Grant Agreement No.: 723174
Call: H2020-ICT-2016-2017
Topic: ICT-38-2016 - MEXICO: Collaboration on ICT
Type of action: RIA

D3.2: SmartSDK IoT and Data Management Enablers

Revision: v.1.0

Work package	WP 3
Task	Task 3.1 & Task 3.2
Due date	31/05/2017
Submission date	31/05/2017
Deliverable lead	INFOTEC
Version	1.0
Authors	Hugo Estrada (INFOTEC), Blanca Vázquez (INFOTEC), Tomas Aliaga (MARTEL), Germán Molina (HOPU), Miguel González (ITESM), Alicia Martínez (CENIDET), Antonio Macías (CICESE).
Reviewers	Federico Facca (MARTEL), Netzahualcóyotl Hernández (CICESE)

Abstract	This deliverable provides an overview of the current status of the components developed for Internet of Things and Data Management.
Keywords	FIWARE, Data Management, IoT Service Enablement

Document Revision History

Version	Date	Description of change	List of contributor(s)
V1.0	31/05/2017	Final version ready for submission	Federico M. Facca (Martel), Blanca Vázquez (INFOTEC)
V0.4	19/05/2017	Add clarification on releases	Federico M. Facca (Martel), Netzahualcóyotl Hernández (CICESE), Hugo Estrada (INFOTEC)
V0.3	15/05/217	Integration of contributions from partners plus minor adjustments.	Francisco Monsanto (UBIWHERE), Germán Molina (HOPU), Miguel González (ITESM), Alicia Martínez (CENIDET), Hugo Estrada (INFOTEC), Blanca Vázquez (INFOTEC), Antonio Macías (CICESE).
V0.2	26/04/2017	Version ready for comments and integration by partners.	Federico M. Facca (Martel)
V0.1	15/04/2017	Table of Contents and initial version.	Hugo Estrada (INFOTEC)

Project co-funded by the European Commission in the H2020 Programme		
Nature of the deliverable:	R	
Dissemination Level		
PU	Public, fully open, e.g. web	✓
CI	Classified, information as referred to in Commission Decision 2001/844/EC	
CO	Confidential to SmartSDK project and Commission Services	

** R: Document, report (excluding the periodic and final reports)*

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

OTHER: Software, technical diagram, etc.

EXECUTIVE SUMMARY

SmartSDK is the FIWARE's "cookbook" for developing smart applications in the Smart City, Smart Healthcare, and Smart Security domains. It refines, combines, and develops new FIWARE Generic Enablers (GEs) and FIWARE Data Models into a set of well codified and ready to use solutions. This is very important to facilitate the take up of FIWARE by new developers and its transition from proof of concepts environment to productions ones.
As part of the compromises, SmartSDK initiative aims of developing new components for Internet of Thing (IoT) and Data Management to be used in a variety of scenarios, for example: smart cities, smart health and smart security. These components enable the developers to facilitate the use of physical devices, manage and capture context data, and share it through FIWARE Cloud platform.
This chapter presents the advances in the components developed for Internet of Things and Data Management tasks. The document presents two new hardware components developed for Internet of Things: (I) Cloudino component represents a new component that extends the capabilities of the Arduino platform to manage the connection of IoT devices with FIWARE. (II) Smart Sport component introduces a new generation of IoT that enables the interaction with the environment and the users, it is a device that allows users easily interact each other (e.g., suggestions, mailbox, and co-creation) and / or to obtain additional information from any point of interest (e.g., tourism, infotainment).
Moreover, in context of the Data Management module of FIWARE, this document presents three components that are being developed in the project to improve current components to manage context data produced by sensors and software applications. The first one will be used to retrieve NGSI historical data with the underlying power of modern time-series oriented databases. The second will brings a layer of encryption on top of sensible NGSI attributes, thus, data can be safely transferred to the Context Broker. The third component consists on a NGSI library which is a new component that enables developers to use mobile devices (e.g., smartphones) as context data producers by sharing information related to alerts, such as the event, location and severity of the event.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

BLE	Bluetooth Low Energy
CKAN	Comprehensive Knowledge Archive Network
DNS	Domain Name System
GEs	Generic Enablers
GSM	Global System for Mobile
GPS	Global Positioning System
IoT	Internet of Things
I2ND	Interface to Networks and Devices
LwM2M	Lightweight M2M
SensorML	Sensor Model Language
OCB	Orion Context Broker
OGC	Open Geospatial Consortium
OASC	Open & Agile Smart Cities
OMA	Open Mobile Alliance
MAC	Media Access Control
MQTT	Message Queue Telemetry Transport
POI	Point of Interest
SDK	Software Development Kit

1 INTRODUCTION

This section presents the status of the components developed for Internet of Things and Data Management. As stated before, new hardware components are being developed to manage the integration of physical devices in smart applications and services. The following sections describe the three main components under development in the project: The Cloudino that represents a new IoT component that extends the capabilities of the Arduino platform to manage the connection of devices to IoT and to send the data to the FIWARE Cloud. The Smart Spot represents the new generation of IoT components that interact with the environment and with the users. The ProximiThings Server can be integrated on IoT systems with FIWARE Platform to add proxemics interaction between objects and people.

New software components are being developed to manage the data. The NGSI library is a new software component that enable developers to use smartphones as context data producers, such as alerts or the location of the system users. The Data Series component permits the analysis of context data coming from system applications. The encryption components will enable developers to create secure connections to send sensible information among the applications.

1.1 Structure of the deliverable

The deliverable is structured as follow:

- Section 2 presents the current FIWARE Reference Architecture for IoT and Data Management services.
- Section 3 presents the contributions of the SmartSDK project related to the architectures described in Section 2 about Internet of Things Enablement services. Three of them are strictly related to the use of the Context Broker. The Smart Spot, Cloudino and ProximiThings are presented in this section.
- Section 4 presents the contributions of the Smart SDK project related to the architectures described in Section 2 about Data Management services. These include: NGSI Timeseries, a complementary element in Cosmos which provides an updated version of Comet STH, leveraging on the power of modern time-series databases; an NGSI Encryption Layer, that supports the selecting encryption and decryption of NGSI data, and a SDK Library for NGSI to connect several kinds of smartphones to Orion Context Broker via a NGSI RESTful interface with the objective of sending context data for mobile contexts.
- Section 5 summarizes the contributions presented in this deliverable in the context of SmartSDK.

1.2 Audience

This deliverable is mainly intended for:

- Developers and Operators interested into deploy FIWARE Smart applications in a production context.
- Developers and Data experts interested into adopting FIWARE Data Models or contributing to the initiative.

2 FIWARE REFERENCE ARCHITECTURE FOR IOT AND DATA MANAGEMENT

This section shortly summarizes the current FIWARE architecture for the Internet of Things Enablement and the Data Management services.

2.1 Internet of Things Enablement services

FIWARE offers an original ecosystem that allow things to become available, searchable, accessible, and usable context resources. To achieve that, FIWARE provides an IoT Backend Device Management (IDAS) that is normally the central enabler at the IoT backend for most common scenarios. This enabler allows IoT devices/gateways to connect to FIWARE-based ecosystems. IDAS IoT Agents translate IoT-specific protocols into the NGSI context information protocol that is the FIWARE standard data exchange model. The IoT Backend Device architecture is shown in Figure 1.

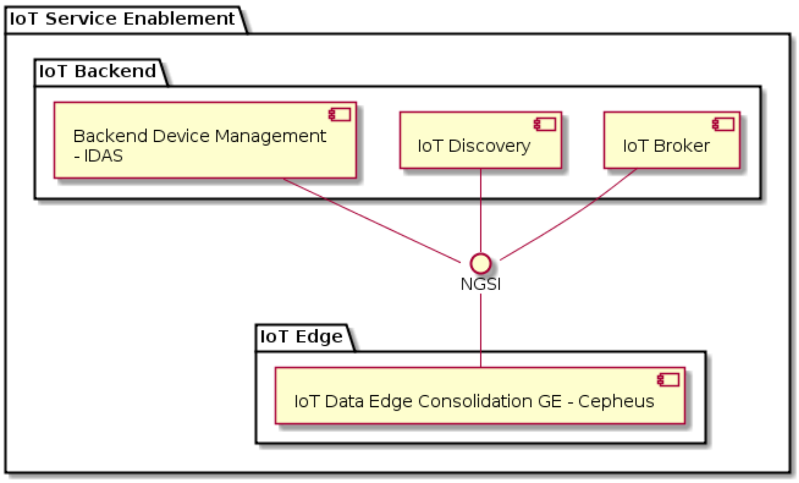

IoT Service Enablement
IoT Backend
Backend Device Management - IDAS
IoT Discovery
IoT Broker
NGSI
IoT Edge
IoT Data Edge Consolidation GE - Cepheus

Figure 1: IoT Backend Device Management Architecture

The main components of the previous architecture are:

- **IoT Agent:** The IoT Agents are the software modules handling South IoT Specific protocols and North OMA NGSI interaction. The minimum configuration of a Backend Device Management GE in a FIWARE ecosystem includes at least one IoT Agent.
- **IoT Manager:** The IoT Agent Manager is an optional module that will interface with all the IoT Agents installed in a datacenter throughout their Administration/Configuration API. This will enable a single point to launch, configure, operate and monitor all IoT-Agents in a FIWARE Ecosystem. It provides IoT Integrators with the ability of transforming devices specific Data Models into the Data Models defined at the NGSI level by different verticals (Smartcities, SmartAgrifood, Smartports, etc.).
- **IoT Edge Management:** The IoT Edge Manager is an optional module that will interface with IoT end-nodes, IoT Gateways and IoT network APIs throughout their IoT Edge API in order to operate and monitor the IoT Edge infrastructure that means connectivity, gateways and devices.

SmartSDK proposes two new IoT components that can be applied in the project scenarios, smart cities, smart security and smart health.

- Smart Spot, which is the infrastructure for the definition of a Smart POI (Point of Interest). Smart POIs are areas of interest, consisting of a set of Smart Spots (the specific point of connection) that broadcast a URL and create a physical space of information where everyone can interact through mobile devices. Smart POIs connect physical objects / places with the smartphone to offer an interactive experience.

- Cloudino, an IoT component that facilitates the connection of microcontroller devices to the Cloud. It is a modular and wireless implementation that integrate a new network layer for microcontroller solutions that need to connect to the Cloud.

2.2 Data/Context Management services

The Data/Context Management Chapter[1], depicted in Figure 1, contains the most relevant components to build the data storage and processing part of the Smart services. The interconnection of these components is implemented through FIWARE's standardized interface NGSIv2 [9]. Application developers, depending on their needs, can select to adopt any given subset of those components in their applications.

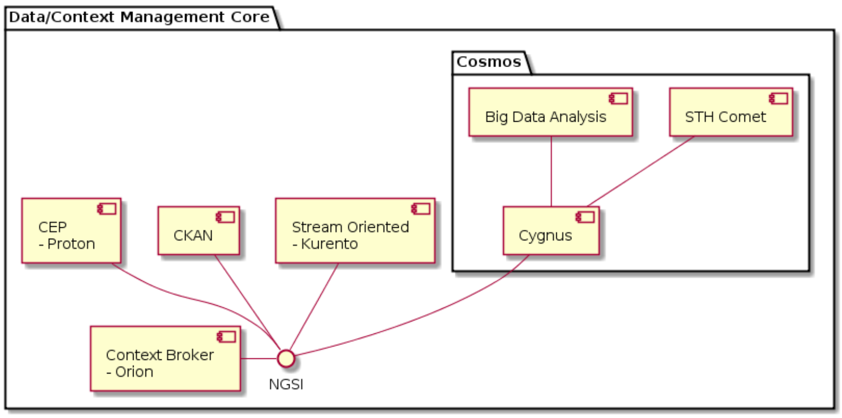


Figure 2: Data / Context Management Architecture

The core components of the Data/Context Management Chapter include:

- Context Broker - Orion, the central element of the Data Context Management architecture, allows both data producers and consumers to exchange data in different ways such as get/put and publish/subscribe services.
- Big Data Analysis - Cosmos, the solution for the analysis of NGSI data sets, made of different tools including the Short Time Historical data storage (STH Comet) and the integration with different datastores (relational databases, big data filesystems, etc.) through the adaptation capabilities provided by Cygnus.
- Stream Oriented - Kurento, an NGSI integrated multimedia server.
- CKAN - a repository for Open Data sets.
- Complex Event Processing (CEP) - Proton, an event-processing tool that identify patterns over NGSI data and generate response over identified patterns.

In Section 3, we explain the contributions of the SmartSDK project related to the Data Management Section. Two of them are strictly related to the use of the Context Broker, one by facilitating the interaction with mobile devices and the other by adding a layer to provide privacy over Orion Context Broker shared data. The third contribution is a complementary element in Cosmos which shares the same goal as the Comet STH; but which leverages on the power of modern time-series databases.

[1] https://catalogue.fiware.org/chapter/datacontext-management

3 INTERNET OF THINGS ENABLEMENT SERVICES

3.1 Introduction and common characteristics

The Internet of Things (IoT) has been defined as a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies [2]. The IoT allows objects to be sensed or controlled remotely across existing network infrastructure [3], creating opportunities for more direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy and economic benefit in addition to reduced human intervention [4]. According to [5][6] the main components of IoT concept are:

- Things: physical and virtual things (cars, person, system...)
- Sensors: sense the physical environment (GPS, speed, gases, RFID...)
- Actuators: affect the physical environment (stability controller, fluid controller, AC motors...)
- Communication platform: type of middleware used to connect IoT components (objects, people, services, etc.) to IoT (FIWARE, OpenIoT, Murano, ThingSpeak…).
- Network: IoT components are tied together by networks, using various wireless and wireline technologies, standards, and protocols to provide pervasive connectivity (Bluetooth, WIFI, Satellite...)
- Services: that processes the data obtained from sensors (Data analytics, store data, access to devices...).

