smar+ en. ci+y

TOWARDS SMART ZERO CO$_2$ CITIES ACROSS EUROPE

VITORIA-GASTEIZ + TARTU + SØNDERBORG

Deliverable 6.4: Interoperability Mechanisms

WP6, Task 6.4

Date of document

28/07/2017 (M 18)

Deliverable Version:	D6.4, V1.0
Dissemination Level:	PU[1]
Author(s):	Aitor Akizu, Natividad Herrasti (ETIC), Felix Larrinaga, Alain Perez (MON/MGEP), Patxi Sáez de Viteri (MON), Jose Luis Izkara (TEC), Alvaro Arroyo, Mauri Benedito (GIS), Urmo Lehtsalu (ET), Jørgen Raun Petersen (VG),

[1] PU = Public
PP = Restricted to other programme participants (including the Commission Services)
RE = Restricted to a group specified by the consortium (including the Commission Services)
CO = Confidential, only for members of the consortium (including the Commission Services)

This project has received funding from the European Union' s Horizon 2020 research and innovation programme under grant agreement No 691883

Document History

Project Acronym	SmartEnCity
Project Title	Towards Smart Zero CO2 Cities across Europe
Project Coordinator	Francisco Rodriguez Tecnalia francisco.rodriguez@tecnalia.com
Project Duration	1st February 2016 - 31st July 2021 (66 months)

Deliverable No.	D6.4 Interoperability mechanisms implementation	
Diss. Level	Confidential / Demo	
Deliverable Lead	*ETIC*	
Status		Working
		Verified by other WPs
	X	Final version
Due date of deliverable	31/07/2017	
Actual submission date	31/07/2017	
Work Package	WP 6 - City Information Open Platform (CIOP)	
WP Lead	*ET*	
Contributing beneficiary(ies)	ETIC, TEC, MON, GIS, VG, ET	

Date	Version	Person/Partner	Comments
22/03//2017	0.1	ETIC	First Draft for the ToC and SOTA
26/05/2017	0.2	GIS	Incorporation of the GIS section
28/06/2017	0.3	ETIC	Second Draft
03/07/2017	0.4	TEC. MON, VG. ET	Content added for all partners
07/07/2017	0.5	ETIC	Integration and revision all of content and contributions
14/07/2017	0.6	ETIC	Integration and revision all of content and contributions
14/07/2017	0.7	ETIC	Final version for review
19/07/2017	0.8	TEC, SON,ET	Reviewed by **partners**
28/07/2017	1.0	ETIC	Final version

Copyright notice

Table of content:

Table of Tables:

Table of Figures:

Abbreviations and Acronyms

Abbreviation/Acronym	Description
AENOR	Asociación Española de Normalización y Certificación
API	Application programming interface
CEN	European Committee for Standardization
CIOP	City Information Open Platform
CityGML	City Geography Mark-up Language
ETL	Extract, Transform and Load
EC	European Commission
GPL	General Public License
GIS	Geographic Information Systems
HDFS	Hadoop Distributed File System
HMI	Human Machine Interface
ICT	Information and Communication Technologies
IoT	Internet of Things
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LGPL	Lesser General Public License
LH	Lighthouse City
M2M	Machine to Machine
MQTT	MQ Telemetry Transport
MySQL	My Structured Query Language
NoSQL	No Structured Query Language
OGC	Open Geospatial Consortium
OS	Operating System
OSM	OpenStreetMap
OWL	Web Ontology Language
RA	Reference Architecture
POC	Proof Of Concept
QGIS	Quantum Sistema de Información Geográfica
RDBMS	Relational Database Management Systems
RDF	Resource Description Framework
REST	Representational State Transfer
SCADA	Supervisory Control And Data Acquisition
SDK	Software Development Kit
SmartEnCity	Towards Smart Zero CO2 Cities across Europe
SOA	Service-oriented architecture
SQL	Structured Query Language
SWE	Sensor Web Enablement
UNE	Una Norma Española
URI	Uniform Resource Identifiers
W3C	World Wide Web Consortium
WCS	Web Coverage Service
WFS	WORLDWIDE FLIGHT SERVICES
WP	Work Package
WMS	Web Map Service
EU	European Union
XML	eXtensible Mark-up Language

Table 1 Abbreviations and Acronyms

0 Publishable Summary

SmartEnCity focuses on the development of a highly adaptable and replicable systemic approach towards urban transformation into sustainable, smart and resource efficient urban environments in Europe, through the planning and implementation of measures aimed at improving energy efficiency in the main consuming sectors in cities and increasing the supply of renewable energy. This approach will be defined in detail, and subsequently laid out and implemented in the three Lighthouse demonstrators (Vitoria-Gasteiz in Spain, Tartu in Estonia and Sonderborg in Denmark), to be further refined and replicated with the development of Integrated Urban Plans (IUPs) in all participant (both Lighthouse and Follower) Cities.

WP6 aims to devise a common ICT platform that will be the reference for the deployment of the "City Information Open Platform" (CIOP) in each one of the pilot lighthouse projects. The platform will provide a standardized data model to accommodate data from each pilot and will also define standardized services and modules for data consumers, especially relevant are those related to the monitoring of SmartEnCity KPIs, those requested by the EC in the call and those identified as ICT solutions for the project.

Deliverable D6.4 presents the results of Task 6.4 "Interoperability mechanisms Implementation" within WP6 of the SmartEnCity project. The main objective for this task is to design and deploy the core/global architecture for the ICT platform, having interoperability and modularity as the key points. The architecture is based on a Reference Architecture and defines the main functional modules and components of the CIOP. It covers the needs of the rest of the tasks of this WP.

The first result of the Task 6.4 is a demonstrator available online. The demonstrator or prototype is an IoT platform deployed in Internet that agrees with the Reference Architecture (RA) described in this document and in the deliverable of T6.3 and that offers the functionality necessary to build the CIOP in the different lighthouses. The main aim of this task is to develop the interoperability mechanisms within this RA. The prototype will be enhanced by integrating the results of other tasks 6.2 and 6.3 and later on with the ones obtained from 6.5 and 6.6. The whole prototype resultant of WP6 will include the data models, the interoperability mechanisms, the technologies for HMI and some added value services.

The second result is this document. This document presents a description of the Reference Architecture (RA) (AENOR, (2015)) proposed for the CIOP mainly focusing in the interoperabity mechanisms aspects. The document also presents:

- The approach or methodology followed to obtain both results (RA and prototype)
- The state of the art of interoperability mechanisms and selection of some of them
- Interoperability mechanisms used for the main elements of the city platform
- Explanation of the demonstrator and its components
- Access to the demonstrator users guide

1 Introduction

1.1 Purpose and target group

This deliverable "D6.4 – Interoperability mechanisms Implementation" is the report that presents a demonstrator that implements the interoperability mechanisms according to the Reference Architecture proposed for the SmartEnCity project. The report presents the description of the demonstrator and the process followed to construct that prototype, which is a specific instantiation of the Reference Architecture.

The main objective is to demonstrate the interoperability mechanisms following a Reference Architecture by means of a demonstrator or prototype presented as the result of this task.

The main activities carried out in this task are listed here:

- General selection of different mechanisms for interchanging data
- Selection of the adequate interoperability mechanisms for providing the elaborated data to the Verticals and applications with the information coming from the infrastructure of the city
- Selection of the adequate interoperability mechanisms for providing data to the Verticals and applications with the information coming from Open Data
- Selection of the adequate interoperability mechanisms to provide the Key Performance Indicators (KPI) from the Verticals

This report is structured in the following sections.

Section 2 presents the objectives pursued in Task 6.4.

Section 3 presents an overall approach of the interoperability mechanisms.

Section 4 presents the state of the art (SOTA) of the interoperability mechanisms.

Section 5 presents the interoperability mechanisms by using APIs where vertical applications get the data and information from the platform.

Section 6 presents the interoperability mechanisms by using GIS data where vertical applications get the data and information from the platform.

Section 7 presents the interoperability mechanisms for integrating Open Data.

Section 8 presents the interoperability mechanisms for providing KPIs from the vertical applications.

Section 9 presents security issues and mechanism at the Interoperability Layer.

Section 10 presents the SmartEnCity demonstrator.

Section 11 presents the conclusions of the task and the integration with the rest of the tasks of the WP6.

Section 12 presents the References of this deliverable.

Main target group of the information, the demonstrator and the conclusions collected in this deliverable are the partners in charge of the development of the CIOP platform at use case level. That is at city level in Work Packages 3, 4 and 5. Follower cities could also take advantages of the findings and results produced in this task.

1.2 Contributions of partners

The following Table 2 Contribution of partners

depicts the main contributions from participant partners in the development of this deliverable.

Participant short name	Contributions
ETIC	Task Leader. Responsible of the demonstrator (deliverable). Responsible of the content in this document. Main contributor in Section 1 (Introduction), Section 2 (Objectives and Guiding Principles), Section 3 (Overall Approach), several subsection of Section 4 (SOTA of interoperability mechanisms), Section 5 (Interoperability mechanisms by using APIs), Section 6 (Interoperability mechanisms by using GIS data), Section 7 (Interoperability mechanisms by using Open Data), Section 8 (Interoperability mechanisms for KPIs),, Section 9 (Security at Interoperability Layer), Section 10 (SmartEnCity Demonstrator), Section 11 (Conclusions). Has reviewed contributions to all the sections
TEC	Main contributor in Section 8 (Interoperability mechanisms by using GIS data)
MON	Main contributor in Section 3 (Overall Approach), several subsection of Section 4 (SOTA of interoperability mechanisms), Section 5 (Interoperability mechanisms by using APIs), Section 8 (Interoperability mechanisms for KPIs),, Section 9 (Security at Interoperability Layer), Section 10 (SmartEnCity Demonstrator), Section 11 (Conclusions).
GIS	Main contributor in Section 6 (Interoperability mechanisms by using GIS data) and Section 10 (SmartEnCity Demonstrator).
VG	Main contributor in Section 3 (Overall Approach) and Section 10 (SmartEnCity Demonstrator).
ET	Main contributor in Section 3 (Overall Approach) and Section 10 (SmartEnCity Demonstrator).

Table 2 Contribution of partners

1.3 Relation to other activities in the project

The following Table 3 depicts the main relationship of this deliverable to other activities (or deliverables) developed within the SmartEnCity project and that should be considered along this document for further understanding of its contents.

Deliverable Number	Contributions
D6.1	This deliverable provides the requirements identified for SmartEnCity
D6.2	This deliverable provides the data models necessary for SmartEnCity.
D6.3	This demonstrator extends D6.2 considering the data models necessary for SmartEnCity.
WP3, WP4 and WP5	The implementation in each lighthouse will agree with the Reference Architecture and the layers and modules defined in it. Data models will be implemented there
WP7	KPIs are defined in that work package. Data Models for KPIs have been built according to D7.2 and data flow construction in SmartEnCity CIOP is outlined in D7.9 (Task 7.3)

Table 3 Relation to other activities in the project

2 Objectives and Guiding Principles

As stated in the Grant Agreement, the overall objective in this work package is to devise a common ICT platform that will be the reference for the deployment of the “City Information Open Platform” in each one of the pilot lighthouse projects. The detailed objectives of the work package are:

- Define the specifications of the platform. Functional and non-functional requirements must be identified considering the overall expected performance of the platform. (Done in (SmartEnCityD6.1, 2016)).
- Define and provide the infrastructure or technological architecture that will enable gathering information from the different Verticals (building retrofitting, district heating, smart grid, smart mobility) and offer data to the consumer applications (web applications, reports, control algorithms…) Main objective of this task/deliverable
- Provide a data model that will accommodate data from different sources such as electric vehicle charging points, appliances and lighting systems in dwellings, district heating Supervisory Control And Data Acquisition (SCADA) systems, data collected by utilities with smart meters, data from building elements (lifts, lighting systems…). (Deliverable 6.3)
- Provide the mechanisms and protocols to ease interconnection between platform modules and to allow data uploading/consuming from the different sources, enhancing interoperability between the platform and other systems. (Deliverable 6.4)
- Provide the mechanisms to build ICT solutions for different stakeholders offering actionable information and recommendations, to empower citizens on decision making in relation to home energy consumption and mobility and to encourage them to reduce their environmental and resources footprint. (Deliverable 6.5)
- Provide mechanisms to build added value service linking the platform to social networks with the objective to boost engagement of stakeholders with the ICT platform and more importantly raise awareness about energy consumption. Also provide mechanisms to build added value services offering data analysis of monitored data, through machine learning big data techniques or business intelligence techniques. (Deliverable 6.6)
- Integrate and validate the different modules of the ICT platform. (Deliverable 6.7)

The overall objective for this task (T6.4) and its deliverable (D6.4) is to develop interoperability mechanisms that will allow integrating data coming from different sources. Two aspects were handled: semantic or data level interoperability (systems modelling data in different formats) and communication level interoperability (systems using different communication protocols). Different interoperability levels have been evaluated. Since most of the devices usually rely on specific or proprietary databases, data transformations are needed to convert the data into the format defined in the platform. Finally, interoperability with external services providers implies developing connectors to link the platform to external service providers like social networks, transportation systems, weather data providers, etc. Most of the interconnectivity mechanisms have been developed based on web services.

This deliverable will provide a demonstrator or prototype that accommodates the technological modules and functionality identified in that Reference Architecture.

The main activities performed in this task are:

- Identify the common and more used mechanisms of interoperability in Smart Cities architectures.
- Select the more suitable interoperability mechanisms according to the requirements define in T6.1, the Reference Architecture of T6.2 and the Data Models of T6.3.
- Identify modules and components that will allow the implementation of the selected interoperability mechanisms.
- Demonstrate the interoperability mechanism in the demonstrator/prototype
- Validation of technologies and tools
- Integration with the rest of the modules within WP6

2.1 Principles

This section includes the principles to be followed in the construction of the demonstrator.

These principles are:

Keep it simple: While designing the architecture select simple solutions instead of complex ones. Be as practical as possible. Avoid unnecessary content focusing on addressed concerns and added value for architecture stakeholders. Avoid repetition.

Re-use: Consider existing solutions and best practices and reuse them instead of re-inventing them. Rely on state-of-the-art and –practice approaches, including standards, frameworks, and results of related projects. Avoid the invention of components to deviate from existing solutions if they are not strictly necessary. Make the design results as understandable as possible. The goal is not to come up with one reference architecture that needs to be complied up to 100%.

Understandability: Architecture documentation must provide the possibility to get a quick overview of the realization of a system at a manageable level of abstraction. Documents must be understandable by stakeholders considering the respective context and needs.

Divide and conquer: Since architecture imposes the structure on the solution and makes the complexity rise, the overall problem must be broken down into separately solvable sub-problems. This may mean decomposing functionality into small partitions, but it may also mean addressing quality requirements separately with appropriate architectural solutions. Architecture also has to provide the means for recomposing the parts back into the overall solution.

Abstraction: Abstraction is the central means for coping with complexity in large software systems. This means that always only so much details are shown as is necessary to understand the information that should be transported with a view. So the typical strategy within a particular view is to start on a high level of abstraction, showing only the most important and top-most elements and their external visible properties and relations. Then iteratively more details and internals are revealed to provide more information to stakeholders that need to have an in-depth insight into the single elements. This is done until the particular stakeholder concern that is to be addressed with a view is sufficiently met.

Horizontal and Vertical Traceability: Vertical traceability links requirements to their realization in the architecture and vice versa. Horizontal traceability is needed between different views on architecture to find relationships among related items across work groups or product components for the purpose of avoiding potential conflicts.

