UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola
di Ingegneria

Corso di Laurea in
Ingegneria Informatica

Analisi e sviluppo di una soluzione scalabile per IoT Broker che fornisce una dashboard per dispositivi IoT

Analysis and development of a scalable solution for IoT Broker providing a dashboard for IoT devices

Candidato:
Cosimo Michelagnoli

Relatore:
Prof. Paolo Nesi
Corelatore:
Prof. Pierfrancesco Bellini

Anno Accademico 2019-2020

Contents

List of Figures

Introduction

The Internet of Things marks the beginning of a new technological era that will have a strong impact on our daily lives. The Internet of Things is born from the idea of bringing the objects of our daily experience into the digital world. In the vision of the Internet of Things, objects create a pervasive and interconnected system using multiple communication technologies. The purpose of this type of solution is basically to monitor, control and transfer information and then carry out consequent actions.

There are different Areas Where IoT is growing its roots. In this thesis, we will analyze the paradigm Internet of Things applied to Smart City.

Figure 1: Smart-City

Among the countless fields of application of the IoT, the importance of an integrated approach to urban development is at the birth of the concept of Smart city. This concept encompasses the idea of an urban area that uses different types of sensors and devices for data collection, in order to support efficient and effective management of resources.

It is possible, by simplifying, to find a multi layer structure on which is based the technological architecture of a Smart City. These layers are:

1. Hardware
2. Middleware
3. Application

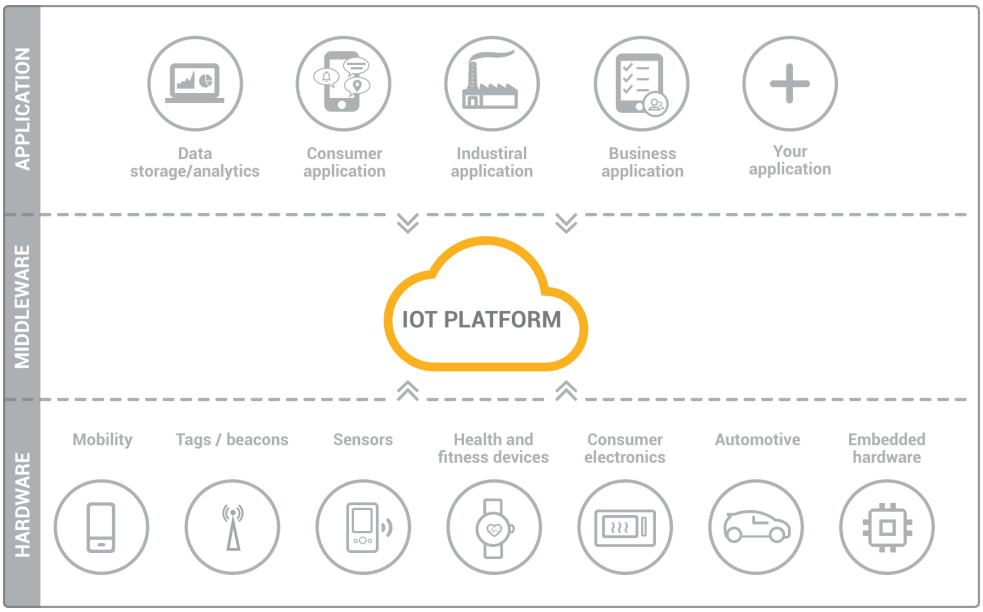

Figure 2: IoT Platform [1]

As for the former, it consists of a network of physical devices capable of connecting to a network and exchanging data. Any "thing" or "smart device" is a gadget with built-in electronics and software that can act as a sensor or actuator.

Considering the second layer, to mask the heterogeneity and distribution problems that we deal with when we interact with devices a software platform defined as middleware is required.

The requirement of this platform is to manage a large number of IoT devices, intelligently store and process gigantic data generated to build intelligent applications. Finally the applications are built on various IoT cloud platforms to provide services to some real life scenarios. For example the IoT Dashboard which is nothing more than a web application that allows the visualization of the data generated by these "smart devices". Turning our attention to the middleware level, we looked at the structure of some solutions already present, finding some weaknesses in the use of a centralized knowledge base structure. In fact, thinking about an iot network in a typical urban scenario there are some challenges that need to be faced such as: 1) the significant fragmentation deriving from the multiple IoT architectures and associated middleware; 2) a huge amount of data to be processed provided in real-time by the IoT devices deployed in the smart systems; 3) to satisfy a variety of data demands for diverse smart city applications. Modern smart city projects

are evolving to a distributed database system, avoiding these critical issues and distributing requests to multiple nodes of the network. In particular, our intention was to allocate memory to specific nodes of the network called "Broker" to redirect some requests directly to them instead of overloading the main database. This thesis work presents the analysis of some FIWARE's components and the development of a web application integrable has a service of the platform Snap4City. Where *FIWARE is a curated framework of open source platform components to accelerate the development of smart solutions.*[2]

Thesis's structure

Chapter 1

The first chapter introduces the SmartCity paradigm and the scenarios to which it is applied. The importance of the Internet of Things which revolutionizes the management of large urban centers is presented.

Chapter 2

The second chapter presents the architecture of the IoT solution built around Orion Context Broker. In this chapter we see in detail the characteristics of the components that have been used.

Chapter 3

The second chapter introduces the use of the Node-Red language for the set up phase. Attention is also paid to the creation of IoT devices and to the simulations of their measurements.

Chapter 4

The fourth chapter explains the protocol used by Orion Context Broker and the other components of FIWARE. It shows the potential of the API to find context data. Finally, it also shows the protocol used by the devices for communicating the measurements.

Chapter 5

This chapter presents a general idea of the project. It shows how the strategy to be adopted to achieve the goal of the thesis was chosen.

Chapter 6

The fifth chapter explains the development of the web application in java that allowed communication between Snap4City and Orion Context Broker.

Chapter 7

In this chapter we show how the operation of the application and its functionality was verified.

Chapter 1

Smart City

A smart city is a framework, predominantly composed of Information and Communication Technologies (ICT), to develop, deploy, and promote sustainable development practices to address growing urbanization challenges[3]. That's mean that today the management and development of cities do not depend only on material infrastructures, but also on the technical development of digital communication and on social participation. Therefore, a smart city is an urban environment where all vital and neuralgic processes are reinterpreted to improve the quality of life, social opportunities, well-being and economic development.

1.1 Introduction

Cities are increasingly congested and therefore need new management and governance models. These new models can enhance public transport and introduce new types of transport models that affect social innovation and raise awareness among citizens. A smart city is able to exploit all modern technologies to obtain energy efficiency and environmental sustainability, home technologies and building automation. A smart city is a lively and dynamic city that values its cultural heritage and returns it to the network as a common good for citizens and visitors.

1.2 Power of IoT interconnection

First, a smart city collects information about itself through sensors, other devices and existing systems. Next, it communicates that data using wired or wireless networks. Third, it analyses that data to understand what's happening now and what's likely to happen next [4]. Through this data collection, sharing and analysis, a Smart City can improve on multiple aspects. However, this is all possible thanks to the Internet of things paradigm. In fact, it's allow machine-to-machine communication between smart devices called IoT devices. IoT devices gather data and are commanded by the City's brain, software and applications running in a Municipal Cloud and the City's Command and Control Centre. IoT devices can connect using the same technologies available to users. In other cases, however, other technologies are more appropriate. An IoT gateway may be required to connect such devices

to IP-based infrastructure. The City's network is the spinal cord linking IoT devices with the software, applications and people that analyse their data and control them.

1.3 Data Visualization

The expansion of big data and the evolution of Internet of Things (IoT) technologies have played an important role in the feasibility of smart city initiatives. Big data is basically huge amounts of data that can be analyzed by businesses to make appropriate strategic moves and business decisions. Big data offer the potential for cities to obtain valuable insights from a large amount of data collected through various sources, and the IoT allows the integration of sensors, radio frequency identification, and Bluetooth in the real-world environment using highly networked services. The combination of the IoT and big data is an unexplored research area that has brought new and interesting challenges for achieving the goal of future smart cities [5]. The role of data visualization is to create an easier way to interpret the enormous amount of data available thanks to a smart city. Data from multiple source are send to be store into databases. Such data can play an important role in enabling smart governance. To easily manage the data's power, smart city display various dashboards. Dashboards display KPIs and metrics using visualizations like tables, line charts, bar charts and gauges. Data dashboards simplify the comprehension of complex data coming from the IoT platform. Data visualization goal is to provide an easy interpretation of data, that you do not have to be an analyst to understand and make a good use of dashboard.

Chapter 2

Architecture

Smart cities are complex and very large distributed systems. In addition, they also have their own unique challenges to provide and support high scalability, efficiency, safety, real-time responses, and smartness (intelligence) requirements. Designing and building applications meeting all these challenges is extremely complex. As central data storage and management system, we make use of FIWARE, which is a framework of open-source platform components created to facilitate the development of smart solutions within various application domains [6].

2.1 IoT Middleware

Middleware technologies have become a necessary part of any distributed environment [7]. Middleware refers to the software layer that resides between the physical layer components(IoT devices) and the upper layer distributed enterprise applications interacting via the network. The first role of middleware is thus to abstract away operating system differences. The aim is to make code portable between different types of IoT devices and their operating system. The second role of middleware is to provide functionality for formatting and sending messages for specific application domains, or for specific types of service. For example, providing formatting for particular network protocols. In summary, middleware is software that connects software components in some way, and is therefore the appropriate technology for large-scale distributed applications.

2.1.1 Contex Broker

The gradual expansion of iot technology is significantly increasing the number of iot devices. This means that for every new iot device there are thousands of data delivered to the network daily. New challenges now concern the ability to analyze and extract useful information for the final user. This scenario opens new opportunities for the introduction of context-awareness features, as it is possible to store context information linked to the data obtained by sensors to make their interpretation simpler and useful for the user. Being "Smart" requires first being "Aware", therefore it has become impossible to develop smart applications without gathering and managing context information. A challenge then is how

to manage all the context information generated by all those different sensor along the time. The Context Broker is the key component of the architecture. Its role is to manage context data flows and as an interface between other architectural components. Basically, Contex Broker interacts with context producer, as sensors, and context consumers, as applications. Among different types of Broker we decided to select Orion Contex Broker because it was already implemented to manage IoT devices in industrial solutions.

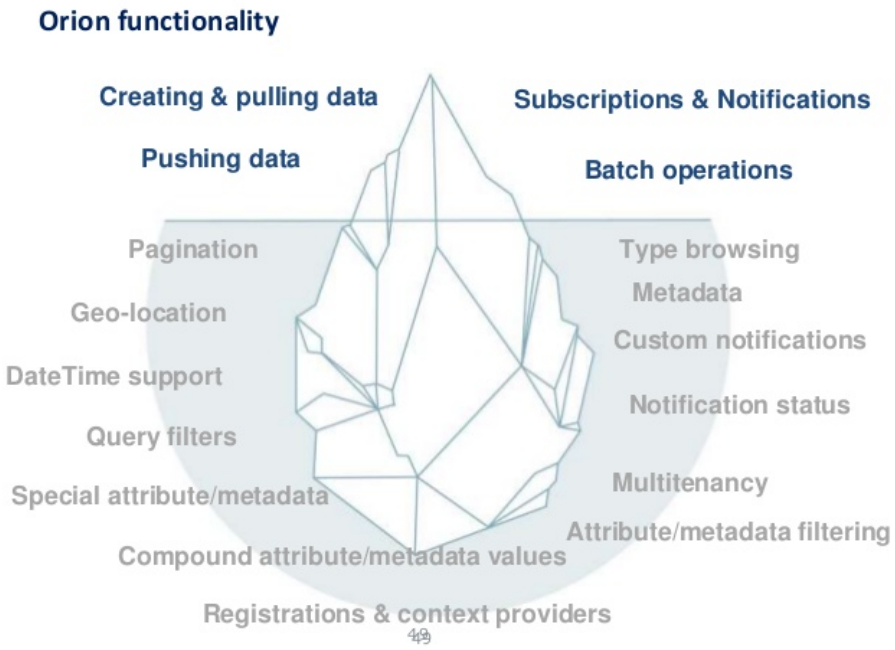

Figure 2.1: Orion functionality

2.1.1.1 Orion functionality

Orion is a C++ implementation of the NGSIv2 REST API binding developed as a part of the FIWARE platform. Representational State Transfer (REST) application programming interfaces (API) using the standardized Next Generation Service Interface (NGSI) is a simple yet powerful standard API for managing Context information complying with the requirements of a smart enabled system. The FIWARE NGSI API defines: 1) a **data model**; 2) a **context data interface**; 3) a **context availability**. The main elements in the NGSIv2 data model are context entities, attributes and metadata, as shown in the figure below [8] .

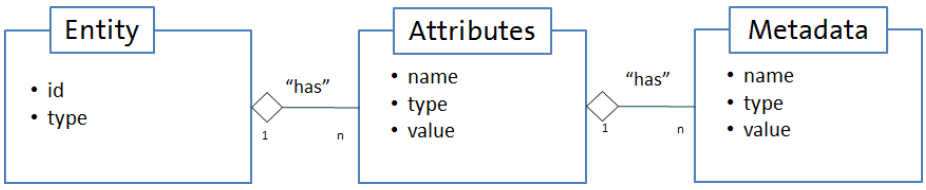

Figure 2.2: The main elements in the NGSI data model

An **entity** represents a sensor or in general a thing, each entity has an entity id and an entity type. Each entity is uniquely identified by the combination of its id and type. Context attributes are properties of context entities, each attribute have a name, an attribute type, an attribute value and metadata. The Orion Context Broker provide an NGSI interface allowing clients to do several operations [9]:

- Query context information. Orion stores context information produced by sensor, so queries are resolved based on that information. e.g. *entity* (a thermometer) *attribute* (the temperature)
- Update context information. Orion can updates entity and attributes values
- Being notified when changes on context information take place or with a given frequency
- Register context provider applications

An important feature is the ability to provide geospatial properties to entities through regular context attributes. The provision of geospatial properties enables the resolution of geographical queries, as will be shown later.

2.1.2 QuantumLeap

It is important to keep in mind that Orion Context Broker is designed to persists only the last pushed value of each attribute. FIWARE provides several solutions that satisfy the need to persist context historical data (e.g. draco, Cygnus, sth comet, quantum leap). Between this components we decided to integrate the **quantum leap** component. QuantumLeap provides NGSIv2 REST API for managing the historic data stored in the database and, thus, adds an efficient time series functionality to the platform. QuantumLeap converts NGSI semi-structured data into tabular format and stores it in a time-series database, associating each database record with a time index and, if present in the NGSI data, a location on Earth. The REST API specification, dubbed NGSI-TSDB, which QuantumLeap implements has been defined with the goal of providing a database-agnostic REST interface for the storage, querying and retrieval of NGSI entity time series that could be as close as possible to the NGSI specification itself [10]. This component must be subscribed to a specific context type managed by Orion. Once QuantumLeap has been subscribed, any modification of the attribute values of any entity belonging to the subscribed type will be persistent in the configured time series database. An important feature of QuantumLeap is the possibility to

transparently support multiple database back ends. For our solution we have used CrateDB because it is easily to scale.

2.1.3 IoT Agent

For seamlessly connecting, managing and gathering data of IoT devices, FIWARE offers a set of IoT Agents that translate IoT specific protocols and message formats, such as Ultralight 2.0, JSON, etc. into the platform-internal NGSI format [11]. This Internet of Things Agent is a bridge that mediates between IoT specific protocol to NGSI context information protocol (**Figure** 2.3). In effect, this brings a standard interface to all IoT

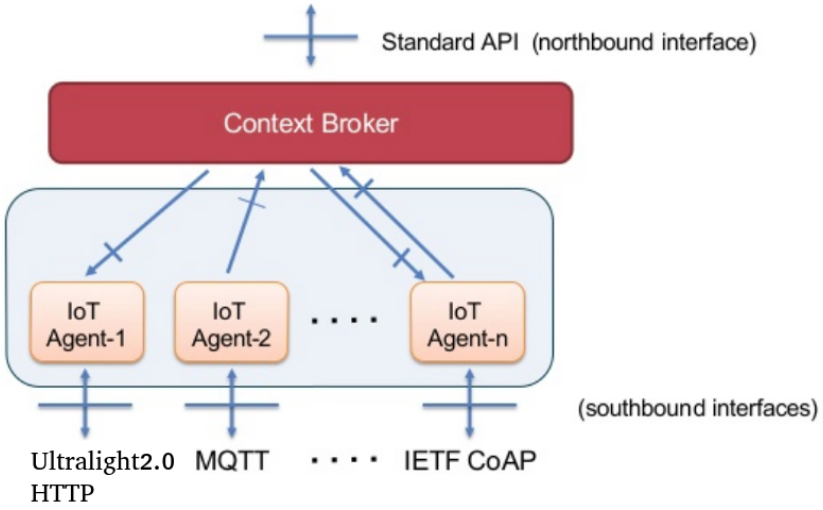

Figure 2.3: Internet of Things Agent

interactions at the context information management level. Each group of IoT devices are able to use their own proprietary protocols and disparate transport mechanisms under the hood whilst the associated IoT Agent offers a facade pattern to handle this complexity [12]. The iot-agent container relies on the presence of Orion Context Broker and uses a MongoDB database to hold device information such as device URLs and keys. Container is listening on port 7896, is exposed to receive ultra-light measurements over HTTP from IoT devices.

2.1.4 Database components

There are two important components that are involved in the data storage role. These components provide the ability to store context data collected through the IoT Agent.

