:

- Via NGSI-10 to the Context or IoT Broker
- Via SPARQL to the Semantic Data Repository

When using the NGSI-10 interface the RDP must follow certain conventions in order to enable semantic interoperability. For instance entity types must link to RDFS classes (rdfs:class or owl:class) and entity id must use the same identifiers as RDF instances. Similar conventions apply for attribute types and names. Details will be described in the upcoming deliverable D6.3 [9].

Alternatively the data can directly pushed to the Semantic Data repository using SPARQL INSERT or UPDATE requests.
This third version of the architecture is a second consolidation of the CPaaS.io logical architecture (architectural Views and Perspectives with requirement tracking) and an update of the two instantiation views that shows how this logical architecture is being instantiated using two different technologies, namely u2 for the Japanese side and FIWARE for the European counterpart. The instantiation View features a new layered model that complies with comments received during the 1st Project Review.

This third version of the CPaaS.io architecture provides much more content in all areas of the IoT-ARM compliant CPaaS.io architecture and it does cover aspects like task distribution, federation and FIWARE-based / u2-based interoperability, which were not tackled yet in the previous version. It also provides Deployment and Operation Views for three u2-based platform and scenarios and two FIWARE-based platform and scenarios.

The final version of the CPaaS.io architecture (D3.7) will be provided in M30 and will cover all remaining issues not yet covered in this version or not fully completed, like for instance:

- Perspectives improvement;
- Ingestion of Government or Public Data and associated FCs;
- Analytics (going beyond Sentiment Analysis);
- Deployment and Operation Views (few are still to be described);
- Full interoperation between both platforms (incl. Semantic interoperability).
7 References


[10] ETSI TR 103 167 Machine-to-Machine Communications (M2M); Threat analysis and countermeasures to M2M service layer

[11] Haller, S et al: “A Domain Model for the Internet of Things”, in iTHINGS'2013 proceedings, Beijing, China (see also IEEE eXplore)


[22] SMARTIE Project Deliverable D2.2 “Requirements”

[23] SPARQL Protocol And RDF Query Language http://www.w3.org/TR/rdf-sparql-query/


[31] www.viewpoints-and-perspectives.info/home/perspectives/ Last accessed 26/01/2017


[42] https://www.w3.org/TR/sparql11-entailment/

[44] http://www.etsi.org/deliver/etsi_gs/CIM/001_099/004/01.01.01_60/gs_CIM004v010101p.pdf
Appendix: Requirements (Volere Template)

The following pages show a snapshot of the updated unified requirements as of June 2018.