Uniformity: To achieve a documentation that is well comprehensible to all stakeholders and therefore fulfil their primary objective, it is essential to document aspects in a uniform manner. The language that used for the documentation should be chosen based on the factors of formality, accuracy and acceptance in practice, but once this has happened, the used elements should exhibit a clearly specified meaning and should be used correspondingly and strictly. Otherwise stakeholders will have problems understanding the meaning of certain elements or relations and errors in any subsequent activities.

3 Overall Approach

In this section, the steps followed to achieve the deliverable are outlined. The methodology and the Reference Architecture selected in (SmartEnCityD6.2, 2017) have been adopted and extended in this deliverable. This report includes the interoperability mechanisms necessary to build some examples of applications and the demonstrator. The results are presented as reports describing the Reference Architecture and as demonstrators. These demonstrators are instantiations of the Reference Architecture created as prototypes that can be easily shared and reused in the project.

In this task, three major approaches have been implemented:

1. Identify and describe the interoperability mechanisms necessary for the SmartEnCity CIOP extending the Reference Architecture using the methodology proposed in (SmartEnCityD6.2, 2017) and using the data models of (SmartEnCityD6.3, 2017). Interoperabity mechanisms are essential to interchange data and information among different layers of the RA:
2. Build a demonstrator that implements those interoperability mechanisms for some examples of applications in Mobility, Energy Efficiency and Citizen's Engagement.
3. Build a demonstrator that implements some of the KPIs proposed in (SmartEnCityD7.2, 2017)

3.1 Reference Architecture and Demonstrator

This section presents the Reference Architecture proposed for SmartEnCity. The Reference Architecture is a layered model based on UNE 178104:2015 (AENOR CTN-178 group standard). Figure 1 Smart Cities Architecture

presents the layers and modules composing the reference architecture. It is worth outlining that the core of the Reference Architecture is an IoT platform.

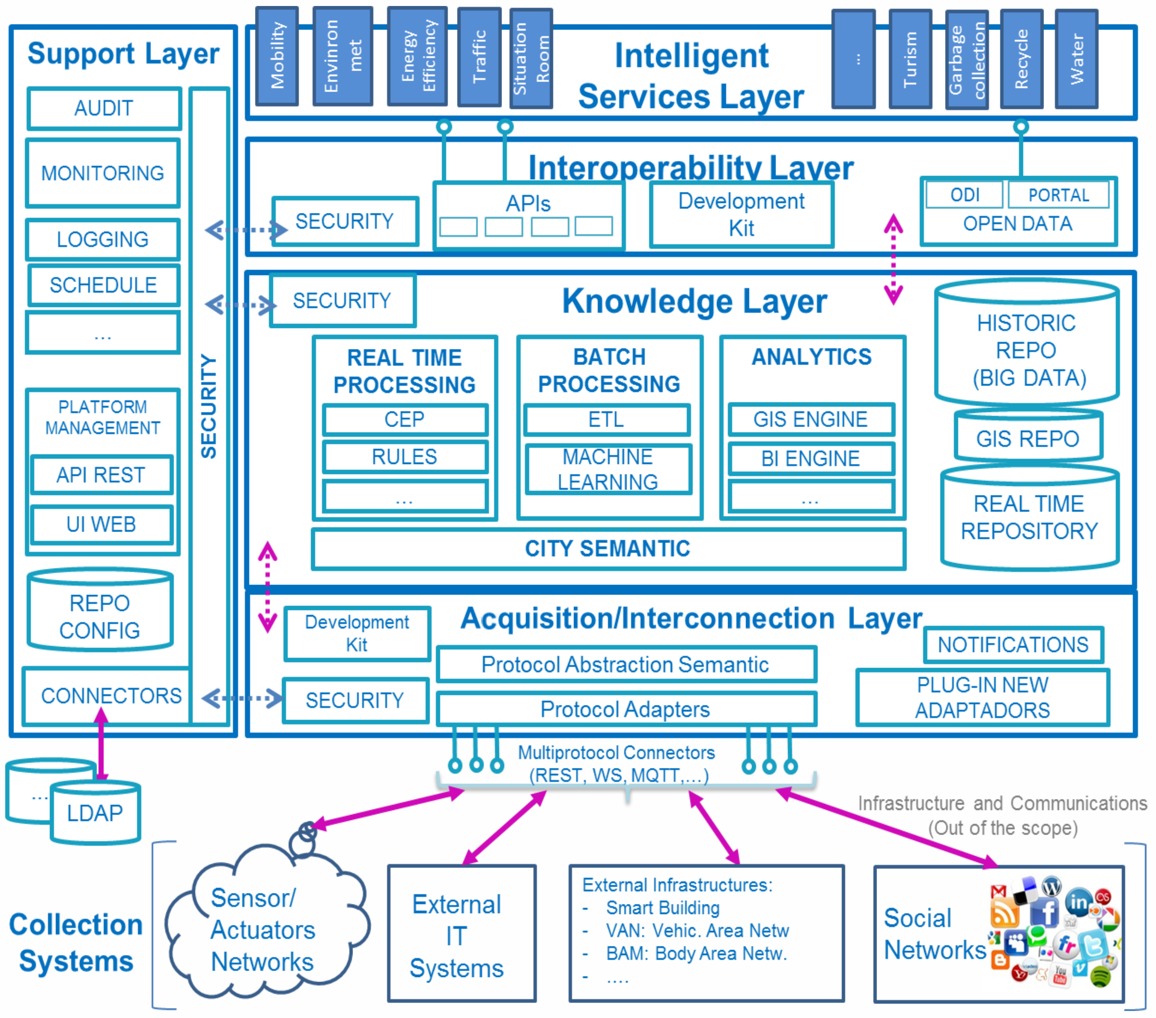


Figure 1 Smart Cities Architecture

In this Reference Architecture, there are two separated places for interoperability, as explained in the subchapters below

3.2 Acquisition/Interconnection Layer

The Acquisition/Interconnect Layer facilitates the capture of data and information coming from the infrastructure of devices, sensors, citizens' smartphones, social networks, city infrastructure, etc.

The Acquisition/Interconnect Layer, shown in the following figure, is responsible for:

a) Integrate the information from the data sources (Collection Systems), which can be:
   - Sensors, actuators, gateways and devices such as traffic lights, buildings, weather stations, etc. from Networks of Sensors
   - Different devices such as smartphones or devices installed in dwellings, vehicles, etc. from public networks
   - Social networks
   - Other IT systems such as SCADAs or management solutions for Vertical domains, which can be proprietary solutions

- External infrastructures, such as, building and city repositories (BIM, CityGML)

b) Supply the Information to the Knowledge Layer independently of the devices connected, providing a semantic view of the acquired data, decoupled from the acquisition protocols.

c) Be independent of the network operator both from the provision of network information and from the control of that information.

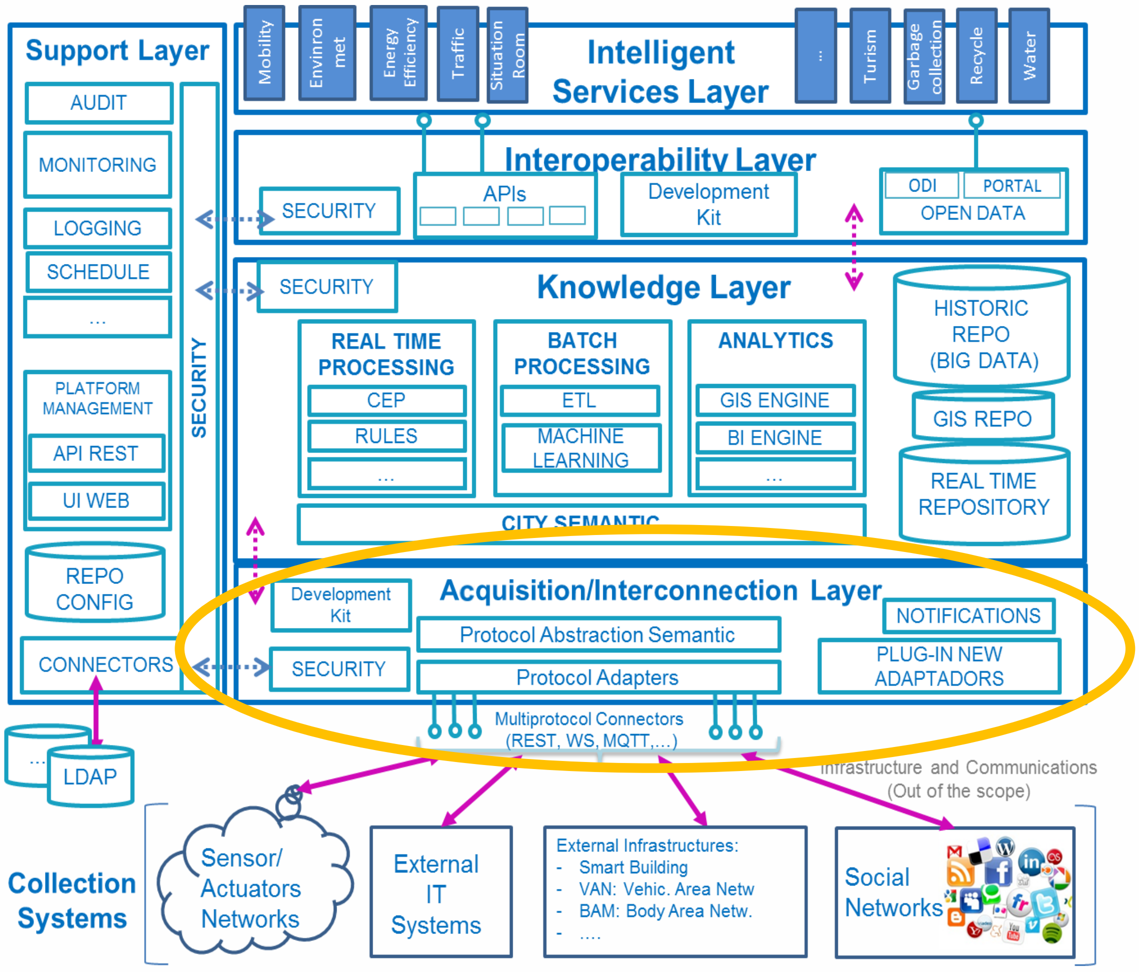


Figure 2 Acquisition/Interconnection Layer

3.3 Interoperability Layer

The Interoperability Layer facilitates the provision of services within the City Intelligent Services by offering interfaces and functionalities, such as the Development Kit and Open Data, which will be used to implement the services that will be delivered to customers:

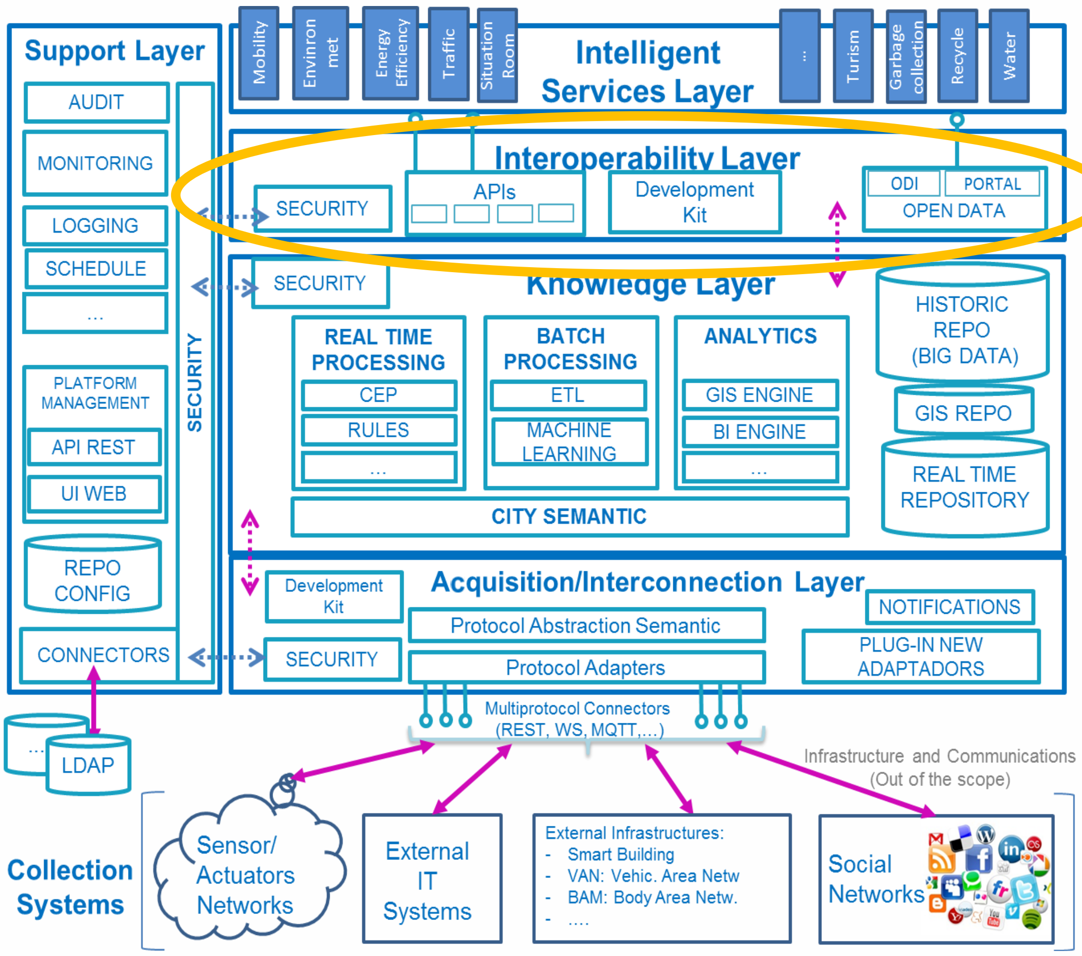


Figure 3 Interoperability Layer

The APIs exposed by the Interoperability Layer will be easy to use by the developer community. REST API interfaces must support different modes of data access, including Push (subscription and notification) and Pull (request and answer) mode. Geo-referenced queries should also be supported. The data access model offered by the API will be agnostic with respect to the specific model of data but better if it is compatible with any existing model.

Based on a set of standards-based APIs the Interoperability Layer must ensure portability of applications between cities and between platforms, in such a way as to create a true ecosystem of applications with critical mass and lower the barrier of access to the application developers.

Some tasks to be done by the Interoperability Layer are:

- Publish APIs that can be consumed from the Intelligent Services Layer
- Interconnection capacity between applications and between platforms
- Access from the platform to external services
- Publish open data through an Open Data Portal
- Through a Development Kit that includes SDK and APIs allows to build Services within the Intelligent Services Layer
- Integrated security in access to APIs, Development Kit, Open Data, etc.
- Service and share geospatial data through standard OpenGIS web services

4 SOTA of interoperability mechanisms

In this section a SOTA (State of the Art) of the more used interoperability mechanisms is presented.

4.1 MQTT (Message Queuing Telemetry Transport)

MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium.

The protocol runs over TCP/IP, or over other network protocols that provide ordered, lossless, bi-directional connections. Its features include:

- Use of the publish/subscribe message pattern which provides one-to-many message distribution and decoupling of applications.
- A messaging transport that is agnostic to the content of the payload.
- Three qualities of service for message delivery:
  - "At most once", where messages are delivered according to the best efforts of the operating environment. Message loss can occur. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after.
  - "At least once", where messages are assured to arrive but duplicates can occur.
  - "Exactly once", where message is assured to arrive exactly once. This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.
- A small transport overhead and protocol exchanges minimized to reduce network traffic.
- A mechanism to notify interested parties when an abnormal disconnection occurs.