2.1.4.1 MongoDB

MongoDB is the NoSQL database used to store context data. Orion uses mongoDB as the data storage, to keep persistence of the context data information such as data entities, subscriptions and registrations. The MongoDB database is used in the same way by the IoT Agent component to hold device information such as URLs and device keys, which are

needed to retrieve the right entity. Whenever a device is registered at an IoT Agent, the agent automatically connects the device and the corresponding data with a specified content in Orion and stores the configuration in the MongoDB.

2.1.4.2 CrateDB

CrateDB is a distributed SQL database management system that integrates a fully searchable document-oriented data store. It is open-source, written in Java, based on a shared-nothing architecture, and is designed for high scalability [13]. This database is used by the quantum leap component as a data sink to hold time-based historical context data.

2.2 FIWARE IoT platform

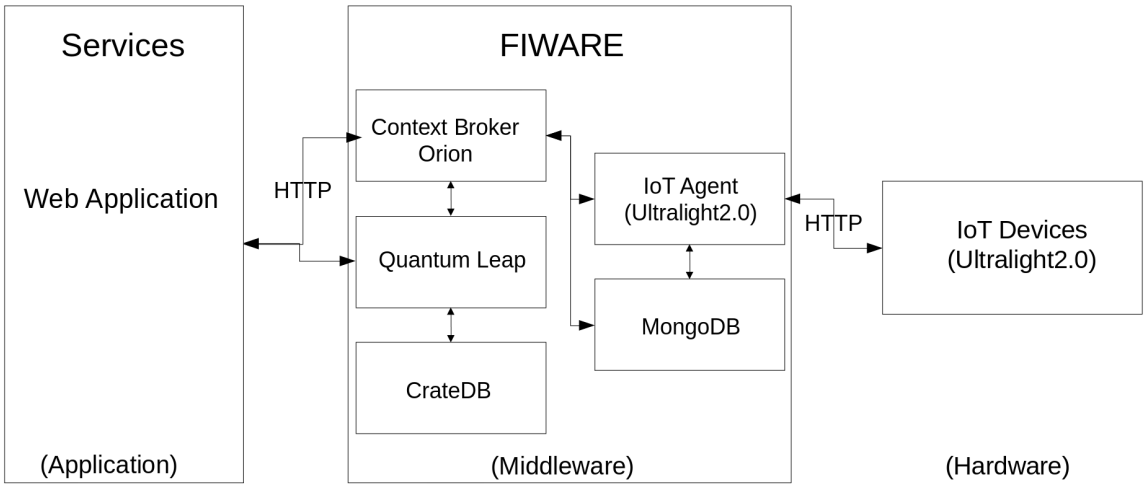

Figure 2.4: FIWARE IoT platform

As shown in figure 2.4, this is the architecture of the distributed IoT platform prototype. The Orion context broker is the central component of the FIWARE IoT platform. It provides update, query, registration and subscription functionality via NGSI API for managing the context information in the platform. This platform has the role of providing data at the application level. The data is produced by the sensors in the hardware layer. In this solution, no physical sensors were used, we simulated their existence by generating a flow of fake measurements using NodeRed. The Hardware layer will be explained in the next chapter.

Chapter 3

Internet of Things using Node-Red

In this chapter will be shown how we made use of Node-Red to simulate the Hardware layer. Node-RED is a flow-based development tool for visual programming developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things [14]. This tool is based on JavaScript and has become very important for the IoT world. Node-RED is a tool born with the idea of managing the world of IoT through the paradigm of data flows. Data packets travel in the flows, preferably via the MQTT protocol, and in JSON format, which jump from a node (the individual objects of the menu on the left) performing actions, calculations, analyzes and events via javascript to websites, dashboards, sensors and every device or service that we manage to interface, all with great simplification and ease.

3.1 Connecting FIWARE platform

To exploit the hardware potential of cloud technology, we embed our platform prototype into a container virtualization using images provided by FIWARE [15]. All the components discussed in the previous chapter will run using Docker Compose. Docker is a container technology that allows you to isolate different components in their respective environments. Docker Compose is a tool for defining and running multi-container Docker applications. All Docker applications can be initialised from the command-line using the docker-compose command. Most of the connections between containers are configured by the YAML file.

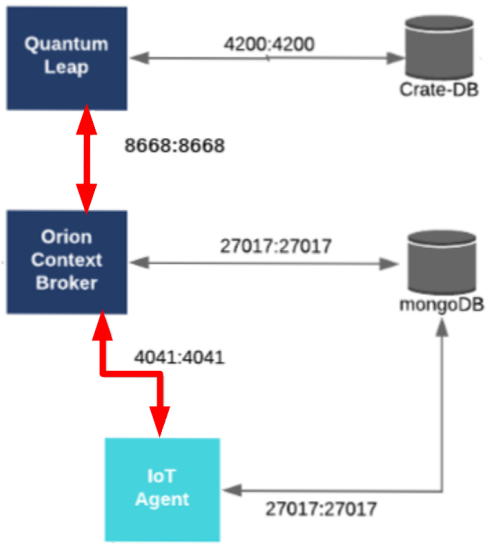

Figure 3.1: Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run

For those marked in red on figure 3.1, is required an HTTP POST request. The connection between Quantum leap, Orion and IoT Agent is made through Node-Red flows. Quantum leap, to persistently store time series data, requires subscription to Orion Context Broker. Subscription must be performed on specified content (**Figure** 3.2), then Quantum leap will automatically store updated data persistently in the connected high-performance CrateDB database.

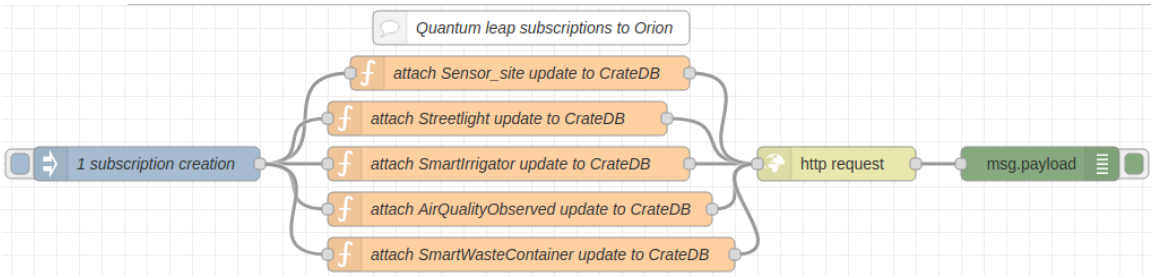

Figure 3.2: Quantum Leap subscriptions for different entity types

This flow contributes to the initialization of the environment, it represents the subscription of the qualitative leap to five different types of entities. This means that Quantumleap can specify under what conditions it wants to be notified about entity changes. Below is the contents of one of the stream nodes (*See code* 3.1). As we can see in section "entities", the value of "idPattern" in the request body ensures that QuantumLeap will be informed of all SensorSite Sensor data changes. Quantumleap, requires Orion to be notified whenever

any of the attributes specified in "condition" undergoes a change. Notification will be send to the URL specified into "notification" section, that must match the exposed port. Quantumleap is also able to select which attributes it wants to receive in orion notifications, this attributes has to be specified inside "attrs" section. The metadata attribute ensures that the time_index column within the CrateDB database will match the data found within the MongoDB database used by the Orion Context Broker rather than using the creation time of the record within the CrateDB itself [16].

```
msg.headers = {
    'Content-Type' : 'application/json',
    'fiware-service' : 'openiot',
    'fiware-servicepath' : '/'
};
msg.payload = {
 "description":"Notify QuantumLeap on any changes on any
   Sensor_site",
 "subject": {
    "entities": [
      {
        "idPattern": "SensorSite.*"
      }
    ],
    "condition": {
      "attrs": [
        "averagespeed",
        "vehicleFlow",
        "anomalylevel",
        "concentration"
      ]
    }
  },
  "notification": {
    "http": {
      "url": "http://quantumleap:8668/v2/notify"
    },
    "attrs": [
      "averagespeed","vehicleFlow","anomalylevel","
   concentration"
    ],
    "metadata": ["dateCreated", "dateModified"]
  },
  "throttling": 1
}
```

```
return msg;
```

Listing 3.1: POST request to the /v2/subscription endpoint of the Orion Context Broker

At this point, only the connection between the IoT Agent and Orion is missing. The IoT agent will initially not know which URL the context broker is responding to. So the first step is to provide the correct URL of our context broker and provide an authentication key which will be required with each measurement. This example provisions an anonymous group of

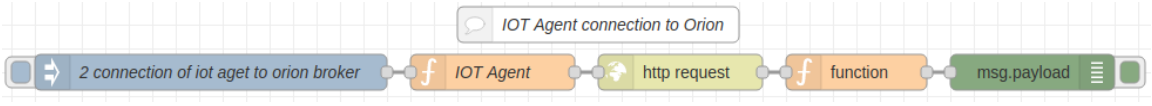

Figure 3.3: IoT Agent connection to Orion Context Broker

devices. It tells the IoT Agent that a series of devices will be sending messages to the port where it is listening for Northbound communications. IoT Agent is even informed that the */iot/d* endpoint will be used and that devices will authenticate themselves by including the token *4jggokgpepnvsb2uv4s40d59ov*. Devices will be sending GET or POST requests to: **http://iot-agent:7896/iot/d?i=<device_id>&k=4jggokgpepnvsb2uv4s40d59ov**

```
msg.headers = {
    'Content-Type' : 'application/json',
    'fiware-service' : 'openiot',
    'fiware-servicepath' : '/'
};
msg.payload = {
    "services": [
   {
     "apikey":      "4jggokgpepnvsb2uv4s40d59ov",
     "cbroker":     "http://orion:1026",
     "timezone":    "Europe/Rome",
     "entity_type": "Thing",
     "resource":    "/iot/d"
   }
 ]
}
return msg;
```

Listing 3.2: Provisioning a Service Group

When a measurement from an IoT device is received on the resource URL it needs to be interpreted and passed to the context broker. The entity_type attribute provides a default type for each device which has made a request (in this case anonymous devices will be known as Thing entities. Furthermore the location of the context broker (cbroker) is needed, so that the IoT Agent can pass on any measurements received to the correct location. There

is no guarantee that every supplied IoT device <device-id> will always be unique, therefore all provisioning requests to the IoT Agent require two mandatory headers:

- **fiware-service** header is defined so that entities for a given service can be held in a separate mongoDB database.
- **fiware-servicepath** can be used to differentiate between arrays of devices.

For example within a smart city application you would expect different fiware-service headers for different departments (e.g. parks, transport, refuse collection etc.) and each fiware-servicepath would refer to specific park and so on. This would mean that data and devices for each service can be identified and separated as needed [17].

3.2 IoT devices

Once all the containers of the platform architecture are properly connected through the exposed port, it is possible to provide new devices in the IoT Agent's registry. The IoT Agent node library offers a simple REST API which provides common functionality to access, provision and decommission devices.

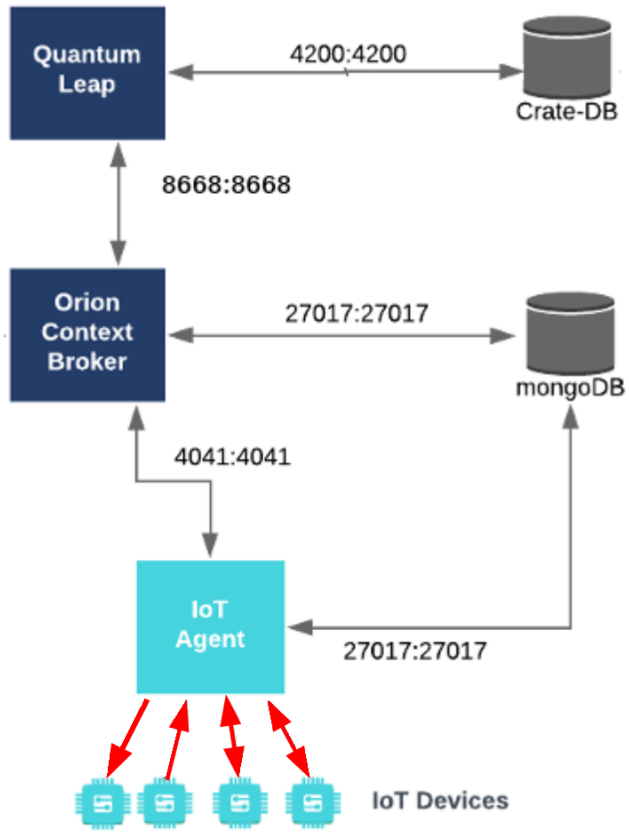

Figure 3.4: IoT devices creation

Since every IoT device is a physical object that exists in the real world, it will eventually

be represented as a unique entity within the context. in our case since we don't have any physical device, only the enity context will be provided.

3.2.1 Registration

Each of the diagram's context entities was created through Node-Red. For every entity type, a stream was created that implemented an HTTP POST request to the exposed port of the IoT agent in the path */iot/devices*.

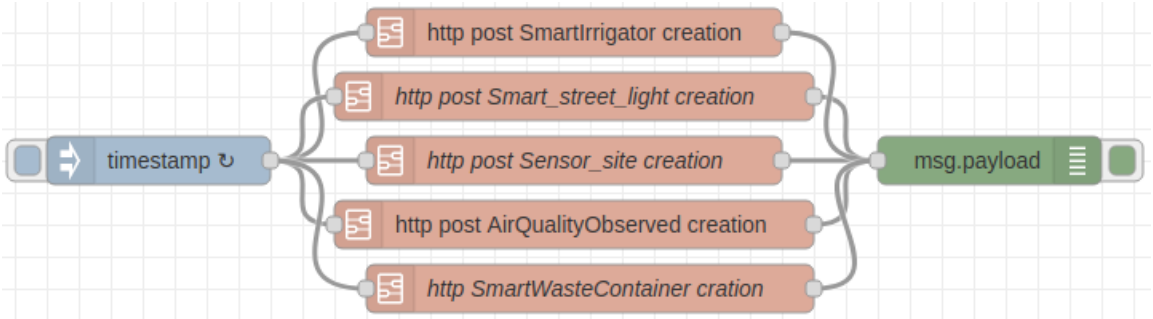

Figure 3.5: Flows that implements HTTP POST requests for entities

In this figure 3.5 there is a "timestamp" node that every 0.2 seconds restarts each stream by increasing a global variable for each type. This feature is helpful to perform a rapidly initialization of the environment. Each Device will be mapped as an Entity associated to a Context Provider: the Device ID will be mapped by default to the entity ID and the type of the entity will be selected by the IoT Agent in a protocol-dependent way (e.g: with different URLs for different types). Both the name and type will be configurable by the user, either by type configuration [18]. Below we can see an example of the code in this nodes.

```
msg.headers = {
    'Content-Type' : 'application/json',
    'fiware-service' : 'openiot',
    'fiware-servicepath' : '/'
};
msg.payload = {
    "devices": [
    {
      "device_id": "streetlight"+iterator,
      "entity_name": "urn:ngsi-ld:Streetlight:"+iterator,
      "timezone": "Europe/Berlin",
      "entity_type": "Streetlight",
      "protocol":    "PDI-IoTA-UltraLight",
      "transport":   "HTTP",
```

```
15        "attributes": [
           {"name": "state", "type":"Text"},
17         {"name":"luminosity", "type":"Integer"},
           {"name":"tension", "type":"Float"}
19        ],
          "static_attributes": [
21          {"name": "category", "type":"Text", "value": ["sensor"
    ]},
            {"name": "city", "type": "Relationship","value": "
    BERLIN"},
23          {"name":"location", "type":"geo:point", "value":(Math.
    random() * (53.000 - 52.000) + 52.000).toFixed(15)+","+(Math
    .random() * (13.000 - 12.000) + 12.000).toFixed(15)}
          ]
25       }
      ]
27  }
    iterator++;
29  global.set('iterator',iterator);
    return msg;
```

Listing 3.3: HTTP POST request to create multiple context entity_type "Streetlight"

Whenever a device is registered, the IoT Agent reads the device's entity information from the request or, if that information is not in the request, from the default values for that type of device. With this information, the IoT Agent sends a new registerContext request to the Context Broker, registering itself as ContextProvider of all the attributes for the device's entity. The registrationId is then stored along the other device information inside the IoT Agent device registry [18]. An example of state information held within each device, as it will eventually be seen within the Context Broker is defined in the diagram below:

3.2.2 Simulation of measurements

After the creation of the devices in the platform, it is possible to pretend that they are really existing sensors using the Node-Red streams. The latter allow us to repeat fake sensor measurements over time. As mentioned earlier since we are using the ultra-light IoT agent, the devices will send requests to:
http://iot-agent:7896/iot/d?i=<device_id>&k=4jggokgpepnvsb2uv4s40d59ov
This brings us to two possible scenarios: 1) receiving a measure from an anonymous device; 2) receiving a measure from a known device. In the first scenario the IoT agent south port is listening to the path defined in the service group and the API key is recognized to match.