In all these components, the relevance of software and hardware is crucial. A physical thing may be represented as a virtual thing (mappings of attributes and values). This thing can be associated a set of sensors for monitoring features (such as temperature, velocity, heart rate). The sensors may be communicated with other devices through a network (such as local or WIFI network) to send data, data storage and data processing. Moreover, currently exists a set of platforms for IoT that provides features as device management, charging and accounting, generic enable capabilities and so on.

FIWARE is a platform that provides a set of generic enablers that ease the development of Smart Applications in multiple vertical sectors. Specifically, FIWARE provides a component to connect IoT devices / gateways to FIWARE-based ecosystems called IDAS IoT Agent.

An IoT Agent is a component that translates IoT-specific protocols into the NGSI context information protocol that is the FIWARE standard data exchange model. The advantages of using an IoT Agent is that devices will be represented in a FIWARE platform as NGSI entities in a Context Broker. This means that a user / device / system can query or subscribe to changes of device parameters status by querying or subscribing to the corresponding NGSI entity attributes at the Context Broker.

Additionally, an IoT agent may trigger commands to actuation devices just by updating specific command-related attributes in their NGSI entities representation at the Context Broker. This way, all developers' interactions with devices are handled at a Context Broker, providing a homogeneous API and interface as for all other non-IoT data in a FIWARE ecosystem.

In this way, the use of an IoT agent provides a backend asset that can collect and store events from physical devices with or without the presence of intermediate gateways. It is also capable of forwarding commands to bidirectional devices (actuators). When the IoT Agent sends data to the Context Broker, it allows us to manage all the whole lifecycle of context information including updates, queries, registrations and subscriptions. In this way, the management of data from sensors through IoT Agents and then to Context Broker is crucial to receive a notification (heat alert, suspicious activities or fall or patient). All these features have achieved that IoT Agent is used in different scenarios: tourism, agrifood, health and so on.

Currently, the IoT Agent component has been deployed in the Smart SDK project. The component is deployed in the Cloudino Connector and the Smart Spot devices. Both devices allow to obtain data from gas sensors, temperature, relative humidity and location and the devices could be used in concrete applications in scenarios of the SmartSDK project: smart city, smart health and smart security.

For instance, in the smart city scenario the objective is to build an application with focus on supporting the citizen mobility in high-polluted cities, like Mexico City, with the aim of improving the life quality of citizens and fostering environmental friendly behaviours by citizens. The application aims to help the final user to determine the best route to follow to reach a destination, considering the user profile (such as health conditions), and the user preferences, such as transport type. To carry out this, this scenario will use the Smart Spot and Cloudino to obtain data from weather condition, pollution, and traffic jam.

The aim of smart security scenario is to develop applications to support the security guard to detect and prevent risk situations. The Smart Spot and Cloudino can be used as mobile sensor to detect risk situations involving cars inside a university campus.

3.1.1 FIWARE Reference Architecture overview

FIWARE provides a set of generic enablers that ease the development of Smart Applications in multiple vertical sectors. The generic enablers are clustered according the FIWARE chapters: Cloud Hosting, Data/Context Management, Internet of Things (IoT) Services Enablement, Applications/Services Ecosystem and Delivery Framework, Security, Interface to Networks and Devices (I2ND) and Advanced Middleware and Web-based User Interface. Each chapter describes a set of high-level descriptions of the APIs that each FIWARE Generic Enabler (GE) exposes to application developers or it uses to connect to another FIWARE GEs.

FIWARE provides a Reference Architecture to associate the different chapters of FIWARE. The Architecture can be instantiated into a concrete architecture by means of selecting and integrating products implementing the corresponding FIWARE GEs (i.e., products which are compliant with the corresponding FIWARE GE Open Specifications). In this section, the reference architecture for FIWARE context-aware, IoT-based and data intensive application have been analysed and extended. This architecture is shown in Figure 3.

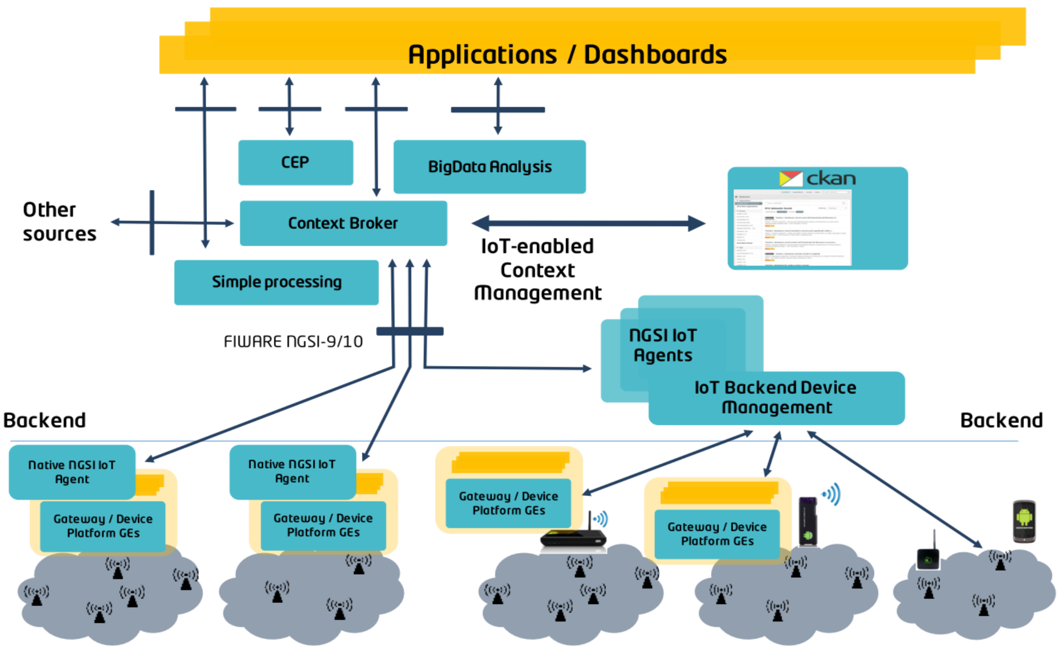

Applications / Dashboards
CEP
BigData Analysis
ckan
Other sources
Context Broker
IoT-enabled Context Management
Simple processing
FIWARE NGSI-9/10
NGSI IoT Agents
IoT Backend Device Management
Backend
Backend
Native NGSI IoT Agent
Gateway / Device Platform GEs
Native NGSI IoT Agent
Gateway / Device Platform GEs
Gateway / Device Platform GEs
Gateway / Device Platform GEs

Figure 3: An FIWARE context-aware, IoT-based and data intensive application. Derived from a presentation by Juan Jose Hierro, FIWARE Chief Architect.

Data coming from remote sensors are collected through the IoT Backend Device Management GE for sensors not offering a NGSI interface, whereas are sent directly to the Orion Context Broker for sensors already NGSI enabled. From Orion Context Broker data can be dispatched to other tools for elaboration, processing and analysis (Complex Event Processing GE or Big Data Analysis GE). Moreover, through the Context Broker data can be exported to CKAN GE, which is an enabler for open data management. Client applications or dashboards query the data in order to provide to the user a visual representation.

As part of the objectives of Smart SDK project is the development of Enablers to support the realization of IoT-based Smart Applications, then two enablers are being developed: Smart Spot and Cloudino Connector. The final objective is each enabler will be a FIWARE IoT Ready. These enablers have extended the Reference Architecture of FIWARE.

The extension is in two main ways. Firstly, the Smart Spot will be a pioneer component in FIWARE IoT Ready, this is because the Smart Spot will implant the hierarchy concept inside the FIWARE components. Secondly, the Cloudino Connector is a component that can connect to the FIWARE Context Broker without an IoT-Agent, using the simple Cloudino configuration Web Interface. The new components have extended the Architecture reference by FIWARE.

Then, these new IoT components have extended the FIWARE Architecture reference. In this extended architecture, the data coming from Smart Spot (for instance gases, temperature, humidity, location) are collected through the IoT Backend Device Management GE for sensors not offering a NGSI interface. On the other hand, the Cloudino connector has a special connection to send data directly to Orion Context Broker, using a NGSI interface. From Orion Context Broker data can be dispatched to other tools for elaboration, processing and analysis (Complex Event Processing GE or Big Data Analysis GE). Moreover, through the Context Broker open data can be exported to CKAN GE. Client applications or dashboards query the data to provide to the user a visual representation. This extended architecture is shown in Figure 4.

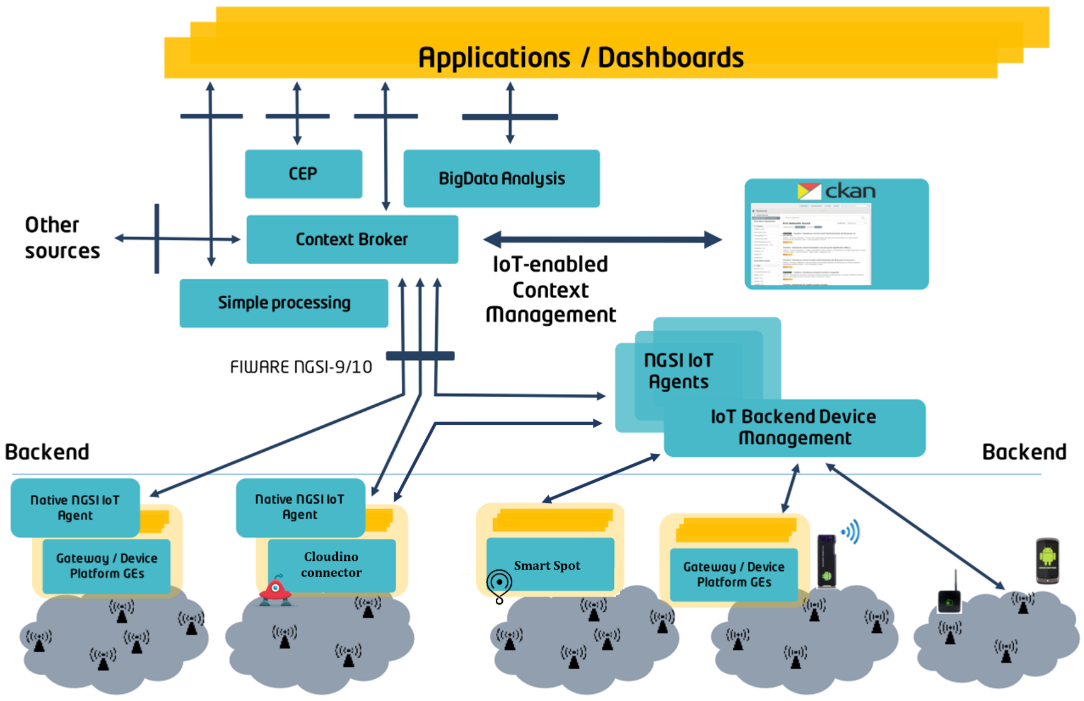


Figure 4: Extended architecture with the Smart Spot and Cloudino connector components to FIWARE IoT & Context Management Architecture

3.2 Smart Spot

A Smart Spot provides the ability of create an area of interaction with citizens and visitors. Thereby, People can connect with online content, discover webs from the physical places and the different digital services linked to the Smart point of interaction (Smart POI), such as booking, payment, participation, real time monitoring, etc.

- ➔ Physical Web technology links physical spaces with Digital Services through Web technology. Interact through your smartphone/tablet without any App required.
- ➔ Smart Spot makes use of Bluetooth Low Energy and WiFi technologies to send "push" messages to any Smartphone.
- ➔ Fully integrated with FIWARE and oneM2M. Creates Open and Agile Smart Cities.

3.2.1 3.2.1 Architecture

The platform and management of the Smart Spot is based entirely on FIWARE and open standards such as OMA LwM2M, Physical Web, OMA NGSI and other technologies based on Open standards that guarantee the system is not locked into proprietary or unique vendor solutions. Figure 5 presents the architecture of the open and agile platform for Smart Cities solutions.