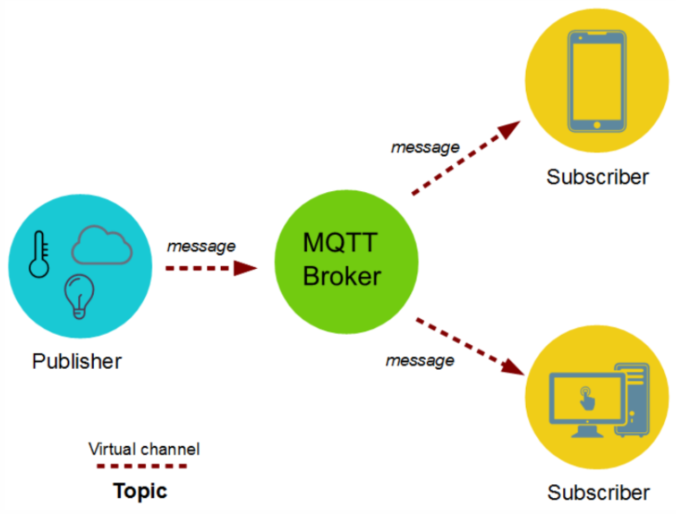


Figure 4 MQTT mechanism

4.2 AMQP (Advanced Message Queueing Protocol)

AMQP is a binary, application layer protocol, designed to efficiently support a wide variety of messaging applications and communication patterns. It provides flow controlled, message-oriented communication with message-delivery guarantees such as at-most-once (where each message is delivered once or never), at-least-once (where each message is certain to be delivered, but may do so multiple times) and exactly-once (where the message will always certainly arrive and do so only once), and authentication and/or encryption based on SASL and/or TLS. It assumes an underlying reliable transport layer protocol such as Transmission Control Protocol (TCP).

The AMQP specification is defined in several layers:

- Type system.
- Symmetric, asynchronous protocol for the transfer of messages from one process to another.
- Standard, extensible message format.
- Set of standardized but extensible 'messaging capabilities.

AMQP key parts:

- Broker (Server): An application – Implementing the AMQP model – that accepts connections from clients for message routing, queuing etc.
- Message: Content of data transferred/ routed including information such as payload and message attributes.
- Consumer: An application which receives messages(s)– put by a producer – from queues.
- Producer: An application which put messages to a queue via an exchange.

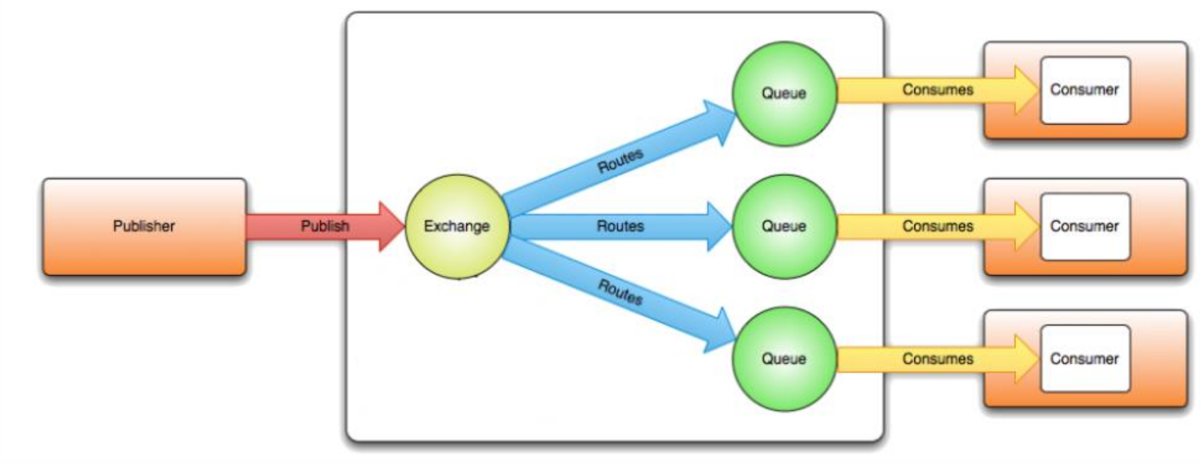


Figure 5 AMQP mechanism

4.3 MQTT vs AMQP

Both provide basic messaging needs; beyond that, AMQP provides a very much richer set of messaging scenarios. AMQP is almost a complete superset, lacking only explicit protocol support for Last-Value-Queues and will messages. However, its deliberate design for extensibility, using an IANA-like approach with a discursive approach, ensures that such features can be added in a forward-compatible, widely agreed upon way.

Both protocols are being promoted for widespread use in the internet:

- MQTT as a low-overhead, simple to implement way to send data, especially from embedded devices.
- AMQP as the asynchronous complement to HTTP.

As such, both are being promoted as being ideal for cloud computing and the IoT. That essential thesis is correct; message queuing, with its asynchronous nature and minimal need for configuration when done right, is perfect for interoperating many different environments.

However, MQTT is constrained to providing basic messaging 'topics' in a single 'namespace', with no long-lived 'store-and-forward' queuing pragmatic. This makes it difficult, if not often impossible, to multi-tenant server resources, or to dynamically migrate them or provide simple 'development to production' switch-over. Even worse, a woefully naive security / user model makes proper resource sandboxing and analysis very limited. AMQP provides for sand-boxed, multi-tenanted or multi-hosted infrastructure, ideal for the modern cloud with multiple user security schemes appropriate to the modern internet. Lastly, it's worth noting that MQTT, intended for telemetry transmission, is used in none of the world's biggest message queue based telemetry projects: Scripps Oceanography's monitoring of the Mid-Atlantic Ridge3, and Smith Electric Vehicle's global fleet management4, both use versions of AMQP.

4.4 STOMP (Simple Text Oriented Message Protocol)

Formerly known as TTMP, is a simple text-based protocol, designed for working with message-oriented middleware (MOM). It provides an interoperable wire format that allows STOMP clients to talk with any message broker supporting the protocol. It is thus language-agnostic, meaning a broker developed for one programming language or platform can receive communications from client software developed in another language. STOMP is an alternative to other open messaging protocols such as AMQP (Advanced Message Queueing Protocol).

4.5 REST (Representational state transfer)

REST (Representational State Transfer) is an architectural style, and an approach to communications that is often used in the development of Web services. The use of REST is often preferred over the more heavyweight SOAP (Simple Object Access Protocol) style because REST does not leverage as much bandwidth, which makes it a better fit for use over the Internet. The SOAP approach requires writing or using a provided server program (to serve data) and a client program (to request data).

REST'S decoupled architecture, and lighter weight communications between producer and consumer, make REST a popular building style for cloud-based APIs, such as those provided by Amazon, Microsoft, and Google. When Web services use REST architecture, they are called RESTful APIs (Application Programming Interfaces) or REST APIs.

REST is often used in mobile applications, social networking Web sites, mashup tools, and automated business processes. The REST style emphasizes that interactions between clients and services is enhanced by having a limited number of operations (verbs). Flexibility is provided by assigning resources (nouns) their own unique Universal Resource Identifiers (URIs). Because each verb has a specific meaning (GET, POST, PUT and DELETE), REST avoids ambiguity.

4.6 SOAP (Simple Object Access Protocol)

SOAP is a protocol specification for exchanging structured information in the implementation of web services in computer networks. Its purpose is to induce extensibility, neutrality and independence. It uses XML Information Set for its message format, and relies on application layer protocols, most often Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

SOAP allows processes running on disparate operating systems (such as Windows and Linux) to communicate using Extensible Markup Language (XML). Since Web protocols like HTTP are installed and running on all operating systems, SOAP allows clients to invoke web services and receive responses independent of language and platforms.

SOAP provides the Messaging Protocol layer of a web services protocol stack for web services. It is an XML-based protocol consisting of three parts:

- An envelope, which defines the message structure and how to process it.
- A set of encoding rules for expressing instances of application-defined datatypes.
- A convention for representing procedure calls and responses.

SOAP has three major characteristics:

1. Extensibility (security and WS-routing are among the extensions under development)
2. Neutrality (SOAP can operate over any protocol such as HTTP, SMTP, TCP, UDP or JMS)
3. Independence (SOAP allows for any programming model)

4.7 REST vs SOAP

REST provides the following advantages, specifically advantages over leveraging SOAP:

- RESTful Web services are easy to leverage by most tools, including those that are free and inexpensive. REST is becoming the dial tone for systems interaction, including the use of RESTful Web services, which are, for the most part, the way cloud providers externalize their cloud services.

- SOAP services are much harder to scale than RESTful services. Thus, REST is often chosen as the architecture for services that are exposed via the Internet (like Facebook, MySpace, Twitter, and most public cloud providers).
- The learning curve seems to be reduced. Developers are able to make use of REST from within applications faster than they can with SOAP. This saves time, which saves money.
- REST uses a smaller message format than SOAP. SOAP uses XML for all messages, which makes the message size much larger, and thus less efficient. This means REST provides better performance, as well as lowers costs over time. Moreover, there is no intensive processing required, thus it’s much faster than traditional SOAP.
- REST is designed for use over the Open Internet/Web. This is a better choice for Web scale applications, and certainly for cloud-based platforms.

Moving forward, REST is likely to continue its growth as enterprises seek to provide open and well-defined interfaces for application and infrastructure services. The growth of public and private cloud computing is driving much of this demand, and will continue to drive growth into the future.

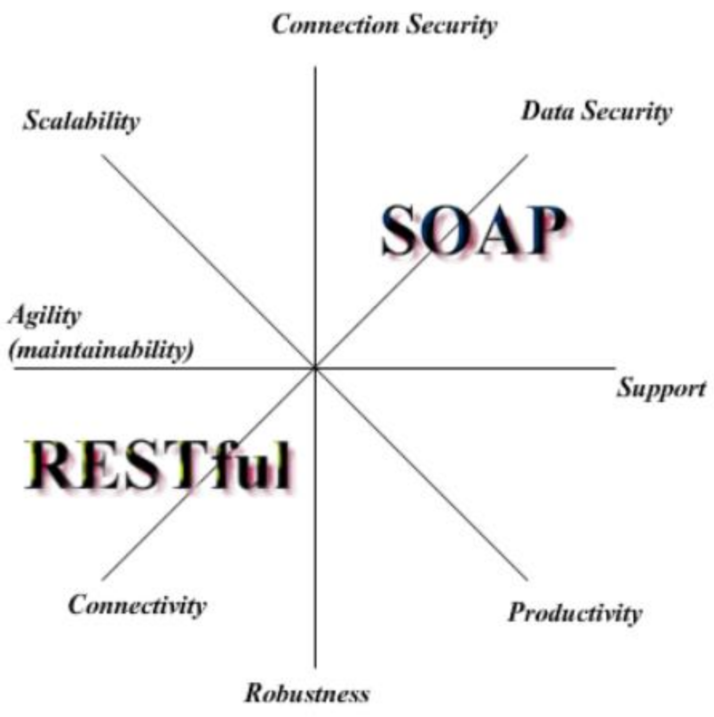


Figure 6 RESTful vs SOAP

4.8 OpenGIS Web Feature Service

The Web Feature Service (WFS) is an international standard promoted by the Geospatial Consortium (OGC), an international voluntary consensus standards organization, originated in 1994. It represents a change in the way geographic information is created, modified and exchanged on the Internet. Rather than sharing geographic information at the file level using, for instance, the File Transfer Protocol (FTP), the WFS offers direct fine-grained access to geographic information at the feature and feature property level. Web feature services allow clients to only retrieve or modify the data they are seeking, rather than retrieving a file that contains the data they are seeking.

The WFS specifies the behaviour of a service that provides transactions on and access to geographic features in a way that is independent of the underlying data store. It specifies discovery operations, query operations, locking operations, transaction operations and operations to manage stored parameterized query expressions.

- Discovery operations allow the service to be interrogated to determine its capabilities and to retrieve the application schema that defines the feature types that the service offers.
- Query operations allow features or values of feature properties to be retrieved from the underlying data store based upon constraints, defined by the client, on feature properties.
- Locking operations allow exclusive access to features for the purpose of modifying or deleting features.
- Transaction operations allow features to be created, changed, replaced and deleted from the underlying data store.
- Stored query operations allow clients to create, drop, list and described parameterized query expressions that are stored by the server and can be repeatedly invoked using different parameter values.

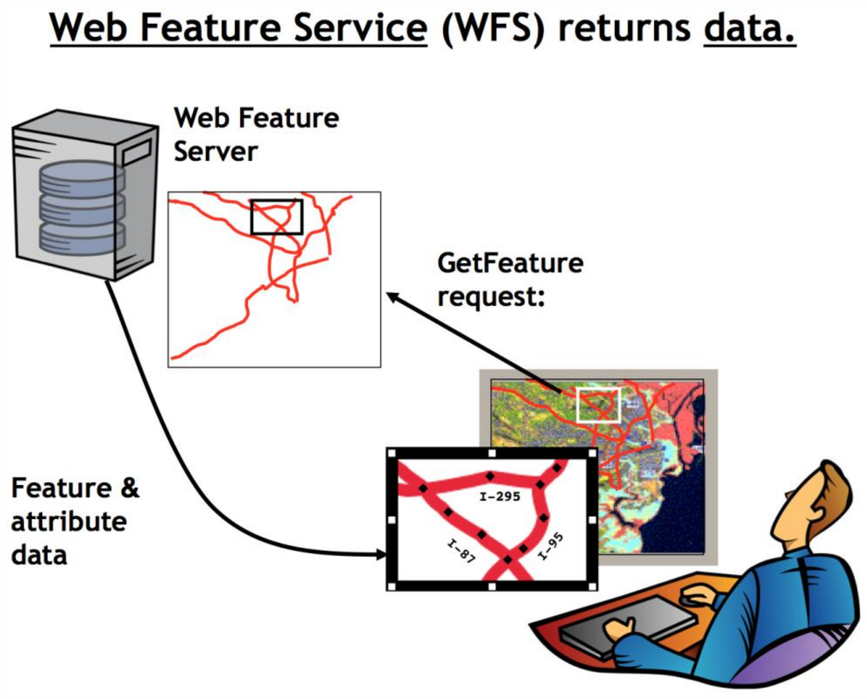


Figure 7 WFS Get Feature mechanism

This International Standard defines eleven operations:

1. GetCapabilities (discovery operation)

2. DescribeFeatureType (discovery operation)
3. GetPropertyValue (query operation)
4. GetFeature (query operation)
5. GetFeatureWithLock (query & locking operation)
6. LockFeature (locking operation)
7. Transaction (transaction operation)
8. CreateStoredQuery (stored query operation)
9. DropStoredQuery (stored query operation)
10. ListStoredQueries (stored query operation)
11. DescribeStoredQueries (stored query operation)

Figure 7 WFS Get Feature mechanism

shows the representation of a client performing a query operation to retrieve feature and attribute data. Servers that implement the OGC Web Feature Service (WFS) Interface Standard return vector source data (points, lines, and polygons) encoded in OGC Geography Markup Language (GML) format.

In the taxonomy of SOAP/REST Web Services, WFS can be classified as a SOAP type service.

4.9 OpenGIS Web Map Service

A Web Map Service (WMS) produces maps of spatially referenced data dynamically from geographic information. This International Standard defines a “map” to be a portrayal of geographic information as a digital image file suitable for display on a computer screen. A map is not the data itself. WMS-produced maps are generally rendered in a pictorial format such as PNG, GIF or JPEG, or occasionally as vector-based graphical elements in Scalable Vector Graphics (SVG) or Web Computer Graphics Metafile (WebCGM) formats.

This International Standard defines three operations: one returns service-level metadata; another operation returns a map whose geographic and dimensional parameters are well-defined; and an optional third operation returns information about particular features shown on a map.