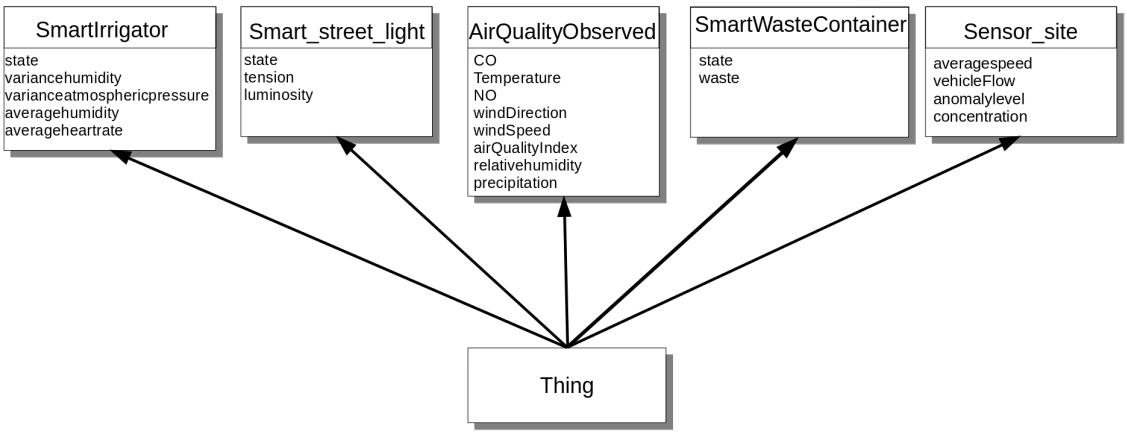

Figure 3.6: State information diagram

As for the device_id it is not recognized so the service group configuration will be used. This means that the measurement will be saved by mapping the device_id provided under the "Thing" category. In the other scenario, the IoT agent south port listens for the path defined in the service group and the API key is recognized to match. Because device_id is also recognized, the provisioned device configuration is used and its settings are combined with the service group. For simplicity, only the scenario in which the sensors present in the network were already known was considered. The simulation of the sensor measurements was carried out by the following flows:

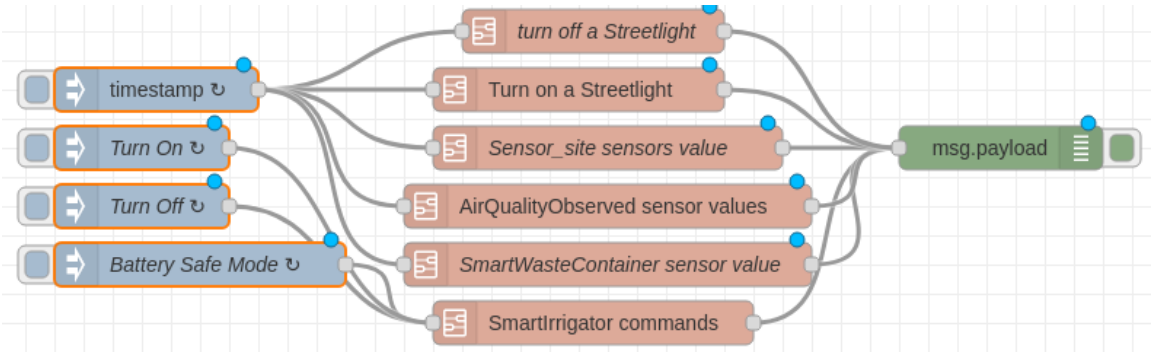

Figure 3.7: Simulation of device measurement on Node-Red

Each node in this stream randomly selects a device from one of the five previously registered entity types. Every IoT device will be using the UltraLight 2.0 protocol running over HTTP. An example of the request payload is shown below:

```
msg.headers = {
    'Content-Type' : 'text/plain'
};
```

```
msg.device =  Math.floor(Math.random() * 50);
var precipitation = Math.round(Math.random());
var CO = Math.floor(Math.random() * (800-200)+200);
var NO = Math.floor(Math.random() * (70-20)+20);
var temperature = (Math.random() * (40 -0) + 0);
var windDirection = Math.floor(Math.random() * (200-50)+50);
var windSpeed =Math.floor( Math.random() * 36);
var airQualityIndex = Math.floor(Math.random() * (100-15)+15);
var relativehumidity = (Math.random()*100).toFixed(2);
msg.payload ='NO|'+NO+'|precipitation|'+precipitation+
             '|CO|'+CO + '|temperature|'+temperature+
             '|windDirection|'+windDirection+'|windSpeed|'+
   windSpeed+
             '|airQualityIndex|'+airQualityIndex+'|
   relativehumidity|'+relativehumidity
return msg;
```

Listing 3.4: HTTP request payload of an AirQualityObserved sensor

This example shows how a real *AirQualityObserved* sensor would send an Ultralight measurement to a IoT Agent, providing the value for each attribute.

Chapter 4

Protocol

Protocol, in computer science, a set of rules or procedures for transmitting data between electronic devices, such as computers. In order for computers to exchange information, there must be a preexisting agreement as to how the information will be structured and how each side will send and receive it [19]. Within a scenario such as SmartCity, the protocols involved are many. In this chapter we will focus on the most important ones. As for the transport layer, although several transport protocols can be used for simplicity, only HTTP was used. Of the remaining protocols, **NGSI** and **Ultralight2**.0 are listed here as the most relevant and used.

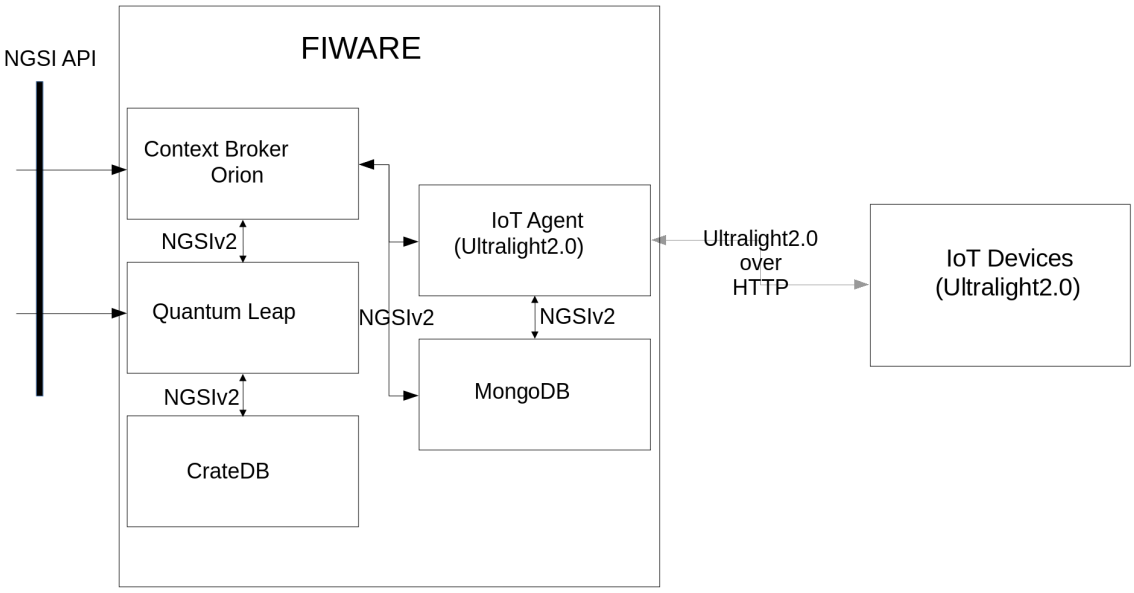

Figure 4.1: Protocol between different containers

4.1 NGSI

NGSI(Next Generation Service Interface) is a protocol developed by OMA to manage Context Information. It provides operations like:

- Manage the Context Information about Context Entities, for example the lifetime and quality of information.
- Access (query, subscribe/notify) to the available Context Information about Context Entities.

The FIWARE version of the OMA NGSI interface is a RESTful API via HTTP. Its purpose is to exchange context information. The two main interaction types are:

- one-time queries for context information
- subscriptions for context information updates (and the corresponding notifications)

The FIWARE NGSI API defines: 1) a **data model** for context information, based on a simple information model using the notion of context entities; 2) a **context data interface** for exchanging information by means of query, subscription, and update operations; 3) a **context availability interface** for exchanging information on how to obtain context information (whether to separate the two interfaces is currently under discussion). Originally NGSIv1, based on standard OMA-NGSI operations, was not really RESTful. In fact, the only method is always POST and the URL does not identify a resource, but a type of operation. Actually NGSIv1 is closer to HTTP-based RPC than RESTful. NGSIv2 was designed from the ground up with RESTful principles in mind.

4.1.1 Advanced Features

The NGSIv2 API offer to Orion Context Broker interesting advanced features. Among the multitude of features, the most important we intend to focus on are:

- Pagination
- Geo-location
- Attributes filtering

4.1.1.1 Pagination

This feature helps to organize query and discovery requests with a large number of response. Four URI parameters:

- *limit*, number of elements per page (default: 20, max: 1000)
- *offset*, number of elements to skip (default: 0)
- *count*, return total elements (default: not return)

- *orderBy*, parameter to specify the attributes or properties to be be used as criteria when ordering results (default: returns are ordered by entity creation date)

e.g.
GET <orion_host>:1026/v2/entities?offset=50&limit=100&options=count&orderBy=temperature
...
(Entities **from** 51 **to** 150 **orders** by temperature in ascending order are returned, along with the 'Fiware-Total-Count: 322' header, which makes the client aware of **how many entities** there are in total and, therefore, the number of subsequent queries to be done)

4.1.1.2 Geo-location

The geospatial properties of a context entity can be represented by means of regular context attributes. The provision of geospatial properties enables the resolution of geographical queries. Two different syntaxes must be supported by compliant implementations [8]:

- Simple Location Format. It is meant as a very lightweight format for developers and users to quickly and easily add to their existing entities.
- GeoJSON. GeoJSON is a geospatial data interchange format based on the JavaScript Object Notation (JSON). GeoJSON provides greater flexibility allowing the representation of point altitudes or even more complex geospatial shapes, for instance multi geometries.

Geographical queries are specified using the following parameters: **georel** is intended to specify a spatial relationship (a predicate) between matching entities and a reference shape (**geometry**). It is composed of a token list separated by ';'.
e.g.
georel=near;maxDistance:1000&geometry=point&coords=-40.4,-3.5
...
(Matching entities must be located (at most) 1000 meters from the reference point)
georel=coveredBy&geometry=polygon&coords=25.774,-80.190;18.466,-66.118;32.321,-64.757;25.774,-80.190
...
(Matching entities are those located within the referred polygon)

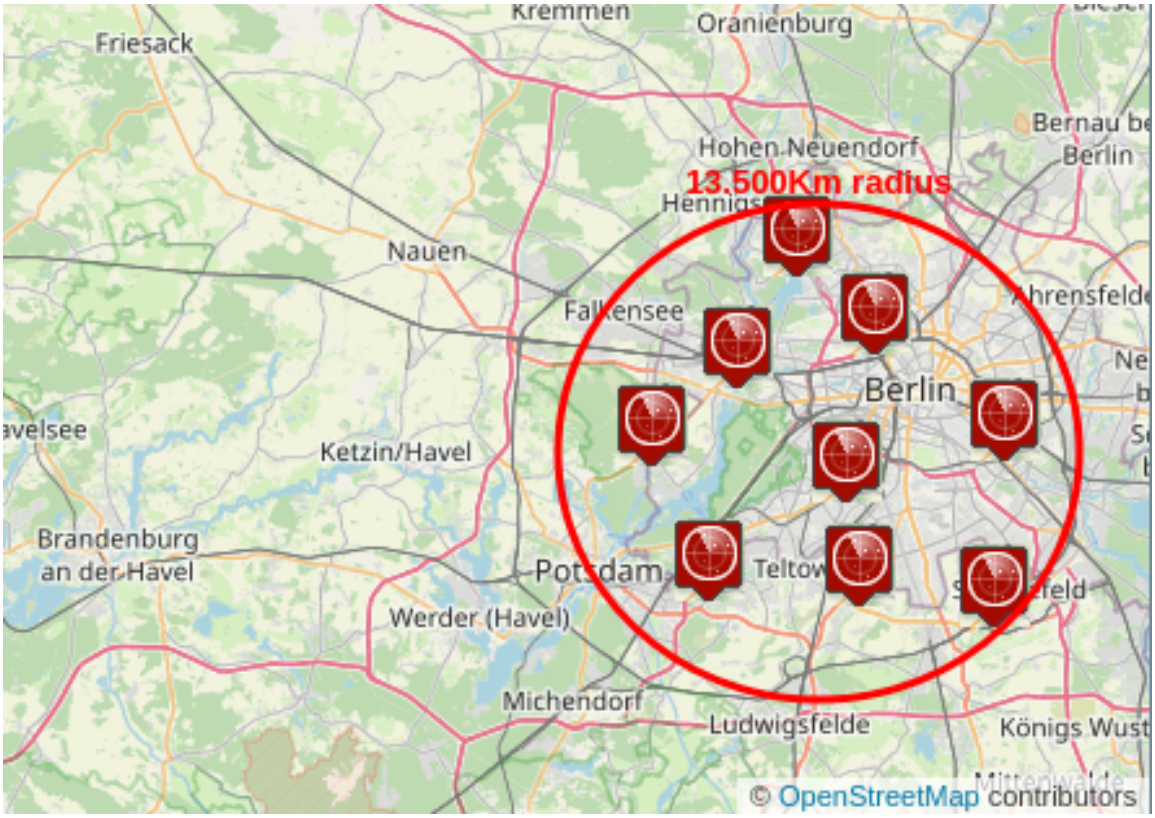

Figure 4.2: An example of geographical queries

4.1.1.3 Attributes filtering

The attrs URL parameter can be used in retrieve operations to specify the list of attributes that must be included in the response. By default, if attrs is omitted then all the attributes are included [8].
e.g.
attrs=temperature,humidity
...
(to include only attributes temperature and humidity)

4.2 Ultralight 2.0

UltraLight 2.0 is a lightweight text based protocol for constrained devices and communications where bandwidth and device memory resources are limited. The payload for measurement requests is a list of key-value pairs separated by the pipe | character. e.g.
<key>|<value>|<key>|<value>|<key>|<value> etc..
For example a payload such as:
t|15|k|abc
Contains two attributes, one named "t" with value "15" and another named "k" with value "abc" are transmitted. Values in Ultralight 2.0 are not typed (everything is treated as a string). Ultralight 2.0 defines a payload describing measures and commands to share be-

tween devices and servers but, does not specify a single transport protocol. Instead, different transport protocol bindings (such as HTTP, MQTT and AMQP) can be used for different scenarios. There are three possible interactions defined in the HTTP binding: requests with GET, requests with POST and commands [20].

4.2.1 Requests with GET requests

A device can report new measures to the IoT Platform using an HTTP GET request to the /iot/d path with the following query parameters [20]:

- **i** (device ID): Device ID (unique for the API Key).
- **k** (API Key): API Key for the service the device is registered on.
- **t** (timestamp): Timestamp of the measure. Will override the automatic IoTAgent timestamp (optional).
- **d** (Data): Ultralight 2.0 payload.

Payloads for GET requests should not contain multiple measure groups.

4.2.2 Requests with POST requests

Another way of reporting measures is to do it using a POST request. In this case, the payload is passed along as the request payload. Two query parameters are still mandatory [20]:

- i (device ID): Device ID (unique for the API Key).
- k (API Key): API Key for the service the device is registered on.
- t (timestamp): Timestamp of the measure. Will override the automatic IoTAgent timestamp (optional).

Chapter 5

Project overview

The purpose of this thesis was to analyze the solutions based on Orion Context Broker provided by the open-source FIWARE platform. Subsequently, once the solution that best meets the requirements of a SmartCity scenario has been identified, testing it with the simulation of some devices. Finally, the main goal to be accomplished remains. The project needed to test whether Snap4City was able to use the data collected by Orion. For this purpose the Snap4City platform was installed locally. To shorten deployment and/or installation tasks, Snap4City provides a set of containers/composes of virtual machines, VMs, devices, and Docker. In our case we used the smallest solution that provides a large-scale smart city. In particular we used Snap4City DataCity-Small released as DOCKER Compose.

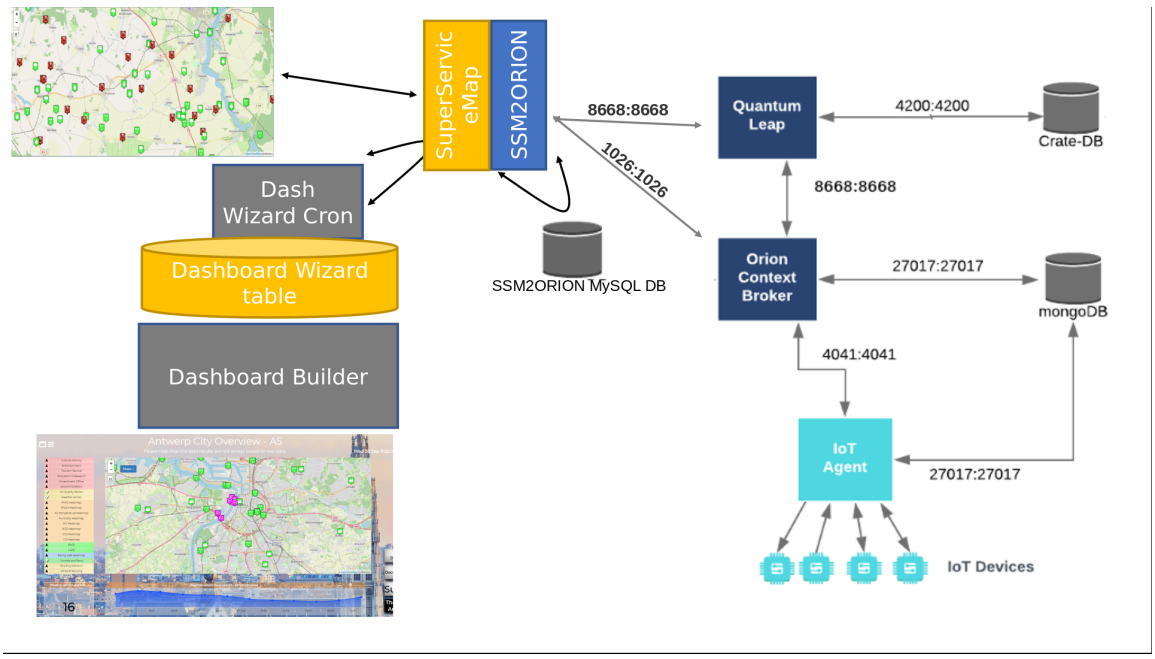

Figure 5.1: Project architecture, Snap4City and Orion Context Broker

5.1 Requirements

Once the platform built around Orion is up and running, all that remains is to connect it to the local version of Snap4City. The connection between the two platforms must be analyzed on two distinct levels:

- **physical**
- **logical**

To allow the connection between platforms at the physical level, all that remains is to configure the databases provided by Snap4city. This is to be able to direct the APIs to the Orion Broker platform. First you need to access the container that contains the MySQL database named MariaDB. Among the various databases the one offending is SuperServiceMap contains the servicemaps table. This table is responsible for providing the address of the service to which APIs should be directed according to the geographical area to which we refer. In fact, in this database we will have to provide a geographic area in the form of a wkt polygon and a url to send the APIs to. The physical connection is completed, it is now possible to focus on the remaining connection on a logical level.