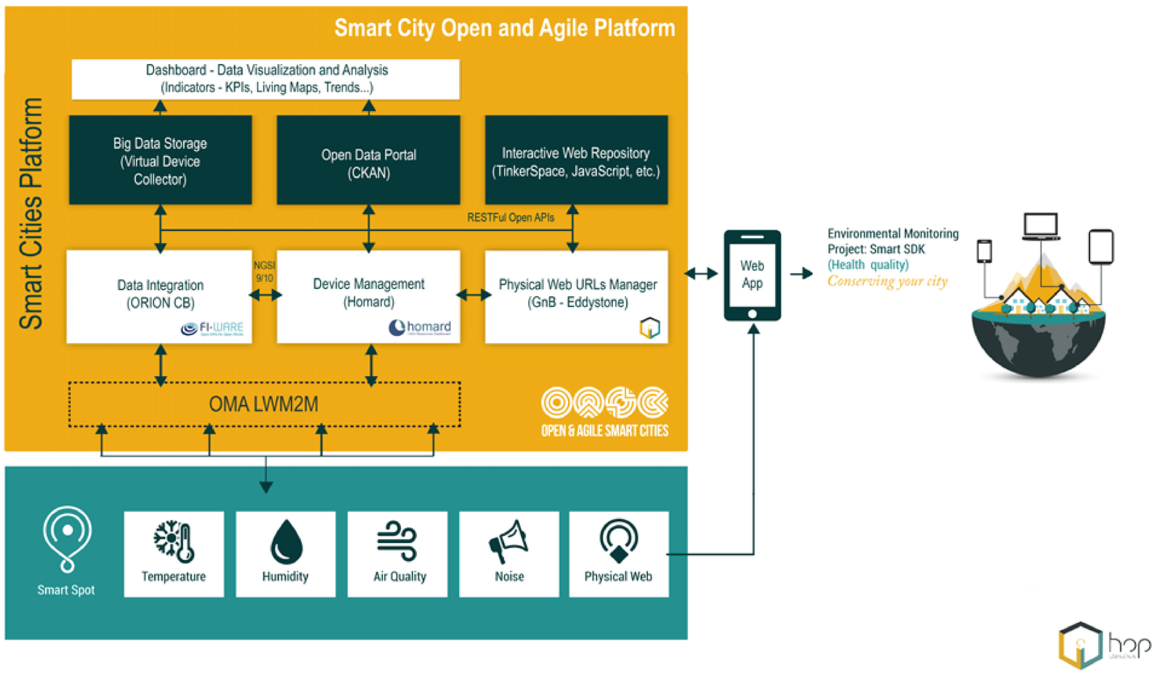

Smart City Open and Agile Platform
Smart Cities Platform
Dashboard - Data Visualization and Analysis (Indicators - KPIs, Living Maps, Trends...)
Big Data Storage (Virtual Device Collector)
Open Data Portal (CKAN)
Interactive Web Repository (TinkerSpace, JavaScript, etc.)
RESTFul Open APIs
Data Integration (ORION CB)
NGSI 9/10
Device Management (Homard)
Physical Web URLs Manager (GnB - Eddystone)
OMA LWM2M
OPEN & AGILE SMART CITIES
Web App
Environmental Monitoring
Project: Smart SDK
(Health quality)
Conserving your city
Smart Spot
Temperature
Humidity
Air Quality
Noise
Physical Web

Figure 5: Architecture of Open & Agile Smart Cities Solutions

The Platform is responsible for interacting, obtaining and integrating data from different environments or data sources with which it must interact. In particular, the following functionalities are offered:

- ➔ **IoT sensors / actuators:** This component interacts with entities such as devices (sensors and actuators) or smartphones, integrating the diversity of formats, protocols or technologies that can be found in a Smart City. In details, the protocol will preferably be OMA LwM2M, OMA NGSI, MQTT and also SensorML of OGC (Open Geospatial Consortium), and that constitutes a model of description of the resources in general, being these devices or other systems. The IoT Devices Integration component has as fundamental responsibility to interact and integrate all those devices belonging to urban services of the city such as: environmental sensors, parking sensors, measures reported by traffic control elements, beacons etc. In addition to supporting the protocols, advanced support for semantic integration is offered that allows integration of the measurements sent by the different heterogeneous systems and offers a common integration via semantics (such as the OMA NGSI data models).
- ➔ **Integrations aligned with the OMA NGSI standard and the FIWARE / OASC**

specifications: The platform is oriented to the implementation of scenarios with control and support of interactivity, such as reaction to events, alarms, etc. based on a modular architecture for context management (according to OMA NGSI) that manage the relevant information and maintain the state for any type of defined entity. This architecture plays a strategic role in the face of interoperability and integration, since it guarantees the horizontality of the information, so that any data of the platform can be consulted through a subscription process based on open standards. This ensures the availability of updated data on the actual state of the ecosystems integrated in the platform. In addition, this component will receive all the data from the various sources and implement the publication / subscription mechanisms that make it possible to circulate information between the producers and the consumers of the same.

The communication between the different actors that interact in the management of contexts is done through a RESTful OMA NGSI interface. Inspired by the standard OMA NGSI specification, which defines an interface capable of handling any type of data, including metadata. At the same way support and integration with other existing platforms is also supported.

➔ **Interaction with the citizen (Physical Web) via personal / mobile devices:** One of the advantages of the proposed architecture in relation to other solutions in the market is its orientation to- wards promoting citizen participation, co-creation and interaction through mobile services. That is why architecture integrates beacons that offer a direct way of interacting with the user, promoting places, offering information and above all collecting the opinions / ideas of citizens.

The following image represents the Smart Spot Hardware Architecture.

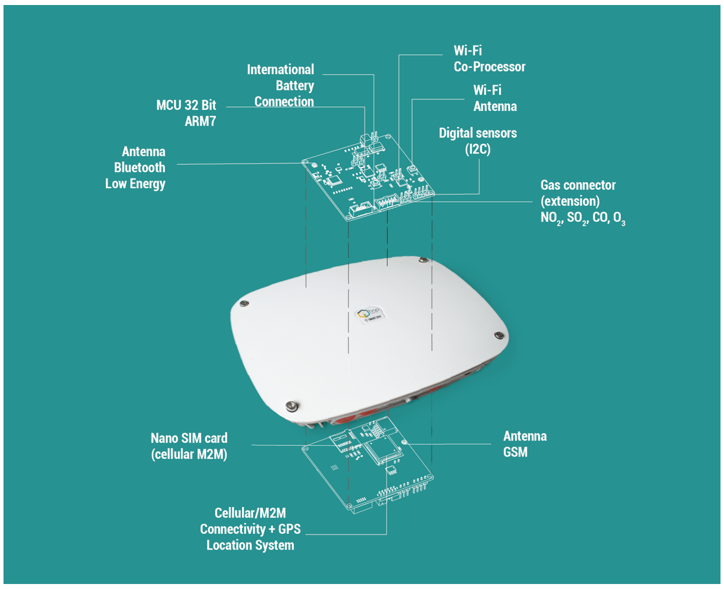

International Battery Connection
Wi-Fi Co-Processor
Wi-Fi Antenna
MCU 32 Bit ARM7
Digital sensors (I2C)
Antenna Bluetooth Low Energy
Gas connector (extension) NO_2, SO_2, CO, O_3
Nano SIM card (cellular M2M)
Antenna GSM
Cellular/M2M Connectivity + GPS Location System

Figure 6: Smart Spot Hardware Architecture

Table 1: Smart Spot Specification Datasheet

Enclosure	Outdoor protection IP55 (Resistant to water and dust)
Radio Interfaces	2 x 802.11 b/g/n/e/i (802.11n @ 2.4 GHz up to 150 Mbit/s) – STA-AP/Sniffer
	1 x Bluetooth Low Energy Beacon (Eddystone / Physical Web)
	1 x M2M Connectivity (Cellular GPRS) - SIM card
MCU Core / Clock Speed	Tensilica Xtensa dual-core 32-bit LX6 / 240MHz

Internal Memory	16 MB
Cellular Quad-band	850/900/1800/1900MHz (GPRS - Class 12 modem)
Hardware accelerated encryption	AES / SHA2 / Elliptical Curve Cryptography / RSA-4096
External Interfaces	I2C Probe and ADC (GPIO) interface for external sensors - Temperature and humidity Probe. - Environmental monitoring Probe. - Noise/acoustic Probe.
Dimensions	80mm x 80mm x 36mm (IP55 encapsulation)
Temperature Range	-20 oC to 80 oC operating temperature
SIM Card Slot	Nano SIM Card Connection -12,3mm x 8,8mm (4FF)
Power Supply	5V (USB compatible)
Battery Charger	Li-ion battery charger
Internal Battery Connection	JST 1.0 Connector (1200mAh Li-on internal battery)
Energy Harvesting	Solar Panel + 10000mAh external battery (IP65 protection)
Bluetooth Low Energy Co-Processor	Texas Instrument CC2541
Wi-Fi Co-Processor	Expressif ESP8266
Cellular GPRS Co-Processor	Simcom SIM868
GPS Outdoor Location	GPS L1 C/A code - 22 tracking/66 acquisition channels
Antennas	External Antenna WiFi 802.11 b/g/n/e/i (STA-AP)
	Internal PCB Antenna WiFi 802.11 b/g/n/e/i (Sniffer - Crowd Monitoring - SmartPhone Detection)
	External Antenna GSM/GPRS (Cellular)
	External Antenna Bluetooth Low Energy
Software Full Stack	IPv4 / IPv6 Connectivity – Internet of Things
	RESTFul (HTTP / CoAP) – Web of Things
	OMA LwM2M Device Management (Firmware Upgrade Over the Air)
	FIWARE NGSI Data Models / ETSI CIM (POI + Device + Extensions)

3.2.2 Interaction with users by URL

Smart POIs (Smart Point of Interaction) are strategic smart areas of interest (POI) where can be accessed digital content geolocated at a specific physical point of interest. This is thanks to a set of Smart Spots (the specific point of connection that use similar beacon technology) that send a URL and create a physical space of information where everyone approaching can collaborate through a smartphone, tablet and with another smart device. Therefore, Smart POIs connect physical objects or places with the smartphone to offer an interactive and multimedia experience. This technology allows to directly open a responsive Web App that contains information designed to answer a specific topic, including text, videos, images and any multimedia material. The devices work by proximity (20 meters) both outdoors and indoors. Smart POIs have a multitude of possibilities for the tourism industry, such ailing the information gaps existing in the cities, and connecting the consumer with services and products related to the sector, proximity marketing, geographic targeting, and content broadcasting.

3.2.3 Connection with Cloud servers

The diagram of the figure 7 represent the current mode for connecting a Smart Spot that represent a multi entity model to the Orion Context Broker.

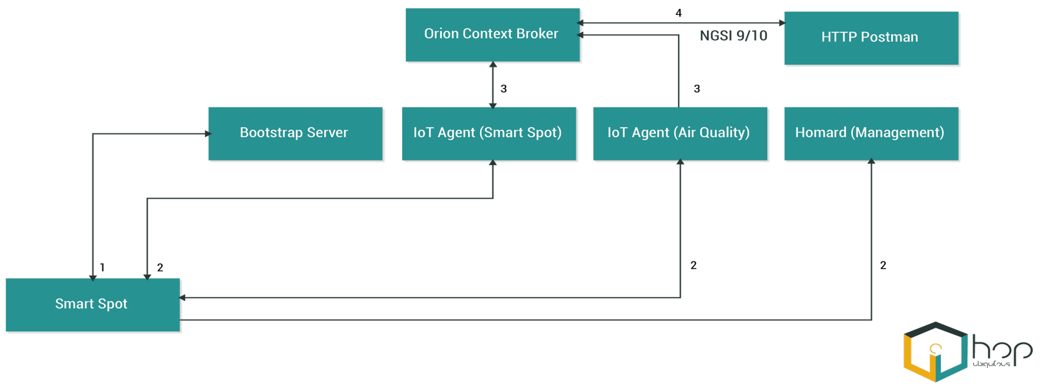


Figure 7: Smart Spot Cloud Connection Architecture

1. The Smart Spot connect with the bootstrap server, this bootstrap server create the configuration need in the Smart Spot for connecting with the desired servers.
2. As the IoT Agent does not support multi entity, The Smart Spot has to connect with different IoT Agent.
3. Each IoT Agent manage a different NGSI entity on the Orion Context Broker.
4. The user can read and change the Smart Spot measurement and status through the Orion Context Broker.

The Method has many disadvantages, the device has to maintain several connections increasing the power and data consume. Several IoT Agents have to be maintained. The Smart Spot configuration has to be written in different services.

For that reasons we have proposed an alternative (Figure 8), this alternative consist in a multi entity IoT Agent, this enabler will resolve all the precious problem, some of them critical for an IoT device like the Data and Power Saving and the single service configuration.

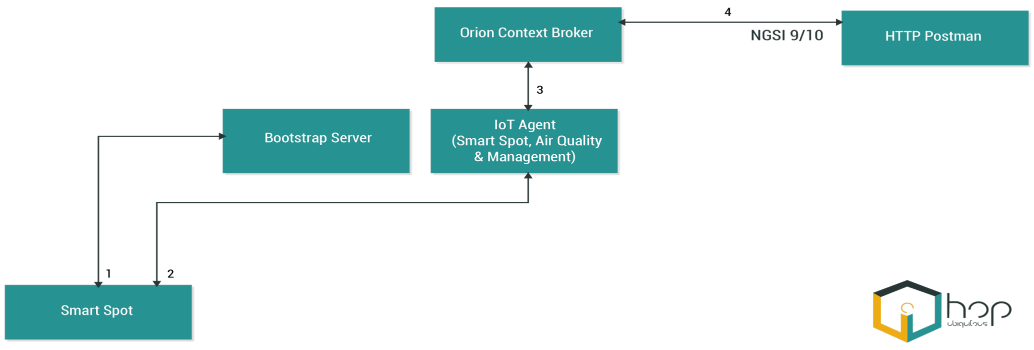


Figure 8: Multi entity IoT Agent for Smart Spot

1. The Smart Spot connect with the bootstrap server, this bootstrap server create the configuration need in the Smart Spot for connecting with the desired servers, in this case only one multi entity IoT Agent.
2. The Smart Spot connect with the multi entity IoT Agent.
3. The IoT Agent manage all the different NGSI entities on the Orion Context Broker.
4. The user can read and change the Smart Spot measurement and status through the Orion Context Broker.

This alternative way for connection the IoT devices is introduced in the Hopu roadmap.

3.2.4 Status and Roadmap

The following features have been implemented in the period covered by this report:

➔ IoT Agent Fixes: For the correct behavior of the device, some corrections were made in this FIWARE enabler, this correction were about the way of manage the LwM2M protocol and they were merged in the master branch by the repository manager.

➔ BLE Physical Web capability in Smart Spots: The device is able to broadcast the desired URL via Bluetooth using the google protocol eddystone URL and giving to the device de capability of send physical webs to the users.