The complete Excel document is available at
<table>
<thead>
<tr>
<th>UNI ID</th>
<th>Requirement</th>
<th>Priority</th>
<th>Category</th>
<th>Description</th>
<th>Fit Criteria</th>
<th>Use Case</th>
<th>View</th>
<th>Perspectives</th>
<th>Functional</th>
<th>Functional/Conceptual</th>
<th>Coverage</th>
<th>Traced in v.3</th>
<th>Relation to Proj. Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFREQ.1</td>
<td>NFREQ.1</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The p/f SHOULD support different service level agreements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.2</td>
<td>NFREQ.2</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The p/f MUST offer mechanisms for balancing processing load (services, events, data, ...)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>NFREQ.3</td>
<td>NFREQ.3</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>Quality of service should be monitored in real time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.4</td>
<td>NFREQ.4</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The p/f SHOULD provide a trusted and secured communication and information management (incl. storage)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.5</td>
<td>NFREQ.5</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The user (including Physical persons) SHOULD be able to access data and services anonymously (under certain restrictions)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.6</td>
<td>NFREQ.6</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>NFREQ.7</td>
<td>NFREQ.7</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The p/f SHOULD provide mechanisms for protecting the data transmission</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.8</td>
<td>NFREQ.8</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The p/f MUST be able to detect some of the threats at runtime</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.9</td>
<td>NFREQ.9</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The p/f SHOULD be able to provide some self-healing mechanisms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.10</td>
<td>NFREQ.10</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The p/f SHOULD provide a trusted and secured communication and information management (incl. storage)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.11</td>
<td>NFREQ.11</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>NFREQ.12</td>
<td>NFREQ.12</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The p/f SHOULD be able to detect some of the threats at runtime</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>NFREQ.13</td>
<td>NFREQ.13</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The p/f SHOULD provide mechanisms for protecting the data transmission</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.14</td>
<td>NFREQ.14</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The p/f SHOULD be able to detect some of the threats at runtime</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.15</td>
<td>NFREQ.15</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The p/f SHOULD be able to detect some of the threats at runtime</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.16</td>
<td>NFREQ.16</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The p/f SHOULD provide mechanisms for protecting the data transmission</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.17</td>
<td>NFREQ.17</td>
<td>M</td>
<td>Platform, Infrastructure</td>
<td>The p/f SHOULD provide mechanisms for protecting the data transmission</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.18</td>
<td>NFREQ.18</td>
<td>M</td>
<td>City Data, City Services</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.19</td>
<td>NFREQ.19</td>
<td>M</td>
<td>City Data, City Services</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.20</td>
<td>NFREQ.20</td>
<td>M</td>
<td>City Data, City Services</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.21</td>
<td>NFREQ.21</td>
<td>M</td>
<td>City Data, City Services</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.22</td>
<td>NFREQ.22</td>
<td>M</td>
<td>City Data, City Services</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.23</td>
<td>NFREQ.23</td>
<td>M</td>
<td>City Data, City Services</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.24</td>
<td>NFREQ.24</td>
<td>M</td>
<td>City Data, City Services</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.25</td>
<td>NFREQ.25</td>
<td>M</td>
<td>City Data, City Services</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.26</td>
<td>NFREQ.26</td>
<td>M</td>
<td>City Data, City Services</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.27</td>
<td>NFREQ.27</td>
<td>M</td>
<td>City Data, City Services</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.28</td>
<td>NFREQ.28</td>
<td>M</td>
<td>City Data, City Services</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.29</td>
<td>NFREQ.29</td>
<td>M</td>
<td>City Data, City Services</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>NFREQ.30</td>
<td>NFREQ.30</td>
<td>M</td>
<td>City Data, City Services</td>
<td>The users SHOULD be able to control the personal data they provide and how their personal data is used</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>
### Deliverable D3.5 CPaaS.io System Architecture (v3)

**Vendors Template**

<table>
<thead>
<tr>
<th>Req No</th>
<th>Req Type</th>
<th>Priority</th>
<th>Category</th>
<th>Description</th>
<th>Relevante</th>
<th>Owner</th>
<th>Filter</th>
<th>Dependencies</th>
<th>Conflicts</th>
<th>View</th>
<th>Prospective</th>
<th>Functional Group</th>
<th>Functional/Conformance</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFREQ.13</td>
<td>NFREQ.13</td>
<td>MEDIUM</td>
<td>Platform</td>
<td>The platform MUST provide a level of security that is sufficient to manage the intervention of the platform users in management duties or mimic interaction of the platform user with the platform itself.</td>
<td>NFREQ.13</td>
<td>NFREQ.13</td>
<td>MEDIUM</td>
<td>Platform</td>
<td>NFREQ.13</td>
<td>NFREQ.13</td>
<td>Functional</td>
<td>Virtual Entity</td>
<td>Virtual Entity</td>
<td>VE Registry</td>
</tr>
</tbody>
</table>

**Trasparency regarding the architectural reference model**

- **Yes** (interactions management enabled and p/f are through the dashboard)
- **No** (only English and Japanese covered)
- **Partly** Obj.1