5.2 Logical connection

This level requires more attention than the physical one. In this layer, you need to understand how you can use Orion data within Snap4City. The complexity of this step is given by the inability to modify the Smart City APIs provided by Snap4City and neither the Orion NGSI APIs. It was therefore decided to opt for the development of a web application that was called SSM2ORION. This application is intended to mediate communications between Snap4City and Orion. The problems this application faces can be grouped into two sub-groups. The first concerns the transformation of Smart City APIs into NGSI APIs and formatting the result, this topic will be deepened in the next chapter. The second problem concerns the different nomenclature adopted by Snap4City and Orion that prevents communication. In fact, the names used by Orion to describe the types of devices do not follow Snp4City rules. To come up against a compromise between freedom of choice of names and compatibility with Snap4City, we decided to add a database to the web application. A user-configurable database, with the intent to map the types contained in orion with the types understood by Snap4City (**figure** 5.2).

```
mysql> SELECT * FROM categories;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    53
Current database: SSM2ORION

+--------------------+--------------------------------+--------------------------+
| orionType          | categoriesSSM                  | macroCategoriesSSM       |
+--------------------+--------------------------------+--------------------------+
| SensorSite         | SensorSite                     | TransferServiceAndRenting |
| Streetlight        | Smart_street_light             | Environment              |
| AirQualityObserved | Air_quality_monitoring_station | Environment              |
| SmartIrrigator     | Smart_irrigator                | Environment              |
+--------------------+--------------------------------+--------------------------+
4 rows in set (0.06 sec)
```

Figure 5.2: Orion types mapped to Snap4City categories

Snap4City divides the devices into macro categories which in turn are divided into categories. This means that for each type that will be inserted in Orion, the macro category will be found and consequently the category to which it corresponds in Snap4City. As for the attributes of the devices, the application allows configuration from the database as for the types. In fact it is possible to freely call the attributes specifying the unit of measurement and the physical quantity.

Chapter 6

Web Application

This chapter explains how the developed web application works. To achieve the goal of integrating this solution into Snap4City we modified the already existing Super Service Map that was already integrated in the **Smart City API** of Snap4City. The idea for this application was that towards Snap4City it behaves exactly like the existing Super Service Map as a black box. The application therefore aims to respond to the APIs directed to Super Service Map and redirect them to the platform of our solution. This application has taken the name of SSM2ORION(Super Service Map to Orion), because SMM2ORION acts as a mediator between Super Service Map and the main component of the platform, Orion Context Broker.

6.1 Development

This chapter will be dedicated to the detailed analysis of the application created. The language with which SSM2ORION was built is java. In particular the Java API for RESTful Web Services (JAX-RS). JAX-RS is a Java programming language API spec that provides support in creating web services according to the Representational State Transfer (REST) architectural pattern. For simplicity SSM2ORION redefines only one of the functions provided by Super Service Map, to be exact *getServices*. As an intermediary between Snap4City and the Orion Context Broker platform, it is possible to summarize two main objectives of this application. The first goal was to readjust the incoming APIs to those provided by Orion NGSIv2 RESTful API. The second was to change the format of the response provided by Orion as if it were Super Service Map responding.

6.1.1 API

The *getServices* method is accessible at the path */api/v1* as defined by the *@Path* annotation provided by JAX-RS.

```
1 @JsonSerialize
  @Path("/api/v1")
3 public class ApiV1Resource {
```

```
    @Context
    private HttpServletRequest requestContext;
    private String quantumPrefix;
    @GET
    public Response getServices(
            @QueryParam("selection") String selection,
            @QueryParam("categories") String categories,
            @QueryParam("maxDists") String maxDists,
            @QueryParam("maxResults") String maxResults,
            @QueryParam("geometry") String geometry,
            @QueryParam("accessToken") String accessToken,
            @QueryParam("serviceUri") String serviceUri,
            @QueryParam("realtime") String realtime,
            @QueryParam("valueName") String valueName,
            @QueryParam("fromTime") String fromTime,
            @QueryParam("toTime") String toTime,
            @QueryParam("format") String format,
            @QueryParam("apikey") String apikey) throws
  Exception {...}
```

Listing 6.1: @QueryParam binds the method parameter to the value of an HTTP query parameter

For the sake of brevity it is not possible to show the entire code of this function. Nevertheless, from this code snippet we can observe the *@QueryParam*, which are the values of the HTTP query parameters. Query parameters are extracted from the request URI query parameters, and are specified by using the javax.ws.rs.QueryParam annotation in the method parameter arguments. If a query parameter "selection" exists(*See code* 6.1) in the query component of the request URI, then the "selection" value will be extracted and parsed as a String and assigned to the selection method parameter. JAX-RS provides this annotations to method parameters to pull information out of the request. This allows us to extract the information contained in the APIs directed to our application to be redirected to Orion Context Broker. It was necessary to understand which parameters of the Orion APIs corresponded to the parameters of the APIs provided by Snap4City. For example, the "selection" parameter matched the "coords" parameter in the Orion NGSIv2 API. The problem was that, for example, the string supplied in the "selection" parameter was not in the form required by the "coords" parameter and so on it was necessary to process the input values to build the correct APIs to Orion. Here is the code to adapt the format in which *"selection"* is passed to the format required by *"coords"*.

```
private String formatsSelection(String selection) {
        String coords;
        if (sel.contains("POLYGON")) {
            coords = selection.replaceFirst("(.*)POLYGON", "");
```

```
            coords = coords.substring(2, sel.length() - 2);
        } else {
            coords = selection.replaceFirst("(.*)LINESTRING", "
");
            coords = coords.substring(1, sel.length() - 1);
        }
        coords = coords.replace(",", ";").replace(" ", ",");
        String[] pieces = coords.split(";");
        StringBuilder result = new StringBuilder();
        List<String> words = new ArrayList<>();
        for (int i = 0; i < pieces.length; ++i) {
            words = Arrays.asList(pieces[i].split(","));
            Collections.reverse(words);
            for (String word : words) {
                result.append(word).append(",");
            }
            if (result.length() > 0)
                result.setLength(result.length() - 1);
            result.append(";");
        }
        if (result.length() > 0)
            result.setLength(result.length() - 1);
        return result.toString();
    }
```

Listing 6.2: Function that formats "selection" provided by Snap4City to "coords" in the format required by Orion

Like this function shown above there are many others with similar functionality. The goal is to make the parameters of the Smart City APIs usable in the Orion APIs. All this in order to then build the correct API to point to Orion as shown below (**figure** 6.1).
Another problem that this application had to solve was related to the architectural limitations of Orion. By analyzing Orion and its features we learned that Orion has an entity limit that it can return to an API. This limit is set at 1000. This conflicts with one of the goals of this solution, scalability. To get around this, however, we used the pagination features that Orion itself provides. In fact, repeating the same API several times and modifying the offset and limit it is possible to report a number much greater than 1000.

The application has been tested with numbers much greater than 1000, but for the sake of brevity in the tests reported in the following chapter a limited sample of 20 units will be used. This is to allow greater clarity and understanding of the functionality of the software created.

```
ClientConfig config = new ClientConfig();
Client client = ClientBuilder.newClient(config);
String httpRequestForwardedFor = "";
httpRequestForwardedFor += ipAddressRequestCameFrom;
String SMQUERY = competentServiceMapsPrefix.get(0) + "/v2/entities?ssm=yes&orderBy=geo:distance&options=count&offset=" + relative
        + (categories == null || categories.isEmpty() ? "" : "&type=" + URLEncoder.encode(categories, s1: "UTF-8"))
        + (selection == null || selection.isEmpty() ? "" : "&georel=" + URLEncoder.encode(georel, s1: "UTF-8"))
        + (!georel.equals("near") ? "" : ";maxDistance:" + URLEncoder.encode(maxDists, s1: "UTF-8"))
        + (geometry == null || geometry.isEmpty() ? "" : "&geometry=" + URLEncoder.encode(geometry, s1: "UTF-8"))
        + (selection == null || selection.isEmpty() ? "" : "&coords=" + sel)
        + (maxResults == null || maxResults.isEmpty() ? "" : "&limit=" + URLEncoder.encode(maxResults, s1: "UTF-8"))
        + (format == null || format.isEmpty() ? "" : "&format=" + URLEncoder.encode(format, s1: "UTF-8"))
        + (apikey == null || apikey.isEmpty() ? "" : "&apikey=" + URLEncoder.encode(apikey, s1: "UTF-8"));


WebTarget targetServiceMap = client.target(UriBuilder.fromUri(SMQUERY).build());

if (requestContext.getHeader( s: "X-Forwarded-For") != null && !requestContext.getHeader( s: "X-Forwarded-For").isEmpty()) {
    httpRequestForwardedFor += requestContext.getHeader( s: "X-Forwarded-For") + ",";
}

r = targetServiceMap.request().header( s: "X-Forwarded-For", httpRequestForwardedFor).header(serviceMapsHeaders.get(0),
        serviceMapsHeaders.get(1)).header(serviceMapsHeaders.get(2), serviceMapsHeaders.get(3)).get();
serviceMapResponse.append(r.readEntity(String.class));
```

Figure 6.1: API construction with correct parameters

Main query parameters APIs

Smart City API	Description
selection	• *Service search near GPS position*; Sample value: 43.7756;11.2490. • *Service search within a GPS (rectangular) area*; Sample value: 3.7741;11.2453;43.7768;11.2515. • *Service search within a WKT described area*; Sample value: wkt:POLYGON((11.25539 43.77339, 11.25608 43.77348, 11.25706 43.77362, 11.25759 43.77328, 11.25755 43.77291, 11.25675 43.77260, 11.25536 43.77270, 11.25539 43.77339)).
categories	The list of categories of the services to be retrieved separated with semicolon, if omitted all kinds of services are returned. It can contain macro categories or categories, if a macro category is specified all categories in the macro category are used. The complete list of categories and macro categories can be retrieved on servicemap.disit.org.
maxDists	Maximum distance from the reference position (selection parameter), expressed in kilometers. This parameter can also be set to inside, in which case services are discovered that have a WKT geometry that covers the reference position. It defaults to 0.1.
maxResults	Maximum number of results to be returned. If it is set to zero, all results are returned. It defaults to 100.
serviceUri	URI of the service of interest. Example: http://www.disit.org/km4city/resource/RT04801702315PO
realtime	It indicates if the last values of the time varying properties should be provided in the result or not. It defaults to true.
fromTime	To be used in requests for real-time values, to indicate the date and time of start of the time interval of interest.
toTime	To be used in requests for real-time values, to indicate the date and time of end of the time interval of interest.

6.1.2 Json response

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. JSON is a text format that is completely language independent.

Even though the getServices method was able to transform the incoming APIs into correct APIs for Orion, the response still had to be readjusted to make it readable for Snap4City. Although Orion's responses were provided in Json format, they had to be manipulated to return readable Json for Snap4City. Below is an example of the structure of a readable json response for Snap4City.

```
"Services": {
    "features": [
        {
            "geometry": {
                "coordinates": [
                    11.23741817474365,
                    43.78316879272461
                ],
                "type": "Point"
            },
            "id": 1,
            "properties": {
                "distance": "0.1606",
                "multimedia": "",
                "name": "The Best S.r.l.",
                "photoThumbs": [],
                "serviceType": "Accommodation_Hotel",
                "serviceUri": "http://www.disit.org/km4city/
   resource/fdbb23055e14557df5ac0541bd52b765",
                "tipo": "Hotel",
                "typeLabel": "Hotel"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    11.23850536346436,
                    43.7859992980957
                ],
                "type": "Point"
            },
            "id": 2,
```

```
            "properties": {
                "distance": "0.1684",
                "multimedia": "",
                "name": "Charly 07 S.r.l.",
                "photoThumbs": [],
                "serviceType": "Accommodation_Hotel",
                "serviceUri": "http://www.disit.org/km4city/
    resource/b2e851a740de67c47d4aa6a4d53d185e",
                "tipo": "Hotel",
                "typeLabel": "Hotel"
            },
            "type": "Feature"
        }
    ],
    "fullCount": 4,
    "type": "FeatureCollection"
}
```


Listing 6.3: Response to https://www.disit.org/superservicemap/api/v1/

This json is the response of the API:
https://www.disit.org/superservicemap/api/v1/? selection=43.7845;11.2382 & maxDists=0.5 & categories=Hotel& maxResults=2
which, thanks to the getServices function, would be transformed in the API:
http://<ip-machine>:1026/v2/entities? coords=43.7845,11.2382 & georel=near;maxDistance:5000 & type=Hotel & geometry=point & limit=2 & options=count

6.2 Using a MySQL Database

MySQL is a popular open source database management system commonly used in web applications due to its speed, flexibility and reliability. MySQL employs SQL, or Structured Query Language, for accessing and processing data contained in databases. The web application we developed makes use of a MySQL database called SSM2ORION without much imagination. This database is configurable in the settings.xml file below:

```
<?xml version="1.0" encoding="UTF-8"?>
<setting>
  <label id="username">
    <value>username</value>
  </label>
  <label id="password">
    <value>Password123.</value>
```

```
8     </label>
      <label id="urlMySqlDB">
10      <value>jdbc:mysql://127.0.0.1:3306</value>
      </label>
12    <label id="db">
        <value>SSM2ORION</value>
14    </label>
  </setting>
```

Listing 6.4: File to configure SSM2ORION database

The database requires four tables in it:

1. orion_broker
2. quantumleapZone
3. categories
4. valueInfo

The first two tables contain information about the location where Orion Context Broker and Quantumleap are contained, to allow the SSM2ORION application to direct the APIs to them once transformed. Furthermore orion_broker contains the value of *fiware-service* and *fiware-servicepath*, the two headers required by the IoT Agent.
As regards the table of categories it is necessary to allow SSM2ORION to carry out a correct mapping of the categories. In fact, Snap4City divides its "smart things" into macro categories which are in turn divided into categories. As far as Orion is concerned, we have left the freedom to use different categories, creating the possibility of mapping the Orion categories with those already existing in Snap4City thanks to the categories table. The last table has the task of attributing a unit of measurement to the values of the attributes provided by the sensors and providing a value_type. value_type generically indicates a data type to reconcile data that comes from attributes with different names, but which contain the same physical quantity.

Chapter 7

Verification & validation

This chapter aims to show how the information contained on Orion is the same as received by Snap4City via the SSM2ORION application. For this purpose, five entities have been generated for each type of devices shown below. Each of the generated entities received a

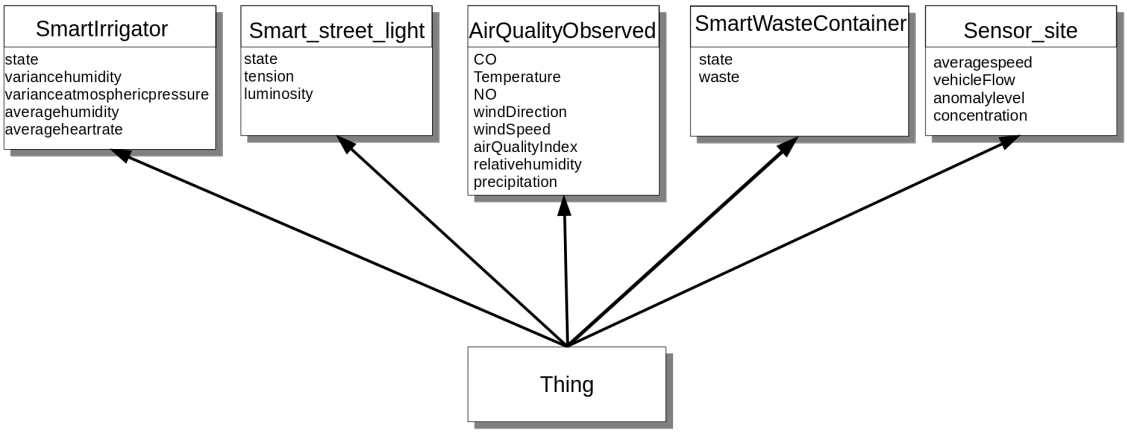

Figure 7.1: State information diagram

random geographic position with variable latitude in the range 52.3000 and 52.6000 and longitude in 13.2000 and 14.0000. This geographical area corresponds to the area occupied by the city of Berlin that we have decided to be the center of our simulation.

7.1 Geoqueries test

Let's start with geographic API tests right away. We can first check if, as set in the Node-Red flows, the entities in the geographical box defined above have really been generated.

