

CAPTOR

Collective Awareness Platform for Tropospheric Ozone Pollution

Work package	WP2
Deliverable number	D2.4
Deliverable title	Open Link Data Repository Development
Deliverable type	R
Dissemination level	PU (Public)
Estimated delivery date	M9
Actual delivery date	10/10/2016
Actual delivery date after EC review	15/09/2017
Version	2.0
Comments	

Authors

Jose M. Barcelo-Ordinas, Albert Cerezo Llaveró, Biel Pieras Morell,
Jorge Garcia-Vidal, Manel Guerrero-Zapata (UPC)

Document History			
Version	Date	Contributor	Comments
V0.1	20/07/2016	Jose M. Barcelo-Ordinas (UPC)	Outline of contents
V0.2	10/09/2016	Jose M. Barcelo-Ordinas, Albert Cerezo Llaveró, Biel Pieras Morell, Jorge Garcia-Vidal, Manel Guerrero-Zapata (UPC)	First version
V0.3	7/10/16	Roger Pueyo Centelles (GUI)	Peer Review
V1.0	10/10/2016	Jose M. Barcelo-Ordinas, Jorge Garcia-Vidal (UPC)	Final Version
V2.0	15/09/2017	Jose M. Barcelo-Ordinas, Jorge Garcia-Vidal (UPC)	Final version after EC review

Table of Contents

List of Abbreviations.....	5
Executive Summary.....	6
1. Research Context.....	7
2. Software repository.....	8
3. Technological Context.....	8
3.1 Semantic Web.....	8
3.2 Linked Data.....	9
3.3 RDF.....	9
3.4 Ontologies.....	9
3.4.1 SSN.....	10
3.4.2 GEO.....	10
3.4.3 FOAF.....	10
3.4.4 CC.....	10
3.4.5 DC.....	10
3.4.6 DUL.....	10
3.5 Django.....	12
3.6 Bootstrap.....	12
3.7 API.....	12
3.8 REST.....	12
3.9 NodeJS.....	12
3.10 Express.....	12
4. Platform Design.....	12
4.1 MySQL database design.....	13
4.2 New ontology for CommSensum platform.....	15
4.3 Model mapping between the database and the extended ontology.....	18
4.4 API.....	19
4.4.1 POST.....	20
4.4.2 GET.....	20
5. Website.....	22
6. Conclusions.....	29

List of Figures

Figure 1. CommSensum system Architecture.....	9
Figure 2. Simplified SSN Ontology Graph.....	11
Figure 3. Entities conceptual map.....	13
Figure 4. CommSensum database design.....	15
Figure 5. CommSensum ontology based model.....	16
Figure 6. API workflow.....	20
Figure 7. Sensor data output example in graphical view.....	21
Figure 8. Website view structure map.....	24
Figure 9. User control panel.....	25
Figure 10. Administrator control panel.....	26
Figure 11. Add stream view.....	27
Figure 12. API output recollected in a graph.....	28
Figure 13. Edit project view.....	29

List of Tables

Tabla 1. Model entities attributes.....	14
Tabla 2. Mapping between MySQL database entities and the extended ontology.....	19

List of Abbreviations

API	Application Programming Interface
DC	Dublin Core
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering
DnS	Descriptions and Situations
DUL	DOLCE + ultra-lite DnS
FLOSS	Free, Libre, Open Source Softwar
FOAF	Friend of a friend
GEO	GeoNames Ontology
HTTP	Hypertext Transfer Protocol
ICT	Information and Communication Technology
IoT	Internet of Things
JSON	Javascript Object Notation
OLD	Open Link Data
OWL	Ontology Web Language
RDF	Resource Description Framework
REST	Representational State Transfer
SQL	Structured Query Language
SSN	Semantic Sensor Network
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	eXtensible Markup Language

Executive Summary

Description of the work

The data collected by the project will be published as Open Linked Data (OLD), using semantic web techniques (RDF/XML and RDF/JSON data syntax) on the *CommSensum* platform. This tool has been developed by UPC as FLOSS, and provides access to the data based on a RESTful API as an extension to the SSN ontology for sensors data.

The project will contribute to provide a scalable and stable Open Linked Data repository available to the scientific and regulatory community and also to the other ICT tools in the project (e.g. the mobile app *AirAct*, developed by EEA and UPC, already uses data from *CommSensum* platform).

The *CommSensum* platform software is published under the Apache 2.0 license and the source code is available on the GitHub platform:

<https://github.com/Commsensum/commsensum>

Objectives

This deliverable covers the following topics and issues:

- Description of the functionalities of the repository platform called CommSensum
- Description of the technologies used in the development of the CommSensum platform
- Description of the back-end of the CommSensum platform, including database types and entities
- Description of the ontology extension to support RDF connections
- Description of the CommSensum front-end

1. Research Context

This deliverable presents *CommSensum*, an Open Link Data community sensing platform built with the purpose of aggregating the users that generate information and the users that are working with it. This platform plays a key role in the CAPTOR project, where the availability and quality of the data is one of the most important factors being worked on.

The software is openly available in <https://github.com/Commsensum/commsensum>.

From the data collection and storage point of view, the *CommSensum* platform is appealing for users because the sensors can be easily linked to it, without technical expertise, facilitating the deployment of these devices. Additionally, the RESTful API provided by the platform to supply the data eases the development of applications.

The user can share her data with other participants, organizations or groups of interest. For example, a user could choose to share data with environmental organizations or with neighbourhood platforms. To make data reuse easier, the system uses standardized protocols and data formats (e.g. it can standardize the predefined words that must be used to obtain a given data such as temperature ($^{\circ}\text{C}$) or ozone (mg/m^3)).

The data available at the *CommSensum* platform may be integrated with data coming from other repositories, forming part of what is called Open Link Data (OLD) in the context of semantic web. This can be achieved thanks to the fact that data is supplied using RDF (Resource Description Framework), a general method for conceptual description or modelling of information that is implemented in web resources, and because data are compliant with standard sensor data ontologies like Semantic Sensor Network (SSN) and many other ontologies described in sections 2 and 3.

There are a number of projects and platforms focused on collective environmental monitoring (Citi-Sense¹, that is part of the Citizen Observatory platform², Air Quality Egg³, CitySense⁴, Aircasting⁵, etc.), but almost all of them provide a closed architecture and are designed for their specific applications. An example is the Air Quality Egg, which shows NO_2 and CO concentrations at the localization of the node. Furthermore, the readings obtained by these platforms can not be used by other applications such as supplementing routine ambient air monitoring, enhancing source compliance monitoring or OpenSense⁶ that provides data for building air pollution maps in an urban area.

The *CommSensum* platform uses Open Linked Data for sharing sensor data in the Web and provides the following general characteristics:

- **Quality of the data:** most of the “cheap” sensors have not been tested under real conditions and, thus, the performance of the measured data may produce meaningless information to the applications. In addition to that, even if the sensors would initially

¹ Citi-Sense: <https://citi-sense.nilu.no>

² Citizen’s Observatory: <http://www.citizen-obs.eu>

³ Air Quality Egg: <http://airqualityegg.com>

⁴ R. N. Murty, G. Mainland, I. Rose, A. R. Chowdhury, A. Gosain, J. Bers, M. Welsh, Citysense: An urban-scale wireless sensor network and testbed, in: IEEE Conference on Technologies for Homeland Security, 2008, pp. 583–588.

⁵ Aircasting: <http://www.aircasting.org>

⁶ OpenSense: <http://www.opensense.ethz.ch/trac>

provide data with enough quality, it is a fact that most sensors lose calibration over time and need to be recalibrated. Calibration is, in fact, one of the biggest challenges in participatory sensing and it is still an open research problem. Techniques to automatically detect the de-calibration of sensors or whether the data are accurate pose a challenge in air pollution participatory networks.