Web Map Service operations can be invoked using a standard web browser by submitting requests in the form of Uniform Resource Locators (URLs). The content of such URLs depends on which operation is requested. In particular, when requesting a map the URL indicates what information is to be shown on the map, what portion of the Earth is to be mapped, the desired coordinate reference system, and the output image width and height. When two or more maps are produced with the same geographic parameters and output size, the results can be accurately overlaid to produce a composite map. The use of image formats that support transparent backgrounds (e.g. GIF or PNG) allows underlying maps to be visible. Furthermore, individual maps can be requested from different servers. The Web Map Service thus enables the creation of a network of distributed map servers from which clients can build customized maps.

This International Standard applies to a Web Map Service instance that publishes its ability to produce maps rather than its ability to access specific data holdings. A basic WMS classifies its geographic information holdings into “Layers” and offers a finite number of predefined “Styles” in which to display those layers.

4.10 OpenGIS Web Coverage Service

The OGC Web Coverage Service (WCS) supports electronic retrieval of geospatial data as "coverages", that is to say, digital geospatial information representing space/time-varying phenomena.

A WCS provides access to coverage data in forms that are useful for client-side rendering, as input into scientific models, and for other clients. The WCS may be compared to the OGC Web Feature Service (WFS) and the Web Map Service (WMS). As WMS and WFS service instances, a WCS allows clients to choose portions of a server's information holdings based on spatial constraints and other query criteria.

Unlike WMS, which returns spatial data to be portrayed as static maps (rendered as pictures by the server), the Web Coverage Service provides available data together with their detailed descriptions; defines a rich syntax for requests against these data; and returns data with its original semantics (instead of pictures) which may be interpreted, extrapolated, etc., and not just portrayed.

Unlike WFS, which returns discrete geospatial features, the Web Coverage Service returns coverages representing space/time-varying phenomena that relate a spatio-temporal domain to a (possibly multidimensional) range of properties. As such, WCS focuses on coverages as a specialized class of features and, correspondingly, defines streamlined functionality.

4.11 OpenGIS Web Processing Service

In many cases, geospatial or location data, including data from sensors, must be processed before the information can be used effectively. The OGC Web Processing Service (WPS) Interface Standard provides a standard interface that simplifies the task of making simple or complex computational processing services accessible via web services. Such services include well-known processes found in GIS software as well as specialized processes for spatio-temporal modeling and simulation. While the OGC WPS standard was designed with spatial processing in mind, it can also be used to readily insert non-spatial processing tasks into a web services environment.

The WPS standard provides a robust, interoperable, and versatile protocol for process execution on web services. It supports both immediate processing for computational tasks that take little time and asynchronous processing for more complex and time consuming tasks. Moreover, the WPS standard defines a general process model that is designed to provide an interoperable description of processing functions. It is intended to support process cataloguing and discovery in a distributed environment.

4.12 OpenGIS 3D Services

The Open Geospatial Consortium (OGC) has recently considered the need for standardizing the processing of massive heterogeneous 3D geospatial datasets to support the streaming and rendering requirements of visualization software. In this regard, the Open Geospatial Consortium (OGC) and the Web3D Consortium have both been working to address the need for interoperability, as well as the content challenges of volume, access speed, and diversity of devices. The Web3D Consortium has focused on open standards for real-time 3D visualization, including streaming, and their members developed a Geospatial Component extension for X3D. The OGC has focused on developing a service interface to provide interoperable access to 3D geospatial data servers. In 2012, a group of OGC members, building on work done in both organizations, completed the 3D Portrayal Interoperability Experiment (3DPIE) to develop and evaluate best practices for 3D portrayal services. Based on the results of the 3DPIE, an OGC 3D Portrayal Service Standards Working group (3D Portrayal Service SWG) was chartered to progress two different OGC proposals to the state of one integrated, adopted OGC standard. The current draft candidate 3D Portrayal Service Standard, a unified web service for 3D portrayal, released in 2015, is intended to make it easy for applications to present, explore, and analyze complex 3D geospatial data from diverse sources.

In parallel to the 3D Portrayal Service Standard, the Open Geospatial Consortium considered in 2016 the start of a new work item for a Community Standard: 3D Tiles. Bringing techniques from graphics research, the movie industry, and the game industry to 3D geospatial, 3D Tiles define a spatial data structure and a set of tile formats designed for 3D, and optimized for streaming and rendering. The initial tile formats are:

- Batched 3D Models – for buildings, terrain, massive models, etc.
- Instanced 3D Models – for trees, bolts, valves, etc.
- Point Clouds – for massive point clouds.
- Vector Data – for 3D points, polylines, and polygons, including extrusions.
- Composite – a tile of tiles to allow aggregation.

4.13 Data and tools

4.13.1 Open Data

The interoperability layer must guarantee the portability of the applications between cities and platforms. This way, a real ecosystem among applications is build, easing the application development.

According to the World Wide Web Consortium (W3C) [https://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData], the Open Data Movement aims at making data freely available to everyone. Therefore, Linked Open Data's goal is to enable linking the Open Data using Semantic Web technologies making the data inter-operable.

The Open Definition project [http://opendefinition.org/] sets out principles that define "openness" in relation to data and content. It makes precise the meaning of "open" in the terms "open data" and "open content" and thereby ensures quality and encourages compatibility between different pools of open material. It can be summed up in the statement that:

"Open means anyone can freely access, use, modify, and share for any purpose (subject, at most, to requirements that preserve provenance and openness)."

The full Open Definition [http://opendefinition.org/okd/] gives precise details as to what this means. To summarize the most important:

- **Availability and Access:** the data must be available as a whole and at no more than a reasonable reproduction cost, preferably by downloading over the internet. The data must also be available in a convenient and modifiable form.
- **Re-use and Redistribution**: the data must be provided under terms that permit re-use and redistribution including the intermixing with other datasets.
- **Universal Participation**: everyone must be able to use, re-use and redistribute - there should be no discrimination against fields of endeavor or against persons or groups. For example, 'non-commercial' restrictions that would prevent 'commercial' use, or restrictions of use for certain purposes (e.g. only in education), are not allowed.

It is important to be clear about what open means and why this definition is used, there's a simple answer: interoperability. It should be mentioned that not all data inside the organization is made public and still the platforms are considered Open Data.

The Open Data approach to be followed in SmartEnCity will consider the following principles:

- Agree with the principles established by the Open Definition Initiative (Availability and Access, re-use and redistribution, and universal participation) to provide interoperability.
- Consider privacy, personal data and other restricted data due to security (mainly related to municipality) from datasets made open.
- Provide open data licenses over datasets made available
- Open datasets that are demanded by stakeholders and not the whole databases (for example those necessary to calculate KPIs)
- Pre-process certain data according to stakeholder expectations
- Identify/address common fears and misunderstandings related to openness

4.13.2 Development Kit

A software development kit or "SDK" is a set of software development tools that allows the creation of applications for a certain software package, software framework, hardware platform, computer system, video game console, operating system, or similar development platform.

To enrich applications with advanced functionalities, most app developers implement specific software development kits. Some SDKs are critical if you want to develop applications in a specific operative system. For example, the development of an Android app requires an SDK with Java, for iOS apps iOS SDK with Swift and for MS Windows the .NET Framework SDK with .NET. SDKs also frequently include sample code and supporting technical notes or other supporting documentation to help clarify points made by the primary reference material.

SDKs may have attached licenses that make them unsuitable for building software intended to be developed under an incompatible license. For example, a proprietary SDK will probably be incompatible with free software development, while a GPL-licensed SDK could be incompatible with proprietary software development. LGPL SDKs are typically safe for proprietary development.

4.13.3 APIs

In computer programming, an application programming interface or “API” is a set of subroutine definitions, protocols, and tools for building application software. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer. An API may be for a web-based system, operating system, database system, and computer hardware or software library.

An API specification can take many forms, but often includes specifications for routines, data structures, object classes, variables or remote calls. Java APIs, Microsoft Windows API, the C++ Standard Template Library are examples of different forms of API.

Just as a graphical user interface makes it easier for people to use programs, application programming interfaces make it easier for developers to use certain technologies in building applications.

- **Libraries and frameworks:** An API is usually related to a software library: the API describes and prescribes the expected behavior (a specification) while the library is an actual implementation of this set of rules.
- **Operating system:** An API can specify the interface between an application and the operating system.
- **Remote APIs:** Remote APIs allow developers to manipulate remote resources through protocols, specific standards for communication that allow different technologies to work together, regardless of language or platform.
- **Web APIs:** Web APIs are the defined interfaces through which interactions happen between an enterprise and applications that use its assets. An API approach is an architectural approach that revolves around providing programmable interfaces to a set of services to different applications serving different types of consumers.

5 Interoperability mechanisms by using APIs

As mentioned before, the Interoperability Layer offers a list of interfaces and functionalities. Application Programming Interfaces (API) is an important part of this layer, because they are the main entry point to access the data, being the connector between the Vertical applications and knowledge layer.

Every Vertical or Intelligence Service with inputs at the Acquisition/Interconnection Layer may have an interface to offer to its own service or even other services (3rd parties) that are interested on consuming those functionalities.

The interaction between both layers (Intelligent Service, Knowledge) must be unidirectional as a client-server relation where Vertical applications and Intelligent Services received data and information from Knowledge Layer.

All APIs have the same structure; Vertical Service name in order to group different sets of services inside the SmartEnCity platform, the functionality name itself with a descriptive name of it service and optional configuration by parameters.

- Domain name: Server name where the service is hosted
- Vertical/service group name: appellation of the API group
- Requested functionality: descriptive name of the offered service
- Customizable parameter: possibility to specify or select a customized petition of information

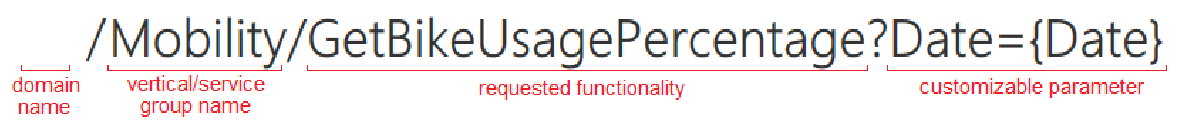
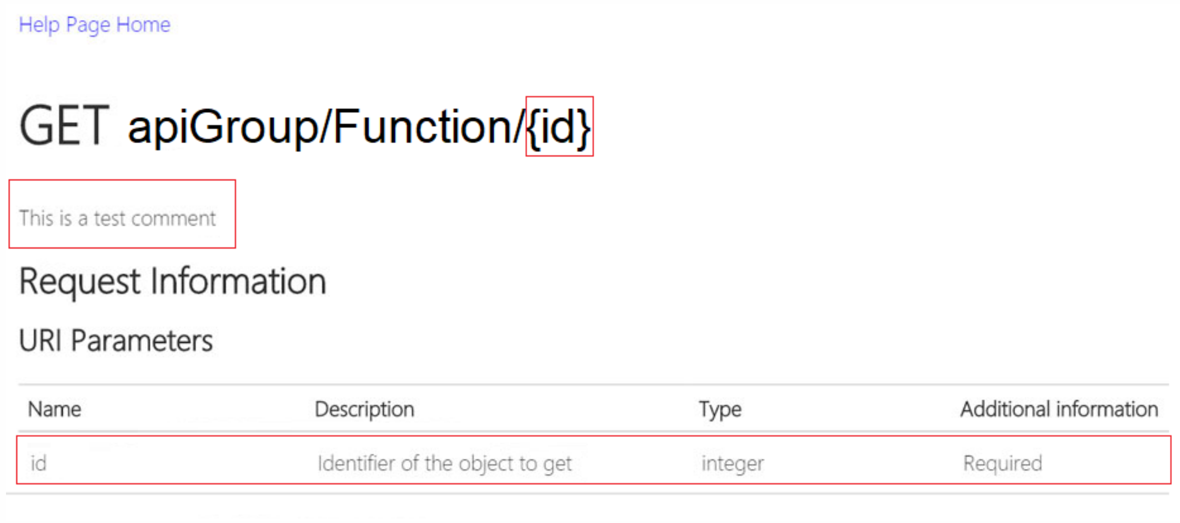

/Mobility/GetBikeUsagePercentage?Date={Date}
domain name
vertical/service group name
requested functionality
customizable parameter

Figure 8 API Get request structure

Every API service contains a documentation file within the service to explain the main purpose of each service. This information is trivial especially in cases where it has customizable parameters.


Help Page Home

GET apiGroup/Function/{id}

This is a test comment

Request Information

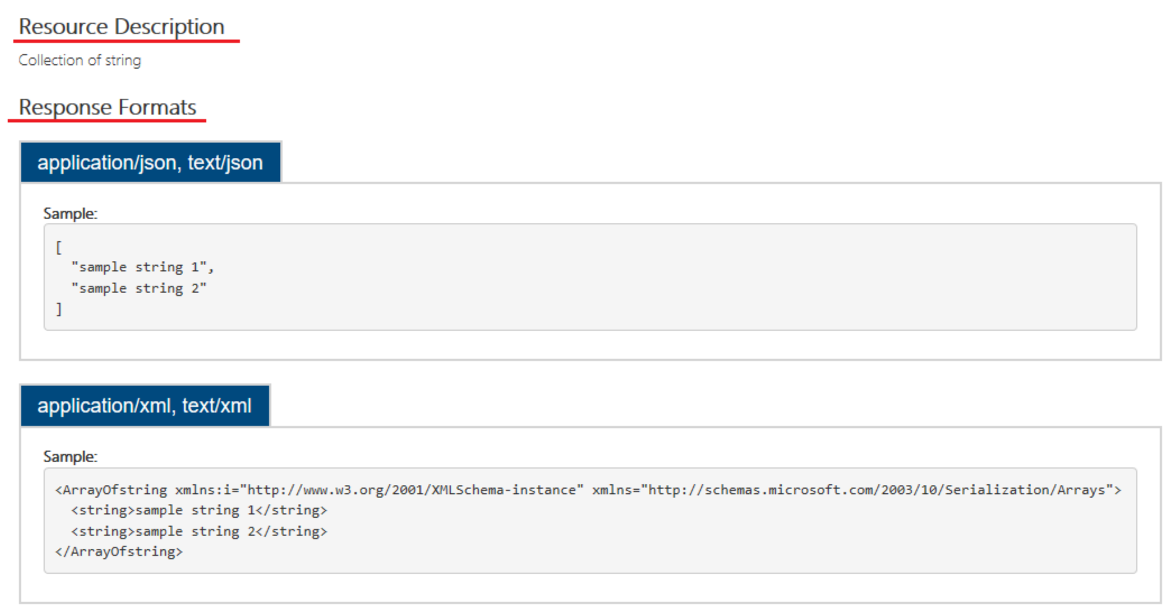
URI Parameters

Name	Description	Type	Additional information
id	Identifier of the object to get	integer	Required

Figure 9 Description example of a customizable request

Making a request, URI parameters are as much important as response content is. Besides the description and an example of the requirements to implement a request, each service has a description of what would be obtained as a response before those calls.

Every API defines its protocol response as well as response format. In any event, documentation brings descriptions of obtained answer and examples of it as shown on the next figure.



Resource Description

Collection of string

Response Formats

application/json, text/json

Sample:

```
[
  "sample string 1",
  "sample string 2"
]
```

application/xml, text/xml

Sample:

```
<ArrayOfstring xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
  <string>sample string 1</string>
  <string>sample string 2</string>
</ArrayOfstring>
```

Figure 10 Example of response

API services even being offered in the SmartEnCity platform does not mean that any Vertical can use them, in some cases functionalities would be available only for some consumers. Restrictions are a security shared pattern in every interface in order to standardize security implementation on the Interoperability Layer. A proposed agreement to provide security using APIs, API-Keys will be used in order to determine which Vertical has permissions to use all the services that the Interoperability Layer has to offer.