7.1.1 Box

Below we show the Smart City API to show all the entities present in the generation box. After which we will show the corresponding API on orion and compare if the data is the same. For purposes of brevity, a limit of five units will be set in response.

```
"Services": {
    "features": [
        {
            "geometry": {
                "coordinates": [
                    13.859457980229442,
                    52.33037715674482
                ],
                "type": "Point"
            },
            "id": 1,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:AirQualityObserved:3",
                "photoThumbs": [],
                "serviceType": "
    Environment_Air_quality_monitoring_station",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:AirQualityObserved:3",
                "tipo": "Air_quality_monitoring_station",
                "typeLabel": "Air_quality_monitoring_station"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.967978738360138,
                    52.33983720270235
                ],
                "type": "Point"
            },
            "id": 2,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:Streetlight:3",
                "photoThumbs": [],
                "serviceType": "Environment_Smart_street_light"
    ,
                "serviceUri": "http://www.disit.org/km4city/
```

```
resource/Orion:1/urn:ngsi-ld:Streetlight:3",
                "tipo": "Smart_street_light",
                "typeLabel": "Smart_street_light"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.988522749007819,
                    52.48291872698824
                ],
                "type": "Point"
            },
            "id": 3,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:SmartIrrigator:1",
                "photoThumbs": [],
                "serviceType": "Environment_Smart_irrigator",
                "serviceUri": "http://www.disit.org/km4city/
resource/Orion:1/urn:ngsi-ld:SmartIrrigator:1",
                "tipo": "Smart_irrigator",
                "typeLabel": "Smart_irrigator"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.903973392521205,
                    52.45880808313917
                ],
                "type": "Point"
            },
            "id": 4,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:SmartIrrigator:3",
                "photoThumbs": [],
```

```
                "serviceType": "Environment_Smart_irrigator",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SmartIrrigator:3",
                "tipo": "Smart_irrigator",
                "typeLabel": "Smart_irrigator"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.852334366186586,
                    52.44857737021909
                ],
                "type": "Point"
            },
            "id": 5,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:AirQualityObserved:0",
                "photoThumbs": [],
                "serviceType": "
    Environment_Air_quality_monitoring_station",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:AirQualityObserved:0",
                "tipo": "Air_quality_monitoring_station",
                "typeLabel": "Air_quality_monitoring_station"
            },
            "type": "Feature"
        }
     ],
     "fullCount": "20",
     "type": "FeatureCollection"
}
```


Listing 7.1: Response to http://localhost:8080/SSM2ORION/rest/api/v1/?selection=52.3000;13.2000;52.6000;14.0000&maxResults=5

Above we observe the answer we get by asking five entities within the box in which all 20 entities were generated. The "fullcount" attribute contains the number of entities inside the box. This API is transformed into NGSI API for Orion to understand. It is possible to check whether Orion would have given the same answer. Then we send the corresponding

NGSIv2 API directly through its exposed port.


```
[
    {
        "CO": {
            "metadata": {},
            "type": "Integer",
            "value": " "
        },
        "NO": {
            "metadata": {},
            "type": "Integer",
            "value": " "
        },
        "TimeInstant": {
            "metadata": {},
            "type": "DateTime",
            "value": "2020-11-22T15:03:44.00Z"
        },
        "airQualityIndex": {
            "metadata": {},
            "type": "Integer",
            "value": " "
        },
        "category": {
            "metadata": {},
            "type": "Text",
            "value": [
                "sensor"
            ]
        },
        "city": {
            "metadata": {},
            "type": "Relationship",
            "value": "BERLIN"
        },
        "id": "urn:ngsi-ld:AirQualityObserved:3",
        "location": {
            "metadata": {},
            "type": "geo:point",
            "value": "52.330377156744824,13.859457980229442"
        },
        "precipitation": {
```

```
            "metadata": {},
            "type": "Boolean",
            "value": " "
        },
        "relativehumidity": {
            "metadata": {},
            "type": "Float",
            "value": " "
        },
        "temperature": {
            "metadata": {},
            "type": "Float",
            "value": " "
        },
        "type": "AirQualityObserved",
        "windDirection": {
            "metadata": {},
            "type": "Integer",
            "value": " "
        },
        "windSpeed": {
            "metadata": {},
            "type": "Integer",
            "value": " "
        }
    },
    {
        "TimeInstant": {
            "metadata": {},
            "type": "DateTime",
            "value": "2020-11-22T15:03:50.00Z"
        },
        "category": {
            "metadata": {},
            "type": "Text",
            "value": [
                "sensor"
            ]
        },
        "city": {
            "metadata": {},
            "type": "Relationship",
```

```
            "value": "BERLIN"
        },
        "id": "urn:ngsi-ld:Streetlight:3",
        "location": {
            "metadata": {},
            "type": "geo:point",
            "value": "52.339837202702356,13.967978738360138"
        },
        "luminosity": {
            "metadata": {},
            "type": "Integer",
            "value": " "
        },
        "state": {
            "metadata": {},
            "type": "Text",
            "value": " "
        },
        "tension": {
            "metadata": {},
            "type": "Float",
            "value": " "
        },
        "type": "Streetlight"
    },
    {
        "TimeInstant": {
            "metadata": {},
            "type": "DateTime",
            "value": "2020-11-22T15:03:45.00Z"
        },
        "averageheartrate": {
            "metadata": {},
            "type": "Float",
            "value": " "
        },
        "averagehumidity": {
            "metadata": {},
            "type": "Float",
            "value": " "
        },
        "category": {
```

```
            "metadata": {},
            "type": "Text",
            "value": [
                "sensor"
            ]
        },
        "city": {
            "metadata": {},
            "type": "Relationship",
            "value": "BERLIN"
        },
        "id": "urn:ngsi-ld:SmartIrrigator:1",
        "location": {
            "metadata": {},
            "type": "geo:point",
            "value": "52.482918726988245,13.988522749007819"
        },
        "state": {
            "metadata": {},
            "type": "Text",
            "value": " "
        },
        "type": "SmartIrrigator",
        "varianceatmosphericpressure": {
            "metadata": {},
            "type": "Float",
            "value": " "
        },
        "variancehumidity": {
            "metadata": {},
            "type": "Float",
            "value": " "
        }
    },
    {
        "TimeInstant": {
            "metadata": {},
            "type": "DateTime",
            "value": "2020-11-22T15:03:49.00Z"
        },
        "averageheartrate": {
            "metadata": {},
```

```
            "type": "Float",
            "value": " "
        },
        "averagehumidity": {
            "metadata": {},
            "type": "Float",
            "value": " "
        },
        "category": {
            "metadata": {},
            "type": "Text",
            "value": [
                "sensor"
            ]
        },
        "city": {
            "metadata": {},
            "type": "Relationship",
            "value": "BERLIN"
        },
        "id": "urn:ngsi-ld:SmartIrrigator:3",
        "location": {
            "metadata": {},
            "type": "geo:point",
            "value": "52.458808083139175,13.903973392521205"
        },
        "state": {
            "metadata": {},
            "type": "Text",
            "value": " "
        },
        "type": "SmartIrrigator",
        "varianceatmosphericpressure": {
            "metadata": {},
            "type": "Float",
            "value": " "
        },
        "variancehumidity": {
            "metadata": {},
            "type": "Float",
            "value": " "
        }
```

```
    },
    {
        "CO": {
            "metadata": {},
            "type": "Integer",
            "value": " "
        },
        "NO": {
            "metadata": {},
            "type": "Integer",
            "value": " "
        },
        "TimeInstant": {
            "metadata": {},
            "type": "DateTime",
            "value": "2020-11-22T15:03:43.00Z"
        },
        "airQualityIndex": {
            "metadata": {},
            "type": "Integer",
            "value": " "
        },
        "category": {
            "metadata": {},
            "type": "Text",
            "value": [
                "sensor"
            ]
        },
        "city": {
            "metadata": {},
            "type": "Relationship",
            "value": "BERLIN"
        },
        "id": "urn:ngsi-ld:AirQualityObserved:0",
        "location": {
            "metadata": {},
            "type": "geo:point",
            "value": "52.448577370219091,13.852334366186586"
        },
        "precipitation": {
            "metadata": {},
```

```
            "type": "Boolean",
            "value": " "
        },
        "relativehumidity": {
            "metadata": {},
            "type": "Float",
            "value": " "
        },
        "temperature": {
            "metadata": {},
            "type": "Float",
            "value": " "
        },
        "type": "AirQualityObserved",
        "windDirection": {
            "metadata": {},
            "type": "Integer",
            "value": " "
        },
        "windSpeed": {
            "metadata": {},
            "type": "Integer",
            "value": " "
        }
    }
]
```


Listing 7.2: Response to http://localhost:1026/v2/entities?georel=coveredBy&geometry=box&limit=5&coords=52.3000,13.2000;52.6000,14.0000

It is clear that the answer provided by Orion turns out to be the same even if more detailed than the answer that Snap4City needs. Now that we are sure that the information that the application returns corresponds to that contained by Orion, it is possible to continue with other tests that show the potential of the software.

7.1.2 Circumference

Below we show how it is possible to get the same answer using different geometric shapes. Let's start by trying the search with a point and a radius.


```
"Services": {
    "features": [
        {
            "geometry": {
                "coordinates": [
```

```
                    13.523025168331678,
                    52.43662419796348
                ],
                "type": "Point"
            },
            "id": 1,
            "properties": {
                "distance": "8.149123",
                "multimedia": "",
                "name": "urn:ngsi-ld:Streetlight:2",
                "photoThumbs": [],
                "serviceType": "Environment_Smart_street_light"
,
                "serviceUri": "http://www.disit.org/km4city/
resource/Orion:1/urn:ngsi-ld:Streetlight:2",
                "tipo": "Smart_street_light",
                "typeLabel": "Smart_street_light"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.52916729725512,
                    52.44695062282087
                ],
                "type": "Point"
            },
            "id": 2,
            "properties": {
                "distance": "9.1279637",
                "multimedia": "",
                "name": "urn:ngsi-ld:AirQualityObserved:1",
                "photoThumbs": [],
                "serviceType": "
Environment_Air_quality_monitoring_station",
                "serviceUri": "http://www.disit.org/km4city/
resource/Orion:1/urn:ngsi-ld:AirQualityObserved:1",
                "tipo": "Air_quality_monitoring_station",
                "typeLabel": "Air_quality_monitoring_station"
            },
            "type": "Feature"
```

```
        },
        {
            "geometry": {
                "coordinates": [
                    13.557755194005033,
                    52.34689344699744
                ],
                "type": "Point"
            },
            "id": 3,
            "properties": {
                "distance": "11.006386",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:0",
                "photoThumbs": [],
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SensorSite:0",
                "tipo": "SensorSite",
                "typeLabel": "SensorSite"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.580030184042405,
                    52.3233853138258
                ],
                "type": "Point"
            },
            "id": 4,
            "properties": {
                "distance": "13.7444764",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:4",
                "photoThumbs": [],
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SensorSite:4",
```

```
                "tipo": "SensorSite",
                "typeLabel": "SensorSite"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.218408874317863,
                    52.44531896576619
                ],
                "type": "Point"
            },
            "id": 5,
            "properties": {
                "distance": "14.6011051",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:1",
                "photoThumbs": [],
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SensorSite:1",
                "tipo": "SensorSite",
                "typeLabel": "SensorSite"
            },
            "type": "Feature"
        }
    ],
    "fullCount": "20",
    "type": "FeatureCollection"
}
```


Listing 7.3: Response to http://localhost:8080/SSM2ORION/rest/api/v1/?selection=52.3990;13.4199&maxDists=40&maxResults=5

The geographical area covered by the circumference with a radius of 40 Km centered in point (52.3990,13.4199) covers the entire box in which the 20 entities were generated. In this answer we can see how the entities are ordered according to the distance from the center of the circumference.

7.1.3 Polygon

Another way to subtend the area where devices were generated is to use polygons.

```
"Services": {
    "features": [
        {
            "geometry": {
                "coordinates": [
                    13.859457980229442,
                    52.33037715674482
                ],
                "type": "Point"
            },
            "id": 1,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:AirQualityObserved:3",
                "photoThumbs": [],
                "serviceType": "
Environment_Air_quality_monitoring_station",
                "serviceUri": "http://www.disit.org/km4city/
resource/Orion:1/urn:ngsi-ld:AirQualityObserved:3",
                "tipo": "Air_quality_monitoring_station",
                "typeLabel": "Air_quality_monitoring_station"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.967978738360138,
                    52.33983720270235
                ],
                "type": "Point"
            },
            "id": 2,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:Streetlight:3",
                "photoThumbs": [],
                "serviceType": "Environment_Smart_street_light"
,
                "serviceUri": "http://www.disit.org/km4city/
```

```
resource/Orion:1/urn:ngsi-ld:Streetlight:3",
                "tipo": "Smart_street_light",
                "typeLabel": "Smart_street_light"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.988522749007819,
                    52.48291872698824
                ],
                "type": "Point"
            },
            "id": 3,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:SmartIrrigator:1",
                "photoThumbs": [],
                "serviceType": "Environment_Smart_irrigator",
                "serviceUri": "http://www.disit.org/km4city/
resource/Orion:1/urn:ngsi-ld:SmartIrrigator:1",
                "tipo": "Smart_irrigator",
                "typeLabel": "Smart_irrigator"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.903973392521205,
                    52.45880808313917
                ],
                "type": "Point"
            },
            "id": 4,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:SmartIrrigator:3",
                "photoThumbs": [],
```

```
                "serviceType": "Environment_Smart_irrigator",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SmartIrrigator:3",
                "tipo": "Smart_irrigator",
                "typeLabel": "Smart_irrigator"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.852334366186586,
                    52.44857737021909
                ],
                "type": "Point"
            },
            "id": 5,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:AirQualityObserved:0",
                "photoThumbs": [],
                "serviceType": "
    Environment_Air_quality_monitoring_station",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:AirQualityObserved:0",
                "tipo": "Air_quality_monitoring_station",
                "typeLabel": "Air_quality_monitoring_station"
            },
            "type": "Feature"
        }
    ],
    "fullCount": "20",
    "type": "FeatureCollection"
}
```

Listing 7.4: Response to http://localhost:8080/SSM2ORION/rest/api/v1/?selection=wkt:POLYGON((13.035848658376796 52.57452235772299,12.968557398611171 52.39597538507152,13.31291012077914 52.19818879958088,13.871324580007656 52.13712117817799,14.061697047048671 52.29992195909934,14.123495142751796 52.54968630982095,13.787038843923671 52.681635909265935,13.346899073415859 52.73613380058605,13.035848658376796 52.57452235772299))&maxResults=5

7.1.4 Linestring

The last figure that remains to be shown is the line. We have created a line that passes over the first two entities of the results obtained above.

```
"Services": {
    "features": [
        {
            "geometry": {
                "coordinates": [
                    13.859457980229442,
                    52.33037715674482
                ],
                "type": "Point"
            },
            "id": 1,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:AirQualityObserved:3",
                "photoThumbs": [],
                "serviceType": "
Environment_Air_quality_monitoring_station",
                "serviceUri": "http://www.disit.org/km4city/
resource/Orion:1/urn:ngsi-ld:AirQualityObserved:3",
                "tipo": "Air_quality_monitoring_station",
                "typeLabel": "Air_quality_monitoring_station"
            },
            "type": "Feature"
        },
```

```
24          {
25              "geometry": {
26                  "coordinates": [
27                      13.967978738360138,
28                      52.33983720270235
29                  ],
30                  "type": "Point"
31              },
32              "id": 2,
33              "properties": {
34                  "distance": "",
35                  "multimedia": "",
36                  "name": "urn:ngsi-ld:Streetlight:3",
37                  "photoThumbs": [],
38                  "serviceType": "Environment_Smart_street_light"
    ,
39                  "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:Streetlight:3",
40                  "tipo": "Smart_street_light",
41                  "typeLabel": "Smart_street_light"
42              },
43              "type": "Feature"
44          }
45      ],
46      "fullCount": "2",
47      "type": "FeatureCollection"
48  }
```


Listing 7.5: Response to http://localhost:8080/SSM2ORION/rest/api/v1/?selection=wkt:LINESTRING(13.296524338187803 52.62087609829679,13.859457980229442 52.33037715674482,13.967978738360138 52.33983720270235,13.565689377250303 52.68752179481433,13.752456955375303 52.30963761493045,13.131729416312803 52.299561006616315,13.796402267875303 52.62754524001399,13.296524338187803 52.62087609829679)

7.1.5 Selection by categories

Now that we have shown all the geometries that can be used to search for entities on the map, we will show how it is also possible to select a specific type of entity. Recall that initially 20 entities were generated (**figure** 7.1). Five entities of each type.

```
mysql> select * from categories;
+--------------------+--------------------------------+--------------------------+
| orionType          | categoriesSSM                  | macroCategoriesSSM       |
+--------------------+--------------------------------+--------------------------+
| SensorSite         | SensorSite                     | TransferServiceAndRenting |
| Streetlight        | Smart_street_light             | Environment              |
| AirQualityObserved | Air_quality_monitoring_station | Environment              |
| SmartIrrigator     | Smart_irrigator                | Environment              |
| SmartWasteContainer | Smart_waste_container         | Environment              |
+--------------------+--------------------------------+--------------------------+
5 rows in set (0.00 sec)
```