- **Open Linked Data:** It is important to have a standard common semantic that enables all participants to produce data with the same representation in a participatory scenario with multiple platforms. The SSN (Semantic Sensor Network) ontology⁷ is an effort to describe sensors and sensor networks for using them in web applications.
- **Data Warehouse:** sensor data have to be stored somewhere. Participants take part in collecting the data but most of them are individuals mainly interested in that this data is shared with others. Some platforms, such as Xively⁸ and Carriots⁹, allow participants to register their nodes and define the representation of the measured data, offering a cloud service to store their own private data. On the other hand, *CommSensum* uses Open Linked Data (OLD) technology that allows users to access the data of other participants, while the applications can also retrieve the data contributed by all the users.
- **Security, privacy and data ownership:** the data has to be secured at communication level and at storage level.

2. Software repository

The software described in this deliverable is made available as a FLOSS project under the Apache 2.0 license at the GitHub platform:

<https://github.com/Commsensum/commsensum>.

3. Technological Context

In this section all the technologies used in the project and the relevant concepts for the understanding in the design of the platform are briefly described. Figure 1 presents the system architecture for CommSensum and the technologies that have been used in the development of the platform. These technologies are briefly described in the following subsections.

3.1 Semantic Web

The Semantic Web is a project that intends to add computer-processable meaning (semantics) to the Word Wide Web. In February 2004, The W3C released the Resource Description Framework (RDF) and the OWL Web Ontology Language (OWL) as W3C Recommendations. RDF is used to represent information and to exchange knowledge in the web. OWL is used to publish and share sets of terms called ontologies, supporting advanced Web search, software agents and knowledge management.

⁷ L. Lefort, C. Henson, K. Taylor, P. Barnaghi, M. Compton, O. Corcho, R. Garcia-Castro, J. Graybeal, A. Herzog, K. Janowicz, et al., "Semantic sensor network final report", W3C Incubator Group Report 28.

⁸ Xively: <https://xively.com/>

⁹ Carriots: <https://www.carriots.com>

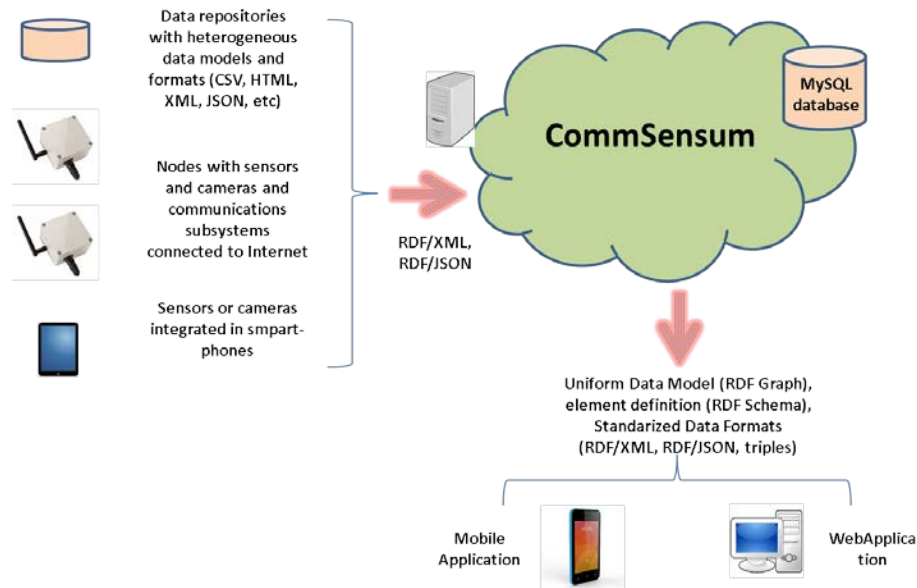


Figure 1. CommSensum system Architecture

3.2 Linked Data

The term Linked Data refers to a set of best practices for publishing and connecting structured data on the Web. Key technologies that support Linked Data are URIs (a generic means to identify entities or concepts in the world), HTTP (a simple yet universal mechanism for retrieving resources, or descriptions of resources), and RDF.

3.3 RDF

Resource Description Framework (RDF) is an abstract model, a way to break down knowledge into discrete pieces, with some rules about the semantics, or meaning, of those pieces. While it is most popularly known for its RDF/XML syntax, RDF can be stored in a variety of formats. Examples of RDF serialisations include RDF/XML, Notation-3 (N3), Turtle, N-Triples, RDF and RDF/JSON. Although RDFs enables the definition of classes, relationships and attributes, these features are in most of the cases insufficient to model a full ontology because of the existence of certain constraints (like cardinality or class dis-jointness among others). That's the reason why the Web Ontology Language (OWL) was defined.

3.4 Ontologies

A specification of a representational vocabulary for a shared domain of discourse - definitions of classes, relations, functions, and other objects - is called an ontology. Ontologies are useful for sharing common understanding of the structure of information among people or software agents, enabling reuse of domain knowledge, making domain assumptions explicit, separating domain knowledge from the operational knowledge and analysing domain knowledge. In our case, ontologies are used specially for its ability to share common understanding of the structure of information among people or software agent. The ontologies used are SSN for the representation of sensor metadata, GEO for the geospatial data representation, FOAF for the representation of users, CC for the copyright information, DC for describing generic metadata and DUL as the supporting upper-level ontology. Upper level ontologies are used to facilitate the semantic integration of domain ontologies and guide the development of new ontologies. For this purpose, they contain general categories that are applicable across multiple domains.

3.4.1 SSN

The SSN ontology is based around concepts of systems, processes, and observations. It supports the description of the physical and processing structure of sensors. Sensors are not constrained to physical sensing devices: rather a sensor is anything that can estimate or calculate the value of a phenomenon, so a device or computational process or combination could play the role of a sensor. The representation of a sensor in the ontology links together what it measures (the domain phenomena), the physical sensor (the device) and its functions and processing (the models). In Figure 2 there's a simplified version of the ontology.

3.4.2 GEO

The GeoNames Ontology makes it possible to add geospatial semantic information to the World Wide Web. All over 8.3 million geo-names toponyms now have a unique URL with a corresponding RDF web service. In our case it's used for describing the latitude, altitude and longitude of a sensing device.

3.4.3 FOAF

FOAF is a machine-readable ontology describing persons, their activities and their relations to other people and objects. It is used to describe people and social relationship on the Web and it's mostly focused on people's existence in the virtual world. In the CommSensum system it is used for describing users and projects.

3.4.4 CC

The Creative Commons Rights Expression Language (CC REL) lets you describe copyright licenses in RDF. In our case it is used for describing the license under which a project is registered.

3.4.5 DC

Dublin Core is a light weight RDFS vocabulary for describing generic metadata. It is ubiquitous in the linked data landscape.

3.4.6 DUL

DUL is a lighter OWL axiomatization of DOLCE and DnS (Descriptions and Situations) which also simplifies the names of many classes and properties, adds extensive inline comments, and thoroughly aligns to the repository of Content patterns. DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) is the first module of the Wonder Web foundational ontologies library. It has a clear cognitive bias, in that it aims at capturing the ontological categories underlying natural language and human common sense. DnS (Descriptions and Situations) is a constructivist ontology that pushes DOLCE's descriptive stance even further. DnS does not put restrictions on the type of entities and relations that one may want to postulate, either as a domain specification, or as an upper ontology, and it allows for context-sensitive re-descriptions of the types and relations postulated by other given ontologies.