➔ Device URL Manager: This service provide to the user the capability of admin the URL broadcasted by the devices seamless, it also provide some statistics like the number of interactions. Now days this service is used by an API REST but we are working in a user interface to facilitate the user's interactions. This tool is used for manage the physical web URL of any device by software. A smartphone will detect an eddystone URL advertisement with a fixed device URL that point to the Device URL Manager, then the Device URL Manager will redirect the request to the real URL.

***shortened mac: is a normal mac without the two dots**

Create Device:

```
Method:         POST
URL:           /api/v1/devices
URL Params:    None
Data Params:   application/json
Body:
```

```
{
        "mac": "001122334455",
        "external_url": "https://google.es/"
}

Successful Response:
Code:           201 (Created)
Content:
{
        "mac": "001122334455",
        "external_url": "https://google.es/"
}

Error Response:
Code: 400 (Bad Request)
Content:
{
        "bad_field_name": [error causes]
}
```

Show Device Data:

```
Method:          GET
URL:            /api/v1/devices/:shortened_mac
URL Params:     shortened_mac=[String]
Data Params:    None

Success Response:
Code:           200 (Ok)
Content:
{
        "mac": "001122334455",
        "external_url": "https://google.es/"
}

Error Response:
Code:           404 (Not Found)
Content:
{
        "detail": "Device Not Found"
}
```

Update Device Data:

```
Method:          PUT
URL:            /api/v1/devices/:shortened_mac
URL Params:     shortened_mac=[String]
Data Params:    application/json
Body:
{
        "mac": "001122334455",
        "external_url": "https://google.es/"
}

Success Response:
Code:           200 (Ok)
Content:
{
        "mac": "001122334455",
        "external_url": "https://google.es/"
}
Error Response:
Code:           404 (Not Found)
```

```
Content:
{
        "detail": "Device Not Found"
}
```

- GSM Module: Thanks to this module, the Smart Spot has the capability of sending data using a micro sim with data. This module also has manage features like connection handle disconnections, use the cheaper connection available, reconnect if is possible and improve the power saving. This module also provide to the Smart Spot the capability of by localized via GPS, this feature is interesting when we are trying to improve mobile entities.
- Smart Spot Data Model: For the correct integration of the Smart Spot in the FIWARE ecosystem, we have been working in a NGSI data models approved and certificated by FIWARE Community. https://github.com/Fiware/dataModels/tree/master/PointOfInteraction

Roadmap for the Smart Spot can be grouped in three steps:

- Sensor integration: For this project will be integrated several sensors, likes temperature, humidity, air quality and accelerometer, some of this sensors have to be defined in the OMA protocol and conveniently parsed to NGSI entities.
- Improve IoT Agent: because of the problem described in the previous section (3.2.3), we have to improve the IoT Agent enabler with the finality of save data and communications.
- Offline gathered data: The device must be able to send the data measured will the connection was unreachable.

3.3 Cloudino

Cloudino is a full stack IoT platform that provides all necessary components to transform the existing microcontroller's solutions (Atmel AVR, Microchip PIC, etc.) to the IoT world.

Cloudino was designed thinking in three main characteristics to take to reality the vision of the Internet of Things: small size, easy to use and low cost hardware, and with these characteristics, the **Cloudino** allows to everyone the possibility to incorporate IoT technologies in their projects without any technical or economical limitations.

3.3.1 Architecture

The **Cloudino** proposes to add a new IoT Chip that works like a configurable Network Layer between the existent hardware solutions (Microcontrollers, Sensors or Actuators) and the Cloud Services, for a simple and fast start to IoT World. The platform consists of three main components, which can work together or independently.

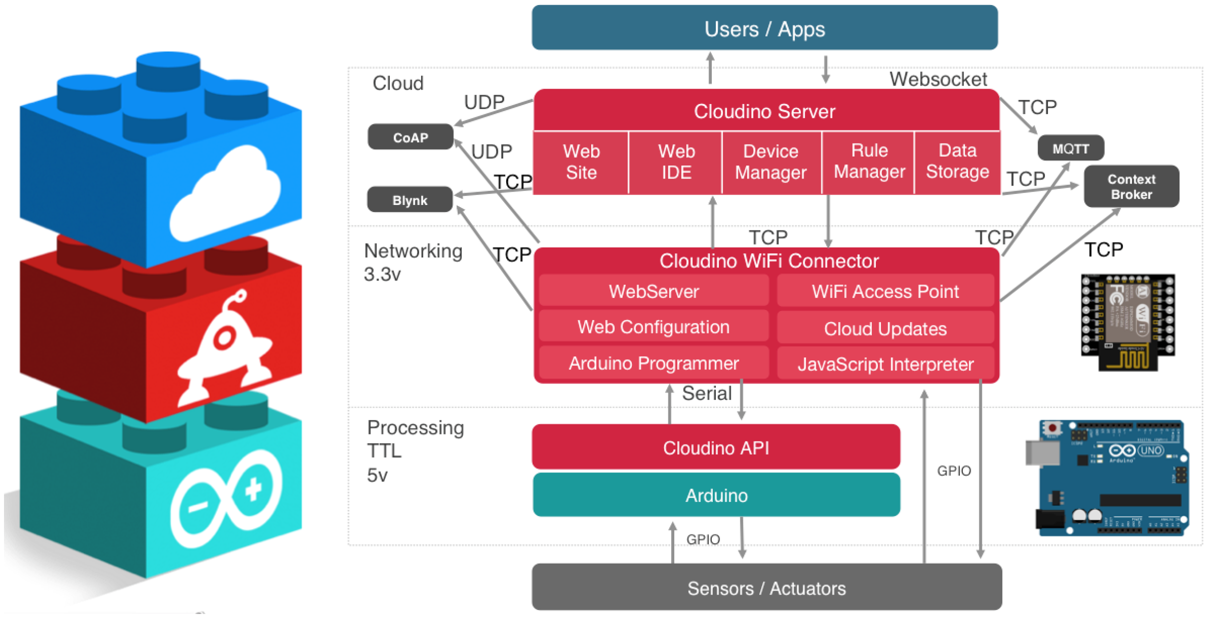
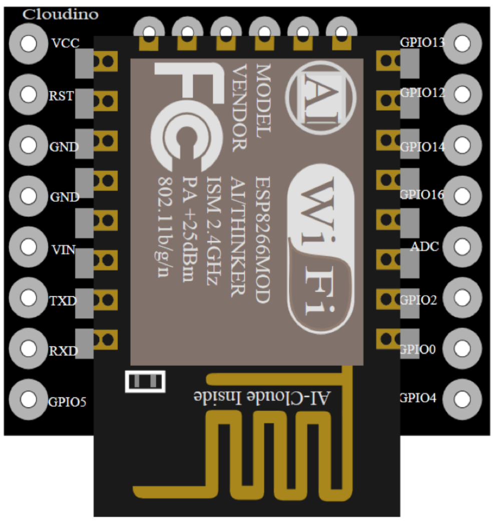


Figure 9: Components of Cloudino

The first component is the **Cloudino API**, which has a specific implementation for different microcontroller solutions (Arduino, Intel Edison, PICs, etc.), the function of the API is to isolate the microcontroller code to the specific IoT protocol, this mean that the programmers can use the same code for send data to a **MQTT Server** or to an **Orion Context Broker** or to the **Cloudino Server** without doing any change to the source code, only configuring the specific protocol in the networking layer.

Another of the main components is the **Cloudino WiFi Connector**, which is a little, inexpensive and powerful IoT Chip, that has preprogrammed the most common IoT protocols like a MQTT or the NGSI for the Orion Context Broker, that allows everyone to start sending information to the Cloud without any additional programming effort. The Cloudino WiFi Connector can be seen as an IoT Router that can be configured using a simple web browser.

Another important characteristic of the **Cloudino WiFi Connector** is that can working in parallel with Arduino and can be used as an Arduino Cloud Programmer.



The **Cloudino WiFi Connector** can be used as an **additional microcontroller dedicated to the network layer**, working in **parallel** with actual microcontroller solutions like Arduino. Also can be used as a **standalone device** for directly communicate the real-life objects to the internet.

The **Cloudino WiFi Connector** has **10 digital GPIOs** and **one analog GPIO**, that we can use for connect sensors and actuator directly to the chip and can be programmed using and **JavaScript** Engine that is working inside the chip.

The **Cloudino WiFi Connector** can be configured to connect to **any cloud service**, however in order to get the best out of the solution, the platform contains the **Cloudino Server**, which includes all the components needed to manage devices from anywhere in the world.

The use of the **Cloudino Server** is optional; however, it has many advantages over existing services, such as **Devices, Rules and User Contexts Manages, and Stream Data Storage.**

The **Cloudino Server** includes also **web IDE** that allows devices to be programmed and debugged via Cloud and it includes an easy "**Blocks programming interface**" for non-experimented programmers.

3.3.2 Connection Cloudino to FIWARE

Cloudino WiFi Connector can be integrated with FIWARE above-described FIWARE IoT ecosystems using different mechanisms:

- Direct Connection.
- Connection via MQTT IoT-Agent.
- Connection via Cloudino.io cloud service.

The simple's way to integrate Cloudino to FIWARE is the **Direct Connection;** this is configuring the Cloudino WiFi Connector to send direct data to the OCB without any IoT Agent. This configuration is very convenient if the solution only contain sensors that periodically reports the status to the server and where do not require any feedback from the server side.

If the solution require feedback from the server side, the **Connection via MQTT IoT Agent** could be used**.** This option is very convenient considering that you have the opportunity to send and receive messages more efficiently; however, another component is required to be configured and maintained.

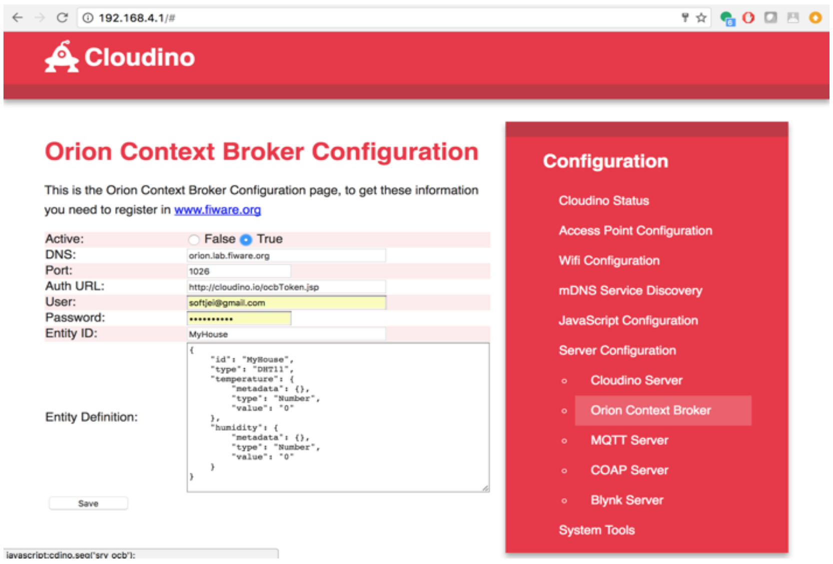
Perhaps the best way to connect the Cloudino to FIWARE is to use the **Service of Cloudino Server**, because Cloudino acts such as an IoT Agent, which not only allows the sending and reception in real time of messages, but also allows the administration of the devices, the possibility to create rules of communication between the devices and besides having a development environment that allows the programming of the devices via cloud.

3.3.3 Direct connection to FIWARE

Cloudino WIFI Connector can connect to the FIWARE Context Broker without an IoT-Agent, using the simple Cloudino **Configuration Web Interface**.

The **Cloudino Connector** starts an **WiFi Access Point** that lets you connect to the configuration web interface at: http://192.168.4.1

To use a **direct connection to FIWARE Context Broker,** the option of **Orion Context Broker** inside the **Server Configuration** need to be selected by the developer, and the following fields need to be configured: The DNS and the Port of the OCB Server, the URL, User and Password for get the authentication token (*https://orion.lab.fiware.org/token*, only in case that the OCB Server needs it), and the entity definition template used for create the initial entities.



Orion Context Broker Configuration

This is the Orion Context Broker Configuration page, to get these information you need to register in www.fiware.org

Active:	False [✓ True]
DNS:	orion.lab.fiware.org
Port:	1026
Auth URL:	http://cloudino.io/ocbToken.jsp
User:	softjei@gmail.com
Password:	••••••••••
Entity ID:	MyHouse

Entity Definition:

```
{
    "id": "MyHouse",
    "type": "DHT11",
    "temperature": {
        "metadata": {},
        "type": "Number",
        "value": "0"
    },
    "humidity": {
        "metadata": {},
        "type": "Number",
        "value": "0"
    }
}
```

javascript:cdino.seql('srv ocb');

Figure 10: Cloudino configuration screen for the direct connection to FIWARE

Once configured the protocol to be used to send data to the cloud, the next step is to configure the **WiFi network** to be used by the device to connect to the internet, this can be done selecting the **WiFi Configuration Option** on the menu.

Once the Cloudino WiFi Connector is configured to access the internet, the next step is to program the specific logic to perform the functions of collection and sending data to the cloud, for which there are two possibilities:

Firstly, it is to use the **Cloudino WiFi Connector as a WiFi Bridge between an Arduino and the Cloud**, connecting the Sensors and Actuators to the Arduino and programming in the Arduino the specific logic to send data to the cloud through the Cloudino WiFi Connector.