Figure 7.2: categories table into SSM2ORION

Above is the mapping table for Orion types. Looking at the table it is clear that there should be 15 entities for the "*Enviorment*" macro category and five for "*TransferService-AndRenting*". Consequently also five for each category. Let's test if we get this result using the box as geometry.

```
"Services": {
    "features": [
        {
            "geometry": {
                "coordinates": [
                    13.859457980229442,
                    52.33037715674482
                ],
                "type": "Point"
            },
            "id": 1,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:AirQualityObserved:3",
                "photoThumbs": [],
                "serviceType": "
    Environment_Air_quality_monitoring_station",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:AirQualityObserved:3",
                "tipo": "Air_quality_monitoring_station",
                "typeLabel": "Air_quality_monitoring_station"
            },
            "type": "Feature"
        },
        {
```

```
            "geometry": {
                "coordinates": [
                    13.967978738360138,
                    52.33983720270235
                ],
                "type": "Point"
            },
            "id": 2,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:Streetlight:3",
                "photoThumbs": [],
                "serviceType": "Environment_Smart_street_light"
,
                "serviceUri": "http://www.disit.org/km4city/
resource/Orion:1/urn:ngsi-ld:Streetlight:3",
                "tipo": "Smart_street_light",
                "typeLabel": "Smart_street_light"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.988522749007819,
                    52.48291872698824
                ],
                "type": "Point"
            },
            "id": 3,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:SmartIrrigator:1",
                "photoThumbs": [],
                "serviceType": "Environment_Smart_irrigator",
                "serviceUri": "http://www.disit.org/km4city/
resource/Orion:1/urn:ngsi-ld:SmartIrrigator:1",
                "tipo": "Smart_irrigator",
                "typeLabel": "Smart_irrigator"
            },
```

```
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.903973392521205,
                    52.45880808313917
                ],
                "type": "Point"
            },
            "id": 4,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:SmartIrrigator:3",
                "photoThumbs": [],
                "serviceType": "Environment_Smart_irrigator",
                "serviceUri": "http://www.disit.org/km4city/
resource/Orion:1/urn:ngsi-ld:SmartIrrigator:3",
                "tipo": "Smart_irrigator",
                "typeLabel": "Smart_irrigator"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.852334366186586,
                    52.44857737021909
                ],
                "type": "Point"
            },
            "id": 5,
            "properties": {
                "distance": "",
                "multimedia": "",
                "name": "urn:ngsi-ld:AirQualityObserved:0",
                "photoThumbs": [],
                "serviceType": "
Environment_Air_quality_monitoring_station",
                "serviceUri": "http://www.disit.org/km4city/
resource/Orion:1/urn:ngsi-ld:AirQualityObserved:0",
```

```
                "tipo": "Air_quality_monitoring_station",
                "typeLabel": "Air_quality_monitoring_station"
            },
            "type": "Feature"
        }
    ],
    "fullCount": "15",
    "type": "FeatureCollection"
}
```


Listing 7.6: Response to http://localhost:8080/SSM2ORION/rest/api/v1/?selection=52.3000;13.2000;52.6000;14.00009&categories=Environment&maxResults=5

As expected, the value of "fullcount" is equal to 15 even if we have decided to show only the first five entities in order not to dwell on it. Now, however, as a counter-proof we use the circumference to look for the other macro category.


```
"Services": {
    "features": [
        {
            "geometry": {
                "coordinates": [
                    13.557755194005033,
                    52.34689344699744
                ],
                "type": "Point"
            },
            "id": 1,
            "properties": {
                "distance": "11.006386",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:0",
                "photoThumbs": [],
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SensorSite:0",
                "tipo": "SensorSite",
                "typeLabel": "SensorSite"
            },
            "type": "Feature"
        },
        {
            "geometry": {
```

```
                "coordinates": [
                    13.580030184042405,
                    52.3233853138258
                ],
                "type": "Point"
            },
            "id": 2,
            "properties": {
                "distance": "13.7444764",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:4",
                "photoThumbs": [],
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SensorSite:4",
                "tipo": "SensorSite",
                "typeLabel": "SensorSite"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.218408874317863,
                    52.44531896576619
                ],
                "type": "Point"
            },
            "id": 3,
            "properties": {
                "distance": "14.6011051",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:1",
                "photoThumbs": [],
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SensorSite:1",
                "tipo": "SensorSite",
                "typeLabel": "SensorSite"
            },
```

```
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.573661389401405,
                    52.51509330334284
                ],
                "type": "Point"
            },
            "id": 4,
            "properties": {
                "distance": "16.5879297",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:2",
                "photoThumbs": [],
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SensorSite:2",
                "tipo": "SensorSite",
                "typeLabel": "SensorSite"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.835989823766955,
                    52.58127420447191
                ],
                "type": "Point"
            },
            "id": 5,
            "properties": {
                "distance": "34.7034",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:3",
                "photoThumbs": [],
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
```

```
    resource/Orion:1/urn:ngsi-ld:SensorSite:3",
                "tipo": "SensorSite",
                "typeLabel": "SensorSite"
            },
            "type": "Feature"
        }
    ],
    "fullCount": "5",
    "type": "FeatureCollection"
}
```

Listing 7.7: Response to http://localhost:8080/SSM2ORION/rest/api/v1/?selection=52.3990;13.4199&maxDists=40&categories=TransferServiceAndRenting

Below is a final example to show how macro categories and categories can be combined by merging results.

```
"Services": {
    "features": [
        {
            "geometry": {
                "coordinates": [
                    13.52916729725512,
                    52.44695062282087
                ],
                "type": "Point"
            },
            "id": 1,
            "properties": {
                "distance": "9.1279637",
                "multimedia": "",
                "name": "urn:ngsi-ld:AirQualityObserved:1",
                "photoThumbs": [],
                "serviceType": "
    Environment_Air_quality_monitoring_station",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:AirQualityObserved:1",
                "tipo": "Air_quality_monitoring_station",
                "typeLabel": "Air_quality_monitoring_station"
            },
            "type": "Feature"
        },
        {
            "geometry": {
```

```
                "coordinates": [
                    13.557755194005033,
                    52.34689344699744
                ],
                "type": "Point"
            },
            "id": 2,
            "properties": {
                "distance": "11.006386",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:0",
                "photoThumbs": [],
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SensorSite:0",
                "tipo": "SensorSite",
                "typeLabel": "SensorSite"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.580030184042405,
                    52.3233853138258
                ],
                "type": "Point"
            },
            "id": 3,
            "properties": {
                "distance": "13.7444764",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:4",
                "photoThumbs": [],
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SensorSite:4",
                "tipo": "SensorSite",
                "typeLabel": "SensorSite"
            },
```

```
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.218408874317863,
                    52.44531896576619
                ],
                "type": "Point"
            },
            "id": 4,
            "properties": {
                "distance": "14.6011051",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:1",
                "photoThumbs": [],
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SensorSite:1",
                "tipo": "SensorSite",
                "typeLabel": "SensorSite"
            },
            "type": "Feature"
        },
        {
            "geometry": {
                "coordinates": [
                    13.573661389401405,
                    52.51509330334284
                ],
                "type": "Point"
            },
            "id": 5,
            "properties": {
                "distance": "16.5879297",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:2",
                "photoThumbs": [],
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
```

```
    resource/Orion:1/urn:ngsi-ld:SensorSite:2",
                "tipo": "SensorSite",
                "typeLabel": "SensorSite"
            },
            "type": "Feature"
        }
    ],
    "fullCount": "10",
    "type": "FeatureCollection"
}
```


Listing 7.8: Response to http://localhost:8080/SSM2ORION/rest/api/v1/?selection=52.3990;13.4199&maxDists=40&maxResults=5&categories=TransferServiceAndRenting;Air_quality_monitoring_station

7.2 Time Series Data Queries (QuantumLeap API)

Another important feature that SSM2ORION app offers is the ability to query a single device. This important functionality is enabled by the QuantumLeap component. In fact on this occasion the API sent to the application by Snap4City is sent to both Orion and QuantumLeap. Through these APIs it is possible to specify a time interval of which we want to know the values measured by the device, on one or on all the attributes. To select a specific device it is necessary to specify the "serviceUri" parameter by inserting the *url* thus generated "http://www.disit.org/km4city/resource/<name-Orion>/urn:ngsi-ld:<device-categories>:<device-number>". There are four additional parameters for accessing sensor data. Two of these, "toTime" and "fromTime", are used to locate a time range. The remaining two parameters are "valueName" to specify a specific attribute and "realtime" which by default is asserted. "realtime" is used to request or not that measurements are provided. Below we show what we get by looking for the SensorSite: 0 device and setting "realtime" to false.


```
"Service": {
    "features": [{
        "geometry": {
            "coordinates": [
                13.557755194005033,
                52.34689344699744
            ],
            "type": "Point"
        },
        "properties": {
            "address": "",
            "avgStars": 0,
```

```
"brokerName": "orionUNIFI",
"comments": [],
"description": "",
"format": "json",
"linkDBpedia": [],
"macaddress": "",
"maintenanceUrl": "",
"maxCapacity": "",
"minCapacity": "",
"model": "SensorSite",
"multimedia": "",
"name": "urn:ngsi-ld:SensorSite:0",
"organization": "DISIT",
"ownership": "public",
"photoOrigs": [],
"photoThumbs": [],
"photos": [],
"producer": "Citta di BERLIN",
"protocol": "ngsi",
"realtimeAttributes": {
    "anomalylevel": {
        "attr_type": "DeviceAttribute",
        "data_type": "Float",
        "healthiness_criteria": "refresh_rate",
        "value_refresh_rate": "1"
    },
    "averagespeed": {
        "attr_type": "DeviceAttribute",
        "data_type": "Float",
        "healthiness_criteria": "refresh_rate",
        "value_refresh_rate": "1"
    },
    "concentration": {
        "attr_type": "DeviceAttribute",
        "data_type": "Float",
        "healthiness_criteria": "refresh_rate",
        "value_refresh_rate": "1"
    },
    "vehicleFlow": {
        "attr_type": "DeviceAttribute",
        "data_type": "Integer",
        "healthiness_criteria": "refresh_rate",
```

```
                    "value_refresh_rate": "1"
                }
            },
            "serviceType": "
    TransferServiceAndRenting_SensorSite",
            "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SensorSite:0",
            "starsCount": 0,
            "typeLabel": "SensorSite",
            "website": "",
            "wktGeometry": ""
        },
        "type": "Feature"
     }],
     "type": "FeatureCollection"
}
```


Listing 7.9: http://localhost:8080/SSM2ORION/rest/api/v1/?serviceUri=http://www.disit.org/km4city/resource/Orion:1/urn:ngsi-ld:SensorSite:0&realtime=false

Running the same API without setting "realtime", it will return the device data and the last acquired value.


```
 "Service": {
        "features": [{
            "geometry": {
                "coordinates": [
                    13.557755194005033,
                    52.34689344699744
                ],
                "type": "Point"
            },
            "properties": {
                "address": "",
                "avgStars": 0,
                "brokerName": "orionUNIFI",
                "comments": [],
                "description": "",
                "format": "json",
                "linkDBpedia": [],
                "macaddress": "",
                "maintenanceUrl": "",
                "maxCapacity": "",
                "minCapacity": "",
```

```
                "model": "SensorSite",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:0",
                "organization": "DISIT",
                "ownership": "public",
                "photoOrigs": [],
                "photoThumbs": [],
                "photos": [],
                "producer": "Citta di BERLIN",
                "protocol": "ngsi",
                "realtimeAttributes": {
                    "anomalylevel": {
                        "attr_type": "DeviceAttribute",
                        "data_type": "Float",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    },
                    "averagespeed": {
                        "attr_type": "DeviceAttribute",
                        "data_type": "Float",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    },
                    "concentration": {
                        "attr_type": "DeviceAttribute",
                        "data_type": "Float",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    },
                    "vehicleFlow": {
                        "attr_type": "DeviceAttribute",
                        "data_type": "Integer",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    }
                },
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SensorSite:0",
                "starsCount": 0,
                "typeLabel": "SensorSite",
```

```
                "website": "",
                "wktGeometry": ""
            },
            "type": "Feature"
        }],
        "type": "FeatureCollection"
    },
    "realtime": {
        "head": {"vars": [
            "anomalylevel",
            "averagespeed",
            "concentration",
            "vehicleFlow",
            "measuredTime"
        ]},
        "results": {"bindings": [{
            "anomalylevel": {"value": "102.726"},
            "averagespeed": {"value": "59.8"},
            "concentration": {"value": "4.336"},
            "measuredTime": {"value": "2020-11-22T21:28:16+01:0
0"},
            "vehicleFlow": {"value": "198"}
        }]}
    }
```


Listing 7.10: http://localhost:8080/SSM2ORION/rest/api/v1/?serviceUri=http://www.disit.org/km4city/resource/Orion:1/urn:ngsi-ld:SensorSite:0

Access to the logged data is only possible by setting the "toTime" and "fromTime" parameters. The "toTime" parameter must be a date in the format "yyyy-MM-dd'T'HH: mm: ss". As far as the "fromTime" parameter is concerned, it can assume different values. For example if we want to retrieve the last n values, "fromTime" will be set as last-n. But now we will show some examples to show the other ways of using the "fromTime" parameter.


```

    "Service": {
        "features": [{
            "geometry": {
                "coordinates": [
                    13.557755194005033,
                    52.34689344699744
                ],
                "type": "Point"
            },
```

```
        "properties": {
            "address": "",
            "avgStars": 0,
            "brokerName": "orionUNIFI",
            "comments": [],
            "description": "",
            "format": "json",
            "linkDBpedia": [],
            "macaddress": "",
            "maintenanceUrl": "",
            "maxCapacity": "",
            "minCapacity": "",
            "model": "SensorSite",
            "multimedia": "",
            "name": "urn:ngsi-ld:SensorSite:0",
            "organization": "DISIT",
            "ownership": "public",
            "photoOrigs": [],
            "photoThumbs": [],
            "photos": [],
            "producer": "Citta di BERLIN",
            "protocol": "ngsi",
            "realtimeAttributes": {
                "anomalylevel": {
                    "attr_type": "DeviceAttribute",
                    "data_type": "Float",
                    "healthiness_criteria": "refresh_rate",
                    "value_refresh_rate": "1"
                },
                "averagespeed": {
                    "attr_type": "DeviceAttribute",
                    "data_type": "Float",
                    "healthiness_criteria": "refresh_rate",
                    "value_refresh_rate": "1"
                },
                "concentration": {
                    "attr_type": "DeviceAttribute",
                    "data_type": "Float",
                    "healthiness_criteria": "refresh_rate",
                    "value_refresh_rate": "1"
                },
                "vehicleFlow": {
```

```
                        "attr_type": "DeviceAttribute",
                        "data_type": "Integer",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    }
                },
                "serviceType": "
TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
resource/Orion:1/urn:ngsi-ld:SensorSite:0",
                "starsCount": 0,
                "typeLabel": "SensorSite",
                "website": "",
                "wktGeometry": ""
            },
            "type": "Feature"
        }],
        "type": "FeatureCollection"
    },
    "realtime": {
        "head": {"vars": [
            "anomalylevel",
            "averagespeed",
            "concentration",
            "vehicleFlow",
            "measuredTime"
        ]},
        "results": {"bindings": [
            {
                "anomalylevel": {"value": "101.558"},
                "averagespeed": {"value": "79.0"},
                "concentration": {"value": "9.669"},
                "measuredTime": {"value": "2020-11-22T20:12:56+
01:00"},
                "vehicleFlow": {"value": "155"}
            },
            {
                "anomalylevel": {"value": "100.056"},
                "averagespeed": {"value": "46.4"},
                "concentration": {"value": "2.992"},
                "measuredTime": {"value": "2020-11-22T20:12:50+
01:00"},
```

```
                "vehicleFlow": {"value": "27"}
            },
            {
                "anomalylevel": {"value": "104.860"},
                "averagespeed": {"value": "49.2"},
                "concentration": {"value": "3.103"},
                "measuredTime": {"value": "2020-11-22T20:12:40+
    01:00"},
                "vehicleFlow": {"value": "495"}
            },
            {
                "anomalylevel": {"value": "99.693"},
                "averagespeed": {"value": "44.3"},
                "concentration": {"value": "6.683"},
                "measuredTime": {"value": "2020-11-22T20:12:34+
    01:00"},
                "vehicleFlow": {"value": "631"}
            },
            {
                "anomalylevel": {"value": "102.790"},
                "averagespeed": {"value": "55.0"},
                "concentration": {"value": "8.620"},
                "measuredTime": {"value": "2020-11-22T20:12:24+
    01:00"},
                "vehicleFlow": {"value": "514"}
            },
            {
                "anomalylevel": {"value": "100.256"},
                "averagespeed": {"value": "33.5"},
                "concentration": {"value": "7.568"},
                "measuredTime": {"value": "2020-11-22T20:12:08+
    01:00"},
                "vehicleFlow": {"value": "299"}
            },
            {
                "anomalylevel": {"value": "104.032"},
                "averagespeed": {"value": "52.0"},
                "concentration": {"value": "4.694"},
                "measuredTime": {"value": "2020-11-22T20:11:40+
    01:00"},
                "vehicleFlow": {"value": "956"}
            },
```

```
128             {
129                 "anomalylevel": {"value": " "},
130                 "averagespeed": {"value": " "},
131                 "concentration": {"value": " "},
132                 "measuredTime": {"value": "2020-11-22T16:03:44+
    01:00"},
133                 "vehicleFlow": {"value": "null"}
134             }
135         ]}
136     }
```


Listing 7.11: http://localhost:8080/SSM2ORION/rest/api/v1/?serviceUri=http://www.disit.org/km4city/resource/Orion:1/urn:ngsi-ld:SensorSite:0&toTIme=2020-11-22T20:13:00