---

**H2020 EUJ-02-2016 CPaaS.io Page 108 of 110**
<table>
<thead>
<tr>
<th>UNI ID</th>
<th>RQ ID</th>
<th>Priority</th>
<th>Category</th>
<th>Description</th>
<th>Rationale</th>
<th>Owner</th>
<th>Fit Criteria</th>
<th>Confidence</th>
<th>View</th>
<th>Perspective</th>
<th>Functionality Group</th>
<th>Functional-Completeness</th>
<th>Traceability regarding the architectural reference model</th>
<th>Relevance to Req. Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ02.21</td>
<td>RQ02.21</td>
<td>MEDU M</td>
<td>Security, Privacy &amp; Trust</td>
<td>Personal data in servers should be encrypted if it can be done so securely at a reasonable cost.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.13, Freq.19</td>
<td>Functional View</td>
<td>Security, Privacy</td>
<td>Security</td>
<td>ERM to be checked between EU and Japan</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.22</td>
<td>RQ02.22</td>
<td>MEDU M</td>
<td>Security, Privacy &amp; Trust</td>
<td>Secure storage of data should be ensured.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.13, Freq.19</td>
<td>Functional View</td>
<td>Security, Privacy</td>
<td>Security</td>
<td>IoT Service, Virtual Entity, Security</td>
<td>Semantic Web Data Repository, VE Registry, Semantic Data Repository, ERM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.23</td>
<td>RQ02.23</td>
<td>HIGH</td>
<td>Security, Privacy &amp; Trust</td>
<td>Data owners should be able to set access-control rights policies (set up by data owners) to their data stored on servers.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.2, Freq.20</td>
<td>Functional View</td>
<td>Security, Privacy, Security</td>
<td>Security, Management</td>
<td>AuthN, Web Front End</td>
<td>unclear</td>
<td>in the current implementation policies are not encrypt. To be discu further</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.24</td>
<td>RQ02.24</td>
<td>MEDU M</td>
<td>Security, Privacy &amp; Trust</td>
<td>Access-control rights/policies (set up by data owners) should be able to be deleted or modified.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.21</td>
<td>Functional View</td>
<td>Security, Privacy</td>
<td>Security</td>
<td>AuthN, ERM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.25</td>
<td>RQ02.25</td>
<td>HIGH</td>
<td>Security, Privacy &amp; Trust</td>
<td>Committed data must remain confidential if needed.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.23</td>
<td>Functional View</td>
<td>Security, Privacy</td>
<td>Security, Communication</td>
<td>ERM, Communication Channel</td>
<td>yes (encryption)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.26</td>
<td>RQ02.26</td>
<td>HIGH</td>
<td>Security, Privacy &amp; Trust</td>
<td>Service providers must be able to set access-control rights/policies (set up by service owners) to their services.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.24</td>
<td>Functional View</td>
<td>Security, Privacy</td>
<td>IoT Service, Virtual Entity, AuthN, Auth2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.27</td>
<td>RQ02.27</td>
<td>HIGH</td>
<td>Security, Privacy &amp; Trust</td>
<td>Data owners should be able to set access-control policies (set up by data owners) to their data stored on servers.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.25</td>
<td>Functional View</td>
<td>Security, Privacy</td>
<td>IoT Service, Virtual Entity, AuthN, Auth2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.28</td>
<td>RQ02.28</td>
<td>HIGH</td>
<td>Privacy</td>
<td>Users and devices must be identified.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.26</td>
<td>Functional View</td>
<td>Security, Privacy, Security</td>
<td>Security</td>
<td>AuthN, Identity Management or associated with an ID, or authenticated? (to be clarified)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.29</td>
<td>RQ02.29</td>
<td>MEDU M</td>
<td>Security, Privacy &amp; Trust</td>
<td>Chas.IO Must provide an authentication policy to data, object and service for the different users or devices which in turn will have to be authenticated and authorized before accessing them.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.27</td>
<td>Functional View</td>
<td>Security, Privacy</td>
<td>IoT Service, Virtual Entity, AuthN, Auth2, Web Front End</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.30</td>
<td>RQ02.30</td>
<td>HIGH</td>
<td>Identity Management</td>
<td>User (CPaaS client) or device as part of the platform deployment MUST be registered before using any services provided by the platform.</td>
<td>Needed for authentication and authorization</td>
<td>N.Freq.28</td>
<td>Functional View</td>
<td>Security, Privacy, Security</td>
<td>Identity Management, Web Front End</td>
<td>S1S-13</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.31</td>
<td>RQ02.31</td>
<td>HIGH</td>
<td>Management, Security</td>
<td>Users and devices MUST be identified.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.29</td>
<td>Functional View</td>
<td>Security, Privacy, Security</td>
<td>Security</td>