Example of Arduino Code to Post Temperature and Humidity

```
#include <Cloudino.h>
#include <dht11.h>

#define DHT11PIN 8

Cloudino cdino;                      //Cloudino Library
dht11 DHT11;                         //DHT11 Library

void getSensor()
{
   int chk = DHT11.read(DHT11PIN);
   cdino.post("temperature",String((float)DHT11.temperature,2));
   cdino.post("humidity",String((float)DHT11.humidity,2));
   cdino.print("Timer done!");    //Send to console
}

void setup()
{
   cdino.setInterval(10000,getSensor);  //Timer every 10 seconds
   cdino.begin();
}

void loop()
{
   cdino.loop();
}
```

Secondly, it is to use the Cloudino WiFi Connector to directly connect the sensors and **actuators** to the device and programming in it the specific logic for collecting and sending data to the cloud using the JavaScript interpreter integrated in the Cloudino WiFi Connector.

Example of CloudinoJS to Post Temperature and Humidity

```
//import Cloudino, Timer and DHT11
require("Cloudino");
require("Timer");
require("DHT11");

//Create timer every second 5s(5000ms)
setInterval(function(){
  //Read DHT11 on GPIO 14
  var sens=DHT11.read(14);
  //Post temperature an humidity data to defined Server
  Cloudino.post("temperature",sens.temperature);
  Cloudino.post("humidity",sens.humidity);
},5000);
```

Example of request to FIWARE Context Broker

```
curl orion.lab.fiware.org:1026/v2/entities/MyHouse -X GET -s -S \
        --header 'Accept: application/json'\
        --header "X-Auth-Token: $AUTH_TOKEN" | python -mjson.tool
```

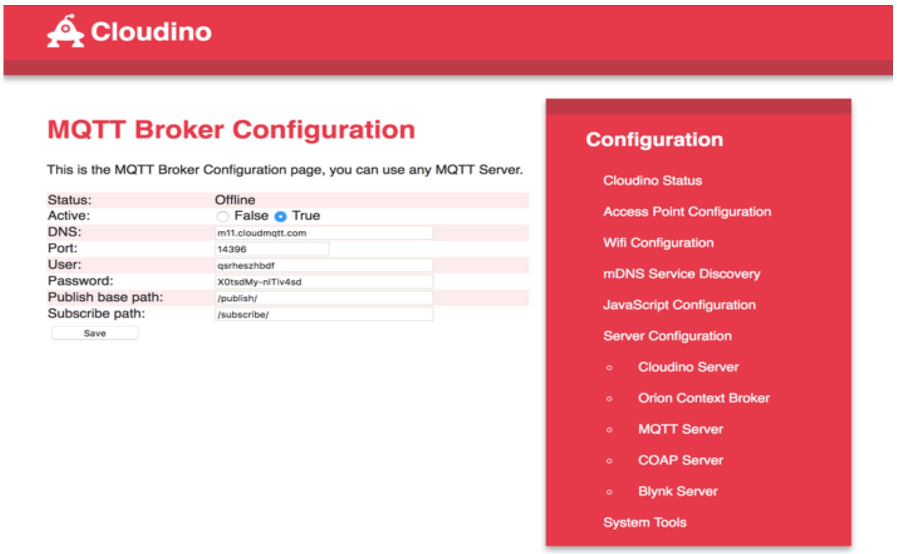
3.3.4 Connection through and MQTT IoT Agent

Cloudino WiFi Connector can connect to the FIWARE using MQTT IoT-Agent, using the simple Cloudino Configuration Web Interface.

The Cloudino WiFi Connector starts an access point that lets you connect to the configuration web interface at: http://192.168.4.1

To use a MQTT Protocol to connect to FIWARE Context Broker, the option of MQTT Server inside the Server Configuration need to be selected by the developer, and the following fields need to be configured: The active field need to be true, the DNS, Port, User and Password of the MQTT IoT Agent need to be specify.

Finally, it is necessary to define the routes of publication and subscription for filtering messages, based on these routes the device will send and receive messages or properties that would be part of the entity stored in the Orion Context Broker. However, the end configuration to connect to the Orion Context Broker will have to be defined on the side of the MQTT Agent.



MQTT Broker Configuration

This is the MQTT Broker Configuration page, you can use any MQTT Server.

Status:	Offline
Active:	False [✓ True]
DNS:	m11.cloudmqtt.com
Port:	14396
User:	qsrheszhbdf
Password:	X0tsdMy-nITiv4sd
Publish base path:	/publish/
Subscribe path:	/subscribe/

Figure 11: Cloudino configuration screen for connection to FIWARE using MQTT Agent

Once configured the protocol to be used to send data to the cloud, the next step is to configure the

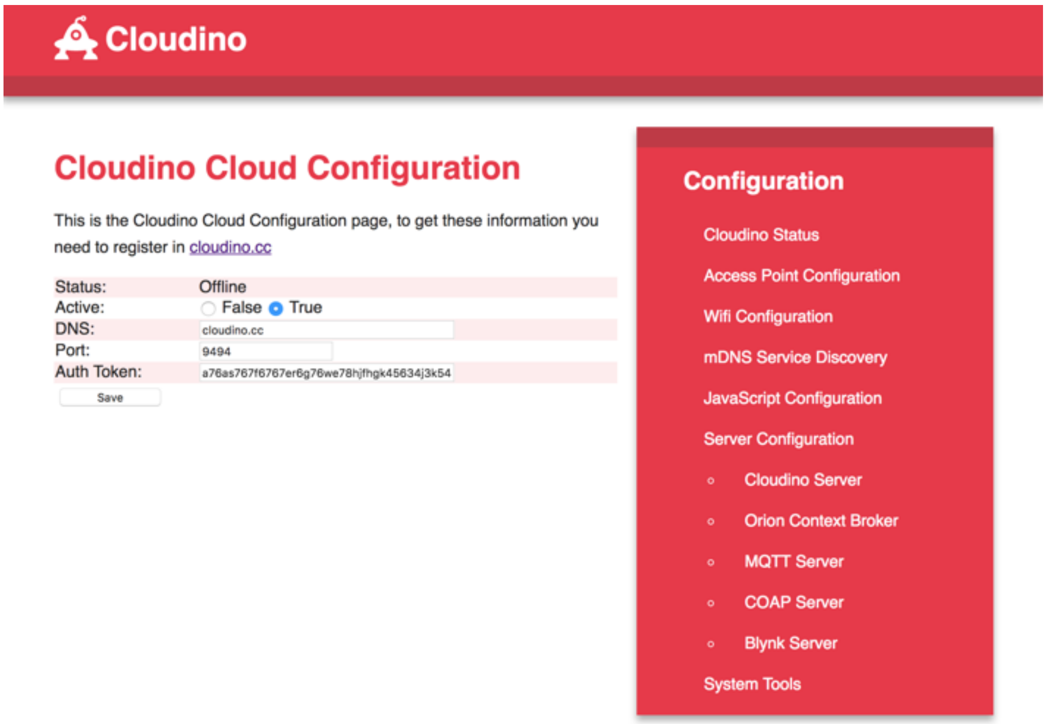
WiFi network and program the **specific logic** to perform the functions of collection and sending data to the cloud, for do this you can follow the steps described in the section of **Direct Connection to FIWARE**

3.3.5 Connection through Cloudino Cloud Service (Work in progress)

Cloudino Connector can connect to the **FIWARE** using **Cloudino Cloud Service**, using the simple Cloudino Configuration Web Interface.

The **Cloudino Wifi Connector** starts an access point that lets you connect to the configuration web interface at: http://192.168.4.1

To use the **Cloudino WiFi Connector** to connect to **FIWARE Context Broker** through **Cloudino Server**, the option of **Cloudino Server** inside the **Server Configuration** need to be selected by the developer, and the following fields need to be configured: The active field need to be true, the DNS, Port and Auth Token need to be specify.



Cloudino Cloud Configuration

This is the Cloudino Cloud Configuration page, to get these information you need to register in cloudino.cc

Status:	Offline
Active:	False [✓ True]
DNS:	cloudino.cc
Port:	9494
Auth Token:	a76as767f6767er6g76we78hjfhgk45634j3k54

Figure 12: Cloudino configuration screen for connection to FIWARE using Cloudino Cloud Service

The **Authentication Token** can be obtained by **registering** on the **cloudino.io** platform and creating a device to link to the **Cloudino WiFi Connector.**

Once configured the protocol to be used to send data to the cloud, the next step is to configure the **WiFi network** and program the **specific logic** to perform the functions of collection and sending data to the cloud, for do this you can follow the steps described in the section of **Direct Connection to FIWARE**

3.3.6 Status and Roadmap

Currently the development of the **Cloudino** is ongoing; however, it is already in a **fully functional stage**. The main development efforts are focused on the one hand in the development of **new devices** for the platform with other mechanisms of data transmission, such as:

- ➔ Cloudino GSM Connector

- Cloudino Lora Connector
- Cloudino SigFox Connector

On the other hand, the development of **Cloudino Server** is continued to make it not only an **IoT Solution Development Platform**, but also a **Data Broker** that implements the main existing protocols used in the IoT.

The current working prototype of Cloudino can be found at https://github.com/Cloudino

3.4 ProximiThings Server

ProximiThings is a FIWARE-enabled framework for the incorporation of proxemic interaction capabilities in IoT systems. The five dimensions of proxemics for ubicomp defined by Greenberg et al. (see Figure 13) constitute the basis for building proxemic interactions. These five dimensions are distance, orientation, location, movement and identification. These dimensions can be measured by different approaches, and using different types of devices. For instance, the distance between a person and an (smart) object could be measured by a Kinect placed alongside the object, but it could also be measured with a small board having an ultrasonic sensor. In both cases, although the particular device may vary, the principle is the same and consists of measuring the round trip time of the infrared signal. We are using Orion Context Broker as a Context Consumer to get sensor information for further conversion into interaction information.

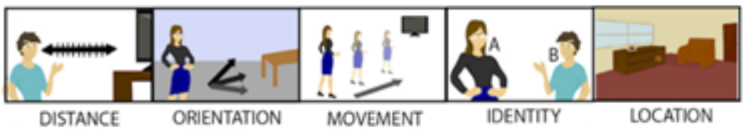


Figure 13: five proxemics dimensions of ubicomp as defined by Greenberg et al.

3.4.1 Architecture

Given the limitations in processing, connectivity and synchronization on the part of the IoT devices, we propose the implementation of a cloud service to overcome such limitations. For this, we are using Orion Context Broker, which allows the storage of context information; it has a RESTful API based on JSON for the CRUD methods of entities and context information. Orion also has methods to notify about updates about context information, using Webhooks with HTTP for this purpose. ProximiThings obtain information from entities through HTTP notification from Orion Context Broker.

ProximiThings Server will store the proxemics dimensions information for further conversion into interaction information. For efficient communication between the Orion Context Broker and IoT devices, we will use IoT Agents (GE) that support the MQTT protocol. This component maps information received by devices in plain text format to an appropriate JSON format to perform context information updates in the Orion Context Broker.

The FIWARE platform does not have any component or GE to allow processing proxemics related information, so this is the main reason why the ProximiThings Server is developed. ProximiThings uses the notification service of the Orion Context Broker to receive updates from the devices concerning the proxemic dimensions; also, through the RESTful API it updates context parameters in the entities.

Information about proxemics dimensions is received in continuous units and is in turn converted into discrete units, turning the Orion Context Broker into a Proxemics Provider. This discrete information is made available to other systems (Proxemics Consumers) through a RESTful API. There are three components or modules in ProximiThings Server: Proxemics Data Conversor, User Interaction Rules and Proxemics Actions.

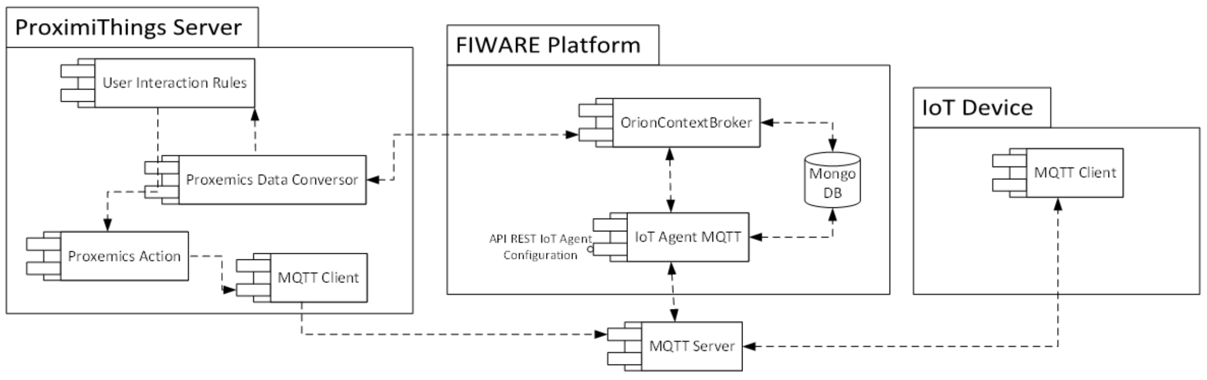


Figure 14: Architectural overview of the ProximiThings framework

Proxemics data conversor is a module of the ProximiThings which allows proxemics data received in continuous units to be converted into discrete units. It is also the module directly connected to the Orion Context Broker to receive updates concerning the measurements of proxemic dimensions. The Orion Context Broker manages context data from the entities received by the devices, and this data can be obtained in several ways. For instance, distance can be measured with IR or optical sensors; identity may be obtained using any type of link, including RFID, NFC, Bluetooth or IR; motion can be detected with a simple IR sensor or with a more complex computer vision system that tracks objects. Therefore, ProximiThings is flexible, as it is not tied to any particular set of sensing devices to generate proxemics information.

User interaction rules is a module intended for developers to set interaction rules based on the discretized proxemic dimensions, in order to trigger actions or commands define in the Proxemics Actions module. Each command may have multiple interaction rules, and these rules are a subset of possible values that a proxemic dimension could have. For instance, if we wish to show information on a display to a specific person, the rules shown in Table 2 should be set.