Above was an example of using the "toTime" parameter. This parameter specifies a time range that goes from the creation of the device up to the time specified in the parameter. If we apply the "fromTime" parameter with the value "last-2" to this then only the last two measurements in the specified renge will be returned.


```
1  "Service": {
2          "features": [{
3              "geometry": {
4                  "coordinates": [
5                      13.557755194005033,
6                      52.34689344699744
7                  ],
8                  "type": "Point"
9              },
10             "properties": {
11                 "address": "",
12                 "avgStars": 0,
13                 "brokerName": "orionUNIFI",
14                 "comments": [],
15                 "description": "",
16                 "format": "json",
17                 "linkDBpedia": [],
18                 "macaddress": "",
19                 "maintenanceUrl": "",
20                 "maxCapacity": "",
21                 "minCapacity": "",
22                 "model": "SensorSite",
23                 "multimedia": "",
24                 "name": "urn:ngsi-ld:SensorSite:0",
```

```
                "organization": "DISIT",
                "ownership": "public",
                "photoOrigs": [],
                "photoThumbs": [],
                "photos": [],
                "producer": "Citta di BERLIN",
                "protocol": "ngsi",
                "realtimeAttributes": {
                    "anomalylevel": {
                        "attr_type": "DeviceAttribute",
                        "data_type": "Float",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    },
                    "averagespeed": {
                        "attr_type": "DeviceAttribute",
                        "data_type": "Float",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    },
                    "concentration": {
                        "attr_type": "DeviceAttribute",
                        "data_type": "Float",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    },
                    "vehicleFlow": {
                        "attr_type": "DeviceAttribute",
                        "data_type": "Integer",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    }
                },
                "serviceType": "
TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
resource/Orion:1/urn:ngsi-ld:SensorSite:0",
                "starsCount": 0,
                "typeLabel": "SensorSite",
                "website": "",
                "wktGeometry": ""
            },
```

```
            "type": "Feature"
        }],
        "type": "FeatureCollection"
    },
    "realtime": {
        "head": {"vars": [
            "anomalylevel",
            "averagespeed",
            "concentration",
            "vehicleFlow",
            "measuredTime"
        ]},
        "results": {"bindings": [
            {
                "anomalylevel": {"value": "101.558"},
                "averagespeed": {"value": "79.0"},
                "concentration": {"value": "9.669"},
                "measuredTime": {"value": "2020-11-22T20:12:56+
01:00"},
                "vehicleFlow": {"value": "155"}
            },
            {
                "anomalylevel": {"value": "100.056"},
                "averagespeed": {"value": "46.4"},
                "concentration": {"value": "2.992"},
                "measuredTime": {"value": "2020-11-22T20:12:50+
01:00"},
                "vehicleFlow": {"value": "27"}
            }
        ]}
    }
```


Listing 7.12: http://localhost:8080/SSM2ORION/rest/api/v1/?serviceUri=http://www.disit.org/km4city/resource/Orion:1/urn:ngsi-ld:SensorSite:0&toTIme=2020-11-22T20:13:00&fromTime=last-2

Another way "fromTime" can be used is by specifying how many minutes, hours or days to go back. If the "toTime" parameter is not specified then we go back from the current time.


```
"Service": {
        "features": [{
            "geometry": {
                "coordinates": [
                    13.557755194005033,
```

```
                52.34689344699744
            ],
            "type": "Point"
        },
        "properties": {
            "address": "",
            "avgStars": 0,
            "brokerName": "orionUNIFI",
            "comments": [],
            "description": "",
            "format": "json",
            "linkDBpedia": [],
            "macaddress": "",
            "maintenanceUrl": "",
            "maxCapacity": "",
            "minCapacity": "",
            "model": "SensorSite",
            "multimedia": "",
            "name": "urn:ngsi-ld:SensorSite:0",
            "organization": "DISIT",
            "ownership": "public",
            "photoOrigs": [],
            "photoThumbs": [],
            "photos": [],
            "producer": "Citta di BERLIN",
            "protocol": "ngsi",
            "realtimeAttributes": {
                "anomalylevel": {
                    "attr_type": "DeviceAttribute",
                    "data_type": "Float",
                    "healthiness_criteria": "refresh_rate",
                    "value_refresh_rate": "1"
                },
                "averagespeed": {
                    "attr_type": "DeviceAttribute",
                    "data_type": "Float",
                    "healthiness_criteria": "refresh_rate",
                    "value_refresh_rate": "1"
                },
                "concentration": {
                    "attr_type": "DeviceAttribute",
                    "data_type": "Float",
```

```
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    },
                    "vehicleFlow": {
                        "attr_type": "DeviceAttribute",
                        "data_type": "Integer",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    }
                },
                "serviceType": "
TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
resource/Orion:1/urn:ngsi-ld:SensorSite:0",
                "starsCount": 0,
                "typeLabel": "SensorSite",
                "website": "",
                "wktGeometry": ""
            },
            "type": "Feature"
        }],
        "type": "FeatureCollection"
    },
    "realtime": {
        "head": {"vars": [
            "anomalylevel",
            "averagespeed",
            "concentration",
            "vehicleFlow",
            "measuredTime"
        ]},
        "results": {"bindings": [
            {
                "anomalylevel": {"value": "101.558"},
                "averagespeed": {"value": "79.0"},
                "concentration": {"value": "9.669"},
                "measuredTime": {"value": "2020-11-22T20:12:56+
01:00"},
                "vehicleFlow": {"value": "155"}
            },
            {
                "anomalylevel": {"value": "100.056"},
```

```
                "averagespeed": {"value": "46.4"},
                "concentration": {"value": "2.992"},
                "measuredTime": {"value": "2020-11-22T20:12:50+
    01:00"},
                "vehicleFlow": {"value": "27"}
            }
        ]}
     }
```


Listing 7.13: http://localhost:8080/SSM2ORION/rest/api/v1/?serviceUri=http://www.disit.org/km4city/resource/Orion:1/urn:ngsi-ld:SensorSite:0&toTIme=2020-11-22T20:13:00&fromTime=53-minute

In this example we are going back 53 minutes from the date specified in "toTime".


```
"Service": {
        "features": [{
            "geometry": {
                "coordinates": [
                    13.557755194005033,
                    52.34689344699744
                ],
                "type": "Point"
            },
            "properties": {
                "address": "",
                "avgStars": 0,
                "brokerName": "orionUNIFI",
                "comments": [],
                "description": "",
                "format": "json",
                "linkDBpedia": [],
                "macaddress": "",
                "maintenanceUrl": "",
                "maxCapacity": "",
                "minCapacity": "",
                "model": "SensorSite",
                "multimedia": "",
                "name": "urn:ngsi-ld:SensorSite:0",
                "organization": "DISIT",
                "ownership": "public",
                "photoOrigs": [],
                "photoThumbs": [],
                "photos": [],
```

```
                "producer": "Ctt[U+FFFD]di BERLIN",
                "protocol": "ngsi",
                "realtimeAttributes": {
                    "anomalylevel": {
                        "attr_type": "DeviceAttribute",
                        "data_type": "Float",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    },
                    "averagespeed": {
                        "attr_type": "DeviceAttribute",
                        "data_type": "Float",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    },
                    "concentration": {
                        "attr_type": "DeviceAttribute",
                        "data_type": "Float",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    },
                    "vehicleFlow": {
                        "attr_type": "DeviceAttribute",
                        "data_type": "Integer",
                        "healthiness_criteria": "refresh_rate",
                        "value_refresh_rate": "1"
                    }
                },
                "serviceType": "
    TransferServiceAndRenting_SensorSite",
                "serviceUri": "http://www.disit.org/km4city/
    resource/Orion:1/urn:ngsi-ld:SensorSite:0",
                "starsCount": 0,
                "typeLabel": "SensorSite",
                "website": "",
                "wktGeometry": ""
            },
            "type": "Feature"
        }],
        "type": "FeatureCollection"
    },
    "realtime": {
```

```
        "head": {"vars": [
            "anomalylevel",
            "averagespeed",
            "concentration",
            "vehicleFlow",
            "measuredTime"
        ]},
        "results": {"bindings": [
            {
                "anomalylevel": {"value": "102.606"},
                "averagespeed": {"value": "71.2"},
                "concentration": {"value": "10.280"},
                "measuredTime": {"value": "2020-11-22T21:34:08+
    01:00"},
                "vehicleFlow": {"value": "717"}
            },
            {
                "anomalylevel": {"value": "104.936"},
                "averagespeed": {"value": "48.2"},
                "concentration": {"value": "0.768"},
                "measuredTime": {"value": "2020-11-22T21:33:52+
    01:00"},
                "vehicleFlow": {"value": "623"}
            },
            {
                "anomalylevel": {"value": "102.113"},
                "averagespeed": {"value": "73.7"},
                "concentration": {"value": "8.556"},
                "measuredTime": {"value": "2020-11-22T21:29:08+
    01:00"},
                "vehicleFlow": {"value": "747"}
            },
            {
                "anomalylevel": {"value": "100.487"},
                "averagespeed": {"value": "67.9"},
                "concentration": {"value": "12.364"},
                "measuredTime": {"value": "2020-11-22T21:29:02+
    01:00"},
                "vehicleFlow": {"value": "588"}
            },
            {
                "anomalylevel": {"value": "103.461"},
```

```
                "averagespeed": {"value": "34.4"},
                "concentration": {"value": "10.837"},
                "measuredTime": {"value": "2020-11-22T21:28:56+
    01:00"},
                "vehicleFlow": {"value": "91"}
            },

            {
                "anomalylevel": {"value": "99.652"},
                "averagespeed": {"value": "66.9"},
                "concentration": {"value": "9.051"},
                "measuredTime": {"value": "2020-11-22T21:25:42+
    01:00"},
                "vehicleFlow": {"value": "612"}
            },
            {
                "anomalylevel": {"value": "103.210"},
                "averagespeed": {"value": "53.8"},
                "concentration": {"value": "3.524"},
                "measuredTime": {"value": "2020-11-22T21:25:40+
    01:00"},
                "vehicleFlow": {"value": "454"}
            },
            {
                "anomalylevel": {"value": "102.155"},
                "averagespeed": {"value": "59.4"},
                "concentration": {"value": "3.365"},
                "measuredTime": {"value": "2020-11-22T21:25:08+
    01:00"},
                "vehicleFlow": {"value": "844"}
            }
        ]}
     }
```


Listing 7.14: http://localhost:8080/SSM2ORION/rest/api/v1/?serviceUri=http://www.disit.org/km4city/resource/Orion:1/urn:ngsi-ld:SensorSite:0&fromTime=1-hour

Above, without specifying the departure date, the current date "2020-11-22T22: 08: 00" was used. Not all measurements have been reported because they were about sixty, but the last and the first have been left. It is also important to remember that the "fromTime" parameter can also be specified in the form of a date.

Conclusions

In conclusion it can be said that the primary objective of the thesis has been achieved. As noted in the previous chapter, the tests carried out to verify that the data on Orion was returned correctly by the application gave positive results. However, although correct communication between Orion and SSM2ORION has been ascertained, there is still one last fundamental step. The final step of the thesis was to verify that the solution developed around Orion Context Broker could be integrated with Snap4city. The only way to verify this was to create a dashboard. This dashboard was built on the Snap4City platform following the tutorials that Snap4City provides.
The central element of the dashboard is a map. In our case this map was centered on Berlin because the devices were created with random coordinates in the Berlin area. Selecting the categories corresponding to the sensors created on Orion you will get the map view of the sensors position. Each sensor has the image of the category to which it belongs (**figure** 7.3). The dashboard allows you to select which sensors to show on the map. For

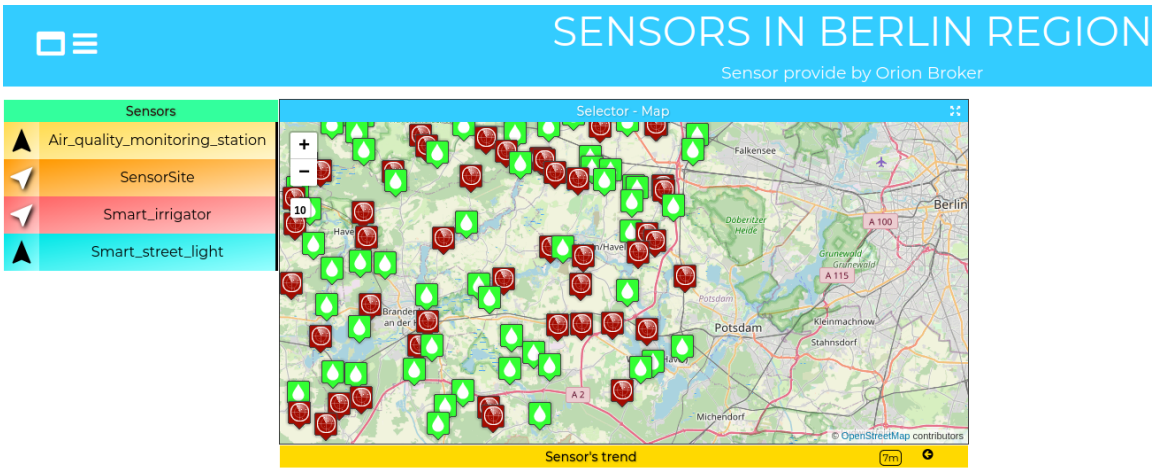

Figure 7.3: Graphical representation of sensors on map

example, by selecting the "SensorSite" type, only the sensors that are currently contained in the area where the map is positioned will be shown. At the start the map is positioned in correspondence of Berlin, however with the mouse it is possible to move there. The dashboard also has other features, it is possible in fact by selecting a sensor directly on the map, to access the history of its attributes.As shown in the figure, it is possible to select four time ranges. Next to the attribute name is the last measured value of the attribute.

Otherwise selecting one or of the buttons such as "last 4hours", under the map a graph will be drawn of the values that that attribute has assumed in that period of time.

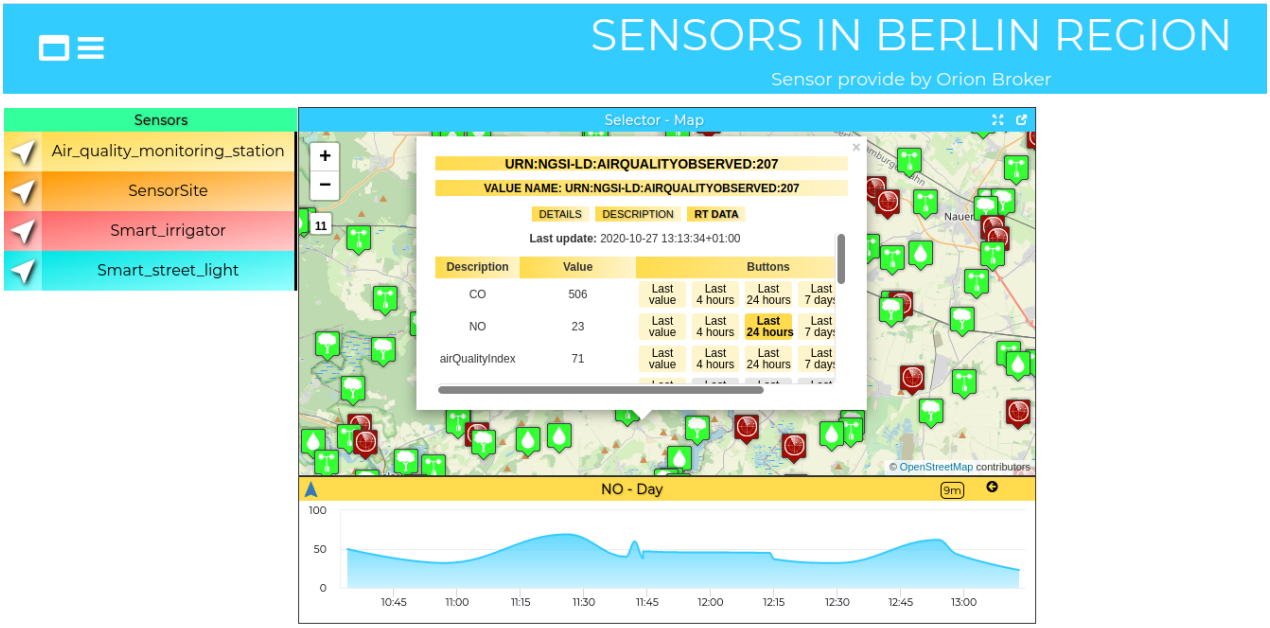

Figure 7.4: Graphical representation of the data history

Appendix A

User manual for SSM2ORION

SSM2ORION quick configuration guide

A.1 Overview

The software consists of:

- **SSM2ORION.war**, web application to be installed on the web server.
- **DataCity-Small**, is the docker implementation of the "DataCity-Small" configured as described in https://www.snap4city.org/drupal/node/471 .
- **SSM2ORION MySQL DB**, database used by the SSM2ORION application for connection with Snap4city and Orion.
- **Time Series Data (CrateDB)**, docker implementation of a FIWARE Context Broker, suite of complementary FIWARE components. One of these components is FIWARE QuantumLeap which allows you to historicize data in a CrateDB database. This suite is available on github: https://github.com/FIWARE/tutorials.Time-Series-Data.git .