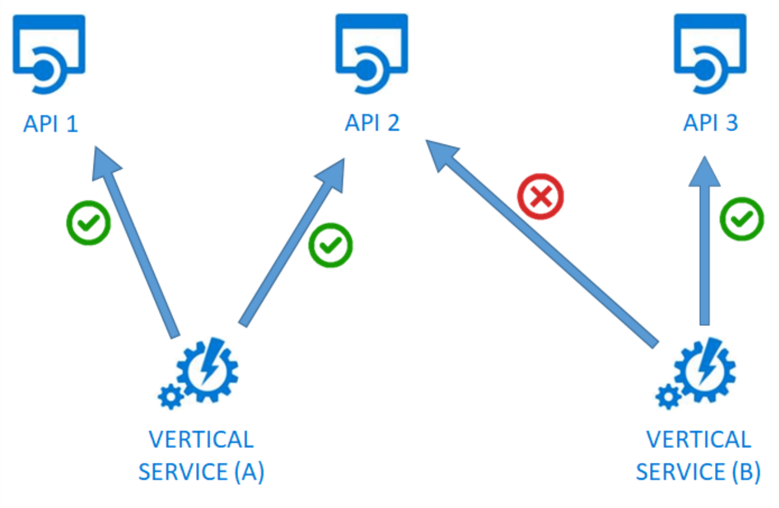


Figure 11 Verticals trying to Access API resources

6 Interoperability mechanisms by using GIS data

6.1 Introduction

In this section, the interoperability mechanisms for incorporating GIS data in the Intelligent Services will be explained. These interoperability mechanisms are only used for the applications and services that need geographical data.

The GIS repository and the structural data repository offer interoperability mechanisms but their implementations have been done with different geospatial tools. The GIS repository development is based in GeoServer tool while structural repository is based in deegree tool.

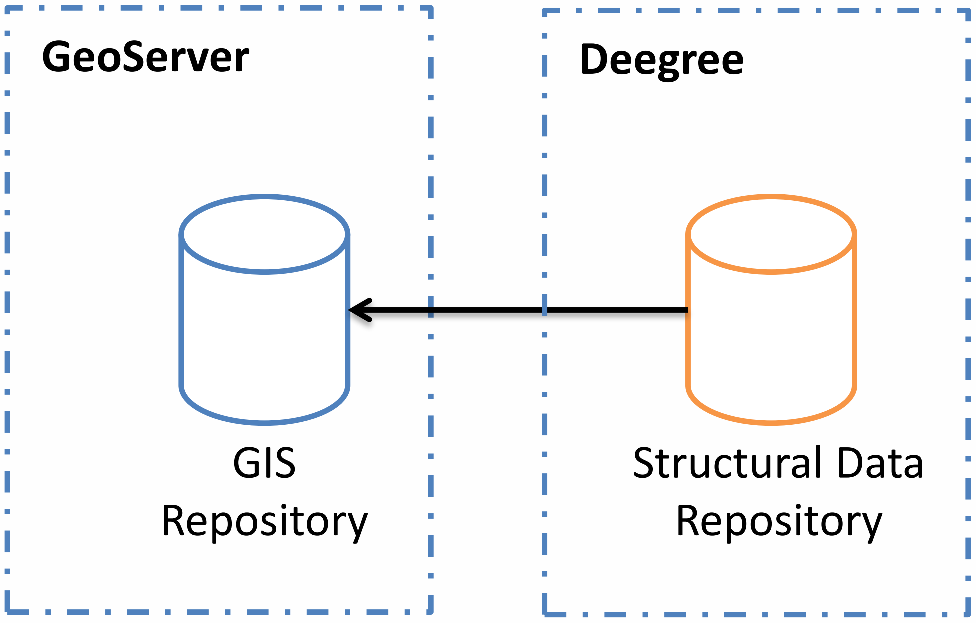


Figure 12 Components of GIS information

The following draw represents the different components for incorporating GIS data to Intelligent Services.

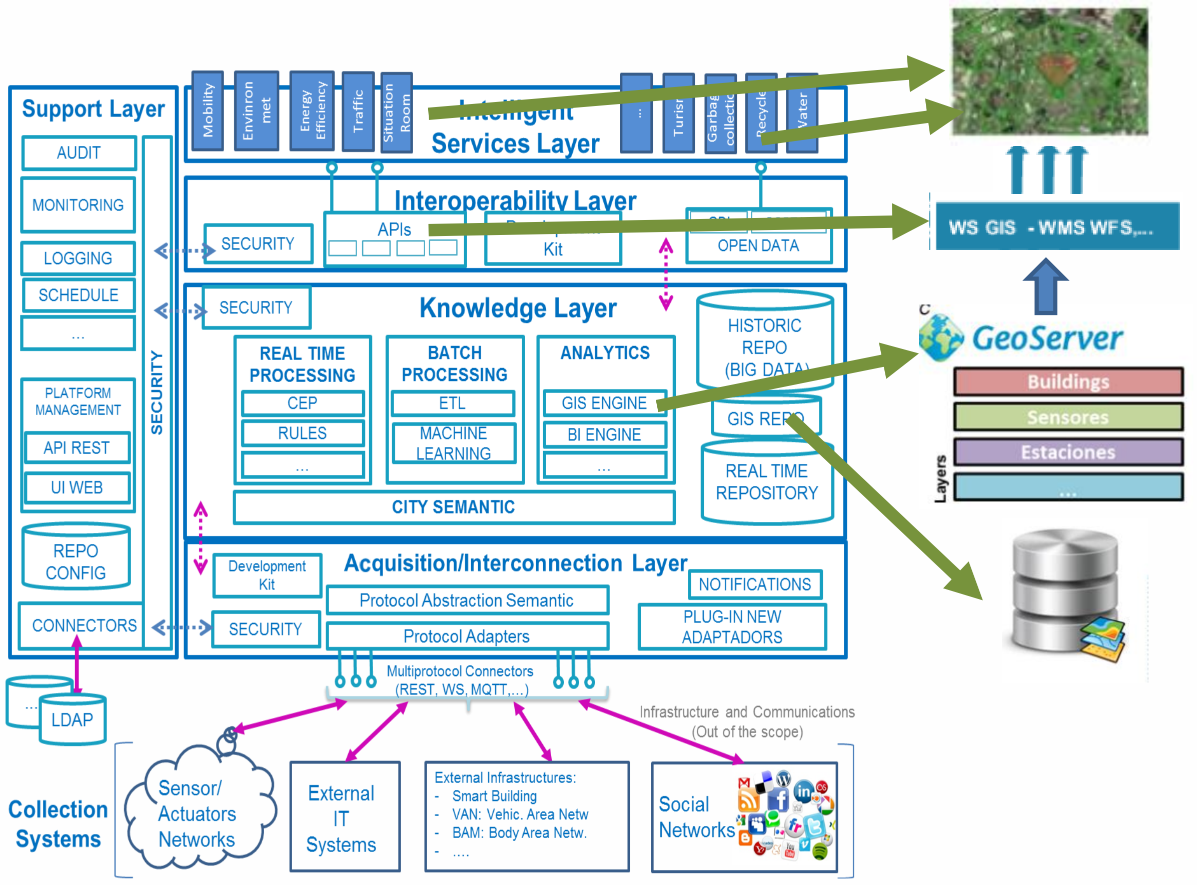


Figure 13 Implementation of GIS access

6.2 Tools

6.2.1 Geoserver

Geoserver is an Open source geospatial server implemented in Java. Geoserver allows to access to its geographical data through OGC Standard Services like WMS (Web Map Service) or WFS (Web Feature Service).

The role of the GIS Server Geoserver is to query the geographic information stored in the GIS Repo and publish it using different styles. As said before, this information is accessible with the use of open standards so it allows the interoperability.

The online resource of each operation supported by a WMS or WFS server is an HTTP URL. Using a web browser and adding parameters to this URL, it is possible to make a request to perform any operation defined in the standards. The responses are also returned by the web browser.

In Section 6.3, we present more in detail the format of the requests for the operations that are supported by the SmartEnCity WMS and WFS server and also an example of an URL that can be used to retrieve the data.

6.2.2 Deegree

Deegree is open source software for spatial data infrastructures and the geospatial web and offers components for geospatial data management, including data access, visualization, discovery and security. deegree web services are implementations of the Geospatial Web Service Specifications of the Open Geospatial Consortium (OGC) and the INSPIRE Network Services.

Deegree is a Java API for geographic information systems, which is a large set of Java classes that can be used to create a GIS tools. There's already a main application developed by deegree, which are the deegree OGC-conform Web Services, which includes most comprehensive set of OGC Web services, such as: WFS, WMS, WMTS, CSW or WPS.

6.3 GIS Repo Services

These are the requests that can be performed with WMS server deployed for SmartEnCity project:

- **GetCapabilities**: Describe the service capabilities and retrieves metadata about the service including supported operations and parameters, and a list of the available layers.
- **GetMap**: This operations generates map image for a specified area and content
- **GetFeatureInfo (optional)**: Retrieves the underlying data, including geometry and attribute values, for a pixel location on a map.
- **DescribeLayer (optional):** Indicates the WFS or WCS to retrieve additional information about the layer.
- **GetLegendGraphic (optional)**: Gets the legend of the map.

GetCapabilities

This operation provides metadata about the capabilities of a WMS service. It contains among others a list of the available layers and operations supported:

The parameters for the GetCapabilities operation are:

- **service:** Service name.
- **version**: Service version.
- **request**: Operation name.

Example of WMS GetCapabilities request:

http://geoservergis.azurewebsites.net/geoserver/wms?
SERVICE=WMS
&VERSION=2.0.0
&REQUEST=GetCapabilities
&

GetMap

This operation retrieves the corresponding map image specified in the request.

The parameters for GetMap request operation are:

- **service:** Service name.
- **version:** Service version.
- **request:** Operation name.
- **layers:** Id of the layers that are going to be retrieved.
- **styles**: Styles in which layers are to be rendered.
- **srs/crs:** Spatial Reference System for map output.
- **bbox**: Bounding box for map extent.
- **width**: Width in pixels.
- **height:** Height in pixels.
- **format**:: Format for the map output.
- **transparent (optional):** Indicates if the background is transparent or not.
- **bgcolor (optional):** Background color for the map image.
- **exceptions (optional):** Format to report the exceptions.
- **time (optional):** Time value or range for map data.
- **sld (optional):** Reference to a StyledLayerDescriptor XML file which controls or enhances map layers and styling.
- **sld_body(optional)**: A URL-encoded StyledLayerDescriptor XML document which controls or enhances map layers and styling.

Example of WMS GetMap request:

http://geoservergis.azurewebsites.net/geoserver/wms?
SERVICE=WMS
&VERSION=2.0.0
&REQUEST=GetMap
&layers=smartencity:bench
&styles=point
&crs=EPSG:25830
&bbox=520527.38458945166,4741647.821567042,529572.1722914433,4747224.15764
44805
&width=100
&height=100
&format=image/png
&

GetFeatureInfo

This operation provides the information about features at a given location in the map.

The parameters for the GetFeatureInfo operation are:

- **service:** Service name.
- **version**: Service version.
- **request**: Operation name.
- **layers:** Id of the layers that are going to be retrieved.
- **styles**: Styles in which layers are to be rendered.
- **srs/crs:** Spatial Reference System for map output.
- **bbox**: Bounding box for map extent.

- **width**: Width in pixels.
- **height:** Height in pixels.
- **query_layers:** Id of the layers that are going to query to get the information.
- **x/i:** X coordinate for the point to query.
- **y/j:** Y coordinate for the point to query.
- **exceptions (optional):** Format to report the exceptions.
- **info_format (optional):** Selected format to get the response.
- **feature_count (optional):** Maximum number of features to be returned.

Example of WMS GetFeatureInfo request:

http://geoservergis.azurewebsites.net/geoserver/wms?
SERVICE=WMS
&VERSION=2.0.0
&REQUEST=GetFeatureInfo
&layers=smartencity:bench
&styles=point
&crs=EPSG:25830
&bbox=520527.38458945166,4741647.821567042,529572.1722914433,4747224.15
76444805
&width=200
&height=200
&query_layers=smartencity:bench
&x=100
&y=100
&

DescribeLayer

This operation provides information about the structure of the data.

The parameters for the DescribeLayer operation are:

- **service:** Service name.
- **version**: Service version.
- **request**: Operation name.
- **layers:** Id of the layers that are going to be retrieved.
- **exceptions (optional):** Format to report the exceptions.
- **outputFormat(optional):** Format to code the output.

Example of WMS DescribeLayer request:

http://geoservergis.azurewebsites.net/geoserver/wms?
SERVICE=WMS
&VERSION=1.1.1
&REQUEST=DescribeLayer
&layers=smartencity:trees
&outputFormat=application/json
&

GetLegendGraphic

This operation generates the legend graphic of the map as image. The legend obtained is based on the styles that are defined on the server or can use a SLD supplied by the user in the request.

Although the legend appearance can be controlled using a “legend_options” parameter, in terms of simplicity we are going to focus this analysis on the standard request parameters.

The parameters for the GetLegendGraphic operation are:

- **request**: Operation name.
- **layer:** Id of the layer to generate the legend.
- **format:** Specifies the format to get the legend output.
- **style (optional):** Style to generate the legend. If not indicated it is used a default style
- **featureType (optional):** Feature type to generate de legend in case the layer has more than one feature types
- **rule (optional):** Rule of style to produce the legend
- **scale (optional):** Select a suitable representation
- **sld (optional):** Specifies an external sld for styling the legend
- **sld_body (optinal):** Allows an SLD document to be included directly in an HTTP-GET request.
- **width (optional):** Output legend width (in pixels).
- **height (optional):** Output legend height (in pixels)
- **language (optional):** Allows set the labels in the specified language.

Example of WMS GetLegendGraphic request:

http://geoservergis.azurewebsites.net/geoserver/wms?
&VERSION=2.0.0
&REQUEST=GetLegendGraphic
&layer=smartencity:trees
&FORMAT=image/png
&

6.4 WFS Requests

These are the requests that can be performed with WFS server deployed for SmartEnCity project:

- **GetCapabilities**: Describe the service capabilities and retrieves metadata about the service including supported operations and parameters, and a list of the available layers.
- **DescribeFeatureType**: This operation describes the structure of the feature types that are supported by the server.
- **GetFeature**: Retrieves the underlying data, including geometry and attribute values, for a given feature.

- **Transaction (optional):** Allows transaction operations to edit the features by creating, updating or deleting.
- **GetPropertyValue:** Gets the value of a feature property.
- **CreateStoredQuery :** Creates a stored query.
- **DropStoredQuery:** Deletes a stored query.
- **ListStoredQueries:** List the stored queries.
- **DescribeStoredQueries:** Describes the stored queries.

GetCapabilities

This operation provides metadata about the capabilities of a WFS service. It contains among others a list of the available layers and operations supported:

The parameters for the GetCapabilities operation are:

- **service:** Service name.
- **version**: Service version.
- **request**: Operation name.

Example of WFS GetCapabilities request:

http://geoservergis.azurewebsites.net/geoserver/wfs?
SERVICE=WFS
&VERSION=2.0.0
&REQUEST=GetCapabilities
&

DescribeFeatureType

This operation requests for the information about an individual feature type:

The parameters for the DescribeFeatureType operation are:

- **service:** Service name.
- **version**: Service version.
- **request**: Operation name.
- **typeNames**: Name of the feature type to describe.
- **exceptions (optional):** Format to report the exceptions.
- **outputFormat (optional):** Format to code the output.