Proxemics actions. When a user-defined proxemic interaction rule is met, a command stored in this module is executed. There can be two types of command: 1) a message to be transmitted via MQTT to IoT devices or 2) an HTTP callback (webhook) including the command and values of the proxemic dimensions of the entities. Each IoT device supports different functions and the MQTT message should specify the function and the data needed for executing that function. Some of the functions executed by ProximiThings in the devices are the transmission of codes via IR, and the interruption of electrical flow.

The HTTP callback allow the incorporation of other cloud services, such as IFTT, so ProximiThings can then be linked to services such as Facebook, Twitter, Dropbox, etc. Another important element of ProximiThings is its RESTful API, which allows developers to incorporate proxemics capabilities into their IoT systems. This API has support for CRUD operation in interaction rules (User Interaction Rules), actions to be executed (Proxemics Actions), as well as for user profiles and to consult information from any entity of a IoT environment. This is performed as a REST service which responds in a JSON format. This way, developers can have a REST client to update or consult information, but also create their own interfaces to show relevant information.

Table 2: The RESTful API provided by ProximiThings

Method	PATH	Description
GET	/config/rules	Show prox. interaction rules being monitored for execution of command
POST	/config/rules	Create a new proxemic interaction rule
POST	/config/rules/{RuleID}	Updates a proxemic interaction rule
DELETE	/config/rules/{RuleID}	Deletes a proxemic interaction rule
GET	/config/actions	Show actions and commands to be executed
POST	/config/actions	Create a new action
POST	/config/actions/{ActionID}	Updates an action or command to be executed in a device
DELETE	/config/actions/{ActionID}	Deletes an action or command to be executed in a device
GET	/dev/{EntityID}	Show proxemic information of an entity by providing its ID
POST	/dev	Creates a new device in the server and in the FIWARE platform

3.4.2 Enabling OCB notifications to ProximiThings

In order to integrate ProximiThings to OCB, we need to create a subscription on OCB to send proxemics dimension data to ProximiThings every time that devices update measurements.

Example to create a notification on OCB to send data to ProximiThings

```
                                    curl -X POST \
  http://ocb-local:1026/v2/subscriptions \
  -H 'accept: application/json' \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -H 'fiware-service: IoTHumanService' \
  -H 'fiware-servicepath: /ioths' \
  -d '{
  "description": "Obtener la orientacion de la persona",
  "subject": {
    "entities": [
      { "type": "People" },
      {"type": "GatewayIP2IR"},
      {"type": "ObjectAbstraction"},
      { "type": "Switch"}
    ],
    "condition": {
      "attributes": [
        "distance","orientation","location_id","movement"
      ]
    }
  },
  "notification": {
    "httpCustom": {
      "url": "http://proximithings-server:6253/subcriber_ocb"
    },
    "attrs": [
      "distance","orientation","location_id","movement"
    ]
  },
  "expires": "2040-01-01T14:00:00.00Z",
  "throttling": 30
}'
```

3.4.3 Using proxemics interaction info from ProximiThings

After ProximiThings detects an interaction, using proxemics dimensions measurements comparing with proxemics rules. If these rules are equal to proxemics dimensions measurements, then ProximiThings executes one of these:

1. MQTT messages to devices
2. Webhooks

MQTT Message to device: We can provide a message to a device to change its configuration or some functionality. For example, a user is near and in front of her computer (inside the personal proxemic zone). We can send a message to her computer to unlock or turn on the computer.

WebHooks: We can execute an URL via HTTP to callback another webservice or RESTful API.

3.4.4 3.4.4 Roadmap and Status

The current working prototype of ProximiThings (not a stable version) can be found at `https://github.com/faxterol/ProximiThings-Server`

In the period of this report, we made the architecture of ProximiThings and we made an evaluation of components from FIWARE Platform to use on ProximiThings.

ProximiThings is developed in three big steps:

- ➔ API REST for create, update, read and delete (CRUD operations) Rules Interactions and Actions.
- ➔ Engine for receive data from OCB and convert sensor data measurements on discrete information.
- ➔ A service to execute HTTP callbacks and send data throught MQTT.

Now, we are working on the first step and we will publish the code on the repository of GitHub. Some resources from API REST of ProximiThings has been implemented. For example:

Example of add a RuleInteraction on ProximiThings API REST

```
 curl -X POST \
 http://127.0.0.1:6253/config/rules \
 -H 'accept: application/json' \
 -H 'cache-control: no-cache' \
 -H 'content-type: application/json' \
 -H 'postman-token: bb257ce2-dc0c-c9c0-24a3-03c33be9961b' \
 -d '{
        "name" : "Water save on sink",
        "description": "Interaction rules for save water on sink when Prof. Smith is not
using it.",
        "entities" : [
                {
                        "entity_id" : "ProfSmith",
                        "proxemics_rules" : {
                                "zone" : "PERSONAL|INTIMATE",
                                "orientation" : "FRONT_OF:SinkKitchen",
                                "movement" : "*",
                                "interaction_phase" : "DIRECT",
                                "location" : "ProfSmithKitchen"
                        }
                },
                {
                        "entity_id":"SinkKitchen",
                        "proxemics_rules" : {
                                "zone" : "PERSONAL|INTIMATE",
                                "orientation" : "FRONT_OF:ProfSmith",
                                "movement" : "IDLE",
                                "interaction_phase" : "DIRECT",
                                "location" : "ProfSmithKitchen"
                        }
                }
        ],
        "commands_rules_apply" : [
                {
                        "entity_id" : "SinkKitchen",
                        "command" : "locker_faucet_open"
                }
        ],
        "commands_rules_not_apply" : [
                {
                        "entity_id" : "SinkKitchen",
                        "command" : "locker_faucet_closed"
                }
        ]
}'
```

Example of add a Proxemics Action on ProximiThings API REST

```
 curl -X POST \
 http://127.0.0.1/%7D:6253%7D/config/actions \
 -H 'accept: application/json' \
 -H 'cache-control: no-cache' \
 -H 'content-type: application/json' \
 -H 'postman-token: 93c9e03a-dab3-5990-9df7-4e9d902771d1' \
 -d '{
        "identifier" : "locker_faucet_open",
        "entity_id" : "SinkKitchen",
        "name" : "Open water flow of faucet",
        "description": "This command open water flow on faucet to save water when Prof.
Smith is in the kitchen.",
        "type_action" : "mqtt_msg",
        "action" : {
                "publish_message" : "{'\''water_flow'\'':'\''on'\''}"
        }
}'
```

The next 6 months, we will have developed the three big steps so next to made an evaluation of ProximiThings on some IoT environments.

3.5 Roadmap of Internet of Things Enablement Services

This section presents the Roadmap and the next steps to follow inside the IoT Backend Device Management Architecture Chapter. The components presented in the section are under development and will be completed in second phase of the project.

3.5.1 Data Storage module for NGSI

The objective of this component is to allow the IoT devices to store data context when the device does not have access to some internet connection to send the data online. In this context, this is a software component that will allow developers to configure a specific sensor (which contain a hardware component to save information in internal or external memory) to enable the store of information when the device is disconnected. Due that, this component is really close to hardware configuration of the sensor; in this project, we will provide the component only for Smart Spot and Cloudino technologies.

3.5.2 Automatic connection module for NGSI

The objective of this component is create a module to enable the automatic upload of gathered data when the hardware component detects an internet connection. One of the possible scenarios we can find in practice is that mobile sensor could be located in placed without any internet connection. In this scenario, it is necessary to configure the device to enable the automatic connection when an internet connection is available. Due that, this component is really close to hardware configuration of the sensor, in this project, we will provide the component only for Smart Spot and Cloudino technologies.

3.5.3 Energy saving configuration module for NGSI

The objective of this component is configure the electronic device to save energy according with the rules defined in the configuration. Each scenario has different option to indicate the conditions when a device needs to save energy, so a configuration module need to be defined. Due that, this component is really close to hardware configuration of the sensor, in this project, we will provide the component only for Smart Spot and Cloudino technologies.

4 4. DATA CONTEXT MANAGEMENT SERVICES

4.1 Introduction

During the last decades, societies have been producing tremendous amount of digitalized data, generated by different kind of devices in all sort of domains. This massive amount of generated data paired with the continuously increasing computing capabilities have enabled what are called the Big Data economies. Moreover, the use of standardized APIs in this Data processing will boost the development of service-oriented business which can complement each other building on top of these APIs.

Aware of this reality, FIWARE defines what is known as the Data Management Chapter, a group of Generic Enablers (GE) aimed at facilitating the processing and analysis of context data, all of them harmonized using the NGSI standard. The services defined in this Chapter are a natural complement to those already defined in section 2, whose purpose was restricted to the on-field sensing and data generation. The context data services on the other hand enable high performance data processing using different open source technologies. The computing requirements of these services are much higher than the one data-generator devices are able to provide, and hence the need to deploy them on a different infrastructure, typically in a cloud environment such as FIWARE Lab. This technical difference draws the line between the two mentioned FIWARE Chapters.

As defined in the official reference documentation [8], the services of the Data Management Chapter enable users to:

- Generate, subscribe for being notified about and query for context information coming from different sources.
- Model changes in context as events that can be processed to detect complex situations that will lead to generation of actions or the generation of new context information (therefore, leading to changes in context also treatable as events).
- Processing large amounts of context information in an aggregated way, using Big Data Map&Reduce techniques, in order to generate new knowledge, and to interact with the store to support off the self-bundles of data, algorithms and infrastructure. Use context data and social networks data to perform analysis.
- Process data streams (particularly, multimedia video streams) coming from different sources in order to generate new data streams as well as context information that can be further exploited.
- Manage some context information, such as location information, presence, user or terminal profile, etc., in a standard way.
- Manage and publish open data, in particular as context data in real time.
- Use existing data and media to enrich applications.

The services are offered by different projects, known as Generic Enablers, and are in dynamic growth and constantly evolving. Consequently, there is always room for enhancements and introduction of complementary features to these components. In particular, the SmartSDK plans three tasks aimed to enhance the services of the FIWARE Data/Context Management Chapter by developing a new prototype to give historical data retrieval of context data in the new NGSI v2; a way to encrypt sensitive data being sent to the Context Brokers and a library to perform NGSI operations natively from mobile devices. These tasks are presented in more details in the following subsections.

4.2 Time Series for NGSI

4.2.1 Introduction

The main goal of this activity is to develop a software component that would allow efficient management (i.e., persistence and retrieval) of historical data generated by NGSI notification sources such as Orion Context Broker.

The reader might wonder if this was not the purpose of the already existent Generic Enabler called Comet. In fact, Comet was developed to attend this goal, but in a time where the industry of timeseries databases was in its early development stages and therefore it had to make some technical design compromises in order to have some of the features the backend technologies were not providing at that time. In concrete, Comet was designed on top of MongoDB, which is a modern technology for databases but known to be not ideal for time series datasets. Comet attends its goals in the sense that allows users to keep historical track of attribute values when Orion is only able to store the latest. However, its responses are still not fully oriented to time-based indexes and it lacks advanced features of modern time series databases such as complex aggregations, adaptive resolution and ease of horizontal scalability.

The industry has seen significant progress in the timeseries databases arena in the last five years. Many projects of this kind emerged and some were open-sourced after a certain maturity was reached. Examples include Graphite, InfluxDB, CrateDB but the list goes on. With this task, we want to review these technologies and construct a component that would allow us to use NGSI queries leveraging on their features.

It is worth noting that the implementation approach described in this section has been presented to the FIWARE TSC and ideas has been exchanged with the developers of Comet to gather insight and try make the most out of this task.

The overall development of this task is tracked in the project's issue tracker (JIRA) issue named SMAR-82; https://jira.fiware.org/browse/SMAR-82.

4.2.2 Architecture

The envisioned software component to be developed can be seen as a context data consumer of NGSI notifications, which interprets the notifications and is able to respond to queries asking for historical data in different ways. The overall architecture is composed of the subcomponents show in Figure 15.

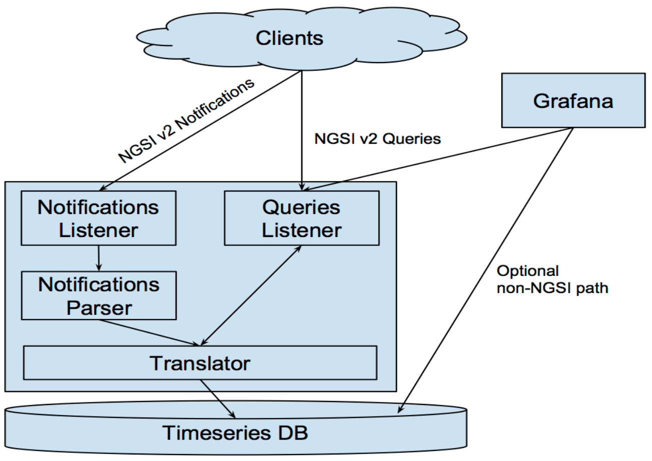


Figure 15: Architecture overview of NGSI TSDB Component

From the top of the Figure 15, as the main source of data input we have an API endpoint, which receives NGSI notifications alerts. This means, someone previously registered this endpoint to a context data generator such as Orion.

Then there is the notifications parser, whose purpose is to interpret the notification details and understand which entity or entities attributes are being updated with this notification.

Below that parser is the Translator, a component responsible for translating the NGSI semantics into the specifics of the underlying time-series database requirements. Since different database technologies are using different type systems and protocols for querying, this Translator should be easily replaceable. Once translated, notifications can be persisted in the underlying tsdb (time-series database).