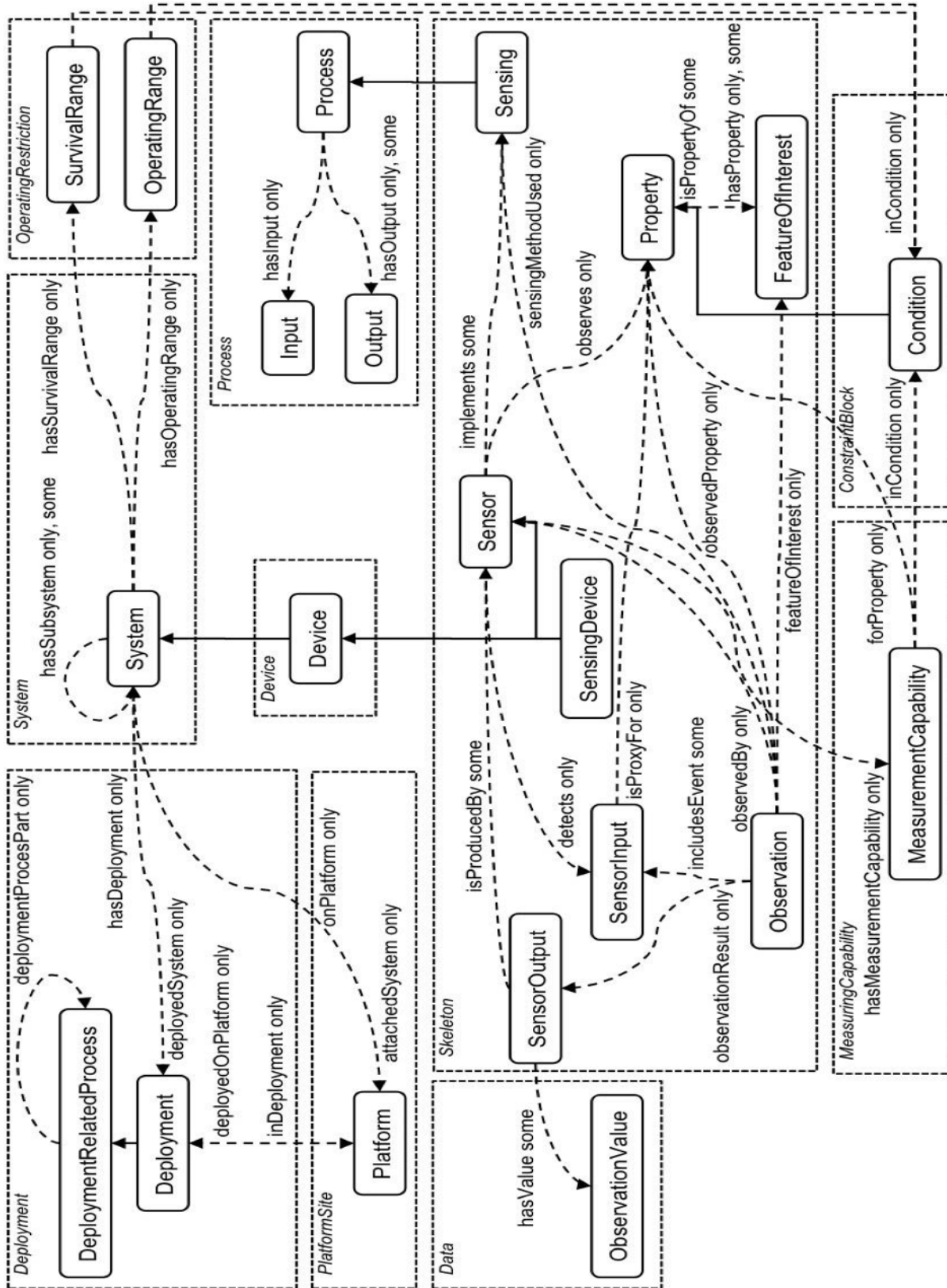


Figure 2. Simplified SSN Ontology Graph

3.5 Django

Django is an open source web application framework, written in Python, which follows the model-view-controller architectural pattern. It was originally developed to manage several news-oriented sites for The World Company of Lawrence, Kansas, and was released publicly under a BSD license in July 2005. Django's primary goal is to ease the creation of complex database-driven websites. It emphasizes on reusability, pluggability of components and rapid development.

3.6 Bootstrap

Bootstrap is a free collection of tools for creating websites and web applications developed by Mark Otto for Twitter. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions.

3.7 API

An application-programming interface (API) is a set of programming instructions and standards for accessing a web-based software application. A software company releases its API to the public so that other software developers can design products that are powered by its service.

3.8 REST

Representational state transfer (REST) is a software architectural style consisting of a coordinated set of architectural constraints applied to components, connectors, and data elements, within a distributed hypermedia system. RESTful APIs are defined with these aspects:

- base URI such as <http://commsensum.pc.ac.upc.edu/>
- an Internet media type for the data which is often JSON
- standard HTTP methods (e.g., GET, PUT, POST, or DELETE)
- hypertext links to reference state
- hypertext links to reference related resources

3.9 NodeJS

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices. Web development in a dynamic language as JavaScript on a virtual machine like V8 is much faster than Ruby, Python, or Perl. It has the ability to handle thousands of concurrent connections with minimal overhead on a single process.

3.10 Express

Express is a minimal and flexible node.js web application framework, providing a robust set of features for building single and multi-page, and hybrid web applications. It provides a myriad of HTTP utility methods and Connect middleware which allow creating a robust user-friendly API be quick and easy.

4. Platform Design

In order to store all the information that the sensors gather, it is required to create a model determining which entities from the real world exist and act in our system. A first model

implements a MySQL database as described in section 3.1. Secondly, in subsection 3.2 it is explained the ontology chosen for the CommSensum platform. Finally, in subsection 3.3 the mapping between the MySQL database and the RDF with the ontology defined is described.

4.1 MySQL database design

The first model designed is explained in this simple conceptual map in Figure 3. A user can create a project or join an existing one which was created by another user. This relationship is determined by a role which can be: **Administrator**, only the administrators can control the users that join the project and they also can send and read all the sensor data related to the project; **Collaborator**, collaborators can send and read all the sensor data related to the project; **Observer**, observers are only allowed to read the sensor data related to the project.

A project has several streams of data which are considered a collection of data related to each other coming from one unique source, e.g., a sensor may have 2 streams where in the first one the battery level measurements are sent and in the latter the humidity level in the air.

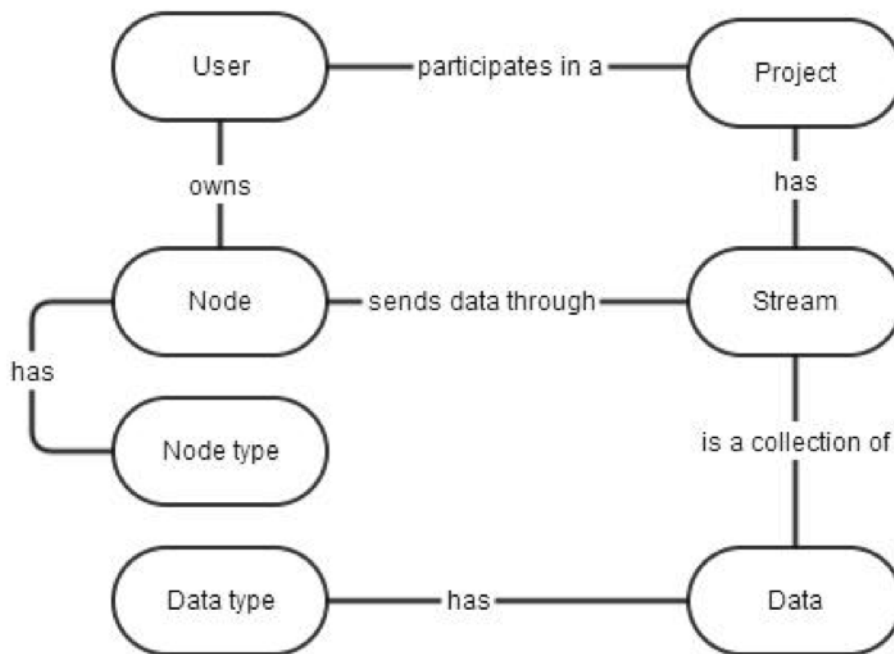


Figure 3

Figure 3. Entities conceptual map.