Example of WFS DescribeFeatureType request:

http://geoservergis.azurewebsites.net/geoserver/wfs?
SERVICE=WFS
&VERSION=2.0.0
&REQUEST=DescribeFeatureType
&typeNames=smartencity:bike_parkingType
&

GetFeature

This operation returns the underlying data for a selection of features from the data source. Using only the required parameters for this operation it will return the geometries and data for all features on the specified *typeNames* parameter. This is a presumably large amount of data so, to avoid this, but this operation also defines some optional parameters that can be combined to filter the results to get the data based on different properties.

The parameters for the GetFeature operation are:

- **service:** Service name.
- **version**: Service version.
- **request**: Operation name.
- **typeNames**: Name of the feature type to describe.
- **featureID (optional)**: ID of the specific feature.
- **count (optional):** Define the maximum number of the features returned.
- **sortBy(optional):** Sort (in ascending order) the features returned by this parameter. Adding "+D" to the parameter will return the features sorted in descending mode, ie: sortBy = someattribute+D
- **propertyName (optional):** Retrieve the results based on attribute instead of based on a feature.
- **srs/crs (optional):** Spatial Reference System for map output.
- **bbox (optional)**: Bounding box for map extent.

Example of WFS GetFeature requests combining different parameters:

http://geoservergis.azurewebsites.net/geoserver/wfs?
SERVICE=WFS
&VERSION=2.0.0
&REQUEST=GetFeature
&typeNames=smartencity:bike_parking
&

http://geoservergis.azurewebsites.net/geoserver/wfs?
SERVICE=WFS
&VERSION=2.0.0
&REQUEST=GetFeature
&typeNames=smartencity:bike_parking
&count=10
&sortBy=CODIGO
&

GetPropertyValue

This operation returns the value of a given property specified in the url.

The parameters for the GetPropertyValue operation are:

- **service:** Service name.
- **version**: Service version.
- **request**: Operation name.
- **typeNames**: Name of the feature type to describe.
- **valueReference**: Name of the attribute to get the results.

Example of WFS GetPropertyValue request:

> http://geoservergis.azurewebsites.net/geoserver/wfs?
> SERVICE=WFS
> &VERSION=2.0.0
> &REQUEST=GetPropertyValue
> &typeNames=smartencity:water_supply
> &valueReference=TIPO_SALID
> &

CreateStoredQuery

This operation creates a Stored Query in the server.

In this case we are going to show a POST request to create a new stored query in the WFS server.

```
<wfs:CreateStoredQuery service='WFS' version='2.0.0'
 xmlns:wfs='http://www.opengis.net/wfs/2.0'
 xmlns:fes='http://www.opengis.org/fes/2.0'
 xmlns:gml='http://www.opengis.net/gml/3.2'
 xmlns:myns='http://www.someserver.com/myns'
 xmlns:topp='http://www.openplans.org/topp'>
  <wfs:StoredQueryDefinition id='water_area'>
    <wfs:Parameter name='AreaOfInterest' type='gml:Polygon'/>
    <wfs:QueryExpressionText
     returnFeatureTypes= smartencity:water_supply'
     language='urn:ogc:def:queryLanguage:OGC-WFS::WFS_QueryExpression'
     isPrivate='false'>
      <wfs:Query typeNames='smartencity:water_supply>
        <fes:Filter>
          <fes:Within>
            <fes:ValueReference>the_geom</fes:ValueReference>
             ${AreaOfInterest}
          </fes:Within>
        </fes:Filter>
      </wfs:Query>
    </wfs:QueryExpressionText>
  </wfs:StoredQueryDefinition>
</wfs:CreateStoredQuery>
```

DropStoredQuery

This operation drops a stored query that already exists in the server

The parameters for the DropStoredQuery operation are:

- **request**: Operation name.
- **storedQuery_Id**: Id of the stored query to drop.

Example of WFS DropStoredQuery request:

http://geoservergis.azurewebsites.net/geoserver/wfs?
request=DropStoredQuery
&storedQuery_Id=water_area
&

ListStoredQueries

This operation returns a list of the stored queries available on the server.

The parameters for the ListStoredQueries operation are:

- **service:** Service name.
- **version**: Service version.
- **request**: Operation name.
- **storedQuery_Id**: Id of the stored query to drop.

Example of WFS ListStoredQueries request:

http://geoservergis.azurewebsites.net/geoserver/wfs?
service=WFS
&version=2.0.0
&request=ListStoredQueries
&

DescribeStoredQueries

This operation returns the metadata associated to each stored query in the server

The parameters for the DescribedStoredQueries operation are:

- **request**: Operation name.
- **storedQuery_Id**: Id of the stored query to describe.

Example of WFS DescribeStoredQueries request:

http://geoservergis.azurewebsites.net/geoserver/wfs?
request=DescribeStoredQueries

&storedQuery_Id=urn:ogc:def:query:OGC-WFS::GetFeatureById
&

6.5 Structural data Repository Services

6.5.1 WFS Request of Building

Example of WFS GetFeature request in order to query the information of a certain building.

Query:

```
<wfs:GetFeature outputFormat='text/xml; subtype=gml/3.2.1'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:ogc='http://www.opengis.net/ogc' xmlns:wfs='http://www.opengis.net/wfs'>
        <wfs:Query typeName='app:building_smartengasteizv1'>
                <ogc:Filter>
                        <ogc:PropertyIsLike wildCard="*" singleChar="#" escapeChar="!">
                                <ogc:PropertyName>app:id</ogc:PropertyName>
                                <ogc:Literal>1511</ogc:Literal>
                        </ogc:PropertyIsLike>
                </ogc:Filter>
        </wfs:Query>
</wfs:GetFeature>
```

Response:

```
<gml:FeatureCollection xsi:schemaLocation=">amp;http://www.opengis.net/gml/3.2
http://schemas.opengis.net/gml/3.2.1/deprecatedTypes.xsd http://www.deegree.org/app
http://3dcity.tecnalia.com/ServiciosWeb/services/smartengasteizv1?SERVICE=WFS&amp;V
ERSION=1.1.0&amp;REQUEST=DescribeFeatureType&amp;OUTPUTFORMAT=text%2Fx
ml%3B+subtype%3Dgml%2F3.2.1&amp;TYPENAME=app:building_smartengasteizv1&amp;
NAMESPACE=xmlns(app=http%3A%2F%2Fwww.deegree.org%2Fapp)" gml:id="WFS_RES
PONSE" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:gml="http://www.opengis.net/gml/3.2">
  <gml:featureMember>
    <app:building_smartengasteizv1 gml:id="APP_BUILDING_SMARTENGASTEIZV1_151
1" xmlns:app="http://www.deegree.org/app">
      <app:id>1511</app:id>
      <app:building_root_id>1480</app:building_root_id>
      <app:building_parent_id>1480</app:building_parent_id>
      <app:storeys_above_ground>5</app:storeys_above_ground>
      <app:year_of_construction>1974-01-01</app:year_of_construction>
      <app:measured_height>17.414338815308724</app:measured_height>
      <app:storeys_above_ground>5</app:storeys_above_ground>
      <app:lod2_solid_id>5264</app:lod2_solid_id>
    </app:building_smartengasteizv1>
  </gml:featureMember>
</gml:FeatureCollection>
```

6.5.2 WFS Request of thematic surface

Example of WFS GetFeature request in order to query the information of a certain thematic surface.

Query:

```
<wfs:GetFeature outputFormat='text/xml; subtype=gml/3.2.1'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:ogc='http://www.opengis.net/ogc' xmlns:wfs='http://www.opengis.net/wfs'>
        <wfs:Query typeName='app:thematic_surface_smartengasteizv1'>
                <ogc:Filter>
                        <ogc:PropertyIsLike wildCard="*" singleChar="#" escapeChar="!">
                                <ogc:PropertyName>app:building_id</ogc:PropertyName>
                                <ogc:Literal>1511</ogc:Literal>
                        </ogc:PropertyIsLike>
                </ogc:Filter>
        </wfs:Query>
</wfs:GetFeature>
```

Response:

```
<gml:FeatureCollection xsi:schemaLocation=">amp;http://www.opengis.net/gml/3.2
http://schemas.opengis.net/gml/3.2.1/deprecatedTypes.xsd        http://www.deegree.org/app
http://3dcity.tecnalia.com/ServiciosWeb/services/smartengasteizv1?SERVICE=WFS&amp;V
ERSION=1.1.0&amp;REQUEST=DescribeFeatureType&amp;OUTPUTFORMAT=text%2Fx
ml%3B+subtype%3Dgml%2F3.2.1&amp;TYPENAME=app:thematic_surface_smartengasteiz
v1&amp;NAMESPACE=xmlns(app=http%3A%2F%2Fwww.deegree.org%2Fapp)" gml:id="W
FS_RESPONSE" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:gml="http://www.opengis.net/gml/3.2">
  <gml:featureMember>
    <app:thematic_surface_smartengasteizv1 gml:id="APP_THEMATIC_SURFACE_SMAR
TENGASTEIZV1_1512" xmlns:app="http://www.deegree.org/app">
      <app:id>1512</app:id>
      <app:building_id>1511</app:building_id>
      <app:objectclass_id>34</app:objectclass_id>
      <app:lod2_multi_surface_id>5277</app:lod2_multi_surface_id>
    </app:thematic_surface_smartengasteizv1>
  </gml:featureMember>
  <gml:featureMember>
    <app:thematic_surface_smartengasteizv1 gml:id="APP_THEMATIC_SURFACE_SMAR
TENGASTEIZV1_1513" xmlns:app="http://www.deegree.org/app">
      <app:id>1513</app:id>
      <app:building_id>1511</app:building_id>
      <app:objectclass_id>34</app:objectclass_id>
```

```
          <app:lod2_multi_surface_id>5279</app:lod2_multi_surface_id>
      </app:thematic_surface_smartengasteizv1>
   </gml:featureMember>
…
```

6.5.3 WFS Request of surface geometry

Example of WFS GetFeature request in order to query the information of a certain surface geometry.

Query:

```
<wfs:GetFeature                    outputFormat='text/xml;                    subtype=gml/3.2.1'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:ogc='http://www.opengis.net/ogc' xmlns:wfs='http://www.opengis.net/wfs'>
        <wfs:Query typeName='app:surface_geometry_smartengasteizv1'>
                <ogc:Filter>
                        <ogc:PropertyIsLike wildCard="*" singleChar="#" escapeChar="!">
                                <ogc:PropertyName>app:root_id</ogc:PropertyName>
                                <ogc:Literal>1511</ogc:Literal>
                        </ogc:PropertyIsLike>
                </ogc:Filter>
        </wfs:Query>
</wfs:GetFeature>
```

Response:

```
<gml:FeatureCollection xsi:schemaLocation=">amp;http://www.opengis.net/gml/3.2
http://schemas.opengis.net/gml/3.2.1/deprecatedTypes.xsd http://www.deegree.org/app
http://3dcity.tecnalia.com/ServiciosWeb/services/smartengasteizv1?SERVICE=WFS&amp;V
ERSION=1.1.0&amp;REQUEST=DescribeFeatureType&amp;OUTPUTFORMAT=text%2Fx
ml%3B+subtype%3Dgml%2F3.2.1&amp;TYPENAME=app:surface_geometry_smartengastei
zv1&amp;NAMESPACE=xmlns(app=http%3A%2F%2Fwww.deegree.org%2Fapp)" gml:id="
WFS_RESPONSE" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:gml="http://www.opengis.net/gml/3.2">
   <gml:featureMember>
      <app:surface_geometry_smartengasteizv1 gml:id="APP_SURFACE_GEOMETRY_SM
ARTENGASTEIZV1_1511" xmlns:app="http://www.deegree.org/app">
         <app:id>1511</app:id>
         <app:root_id>1511</app:root_id>
      </app:surface_geometry_smartengasteizv1>
   </gml:featureMember>
   <gml:featureMember>
      <app:surface_geometry_smartengasteizv1 gml:id="APP_SURFACE_GEOMETRY_SM
ARTENGASTEIZV1_1512" xmlns:app="http://www.deegree.org/app">
```

```
          <gml:boundedBy>
            <gml:Envelope srsName="EPSG:25830">
              <gml:lowerCorner>526543.547 4744452.550</gml:lowerCorner>
              <gml:upperCorner>526543.547 4744452.550</gml:upperCorner>
            </gml:Envelope>
          </gml:boundedBy>
          <app:id>1512</app:id>
          <app:parent_id>1511</app:parent_id>
          <app:root_id>1511</app:root_id>
          <app:geometry>
</!--Inlined geometry
'APP_SURFACE_GEOMETRY_SMARTENGASTEIZV1_1512_APP_GEOMETRY'-->
            <gml:Polygon gml:id="APP_SURFACE_GEOMETRY_SMARTENGASTEIZV1_151
2_APP_GEOMETRY" srsName="EPSG:25830">
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList>526543.547 4744452.550 516.702 526543.547 4744452.550
533.835 526543.547 4744452.550 533.835 526543.547 4744452.550 516.702 526543.547
4744452.550                                              516.702</gml:posList>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>
          </app:geometry>
        </app:surface_geometry_smartengasteizv1>
    </gml:featureMember>
</gml:FeatureCollection>
```

7 Interoperability for integrating Open Data

7.1 Introduction

In this section the interoperability mechanisms for incorporating Open Data in the Intelligent Services will be explained. These interoperability mechanisms are only used for the applications and services that need external data that is not captured from the infrastructure of SmartEnCity project. For example, Open Data is related to meteorological data, demographic data, pollution data, traffic data, etc.

The following draw represents the different components for incorporating Open Data to Intelligent Services.

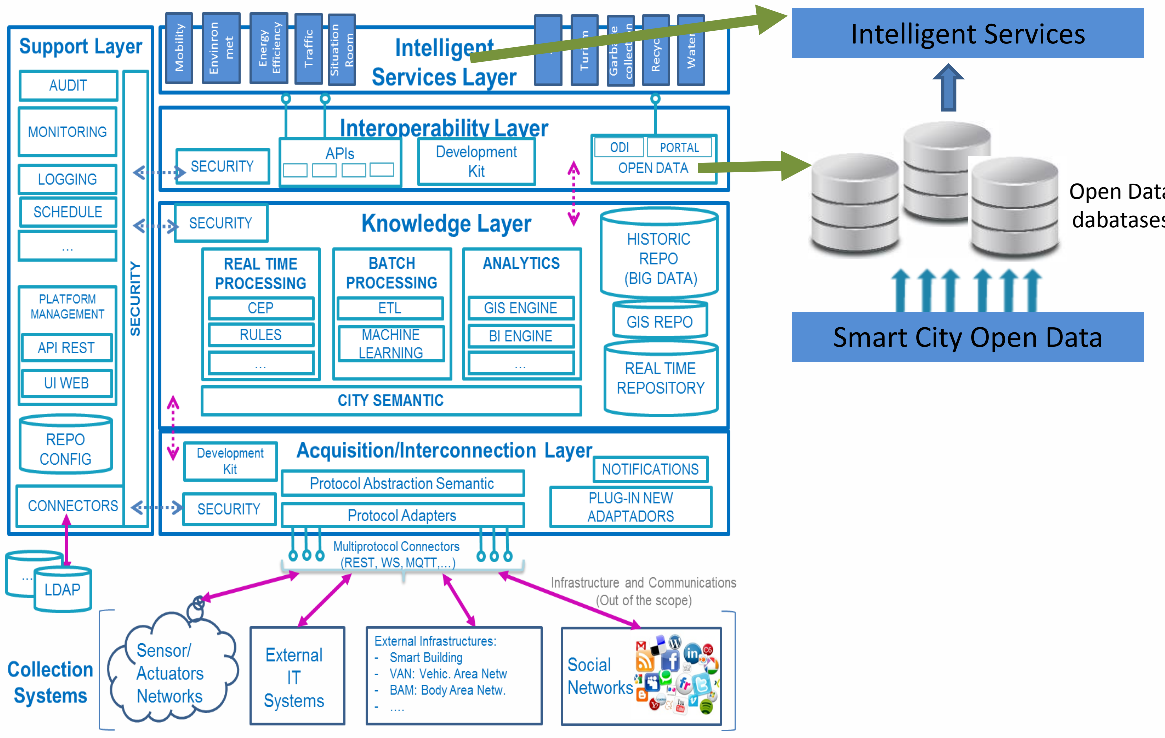


Figure 14 Implementation of Open Data access

Open Data is a general way to talk about a lot of different sources of data that can help to develop powerful applications and intelligent services for private companies, public administrations, citizens and the whole city.