An exposed API endpoint will allow for queries to be made in the NGSI v2 format to retrieve data stored in the database. These endpoints are either extensions of the official NGSI API or proposed new alternatives.

A visualization layer using state-of-the-art tools like Grafana can be built either on top of the exposed NGSI API or directly on top of the underlying database by reusing some of the existing Grafana plugins * (sources: https://grafana.com/plugins/influxdb, https://grafana.com/plugins/crate-datasource)

4.2.3 Status

As mentioned in the introduction, the first steps of this task involved developing a state-of-the-art (SOTA) analysis of the available databases with support for timeseries datasets. Special interest was given to those solutions who easily support scaling within a dockerized environment, since this criteria aligns well with the vision of dockerized solutions SmartSDK presents in deliverable D3.1. In addition, aligned with the FIWARE vision of openness, the analysis focused only on open source databases.

Then, a benchmarking testbed was developed to achieve two fundamental goals:

- Compare performance of different databases solutions
- Test the bidirectional translations to guarantee NGSI data integrity

Given the aforementioned requirements, we brought to comparison InfluxDB, CrateDB and RethinkDB. However, special care was taken to develop such benchmarking in a way that introducing a new database solution to be compared requires the least of the efforts.

Details of the initial comparison can be found in the Appendix A. Nevertheless, the overall impression is that InfluxDB performed very well, but its lack of support for geodata and additional datetime columns turned out to be a showstopper. CrateDB on the other hand, although not the fastest, proved to be very flexible for the amount of requirements we have. RethinkDB seems to be a compromise among the two ends, but with no clear advantages over CrateDB. Thus, the work on this task will move on with CrateDB for now.

4.2.4 Roadmap

The development of this epic was broken down into the following tasks, which at the same time, are setting the trace for the roadmap of this activity. The first three were already finished by the time of writing this report, the rest are yet to be done within the upcoming sprints.

- Perform a SOTA analysis of the modern available time-series databases.
- Develop a flexible benchmarking testbed.
- Develop and validate first version of custom translators.
- Scale out: test performance on distributed environment with bigger datasets.
- Develop the first version of the notifications parser.
- Develop a microservice which:

- Receives NGSI v2 notifications and feeds the parser.
- Uses the parsed entities to feed the translator.
- Responds to queries interacting with the translator.

➔ Extend the NGSI API accordingly using well-known API manages (swaggers). This is to hook those extended API endpoints with the previously mentioned microservice.

4.3 NGSI Encryption Layer

As more sensitive data is shared and stored, particularly within the Orion Context Broker, there is a need to encrypt such data when specific attributes are related to sensitive information. One drawback of encrypting data, is that it can be selectively shared only at a coarse-grained level (i.e., giving another party your private key). This is an opportunity to allow, in a selective way, the protection of specific sensitive data using an encryption key and maintaining the visibility of the rest of the attributes.

In this way, the main goal of this activity is to develop a software component allowing NGSI data encryption on the related attributes in a partial or a total manner.

This fine-grained sharing of NGSI attribute encryption (NGSI-AE) could be seen as cipher-texts that will be labeled with sets of attributes and private keys are associated with access structures that control which ciphertexts a user is able to decrypt.

In this NGSI-AE, a user's key and cipher-texts are labeled with sets of descriptive attributes and a particular key can decrypt a particular ciphertext only if there is a match between the attributes of the cipher-text and the user's key.

4.3.1 Architecture

The proposed software component to be developed is an attribute encryption oriented for the NGSI specification. The overall architecture is composed of the subcomponents shown in Figure 16.

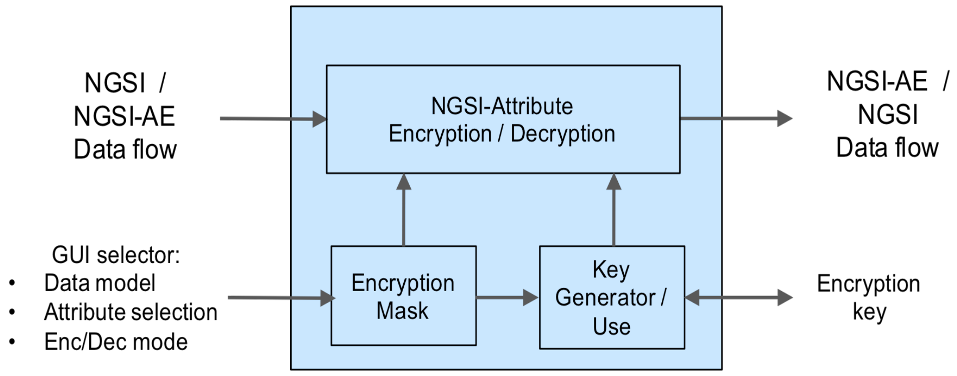
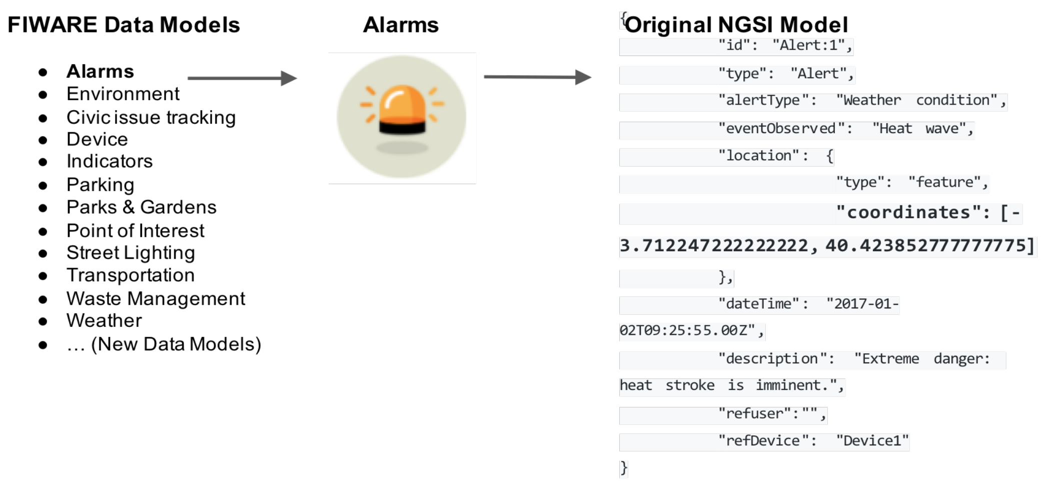


Figure 16: Architecture overview of NGSI Attribute Encryption module.

The GUI selector shows a catalogue of data models according to the application and to receive the correct NGSI data flow. In the Figure 17, there is an example of NGSI data models using the Alarms' data model.

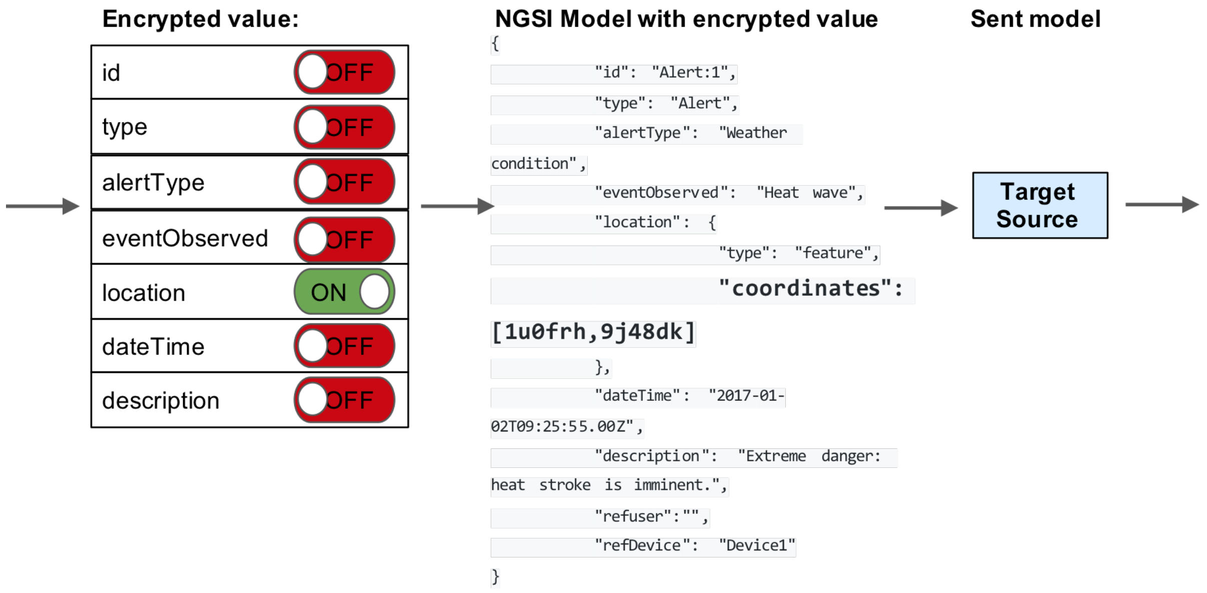


```
{
            "id": "Alert:1",
            "type": "Alert",
            "alertType": "Weather condition",
            "eventObserved": "Heat wave",
            "location": {
                        "type": "feature",
                        "coordinates": [-
3.712247222222222, 40.423852777777775]
            },
            "dateTime": "2017-01-
02T09:25:55.00Z",
            "description": "Extreme danger:
heat stroke is imminent.",
            "refuser":"",
            "refDevice": "Device1"
}
```

Figure 17: Example of selection of a data model without encryption.

The encryption mask is related to the data model, previously selected by the user, and establishes what attributes must be encrypted. The user configures the encryption of these attributes using the GUI selector by means of a panel. The Key Generator / Use produces the encryption key, once the data model and the encryption mask were configured. The NGSI Attribute Encryption / Decryption oversees the use of an encryption key for the attribute encryption for the selected data model.

Figure 18 shows an example in which the alerts' data model will use a mask in which the location will be encrypted.



Encrypted value:	
id	OFF
type	OFF
alertType	OFF
eventObserved	OFF
location	ON
dateTime	OFF
description	OFF

```
{
            "id": "Alert:1",
            "type": "Alert",
            "alertType": "Weather
condition",
            "eventObserved": "Heat wave",
            "location": {
                        "type": "feature",
                        "coordinates":
[1u0frh,9j48dk]
            },
            "dateTime": "2017-01-
02T09:25:55.00Z",
            "description": "Extreme danger:
heat stroke is imminent.",
            "refuser":"",
            "refDevice": "Device1"
}
```

Figure 18: Example of selection of attribute to be encrypted. In this case, the attribute location will be encrypted in the NGSI frame.

4.3.2 Status and Roadmap

To develop this software component, the first step will be focused in developing a state-of-the-art (SOTA) analysis of the available data encryption algorithms, specific for attribute encryption. The main idea is to have this encryption within the same component, avoiding the need of an external data encryption key generator and management.

With this main requirement established, we can identify the following tasks, tracing the roadmap as well:

- Develop a SOTA for attribute encryption algorithms
- Compare performance of different attribute encryption algorithms
- Develop a first version of the NGSI-AE
- Test the encryption / decryption to guarantee NGSI data integrity

4.4 SDK Library for NGSI

The objective of this component will be generate a generic library (JavaScript library) to connect several kind of smartphones to Orion Context Broker via a NGSI RESTful interface with the objective of sending context data for mobile contexts. In this context, the mobile devices will play a Context Producer role. This library will enable developers to update context information that can be geolocated. The objective is that the library can be properly used in several mobile platforms such as Android, Windows or IOS. The proposed approach is the developers create specific interfaces in the mobile devices (according with the different application scenarios of the project) and use the SDK library as the basis to communicate the data to the Orion Context Broker.

In order to create the library in an incrementally manner, a first version of the library has been developed for the Android operating system. This first version permit mobile devices with this operating system to send several context data to the FIWARE Cloud via NGSI. This library is fully functional and it is currently used in the Smart City scenario to indicate the location of mobile pollution monitoring units.

4.4.1 Architecture of the system

The envisioned software component to be developed can be seen as a translator of the data captured by the smartphone sensors to the NGSI specification. Several steps need to be accomplished to take the different data types from the smartphone and transform the data in the NGSI required specification. As stated before, at the present time, the library only consider the connection to Android devices. The overall architecture is composed of the subcomponents shown in Figure 19.