Users need to register their nodes so as to be able to send the data to the system, which allows establishing the ownership of the node. This is important in order to determine which user can modify the node properties and which user should be notified when the system detects that the node is not responsive anymore. Node types are used to cluster nodes by their capabilities. Data meta-data is stored as data type and it's connected to the actual data.

The attributes of each entity in the model explained before are listed in Table 1. It is important to emphasize that in the node entity there's a geo-positioning redundancy since the city, country and region could be obtained through the latitude and longitude, but it was decided to keep both ways of expressing it in order to make it easier for developers.

Entity	Attributes
User	first name last name email APIkey
Project	Name Description License
Node	Name Description Timezone Status Serial Number Enabled Latitude Longitude Altitude City Country Region
NodeType	Name Description
Stream	Name Description Last update
Data	Value Date
DataType	Name Units Description

Tabla 1. Model entities attributes.

Figure 5 represents the MySQL implementation of the model described in section 3.1. Every box represents a table in the database stating all the columns and its data type. Every table has at least one small yellow key that represents the primary key. The red dot symbolizes that this column is a foreign key to another table.

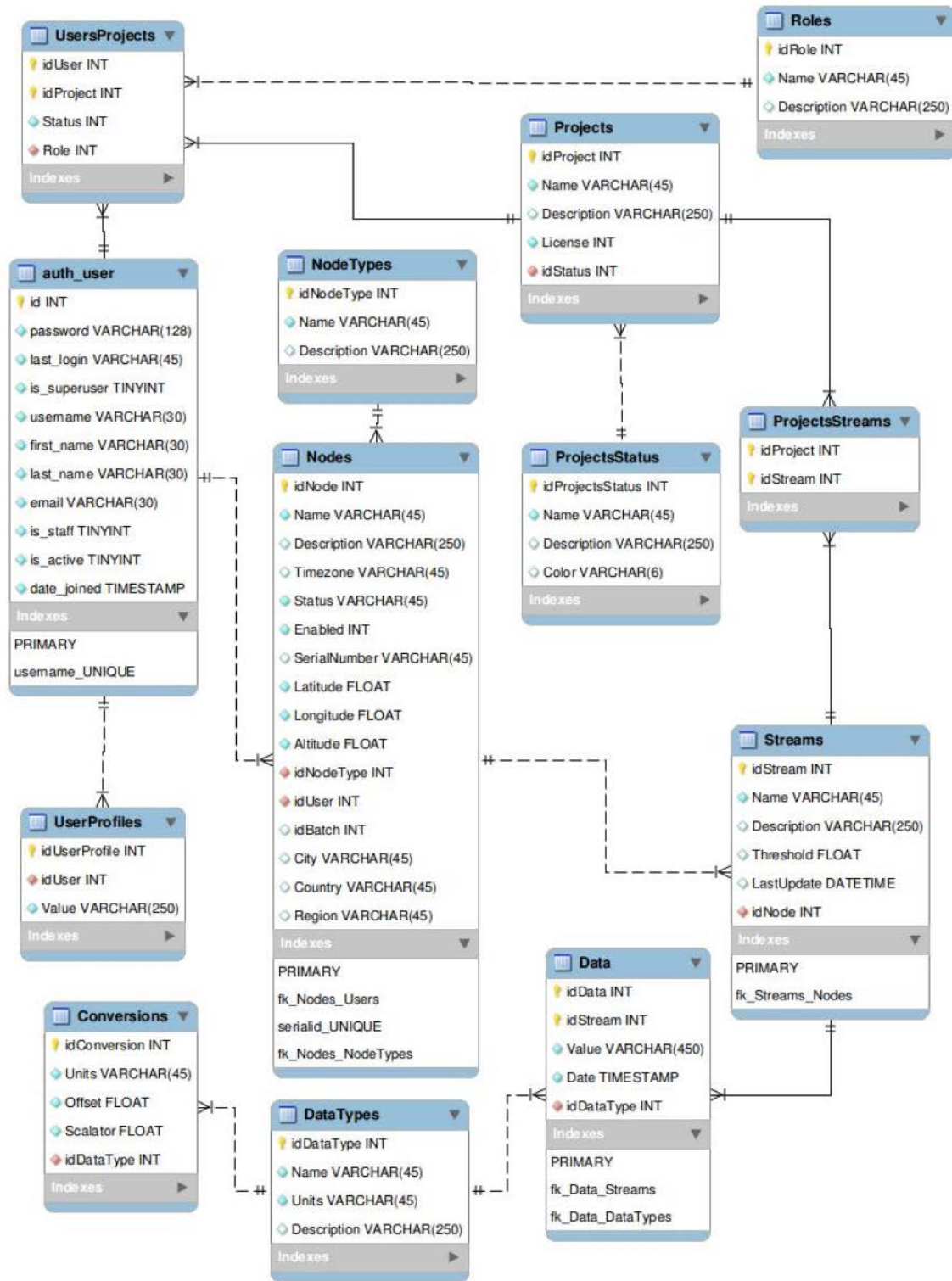


Figure 4. CommSensum database design

4.2 New ontology for CommSensum platform

The model presented in section 3.1 is based on MySQL and does not initially support RDF. In order to represent the data in RDF, the ontologies presented in section 2.4 have to be extended

and an intermediate level that communicates with the MySQL database has to be built.

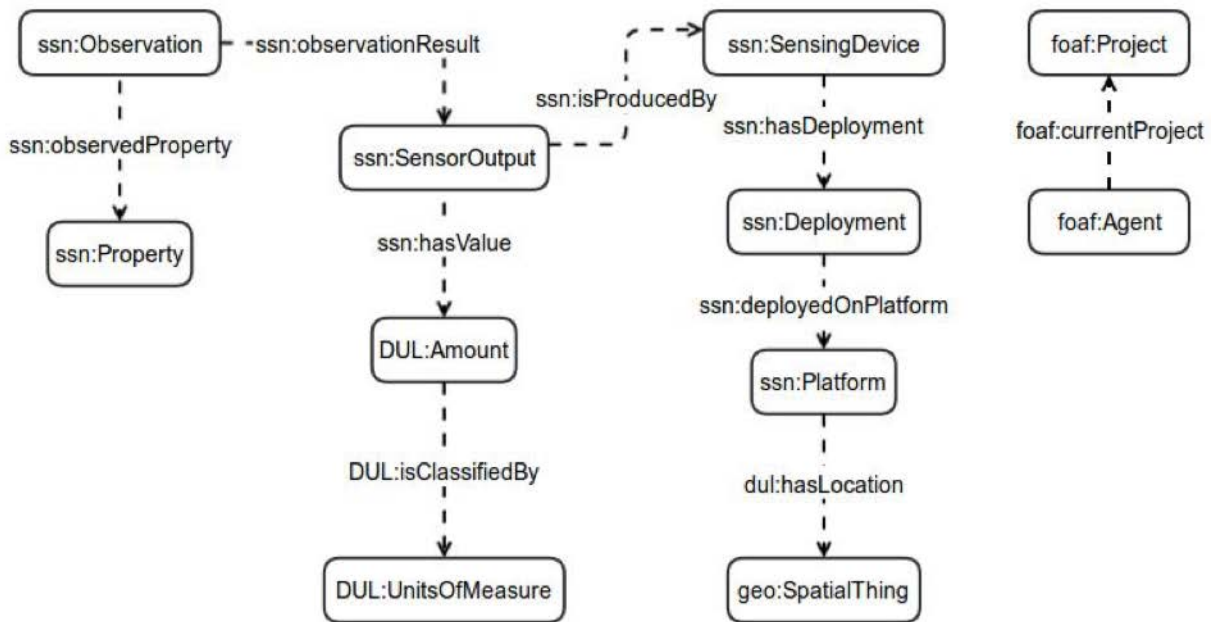


Figure 5. CommSensum ontology based model

To represent all the information required, the system was modelled extending the SSN ontologies. The resulting new ontology can be seen in Figure 5. Each of the entities and their relationships will now be described:

• **ssn:Observation** (<http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#Observation>):

An Observation is a Situation in which a Sensing method has been used to estimate or calculate a value of a Property of a FeatureOfInterest. Links to Sensing and Sensor describe what made the Observation and how; links to Property and Feature detail what was sensed; the result is the output of a Sensor; other metadata details times etc.