<td>AuthN, Identity Management or associated with an ID, or authenticated? (to be clarified)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.32</td>
<td>RQ02.32</td>
<td>HIGH</td>
<td>Privacy</td>
<td>User’s data MUST be processed only with the user’s consent.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.30</td>
<td>Functional View</td>
<td>Security, Privacy, Security</td>
<td>Security</td>
<td>AuthN, Identity Management or associated with an ID, or authenticated? (to be clarified)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.33</td>
<td>RQ02.33</td>
<td>HIGH</td>
<td>Security, Privacy &amp; Trust</td>
<td>Chas.IO Must support context-aware access policies.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.31</td>
<td>Functional View</td>
<td>Security, Privacy, Security</td>
<td>Security</td>
<td>AuthN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.34</td>
<td>RQ02.34</td>
<td>HIGH</td>
<td>Security, Privacy &amp; Trust</td>
<td>Services MUST maintain a log of the operations done by users. This log must not be modified by attackers.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.32</td>
<td>Functional View</td>
<td>Security, Security</td>
<td>Security</td>
<td>Logging</td>
<td>dropped for log part, non-reputation already covered</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.35</td>
<td>RQ02.35</td>
<td>HIGH</td>
<td>Discovery, Tagging</td>
<td>Time stamps MUST be supported.</td>
<td>Basic time data needed for studying bits like data query, time series analysis, casuistry detection.</td>
<td>N.Freq.33, Freq.34</td>
<td>Functional View</td>
<td>Security, Privacy</td>
<td>Security, Organization</td>
<td>Task Deployment, Optimisation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.36</td>
<td>RQ02.36</td>
<td>HIGH</td>
<td>City Data, Platform Infrastructure</td>
<td>The IoT system MUST ensure that processing of information with respect to cost function, e.g. communication, computation, energy, etc.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.35</td>
<td>Functional View</td>
<td>Security, Privacy, Security, Organization, IoT process, Mgmt</td>
<td>Task Deployment, Optimisation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.37</td>
<td>RQ02.37</td>
<td>MEDU M</td>
<td>Load Balancing</td>
<td>The IoT system SHOULD ensure that processing of information with respect to cost function, e.g. communication, computation, energy, etc.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.38</td>
<td>Functional View</td>
<td>Security, Privacy, Security</td>
<td>Security, Organization, IoT process, Mgmt</td>
<td>Task Deployment, Optimisation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.38</td>
<td>RQ02.38</td>
<td>MEDU M</td>
<td>Load Balancing</td>
<td>The CPaaS.io platform MUST be able to provide mechanisms for migrating tasks to nodes participating in the CPaaS.io systems according to its characteristics and availability.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.39</td>
<td>Functional View</td>
<td>Security, Privacy, Security</td>
<td>Security, Organization, IoT process, Mgmt</td>
<td>Task Deployment, Optimisation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.39</td>
<td>RQ02.39</td>
<td>HIGH</td>
<td>Platform, Management</td>
<td>This basic info is needed by the decision making process that decides how and where tasks must be deployed.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.40</td>
<td>Functional View</td>
<td>Security, Privacy, Security</td>
<td>Security, Organization, IoT process, Mgmt</td>
<td>Task Deployment, Optimisation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ02.40</td>
<td>RQ02.40</td>
<td>HIGH</td>
<td>Platform, Management</td>
<td>Chas.Io platform MUST be able to perform analytics, and event detection/prediction using &quot;live&quot; and historical data.</td>
<td>To guarantee data integrity and make sure the data can’t be stolen or replicated.</td>
<td>N.Freq.41</td>
<td>Functional View</td>
<td>IoT Service, Virtual Entity</td>
<td>IoT Service, Analytics, Rule-based Reasoner, CEP Engine</td>
<td>The CEP part is dropped because of a lack of motivation from the scenario prediction is dropped for the same reason.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The CPaaS.io platform MUST provide a live data stream. It can be coupled with ML for inferring production.

Complex Event Processing techniques that are able to infer high level events out of an incoming flow of data, based on various criteria (condition for that event to be triggered).

The ontology must support the concept of event. WA-9.

Networks

Data transmission is modulated with LoRa, operated by The Things Network. Network server. The application needs to be integrated with TTN API.

The platform MUST be able to host and serve IoT Services and VE Services locally. Some IoT Resource owner do not face any infrastructure for serving IoT Services exposing their IoT Resources.

For legal reasons some data may not be stored by the platform directly.

For scenario that include actuation it is highly crucial to provide this functionality.