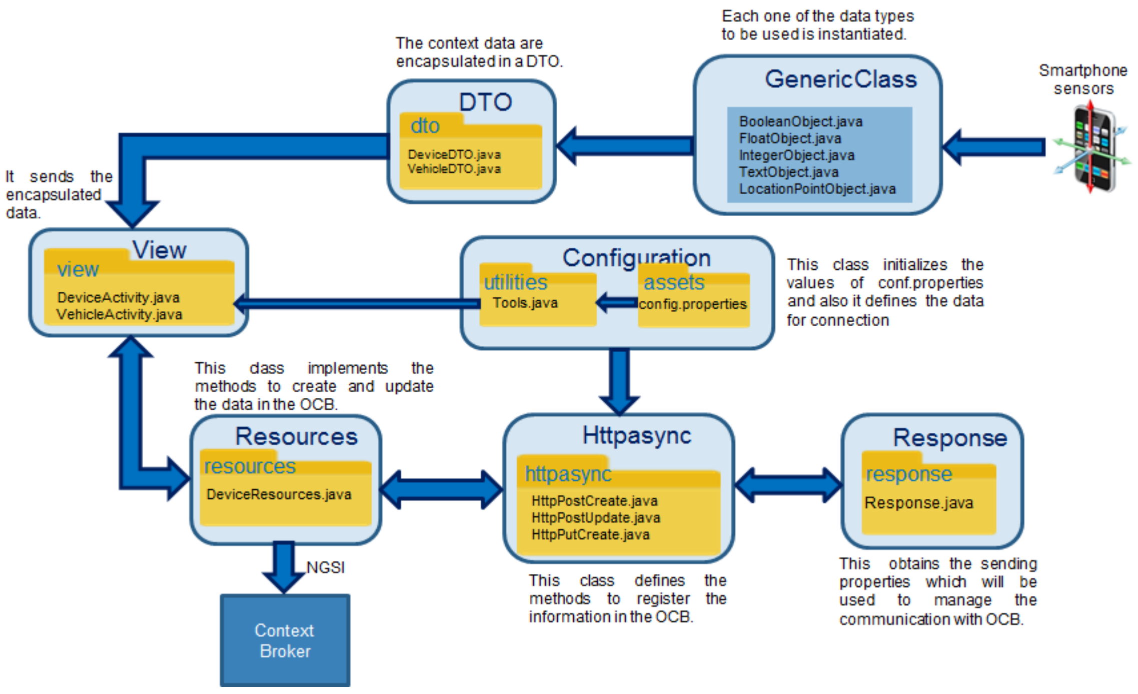
In Generic, class permits the instantiation of each one of the data types that need to be used in the mobile application. The kind of data types completely depends of the requirements of the application.

One the data types were instantiated, a Data Transfer Object is created for each one of the real world objects of the application domain. The class need to define each object in its corresponding attributes.

In Configuration class, the connection parameters to the Context Broker are generated and configured. The View class contain the information about all views that are required by the mobile application.

Response class manages the information that is received from the Context Broker as a response of a request. This information is used to create messages for the user regarding the correct/incorrect updating of data in the Context Broker.

In Resource class, the DTO are converted to the NGSI standard specification to be sent to the Context Broker.



Each one of the data types to be used is instantiated.
The context data are encapsulated in a DTO.
Smartphone sensors
GenericClass
BooleanObject.java
FloatObject.java
IntegerObject.java
TextObject.java
LocationPointObject.java
DTO
dto
DeviceDTO.java
VehicleDTO.java
It sends the encapsulated data.
View
view
DeviceActivity.java
VehicleActivity.java
Configuration
utilities
Tools.java
assets
config.properties
This class initializes the values of conf.properties and also it defines the data for connection
This class implements the methods to create and update the data in the OCB.
Resources
resources
DeviceResources.java
Httpasync
httpasync
HttpPostCreate.java
HttpPostUpdate.java
HttpPutCreate.java
Response
response
Response.java
NGSI
Context Broker
This class defines the methods to register the information in the OCB.
This obtains the sending properties which will be used to manage the communication with OCB.

Figure 19: Architecture of current NGSI Library

4.4.2 Status and Roadmap

The development of this component has been broken in following tasks. First five tasks are fully functional and the rest of task will be completed within the upcoming sprints.

- Read the smartphone data in real time.
- Create data types for each one of the data obtained from smartphone.
- Create classes for configure the connection to the Context Broker.
- Send data updates to the Context Broker via NGSI.
- Obtain response about the update from the Context Broker.
- Review the current libraries for connection to Context Broker.
- Extend current version of library from Git.
- Create a set of test cases for library.

5 CONCLUSION

This document presents the current status of the Internet of Things Enablement and Data / Context Management services developed in SmartSDK. The components developed in this chapter are orthogonal to application domains; therefore, these can be reused in the project scenarios: smart cities, smart health and smart security.

Hardware and software components are the main contributions of this chapter that represent an progress in current FIWARE components to manage the context data produced by sensors and software applications and also, it represents an advance for the FIWARE components that permit the connection of electronic devices to the FIWARE Cloud.

REFERENCES

[1] SmartSDK Consortium. Description of Action. July 2016. SmartSDK project is co-funded by the EU's Horizon2020 programme under agreement number 723174 - ©2016 EC and by CONACYT agreement 737373.

[2] Overview of the Internet of things (2012, June 6). Retrieved 2017, May from http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=y.2060

[3] Internet of Things: Science fiction or Business fact? (2014). Retrieved 2017, May from https://hbr.org/resources/pdfs/comm/verizon/18980_HBR_Verizon_IoT_Nov_14.pdf

[4] Internet of Things - Converging technologies for Smart Environment and Integrated Ecosystems (2013). Retrieved 2017, May from http://www.internet-of-things-research.eu/pdf/Converging_Technologies_for_Smart_Environments_and_Integrated_Ecosystems_IERC_Book_Open_Access_2013.pdf

[5] What Makes Up the Internet of Things? (2015, Marc 6). Retrieved 2017, May from https://www.computer.org/web/sensing-iot/content?g=53926943&type=article&urlTitle=what-are-the-components-of-iot-

[6] From the Internet of Computers to the Internet of Things. Retrieved 2017, May from http://www.vs.inf.ethz.ch/publ/papers/Internet-of-things.pdf

[7] Physical Web overview and official website https://google.github.io/physical-web/

[8] FIWARE Data/Context Management. Retrieved 2017, May from https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Data/Context_Management

[9] FIWARE-NGSI v2 Specification. Retrieved 2017, May from http://telefonicaid.github.io/fiware-orion/api/v2/stable/

[10] InfluxDB Version 1.2 Documentation. Retrieved 2017, May from https://docs.influxdata.com/influxdb/v1.2/

[11] WHAT IS CRATEDB? Retrieved 2017, May from https://crate.io/overview/

[12] RethinkDB. Retrieved 2017, May from https://www.rethinkdb.com/

[13] RIAK TS. Retrieved 2017, May from http://basho.com/products/riak-ts/

[14] Documentation for OpenTSDB 2.3. Retrieved 2017, May from http://opentsdb.net/docs/build/html/index.html

[15] ngsi-timeseries-api. Retrieved 2017, May from https://github.com/smartsdk/ngsi-timeseries-api/tree/benchmark

APPENDIX A – TIME-SERIES FOR NGSI INITIAL STUDY

A.1 Introduction

This appendix complements section X by giving more details on the first tasks of the "**Time Series for NGSI"**. In concrete, the first steps of this epic consisted on investigating the state-of-the-art in different modern databases used for timeseries data. Timeseries data refers to datasets which are always indexed by time, or in other words, measurements of some type that happened at certain different points in time. As mentioned earlier, the selected candidates for the testing were InfluxDB, CrateDB and RethinkDB [10], [11], [12]. Other solutions unfortunately not explored in the testing due to time constraints, but definitively worth considering in a future opportunity are Riak-ts [13] and TSDB [14].

Complementing this SOTA, a coding testbed has been developed to try the different alternatives. The goals of the testbed are twofold. On the one hand, it helps validate the translation of the basic NGSI data types to the specifics of each database solution. Correctness checks are required due to the presence of data types conversions, which are common in the storage process. On the other hand, the testbed allows us to define a set of isolated and automated tests to basic database operations so that comparable metrics can be extracted out of their execution.

A.2 The procedure

The syntaxes and protocols used to manipulate data differs from one database solution to the other. Thus, to have an harmonized testbed, we decided to test the solutions using their Python3 client drivers. Python was chosen not only because it was one of the few languages in which drivers were available for all the tested databases; but also because of its benefits for fast prototyping and flexibility for changes. Also, for a fair comparison, only officially supported drivers were considered.

The source code and work in progress of this epic is being kept in the SmartSDK's github repository called ngsi-timeseries-api [15]. In all the cases, the databases are run locally, each having one table on a single instance on its own Docker container. This helped keep complexity low at this stage of testing, particularly because the scalability of each solution is different and to be evaluated at a later point, as explained in the roadmap of section X.

The following actions were measured for each candidate database. In parenthesis, the reference codename for the figures):

- ➔ An insert of a single NGSI notification, i.e all attributes of 1 entity. (Insert 1)
- ➔ An insert of 1000 NGSI notifications, i.e a batch of 1000 updates. (Insert N)[2]
- ➔ A query for 1 attribute of 1 entity (Query 1A1E)
- ➔ A query for all attributes of 1 entity (Query NA1E)
- ➔ A query for 1 attribute of all entities (Query 1ANE)
- ➔ A query for all attributes of all entities (Query NANE)
- ➔ An aggregation (average) of 1 attribute for one entity
- ➔ An aggregation (average) of 1 attribute for all entities

One measure the benchmark did not include and might be worth taking into account when testing more complex deployments is the time of data availability. This means, the time it takes for a piece of

[2] Note, after this point, the number of total updates is raised to 100000 (100 entities, 1000 updates each).

data to be retrievable by a query after it was inserted. Due to internal implementation details such as caching or replication, this is not always immediate.

The figures below show the results obtained for the metrics mentioned earlier. It is worth noting these timings can be significantly improved by making the translation overhead more efficient, but since all solutions are equally affected by the translation overhead, the relative comparison is still valid.

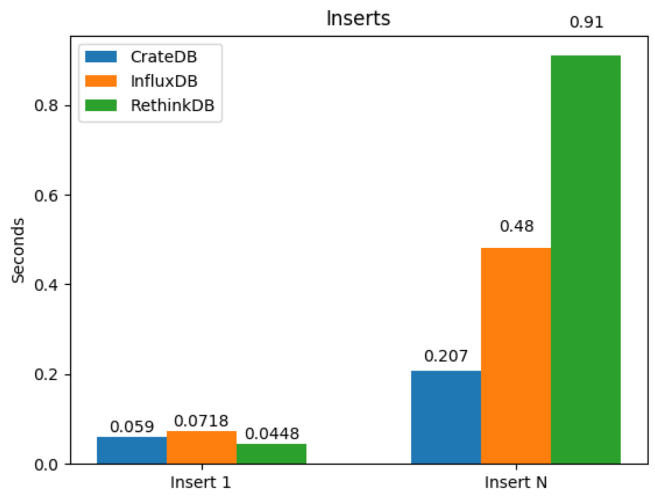


Figure 20: Insert times (in seconds) for 1 and N=1000 updates

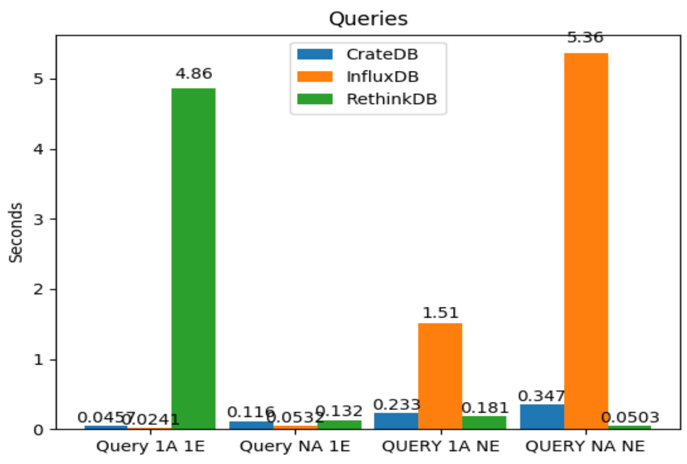


Figure 21: Querying times (in seconds) for all the defined metrics

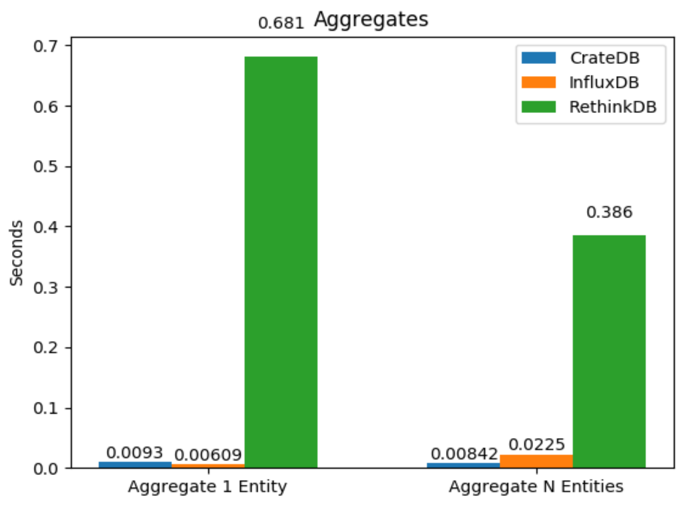


Figure 22: Aggregation (mean) times (in seconds) for attributes of 1 and N=1000 entities

The benchmark was executed on a MacbookPro (early 2015) with a 2.7 GHz Intel Core i5 processor and 8 GB 1867 MHz DDR3 RAM running macOS Sierra v10.12.4 (16E195). The following table shows the database versions used in the comparison, which was, in all cases, the latest available at that time.

Table 3: Tested database versions

	InfluxDB	CrateDB	RethinkDB
Version	1.2.2	1.0.5	2.3.5
Official Docker Image	61a53f6a13f2	ae465cbdc754	c5ed876750b4

A.3 Conclusions

The main findings extracted from the analysis of both the obtained metrics and the implementation experience with each database can be summarized as follows.

The first observation is regarding the lack of support from InfluxDB to having either geodata storage or multiple columns for storing datetimes. Even though it proved to have good insert and query times for single attributes, this lack of support for geodata and extra datetime columns ended up being a showstopper. Moreover, it is clear from Figure 22, that InfluxDB is not well at responding to queries of the type: Give me all attributes of Entity X. This is because its index is "measurements" centric, which are equivalent to NGSI's attributes. Hence, it works better with queries like: give me all temperatures for Entities X.

RethinkDB on the other hand does support geodata attributes, although not as good as CrateDB does [15]. For example, ReQL geometry objects are not pure GeoJSON objects so further conversions are required, and not all geometry types are supported.