Attributes:

- rdfs:label
- ssn:observationResultTime

• **ssn:Property** (<http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#Property>):

An observable Quality of an Event or Object. That is, not a quality of an abstract entity as is also allowed by DUL's Quality, but rather an aspect of an entity that is intrinsic to and cannot exist without the entity and is observable by a sensor.

Attributes:

- rdfs:label
- rdfs:comment

• **ssn:SensorOutput** (<http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#SensorOutput>)

A sensor outputs a piece of information (an observed value), the value itself being represented by an ObservationValue.

• **DUL:Amount** (<http://purl.org/ifgi/dul#Amount>)

A quantity, independently from how it is measured, computed, etc.

• **DUL:UnitsOfMeasure** (<http://purl.org/ifgi/dul#UnitOfMeasure>)

Units of measure are conceptualized here as parameters on regions, which can be valued as data type values.

• **ssn:SensingDevice** (<http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#SensingDevice>)

A sensing device is a device that implements sensing.

Attributes:

- rdfs:label
- rdfs:comment

• **ssn:Deployment** (<http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#Deployment>)

The ongoing Process of Entities (for the purposes of this ontology, mainly sensors) deployed for a particular purpose. For example, a particular Sensor deployed on a Platform, or a whole network of Sensors deployed for an observation campaign. The deployment may have sub processes, such as installation, maintenance, addition, and decommissioning and removal.

• **ssn:Platform** (<http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#Platform>)

An Entity to which other Entities can be attached - particularly Sensors and other Platforms. For example, a post might act as the Platform, a buoy might act as a Platform, or a fish might act as a Platform for an attached sensor.

• **geo:SpatialThing** (http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing)

A point, typically described using a coordinate system relative to Earth, such as WGS84. Uniquely identified by latitude, longitude and altitude.

Attributes:

- geo:lat
- geo:long
- geo:alt

• **foaf:Project** (http://xmlns.com/foaf/spec/#term_Project)

The Project class represents the class of things that are 'projects'. These may be formal or informal, collective or individual. It is often useful to indicate the homepage of a Project.

Attributes:

- dc:title
- dc:description
- cc:license

• **foaf:Agent** (http://xmlns.com/foaf/spec/#term_Agent)

The Agent class is the class of agents; things that do stuff.

Attributes:

- foaf:name
- foaf:nick
- foaf:mbox

• **addr:Address** (<http://wiki.foaf-project.org/w/AddressVocab>) The Address class represents a region in space using political boundaries.

Attributes:

- addr:thoroughfareName
- addr:town
- addr:region
- addr:country

• **ssn:ObservationResult**

(<http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#observationResult>)

Relation linking an Observation (i.e., a description of the context, the Situation, in which the observation in was made) and a Result, which contains a value representing the value associated with the observed Property.

• **ssn:ObservedProperty**

(<http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#observedProperty>)

Relation linking an Observation to the Property that was observed. The observedProperty should be a Property (hasProperty) of the FeatureOfInterest (linked by featureOfInterest) of this observation.

• **ssn:isProducedBy** (<http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#isProducedBy>)

Relation between a producer and a produced entity: for example, between a sensor and the produced output.

• **ssn:hasValue** (<http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#hasValue>)

• **DUL:isClassifiedBy** (<http://purl.org/ifgi/dul#isClassifiedBy>) A

relation between a Concept and an Entity

• **ssn:hasDeployment**

(<http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#hasDeployment>)

Relation between a System and a Deployment, recording that the System/Sensor was deployed in that Deployment.

• **ssn:deployedOnPlatform**

(<http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#deployedOnPlatform>)

Relation between a deployment and the platform on which the system was deployed.

• **DUL:hasLocation** (<http://purl.org/ifgi/dul#hasLocation>) A

generic, relative localization, holding between any entities.

• **foaf:currentProject** (http://xmlns.com/foaf/spec/#term_currentProject)

A currentProject relates a Person to a Document indicating some collaborative or individual undertaking. This relationship indicates that the Person has some active role in the project, such as development, coordination, or support.

• **Foaf:maker** (http://xmlns.com/foaf/spec/#term_maker)

The maker property relates something to an Agent that made it.

4.3 Model mapping between the database and the extended ontology

Using the information stored in the database, which had been implemented following the first model, it was required to add a mapping layer between models. Since both models represent the same part of the world but defining different relations and entities, there should exist a

mapping function between them. Table 2 shows from which former entities implementation do the new entities gather the information. Those new entities that are not in the table don't store any literals but they are the vehicle to relate other entities that store them.

Ontology	Database
ssn:SensingDevice	Node
geo:SpatialThing	Node
foaf:Agent	User
foaf:Project	Project
DUL:UnitsOfMeasure	DataType
DUL:Amount	Data
ssn:SensorOutput	Stream Node
ssn:Observation	Data DataType
ssn:Property	DataType

Tabla 2. Mapping between MySQL database entities and the extended ontology.

4.4 API

The API has been developed using NodeJS in conjunction with the Express web application framework. The CommSensum API serves two purposes: (i) storing the data that the sensors send to the system and (ii) answering to requests that are used to query the data stored in the database.

These two kinds of requests can be sent using the POST and GET HTTP methods. Sensors can send the data directly to the Commsensum platform or, alternatively, this data can be first stored in an intermediate format (e.g. CSV) to be introduced in the platform later.

A security layer on top of both methods has been implemented in order to protect the system from fraudulent data insertion and to protect the user's data they don't want to be shared.

This security layer is provided by the username and the given API key when the user is registered to the system. Nowadays it's implemented as a better password system but a signature should be calculated instead as $S = F(K, R)$, where K is the API key and R is the entire request, including all request parameters. The F function should be a secure message authentication code algorithm, such as AES-CMAC or SHA1-HMAC. The API work-flow is shown in Figure 6.

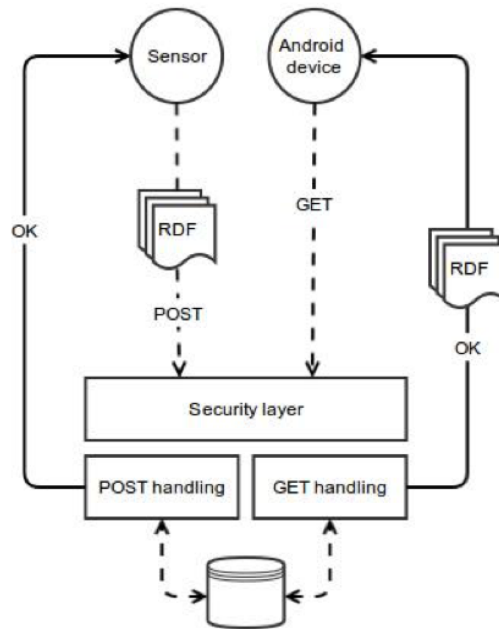


Figure 6. API workflow.