In this context, data is a valuable asset and an essential resource for almost any activity in our society that everyone assumes that it should be shared. Proper management of all the data coming from the city will allow to understanding what is happening and to make taken the right decisions to ensure optimal management of resources, as well as to meet the demands of its people efficiently.

In the context of SmartEnCity, Open Data can be used in two completely different contexts. On the one hand, third party Open Data can be an additional source of information to the platform and, on the other hand, the platform can offer some of its own data to third parties through existing Open Data interoperability standards.

In this demonstrator, third party Open Data will be consumed to provide additional environmental information in order to give more precise recommendations to final users. Open Data related to climate and meteorological historic data and weather forecasts will be used in the demonstrator by using HTTP(S) REST API.

The weather historic data will be added to the Historical Data Repository to be able to calculate those KPIs that are related to weather conditions facilitating the production of more added-value reports and dashboards.

8 Interoperability mechanisms for providing KPIs

Independently and separately to any implementation of the Reference Architecture of any city, a common holistic methodology for the assessment of the performance has to be achieved. To do so, a methodology and the KPIs proposed in the deliverable D7.2 will allow to measure the objectives established in each city from technical, environmental, economic and social points of view. The assessment methodology covers seven protocols which will be applied for the evaluation of different issues, which will be detailed along the current report.

- Energy Assessment
- ICT Assessment
- Life Cycle Analysis
- Mobility
- Social Acceptance
- Citizen Engagement
- Economic performance

At this moment in the project the detail for all ICT applications and systems to be implemented/installed in the Verticals of work packages 3, 4 and 5 is not fully available so the efforts have been put in demonstrate the capability of developing the Reference Architecture and building special databases and other components for the calculation of the KPIs defined in WP7.

In order to calculate the evaluation KPIs selected in WP7, the Reference Architecture has to incorporate an interoperability mechanism to provide the calculated KPIs or the relevant information and variables to allow the KPIs calculation.

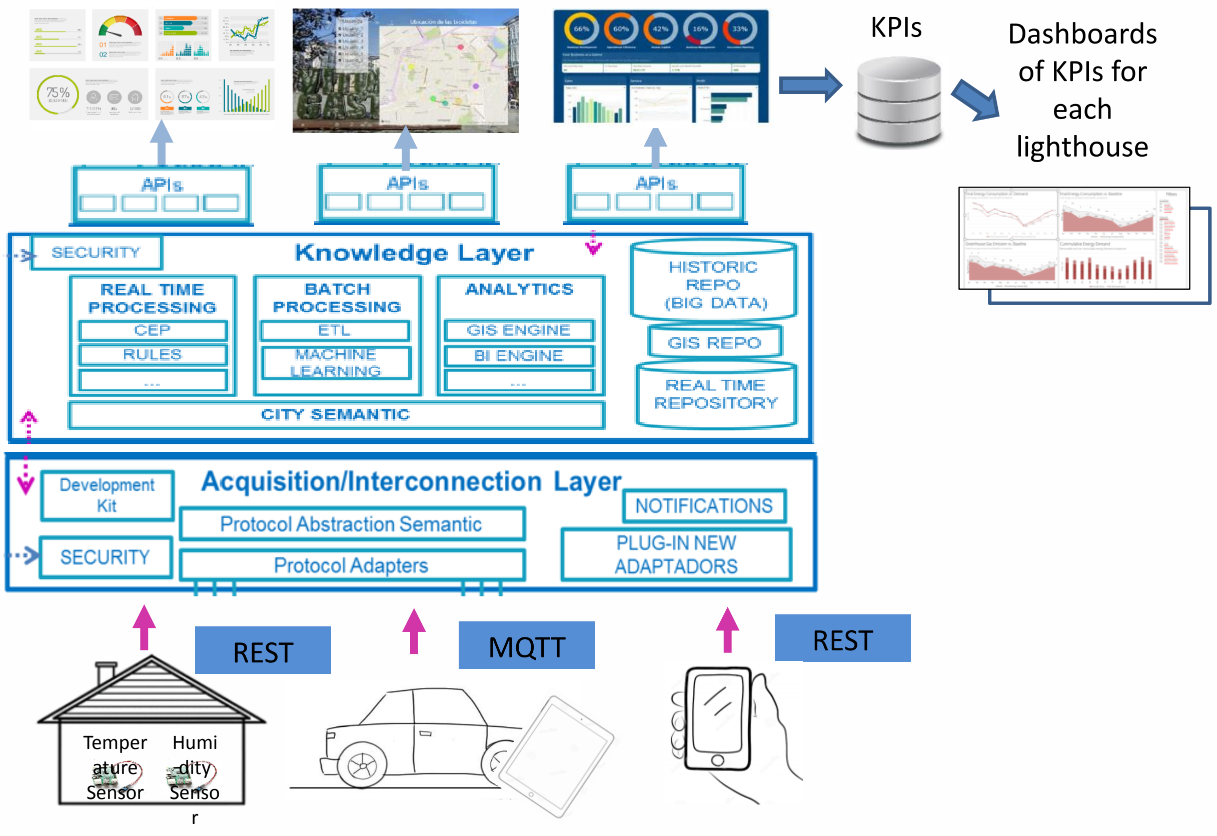


Figure 15 Interoperability mechanisms for KPIs

An external common database is created where the needed data for the calculation of the KPIs or the calculated values of the KPIs provided by each lighthouse are stored. This database will provide the data for the development of dashboards that will show the behaviour and the performance of each lighthouse concerning with the sustainability and the performance achieved after the execution of sustainable interventions.

In WP7 the list of KPIs has been defined for evaluating different aspects of SmartEnCity project and each lighthouse. In WP3, WP4 and WP5 in each one of the lighthouse each one of the Verticals will specify the way to calculate the WP7 KPIs in an independent manner and they will define the specific formula for each KPI.

8.1.1 Mechanisms for providing data for KPIs

There will be two different methods for uploading data to the KPIs database from the Verticals and Intelligent Services. In one hand, the data will be already calculated and only their values are uploaded to the KPIs database. On the other hand, each lighthouse may upload all the needed variables to allow the KPIs calculation be done by a calculator application.

So, there will be two different proceedings:

- **Excel or csv files:** Both options are valid. In one case, there will be an excel template to be fulfilled each month with new calculated KPIs and in the case of a csv file there will be an automated process for generating the data.

- **Automatic KPI calculation**: In this case there will be automated processes for uploading all the variables needed to calculate the KPIs. The Vertical application will be in charge of providing all the needed information.

By suing KPIs database some dashboards and reports will be developed to show the evolution and the behavior of the lighthouses.

9 Security at Interoperability Layer

At any layer of the Reference Architecture there are needed some security components and some authorization interfaces in order to guarantee the access only to validated users and to avoid the attempt of access of intruders and non-authorized users.

The purpose of the security module within the system is to develop a global system of user authentication, one type of authentication and global configuration for all users of the SmartEnCity demonstrators. At the level of the global system there will be a centralized user authentication that will allow the access and give the permissions to get at the different layer of the architecture. In that way the access to the interoperability layer and the management of the APIs is only allowed to the users with the right permissions.

An API built in REST architecture should have URLs for its resources, where the operation executed on a resource is invoked via an HTTP method. If an HTTP GET request is sent, the API would return user data in JSON or XML format. If a POST request is send, user data would be updated. If a POST request is sent (and the user id is not passed to the server as a parameter), a new user would be created. Finally, if a DELETE request is sent, the user with the id specified would be deleted.

Another important characteristic of the REST API architecture is to return HTTP status codes. Some of the most common codes are: 404 not found, 200 OK, 400 bad request, 401 unauthorized.

Hereafter some of the security methods are explained.

- **Access Control**

Access Control is the best method to protect access to REST APIs.

- API Key*:* API Key can be used for every API request. If there are any suspicious behaviours in the API requests, it will be able to be identified by the API Key and revoke the specific API Key for further requests. Furthermore, API rate limits control can also enforce on the API key as source of identifier of every API request.

- *API Rate limits*: The objective of the API Rate limits is to reduce massive API requests that cause denial of services, and also to mitigate potential brute-force attack, or misuses of the services. The following API rate limits mechanism can be considered.

  API rate limits per application or per API: Every API or application can only access the services for defined the number of requests per rate limit window.

  API rate limits per GET or POST request: The allowed access requests may vary based on GET or POST requests per period.

  The results of exceeding API rate limits can be temporarily blacklisted the application/API access or notification alert to relevant users/admin. The service should return HTTP return code. "429 Too Many Requests" - The error is used when there may be DOS attack detected or the request is rejected due to rate limiting.

HTTP error return code: If there are too many error return (i.e. 401, 404, 501...), the identifier of the API (API Key) will be blocked temporarily for further access.

- **Input Validation**

Input validation failures have to be controlled to avoid many failed input validations and sometimes it's good to rate a limit of number of requests per hour or day to prevent abuse.

When POSTing or PUTting new data, the client will specify the Content-Type of the incoming data. The server should never assume the Content-Type; it should always check that the Content-Type header and the content are the same type. A lack of Content-Type header or an unexpected Content-Type header should result in the server rejecting the content with a 406 Not Acceptable response.

XML-based services must ensure that they are protected against common XML based attacks by using secure XML-parsing. This typically means protecting against XML External Entity attacks, XML-signature wrapping, etc.

- **Output Encoding**

To make sure the content of a given resources is interpreted correctly by the browser, the server should always send the Content-Type header with the correct Content-Type.

  - *JSON encoding:* It's vital to use a proper JSON serializer to encode user-supplied data properly to prevent the execution of user-supplied input on the browser.
  - *XML encoding:* XML should always be constructed using an XML serializer. This ensures that the XML content sent to the browser is parseable and does not contain XML injection.

- **Cryptography**

  - *Data in transit:* Unless the public information is completely read-only, the use of TLS should be mandated, particularly where credentials, updates, deletions, and any value transactions are performed. The overhead of TLS is negligible on modern hardware, with a minor latency increase that is more than compensated by safety for the end user.
  - *Data in storage:* In order to correctly handle stored sensitive or regulated data, cryptography is a good tool.

OAuth 2.0 is an open standard for authorization that provides a process for end-users to authorize third-party access to their server resources without sharing their credentials (typically, a username and password pair). The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

OAuth works in the following way:

- The consumer requests a *request token* (usually by passing an *application key* and *application secret*)
- The user is then redirected to a login page, passing the request token to that page
- User logs in and is redirected back to the consumer, passing the request token to the consumer's page
- The consumer exchanges the request token for an *access token*
- If the previous request was valid, the server will return an access token to the consumer. The access token is used for API requests.

During this process, the authorization is processed using multiple predefined URLs, called *endpoints*. There are 3 endpoints:

- Request URI (this endpoint passes the request token)
- Access URI (exchanges request token for an access token)
- Authorize URI (confirms that the access token is valid)

10 SmartEnCity Demonstrator

10.1 Introduction

In deliverable D6.2 the demonstrator of the LH of Tartu using Cumulocity as reference architecture and following the IoT paradigm was presented. In this section the different components of the developed SmartEnCity demonstrator for the LH of Vitoria-Gasteiz will be described and explained. The three different Verticals/Intelligent Services have been implemented in the demonstrator and they are:

- Mobility
- Energy Efficiency
- Citizen engagement

Each Intelligence Service uses different devices and sensors for capturing the data but all of them are connected though the Interoperability Layer to the Knowledge Layer where all the information and data of the city is stored, aggregated, cleaned, managed and processed.

The T6.4 demonstrator is the integration of the three following demonstrators.

10.2 Demonstrator of Mobility

The following draw shows the components of the SmartEnCity architecture implemented in the demonstrator.

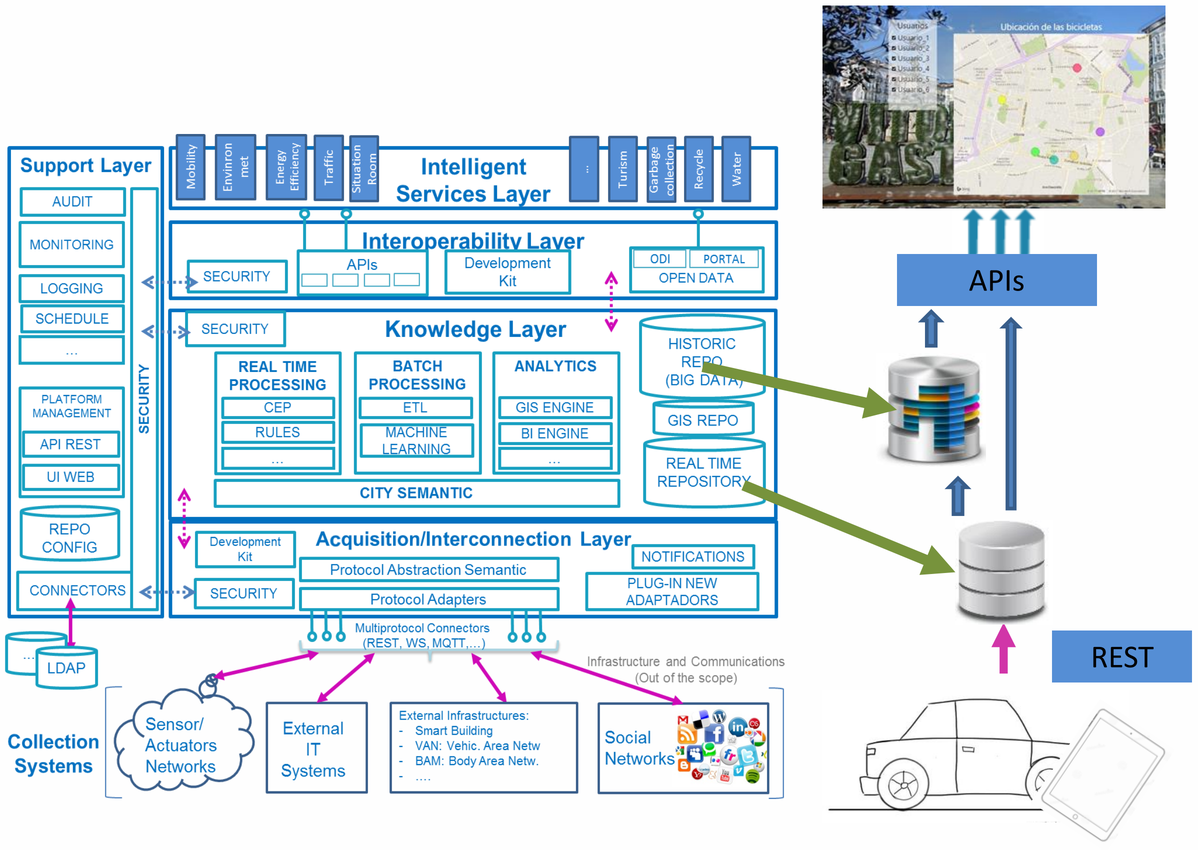


Figure 16 Implementation of Mobility at the demonstrator

The components in each layer are:

- Acquisition/Interconnection Layer

In order to capture information about mobility, some sensors are measuring speed, battery, geological position and so on. All these data are sent by MQTT protocol into an IoT endpoint where the data is checked if the captured information is reliable or not. Access management and status updates are handled in this layer in order to manage permissions for the devices and to check the status of every granted gateway.