A.2 MySQL DB configuration parameters

The settings.xml file present in the .war file is responsible for the connection with the SSM2ORION database. Below is an example of the parameters:

```
<?xml version="1.0" encoding="UTF-8"?>
<setting>
  <label id="username">
    <value>username</value>
  </label>
  <label id="password">
    <value>Password123.</value>
  </label>
```

```
  <label id="urlMySqlDB">
    <value>jdbc:mysql://127.0.0.1:3306</value>
  </label>
  <label id="db">
    <value>SSM2ORION</value>
  </label>
</setting>
```

Listing A.1: File to configure SSM2ORION database

The configuration parameters are the values that the web app uses to access the MySQL-DB SSM2ORION.
e.g.
If the tomcat web server is used, the settings.xml file must be placed in the path */opt/tomcat/bin/*

A.3 SSM2ORION MySQL DB configuration

This database is composed of the tables::

1. categories
2. orion_broker
3. quantumleapZone
4. unitValue

In the **categories** table we have the following columns :

- **orionType**, identifier of the category in the Orion Broker
- **categoriesSSM**, identifier of the category in Snap4City
- **macroCategoriesSSM**, identifier of the macro category in Snap4City

e.g.
INSERT INTO categories (orionType, categoriesSSM, macroCategoriesSSM) VALUES ('AirQualityObserved','Air_quality_monitoring_station','Environment');

In the **orion_broker** table :

- **id**, broker identification name
- **urlPrefix**, url of where Orion Broker is located
- **header**, contains the headers needed to identify the location of where the devices are stored in the mongoDB
- **headerpath**, contains the values of the headers

e.g.
INSERT INTO 'orion_broker' VALUES ('Orion:1','http://localhost:1026','fiware-service','openiot');
INSERT INTO 'orion_broker' VALUES ('Orion:1','http://localhost:1026','fiware-servicepath','/');

In the **quantumleapZone** table we have:

- **url**, url of where Quantumleap is located
- **timeZone**, time zone to be applied if the position of the devices is not along the greenwich meridian

e.g.
INSERT INTO quantumleapZone (port, timeZone) VALUES ('8668','Europe/Berlin');
In the **valueInfo** table we have:

- **valueName**, attribute name
- **measurementUnit**, unit of measurement of the attribute
- **value_type**, generically indicates a data type to reconcile data that come from attributes with different names, but which contain the same physical quantity

e.g.
INSERT INTO 'valueInfo' VALUES ('windSpeed','km/h','speed');

A.4 Deployment of SSM2ORION

In the case of using Tomcat as a web server to deploy through the Tomcat "tomcat manager" interface there are two ways to put the war file in the webapps folder (figure A.1):

1. By choosing the folder where the war
2. By inserting the file into the Tomcat structure and then indicating the name and path of the application

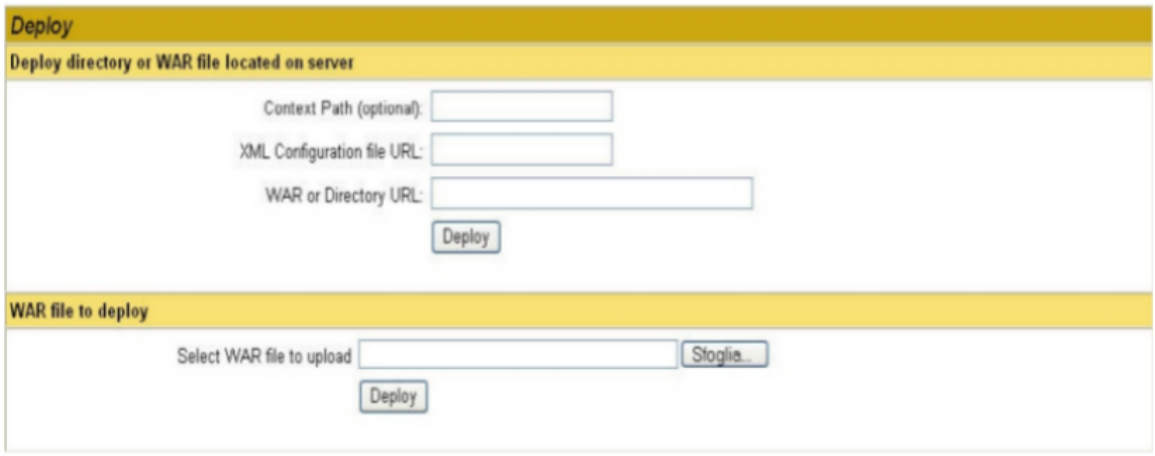

Figure A.1: Tomcat interface

A.5 DataCity-Small configuration

To install this implementation you can follow the steps at: https://github.com/disit/snap4city-docker/blob/master/DataCity-Small/README.md once installation is complete, you need to access the container containing the MySQL-DB "MariaDB"(username = "user" password = "passwordx"). Once logged in, the SuperServiceMap DB must be selected and a row must be added to the servicemaps table.
This table consists of four columns. The two columns of particular interest are the last two:

- **competenceArea**, where a polygon that encloses the service devices must be indicated
- **urlPrefix**, where you need to indicate the url to which the application responds

VALUES ('sm-orion','1.1.1.1','POLYGON(....)','http://<ip dell'host>:<port>/<path>');

e.g.
INSERT INTO servicemaps VALUES ('sm-orion','1.1.1.1','POLYGON((11.988338135956553 53.00382319886812,13.073238038300303 53.00382319886812,13.073238038300303 52.14085136874745,11.988338135956553 52.14085136874745,11.988338135956553 53.00382319886812))','http://192.168.1.102:8080/SSM2ORION/rest');

Subsequently selecting the Dashboard database, it is possible to modify the kbUrl value of the Organizations table where **organizationName** has value *"Organization"*.
e.g.
UPDATE Organizations SET kbUrl = 'http://dashboard/superservicemap/api/v1/' WHERE id = 7;
Once the DB has been updated it is necessary to reload the containers with the *docker-compose restart* command. In order to make the service work, we need to add a line with

<host ip> dashboard in the local /etc/hosts file.
e.g.
192.168.1.102 dashboard Now you can go to http://dashboard/dashboardSmartCity and log in (Username = "userrootadmin" Password = "Sl9.wrE@k").

A.6 Configuring the devices

The IoT Agent node library offers simple REST APIs for accessing, creating and deactivating devices. IOTA_HTTP_PORT (the exposed port where the iot agent for Northbound communications is listening) It is possible to verify that the IoT Agent is running by making an HTTP request to the exposed port.
e.g.
Request:
curl -X GET
'http://localhost:4041/iot/about'
Response:

```
{
    "libVersion": "2.6.0-next",
    "port": "4041",
    "baseRoot": "/",
    "version": "1.6.0-next"
}
```

Listing A.2: Response to http://localhost:4041/iot/about

The first thing to do is to provide a Service Group since it is necessary to provide an authentication key for each device message, and it is also necessary to provide the Iot agent with the url of the broker to which it will respond.
e.g.
Request:
curl -iX POST
'http://localhost:4041/iot/services'
-H 'Content-Type: application/json'
-H 'fiware-service: openiot'
-H 'fiware-servicepath: /'
*-d '**Payload**'*
Headers:

- Content-Type: application/json
- fiware-service: openiot
- fiware-servicepath: /

Payload:

```
{
 "services": [
          {
              "apikey":      "4jggokgpepnvsb2uv4s40d59ov",
              "cbroker":     "http://orion:1026",
              "entity_type": "Thing",
              "resource":    "/iot/d"
            }
          ]
}

```

Listing A.3: Payload sent to http://localhost:4041/iot/services

Response:

HTTP/1.1 201 Created

The details of a Service group can be read by making a GET request to */iot/services* by providing the **resource** parameter.

e.g.

Request:

curl -G -X GET
'http://localhost:4041/iot/services'
-d 'resource=/iot/d'
-H 'fiware-service: openiot'
-H 'fiware-servicepath: /'

Headers:

- Content-Type: application/json
- fiware-service: openiot
- fiware-servicepath: /

Response:

```
{
    "_id": "5b07b2c3d7eec57836ecfed4",
    "subservice": "/",
    "service": "openiot",
    "apikey": "4jggokgpepnvsb2uv4s40d59ov",
    "resource": "/iot/d",
    "attributes": [],
    "lazy": [],
    "commands": [],
```

```
10      "entity_type": "Thing",
11      "internal_attributes": [],
12      "static_attributes": []
13  }
```

Listing A.4: Response to http://localhost:4041/iot/services

It is good practice to use URNs following the NGSI-LD specification (https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.03.01_60/gs_cim009v010301p.pdf) when creating entities. There are two types of attributes:

- **attributes**, are active readings from the device
- **static_attributes**, static data on the device (such as relationships) passed to the broker as the name suggests

e.g.
Request:
curl -iX POST 'http://localhost:4041/iot/devices'
-H 'Content-Type: application/json'
-H 'fiware-service: openiot'
-H 'fiware-servicepath: /'
-d 'Payload'
Headers:

- Content-Type: application/json
- fiware-service: openiot
- fiware-servicepath: /

Payload:

```
1       {
2     "devices": [
3     {
4       "device_id": "airQualityObserved1",
5       "entity_name": "urn:ngsi-ld:AirQualityObserved:1",
6       "entity_type": "AirQualityObserved",
7       "protocol":    "PDI-IoTA-UltraLight",
8       "transport":   "HTTP",
9       "timezone":    "Europe/Berlin",
10      "attributes": [
11       {"name":"CO", "type":"Integer"},
12       {"name":"temperature", "type":"Float"},
13       {"name":"windSpeed", "type":"Integer"},
14       {"name":"relativehumidity", "type":"Float"},
```

```
15          {"name":"NO", "type":"Integer"},
16          {"name":"airQualityIndex", "type":"Integer"},
17          {"name":"precipitation", "type":"Boolean"},
18          {"name":"windDirection", "type":"Integer"}
19
20        ],
21        "static_attributes": [
22          {"name": "category", "type":"Text", "value": ["sensor"]
    },
23          {"name": "city", "type": "Relationship","value": "
    BERLIN"},
24          {"name":"location", "type":"geo:point", "value":"52.778
    5,12.1848"}
25          ]
26      }
27
```

Listing A.5: Payload sent to http://localhost:4041/iot/devices

Response:

HTTP/1.1 201 Created Per avere informazioni sui devices presenti su l'IoT Agent è possibile effettuare una GET request a */iot/devices*.

e.g.

Request:

curl localhost:4041/iot/devices -s -S
-H 'Accept: application/json'
-H 'fiware-service: openiot'
-H 'fiware-servicepath: /' | python -mjson.tool

Response:

```
1  {
2      "count": 1,
3      "devices": [
4          {
5              "attributes": [
6                  {
7                      "name": "CO",
8                      "object_id": "CO",
9                      "type": "Integer"
10                 },
11                 {
12                     "name": "temperature",
13                     "object_id": "temperature",
```

```
                "type": "Float"
            },
            {
                "name": "windSpeed",
                "object_id": "windSpeed",
                "type": "Integer"
            },
            {
                "name": "relativehumidity",
                "object_id": "relativehumidity",
                "type": "Float"
            },
            {
                "name": "NO",
                "object_id": "NO",
                "type": "Integer"
            },
            {
                "name": "airQualityIndex",
                "object_id": "airQualityIndex",
                "type": "Integer"
            },
            {
                "name": "precipitation",
                "object_id": "precipitation",
                "type": "Boolean"
            },
            {
                "name": "windDirection",
                "object_id": "windDirection",
                "type": "Integer"
            }
        ],
        "commands": [],
        "device_id": "airQualityObserved1",
        "entity_name": "urn:ngsi-ld:AirQualityObserved:1",
        "entity_type": "AirQualityObserved",
        "lazy": [],
        "protocol": "PDI-IoTA-UltraLight",
        "service": "openiot",
        "service_path": "/",
        "static_attributes": [
```

```
                    {
                        "name": "category",
                        "type": "Text",
                        "value": [
                            "sensor"
                        ]
                    },
                    {
                        "name": "city",
                        "type": "Relationship",
                        "value": "BERLIN"
                    },
                    {
                        "name": "location",
                        "type": "geo:point",
                        "value": "52.7785,12.1848"
                    }
                ],
                "transport": "HTTP"
            }
        ]
}
```

Listing A.6: Response to http://localhost:4041/iot/devices

A.7 Ultralight measurements over HTTP from the IoT devices

For an Ultralight IoT Agent the devices will send GET and POST requests to: http://iot-agent:7896/iot/d?i=<device_id>k=4jggokgpepnvsb2uv4s40d59ov
e.g.
Request:
*curl -iX POST 'http://localhost:7896/iot/d?k=4jggokgpepnvsb2uv4s40d59ovi=motion001' -H 'Content-Type: text/plain' -d '**Payload**'*
Headers:

- Content-Type: text/plain

Payload:
'NO|24|precipitation|true|CO|240|temperature|15.46|windDirection|75|windSpeed|25|airQualityIndex|87|relativehumidity|57.23'

Response:
HTTP/1.1 200 OK

To check whether the Orion Context Broker has registered or not the values that the devices send to the IoT Agent.

e.g.

Request:

curl -X GET 'http://localhost:1026/v2/entities/urn:ngsi-ld:AirQualityObserved:1'
-H 'Accept: application/json'
-H 'fiware-service: openiot'
-H 'fiware-servicepath: /' | python -mjson.tool

Headers:

- Content-Type: application/json
- fiware-service: openiot
- fiware-servicepath: /

Response:

```
{
    "CO": {
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-10-21T13:52:56.00Z"
            }
        },
        "type": "Integer",
        "value": "240"
    },
    "NO": {
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-10-21T13:52:56.00Z"
            }
        },
        "type": "Integer",
        "value": "24"
    },
    "TimeInstant": {
        "metadata": {},
        "type": "DateTime",
        "value": "2020-10-21T13:52:56.00Z"
    },
```

```
    "airQualityIndex": {
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-10-21T13:52:56.00Z"
            }
        },
        "type": "Integer",
        "value": "87"
    },
    "category": {
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-10-21T13:52:56.00Z"
            }
        },
        "type": "Text",
        "value": [
            "sensor"
        ]
    },
    "city": {
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-10-21T13:52:56.00Z"
            }
        },
        "type": "Relationship",
        "value": "BERLIN"
    },
    "id": "urn:ngsi-ld:AirQualityObserved:1",
    "location": {
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-10-21T13:52:56.00Z"
            }
        },
        "type": "geo:point",
        "value": "52.7785,12.1848"
```

```
    },
    "precipitation": {
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-10-21T13:52:56.00Z"
            }
        },
        "type": "Boolean",
        "value": true
    },
    "relativehumidity": {
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-10-21T13:52:56.00Z"
            }
        },
        "type": "Float",
        "value": "57.23"
    },
    "temperature": {
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-10-21T13:52:56.00Z"
            }
        },
        "type": "Float",
        "value": "15.46"
    },
    "type": "AirQualityObserved",
    "windDirection": {
        "metadata": {
            "TimeInstant": {
                "type": "DateTime",
                "value": "2020-10-21T13:52:56.00Z"
            }
        },
        "type": "Integer",
        "value": "75"
    },
```

```
111     "windSpeed": {
112         "metadata": {
113             "TimeInstant": {
114                 "type": "DateTime",
115                 "value": "2020-10-21T13:52:56.00Z"
116             }
117         },
118         "type": "Integer",
119         "value": "25"
120     }
121 }
```

Listing A.7: Response to http://localhost:1026/v2/entities/<device-id>

Bibliography

[1] What is an iot platform?, 2020, URL: https://www.kaaproject.org/blog/what-is-iot-platform.

[2] What is fiware?, 2018, URL: https://www.fiware.org/about-us/.

[3] What is a smart city?, 2020, URL: https://www.thalesgroup.com/en/markets/digital-identity-and-security/iot/inspired/smart-cities.

[4] The definition of a smart city, 2020, URL: https://rg.smartcitiescouncil.com/readiness-guide/article/definition-definition-smart-city.

[5] Nor Badrul Anuar Kayode Adewole Ibrar Yaqoob Abdullah Gani Ejaz Ahmed Haruna Chiroma Ibrahim Abaker Targio Hashem, Victor Chang. The role of big data in smart city. *International Journal of Information Management*, 32:748–758, October 2016.

[6] Fiware: The open source platform for our smart digital future accessed, 2010, URL: https://www.fiware.org/about-us/ Google Scholar.

[7] N. Mohamed J. Al-Jaroodi. Middleware is still everywhere!!! *Concurrency Comput. Pract. Exper*, 24:1919–1926, Nov. 2012.

[8] Fiware-ngsi v2 specification, 2018, URL: http://telefonicaid.github.io/fiware-orion/api/v2/stable/.

[9] Core context management, 2020, URL: https://github.com/Fiware/catalogue/tree/master/core.

[10] Quantumleap, 2019, URL: https://quantumleap.readthedocs.io/en/latest/.

[11] T. Storek. Application of the open-source cloud platform fiware for future building energy management systems. *J. Phys.: Conf. Ser.*, 1343, 2019.

[12] What is an iot agent?, 2020, URL: https://fiware-tutorials.readthedocs.io/en/latest/iot-agent/index.html.

[13] Wikipedia cratedb, 2020, URL: https://en.wikipedia.org/wiki/CrateDB.

[14] Wikipedia node-red, 2020,URL: https://en.wikipedia.org/wiki/Node-RED.

[15] Soppelsa F and Kaewkasi C. Native docker clustering with swarm. *(Packt Publishing)*, ISBN:1786469758, 9781786469755, 2017.

[16] Persisting and querying time series data (cratedb), 2020, URL: https://github.com/FIWARE/tutorials.Time-Series-Data.

[17] What is an iot agent?, 2020, URL: https://github.com/FIWARE/tutorials.IoT-Agent.

[18] Architecture, 2019, URL: https://github.com/telefonicaid/iotagent-node-lib/blob/master/doc/architecture.md.

[19] The Editors of Encyclopaedia Britannica. Protocol. *Encyclopædia Britannica*, August 31, 2018, URL: https://www.britannica.com/technology/protocol-computer-science.

[20] User & programmers manual, 2019, URL: https://fiware-iotagent-ul.readthedocs.io/en/latest/usermanual/index.html.