4.4.1 POST

When a sensor has data to insert to the system, it can form an HTTP POST request with a valid RDF document containing the data, the username and the API key in the headers, and 3 parameters in the URL: the API version, the project name and the stream name. The first parameter was introduced for compatibility reasons: if some nodes were deployed with the current version of the API but, after some time, the API was extended and the system changed, the old nodes might stop working. If an API version is included in the URL both versions can work simultaneously. The project name and the stream name parameters are necessary to identify where the data should be stored to.

Alternatively, sensors can create CSV files that will be later stored in the platform in an equivalent format as the used when the direct POST method is used.

4.4.2 GET

An HTTP GET request needs to be created in order to retrieve data from the system. This request should include the username and the API key in the headers as the POST request. Unlike the POST request, there are multiple ways of performing this operation depending on the entity you want to obtain and on the constraints you are applying. The required parameters are different if all the sensors from a project are requested in comparison to those needed to obtain the last measure of the humidity sensor closest to, i.e., the White House.

The API has been designed to allow the permutation of the parameters in the URL, thus making the order not matter. All the examples provided below are referenced to the base URL <http://commsensum.pc.ac.upc.edu> in order to make them shorter.

Observations

- URL: /observations
- Input parameters: sensors, projects, samplingTimes, observedProperties and geo.

The *sensor* parameter is used to determine the sensor from which the data is obtained by providing its id. The *project* parameter is used to determine from which project the data will be obtained by providing its id. The *samplingTimes* parameter may be used in 2 different ways: it can be used as a single data with the format YYYYMMDDThhmmss and it can also be used as an interval between two dates separated by two colons. If the *samplingTimes* parameter is used, the observations obtained will correspond to that interval of time. The *observed-properties* parameter is used to determine what property the user is trying to obtain by providing its id. Finally, the *geo* parameter is a string formed by the latitude, longitude and radius separated by a semicolon. The latitude and longitude determine a central point, while the radius is used to establish how far from that point should the data be gathered.

For example, if the user wanted to obtain all the observations made between the 30th of April of 2014 at 06:08:31 and the 1st of May of 2014 at 06:08:31 and that also come from the project with the id 6 (which should be public or only the owner would be able to retrieve this data) the URL would be /observations/projects/6/samplingTimes/20140430T060831::20140501T060831. The RDF/XML is too big to be added to this document but a graph generated from the obtained output is displayed in Figure 7.

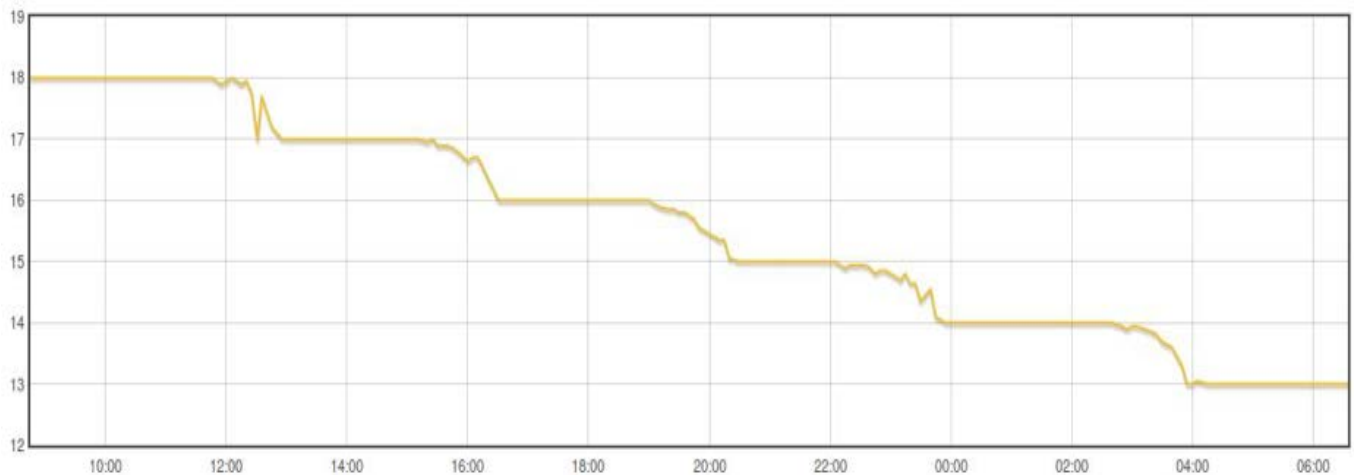


Figure 7. Sensor data output example in graphical view.

Sensors

- URL: /sensors
- Input parameters: country, city, project and geo.

The *country* and *city* parameters are used to respectively determine in which country and city should be located the sensors that are being searched. The *geo* and *project* parameters work like their homonyms in the Observations get function. If no extra parameter is introduced it is possible to query the sensor by id like "/sensors/2" for example.

In case the user wanted to get a list of the sensors that are located in the city of Barcelona the URL would be `/sensors/city/Barcelona`.

Projects

- URL: `/projects`
- Input parameters: none

Agents

- URL: `/agents`
- Input parameters: none

Properties

- URL: `/properties`
- Input parameters: none

UnitsOfMeasure

- URL: `/unitsofmeasure`
- Input parameters: none

Projects, agents, properties and unitsofmeasure can only be queried by id.

5. Website

The website has been implemented using Django, a Python-based web framework that follows the Model-View-Controller (MVC) architectural pattern. Django is a very well-known framework that provides many of the functionality that had to be developed like the administrator panel which allows managing all the entities in the database. The MVC pattern on which Django is based on is an abstract model that separates the data from an application, the user interface and business logic (inputs, generating queries or data) into three different components: model, view and controller. In addition to dividing the application into three types of components, the MVC pattern also defines its interactions.

- The model represents the data structures, and usually provides functions that help you retrieve, insert or update information in the database. It also contains functions to access other data from other web services or APIs.
- The View is the information that is presented to the user.
- The controller acts as intermediary between the Model, the View and other resources.

All the models that Django uses are loaded from the database so there's a class for every entity previously detailed in the database section.

The structure of the views implemented in CommSensum can be seen in Figure 8. From the home page you can access the part of the website that explains what the project is about, the project's blog and the private area. The private area includes the dashboard from which an overview can be seen in Figure 9 and, in case the user is also an administrator of the system, it will also include the administration dashboard from which an overview can be seen in Figure 10.

From within the user dashboard, the user can add projects, subscribe to a project, view the list of all his/her projects and edit them. The user can also add and edit sensors, view the list of all the sensors that he/she has registered, add a stream or edit an existing one. In the dashboard there's also access to a list of all the data gathered by its sensors and a query interface that uses the API which facilitates the results in RDF/XML, formatted in a table or if it's possible, recollected in a graph as it can be seen in Figure 11.

The administrator dashboard allows the administrator to manage all the entities that control the website. These entities are the same as the user can control but there's no restriction, so the admin can change other users' information. There are also some special entities like Group, User, ProjectStatus or DataType that are only available to the administrator. It may be emphasised that most of the "View all ...", "Add ..." and "Edit ..." seen in Figure 8 are the same view but they use different models to populate them. In Figures 12, 13 and 14 there are examples of each of them.

The website design has been developed using Bootstrap as the base and this allowed us to focus on the functionality rather than in the looks of the interface. The website is currently available at <http://commsensum.pc.ac.upc.edu>.

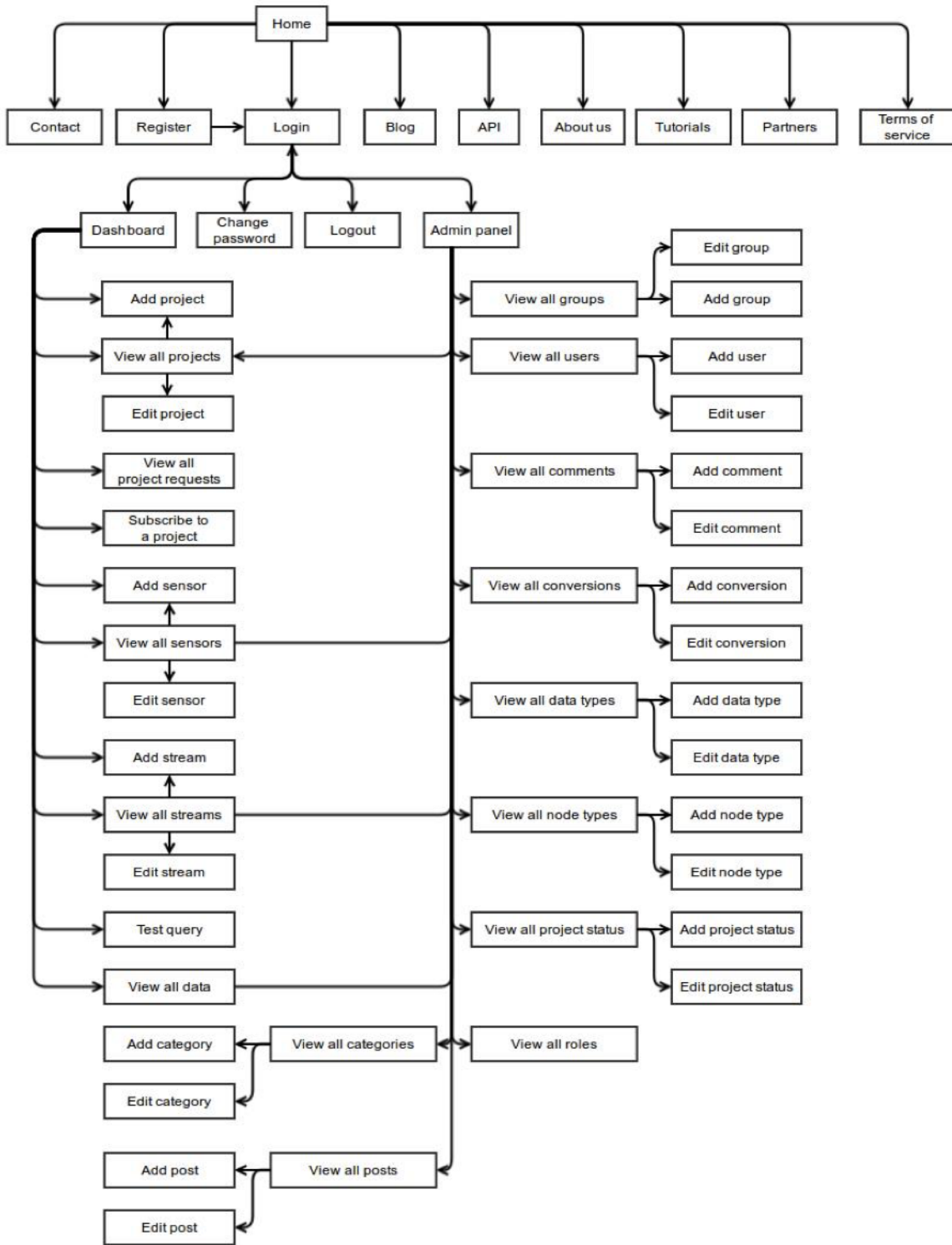


Figure 8. Website view structure map

This figure shows the Control Panel that allows i) to select and manage Projects created by users where each project consists on Sensor nodes and streams of data select specific; ii) to select and manage sensor nodes, iii) to select and manage streams belonging to sensor nodes, and iv) to select and manage data.

CommSensum Beta Projects Blog Platform Control Panel otrullols

- Username: otrullols
- API key: Rrme1YvMeyi3Ftd69qc4Y2CHK2Y14L06olUJaWdVVbM
- 11 new project requests.

Projects

[+Add](#) [★Subscribe](#) [View all](#)

Name	Description	Status	License	Role
waspmotes	Proyecto de medicion de contaminantes	Enabled	Public	Administrator
EuroEnvironData	Public European Environment Data	Enabled	Public	Administrator
bcnoise	test Waspmote con sensor de ruido dB	Enabled	Public	Administrator
ozoalprj	Proyecto de ozono alph	Enabled	Public	Administrator
CAPTOR	Proyecto CAPTOR H2020	Enabled	Public	Administrator
CCAA Andalucia	CCAA Andalucía	Enabled	Public	Administrator
CCAA Pais Vasco	CCAA PaAs Vasco	Enabled	Public	Administrator
CCAA Com. Valenciana	CCAA Com. Valenciana	Enabled	Public	Administrator
CCAA La Rioja	CCAA La Rioja	Enabled	Public	Administrator
CCAA Castilla y Leon	CCAA Castilla y LeAn	Enabled	Public	Administrator

Nodes

[+Add](#) [View all](#)

Name	Status	City	Country
csicmob4	OK		
csicmob3	OK		
csicmob2	OK		
csicmob1	OK		
csicmob5	OK		

Streams

[+Add](#) [View all](#)

Name	Sensor
csicmob4humidity	csicmob4
csicmob4temperature	csicmob4
csicmob4O3r1	csicmob4
csicmob4O3rD	csicmob4
csicmob4NO2r0	csicmob4

Data

[Data query](#) [Test query](#) [View all](#)

Date	Data type	Value	Units
June 1, 2016, 10 a.m.	temperature	28.6552	°C
June 1, 2016, 10 a.m.	humidity	7.2759	%
June 1, 2016, 10 a.m.	O3r	234.7435	KOhms
June 1, 2016, 10 a.m.	O3r	343.6141	KOhms
June 1, 2016, 9:30 a.m.	humidity	9.6786	%

[CommSensum](#)
[About us](#)
[Blog](#)
[Contributors](#)
[Contact](#)

[Platform](#)
[API](#)
[Documentation](#)
[Tutorials](#)

[Support](#)
[Presentations](#)
[Code snippets](#)

[Legal](#)
[Terms of Service](#)
[Privacy](#)
[Security](#)

Figure 9. User control panel

The following figure shows the administrator control panel that allows to manage groups, authorize permissions and add projects and confirm registrations.

CommSensum Beta admin Control Panel otrullols ▾

Home

Site administration Applications ▾

Authentication and Authorization

Groups	+ Add	Change
Users	+ Add	Change

Django_Comments

Comments	+ Add	Change
----------	-------	--------

Projects

Batches	+ Add	Change
Conversions	+ Add	Change
Data types	+ Add	Change
Datas	+ Add	Change
Images	+ Add	Change
Node types	+ Add	Change
Nodes	+ Add	Change
Project status	+ Add	Change

Figure 10. Administrator control panel

The following figure shows how to add a stream in a project.

CommSensum Beta admin Control Panel otrullols -



Home / Projects / Streams / Add stream

Add stream

Fields in **bold** are required.

Name:

Description:

Idnode:  

LastUpdate: Date: [Today](#) | [Calendar](#)

Time: [Now](#) | [Clock](#)

Figure 11. Add stream view

The following figure shows the data plotted for a data stream for a period of time. In this case, the query is performed using RDF language. There also exists the possibility to access the data via a non-RDF query, choosing a project, a node and stream and choosing in a calendar the initial date and the final date that the user wants to visualize.

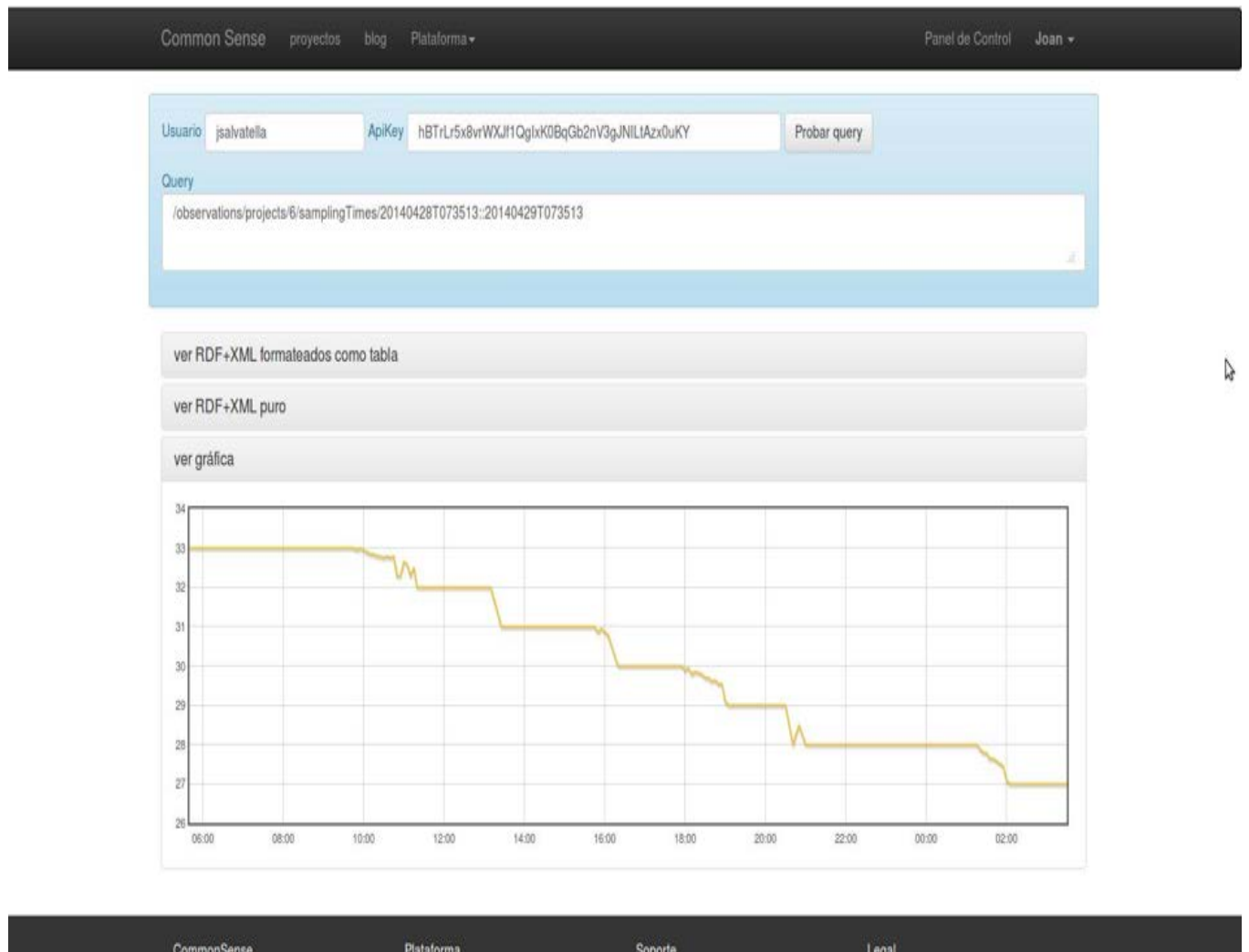


Figure 12. API output recollected in a graph

The following figure shows how to create a project, adding the name, description, user and streams. Since a project consists on streams and nodes, first these ones have to be created.

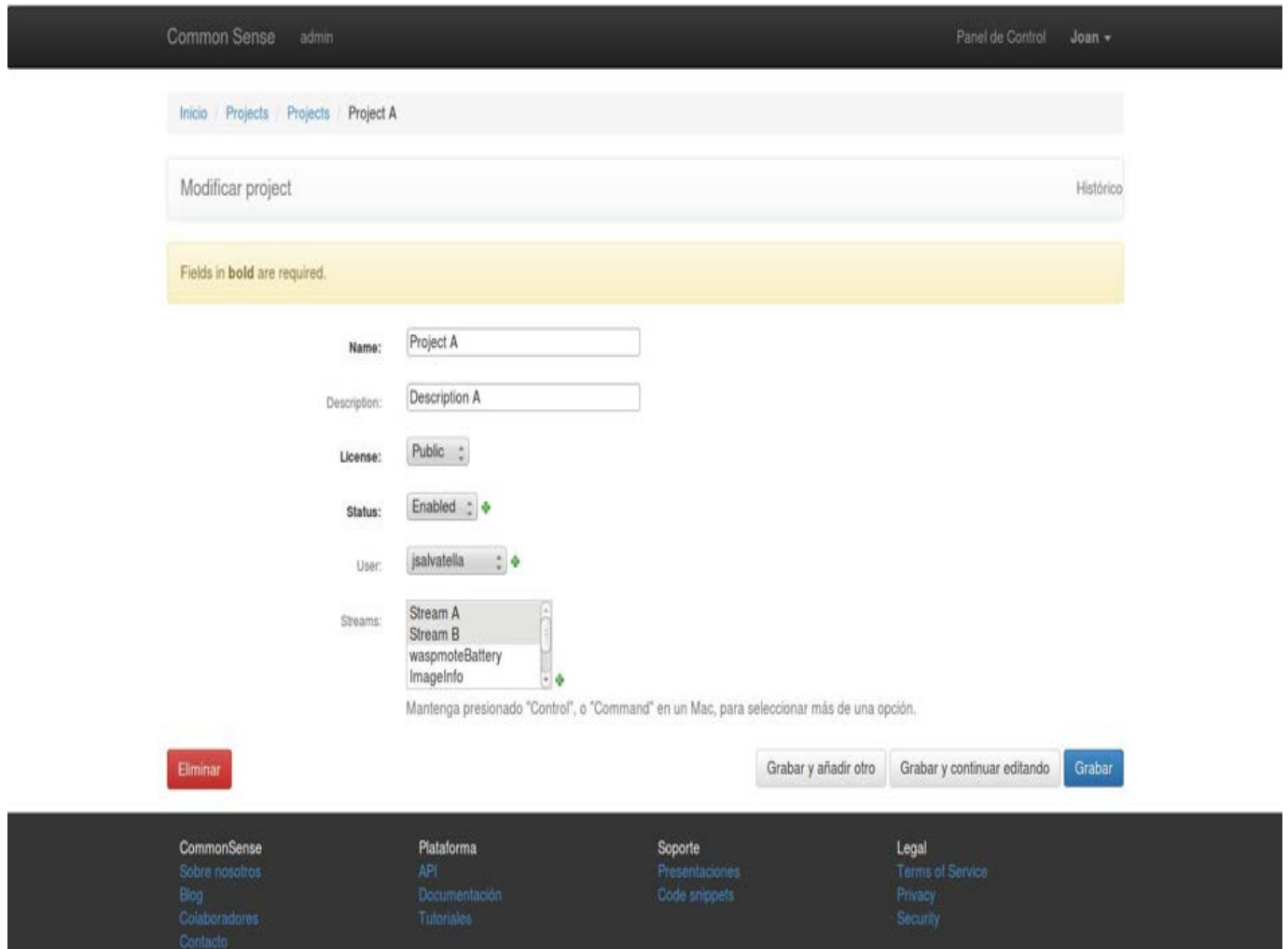


Figure 13. Edit project view

6. Conclusions

In this deliverable we presented *CommSensum*, an Open Link Data community sensing platform. This platform plays a key role in the CAPTOR project as it will be used as a platform for making openly public the results of our scientific experiments. We are currently working in improving the platform after the feedback obtained in our first testbed usage.