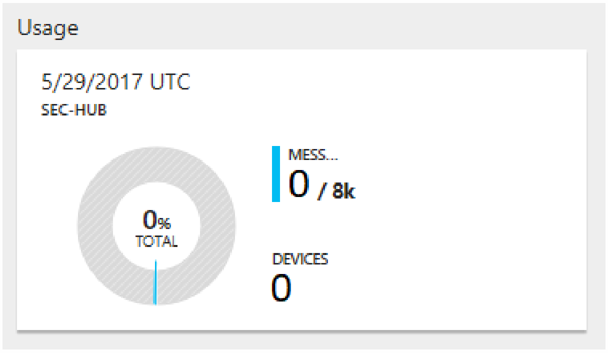
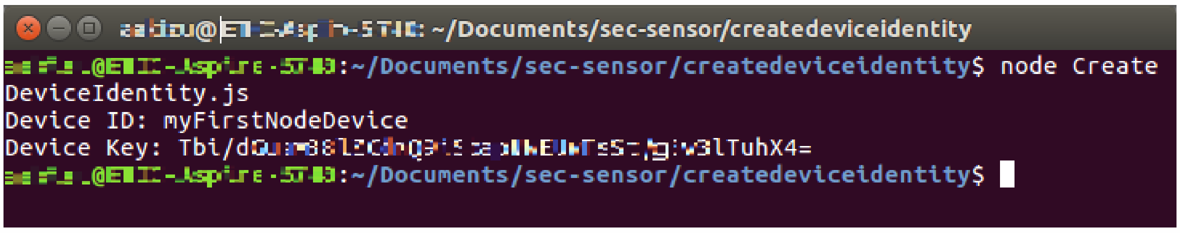


Figure 17 IoT endpoint initial status



```
[redacted]: ~/Documents/sec-sensor/createdeviceidentity
[redacted]:~/Documents/sec-sensor/createdeviceidentity$ node Create
DeviceIdentity.js
Device ID: myFirstNodeDevice
Device Key: Tbi/d[redacted]v3lTuhX4=
[redacted]:~/Documents/sec-sensor/createdeviceidentity$
```

Registering new sensor through a node.js application

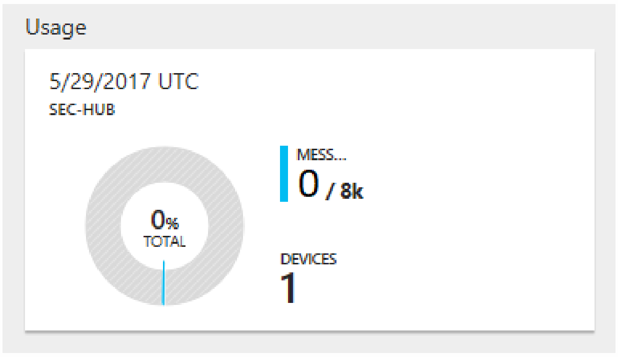


Figure 18 IoT endpoint after registration

- Knowledge Layer

An ETL (Extraction Transformation Load) process gets the received data from the previous layer and stores it on the real-time repository. This data is stored on a *Time Series* table where the schema of each object has message identification, sensor identification, timestamp and the content of the message itself.

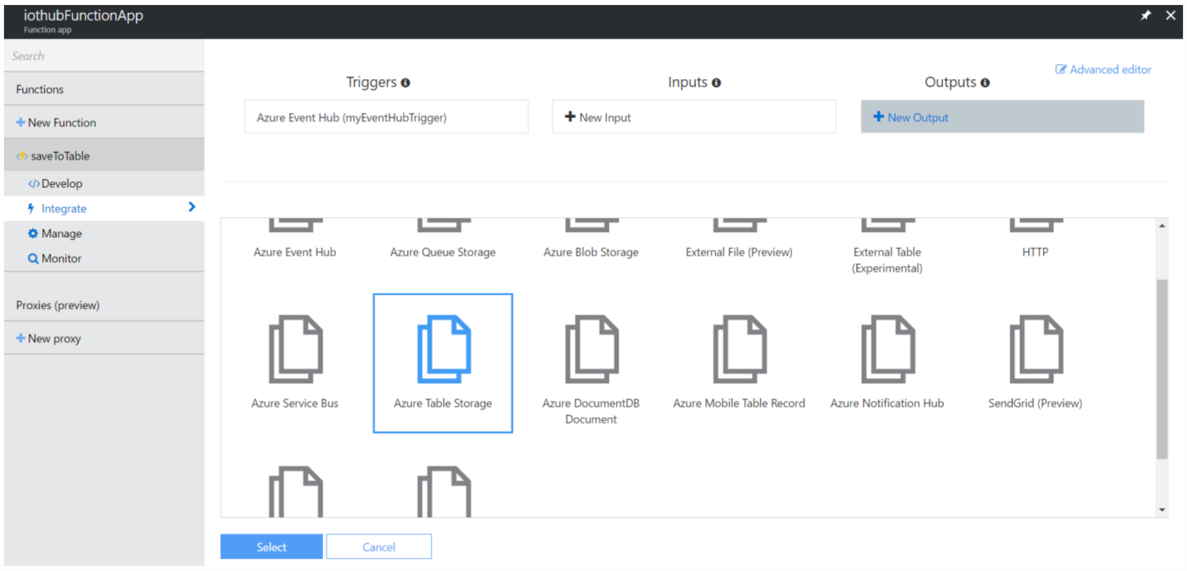
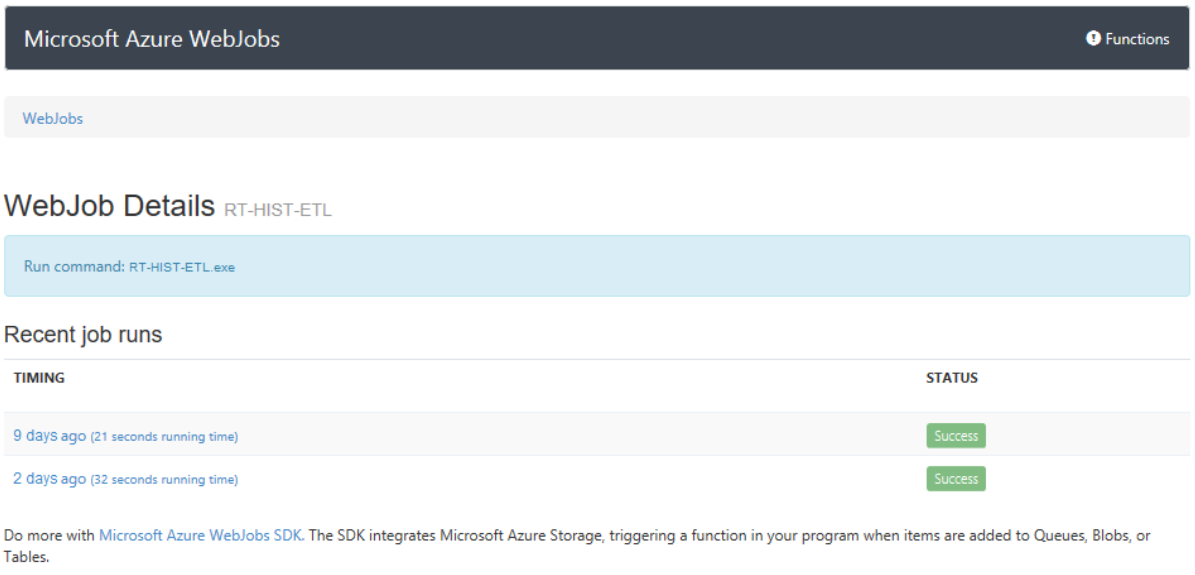


Figure 19 Setting Time Series table output

There are more ETL processes on this layer, such as real-time repository to historic repository saving process. Once per day, all the information stored on the real-time repository is transferred into a historical storage system. When the task finishes successfully, removes 'old' data from the real-time repository for lightweighting purposes.



WebJob Details RT-HIST-ETL

Run command: RT-HIST-ETL.exe

Recent job runs

TIMING	STATUS
9 days ago (21 seconds running time)	Success
2 days ago (32 seconds running time)	Success

Do more with Microsoft Azure WebJobs SDK. The SDK integrates Microsoft Azure Storage, triggering a function in your program when items are added to Queues, Blobs, or Tables.

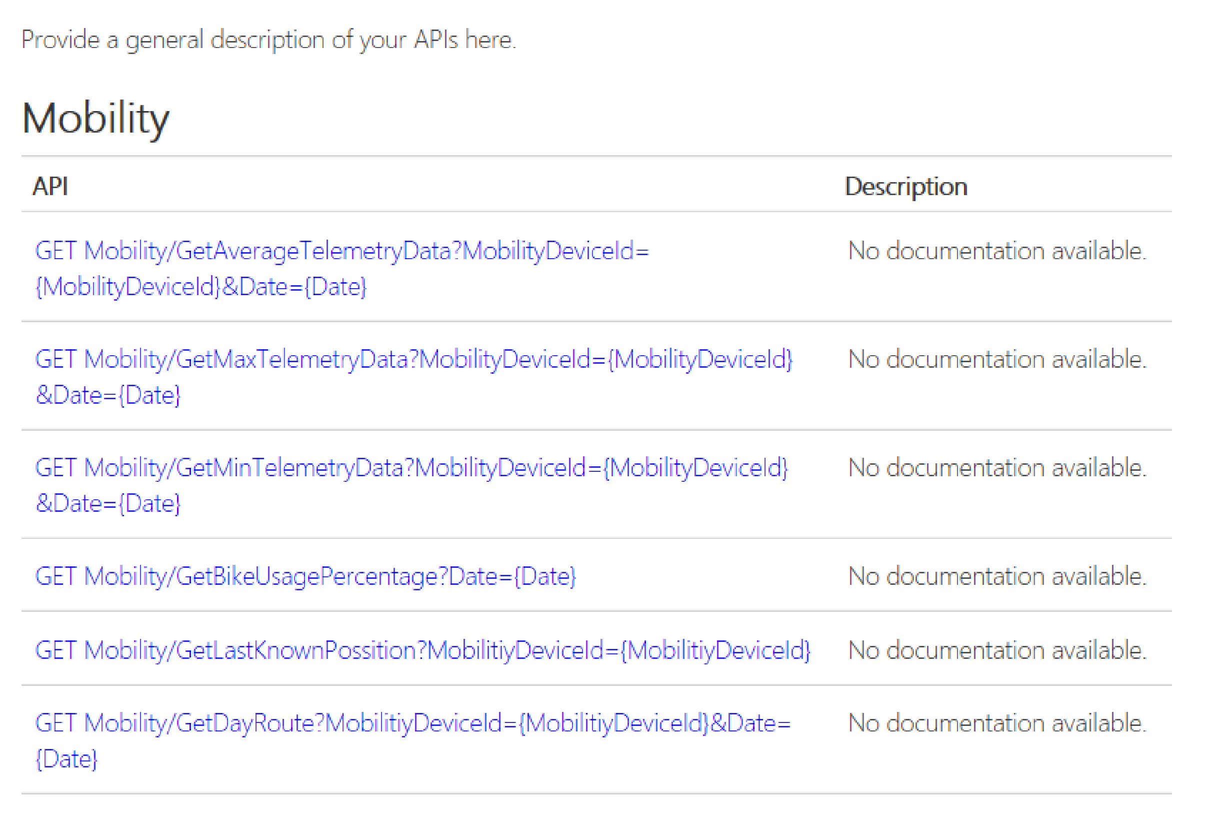
Figure 20 ETL process execution log

- Interoperability Layer

This layer contains all the API used by Vertical applications related with mobility. In the case of mobility services, the following APIs will be available:

  - GetAverageTelemetryData
    - Return the average Telemetry data from the given device and date

- GetMaxTelemetryData
  - Return the maximum Telemetry values from the given device and date
- GetMinTelemetryData
  - Return the minimum Telemetry values from the given device and date
- GetBikeUsagePercentage
  - Returns the percentage usage of electrics bikes from the given date
- GetLastKnownPossition
  - Returns the last position of the mobility device
- GetDayRoute
  - Return generated route from the given device at a specific date


Provide a general description of your APIs here.

Mobility

API	Description
GET Mobility/GetAverageTelemetryData?MobilityDeviceId={MobilityDeviceId}&Date={Date}	No documentation available.
GET Mobility/GetMaxTelemetryData?MobilityDeviceId={MobilityDeviceId}&Date={Date}	No documentation available.
GET Mobility/GetMinTelemetryData?MobilityDeviceId={MobilityDeviceId}&Date={Date}	No documentation available.
GET Mobility/GetBikeUsagePercentage?Date={Date}	No documentation available.
GET Mobility/GetLastKnownPossition?MobilitiyDeviceId={MobilitiyDeviceId}	No documentation available.
GET Mobility/GetDayRoute?MobilitiyDeviceId={MobilitiyDeviceId}&Date={Date}	No documentation available.

Figure 21 API definitions

- Intelligence Service Layer

For example, a Vertical application shows the current status of every mobility related item. The app shows on a quick view a report about mobility devices under the SmartEnCity platform. The principal aim of this application is to handle the assets of the Mobility service with the registration of every alarm/issue. The same Vertical application can show in a map the location, position and status of each mobility device by tracking their movements on the map.

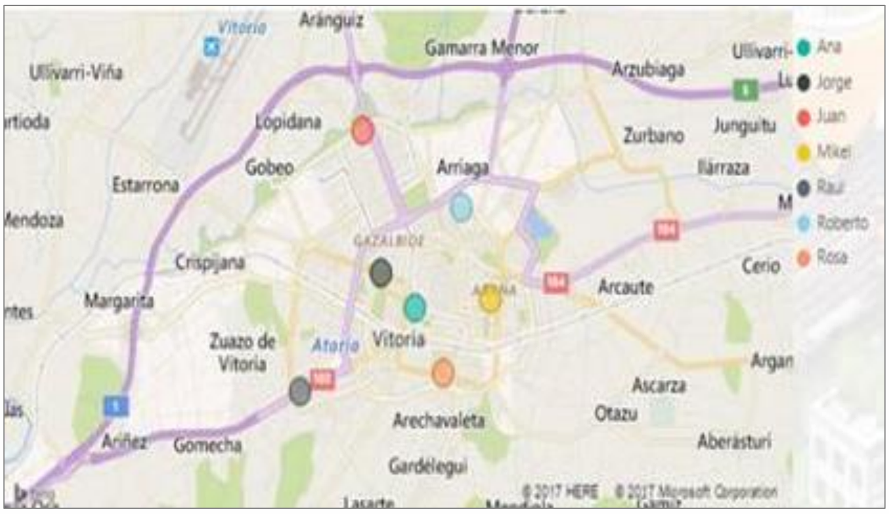


Figure 22 Cars position on the map

10.3 Demonstrator of Energy Efficiency

The following draw shows the components of the SmartEnCity architecture implemented in the demonstrator for Energy Efficiency.

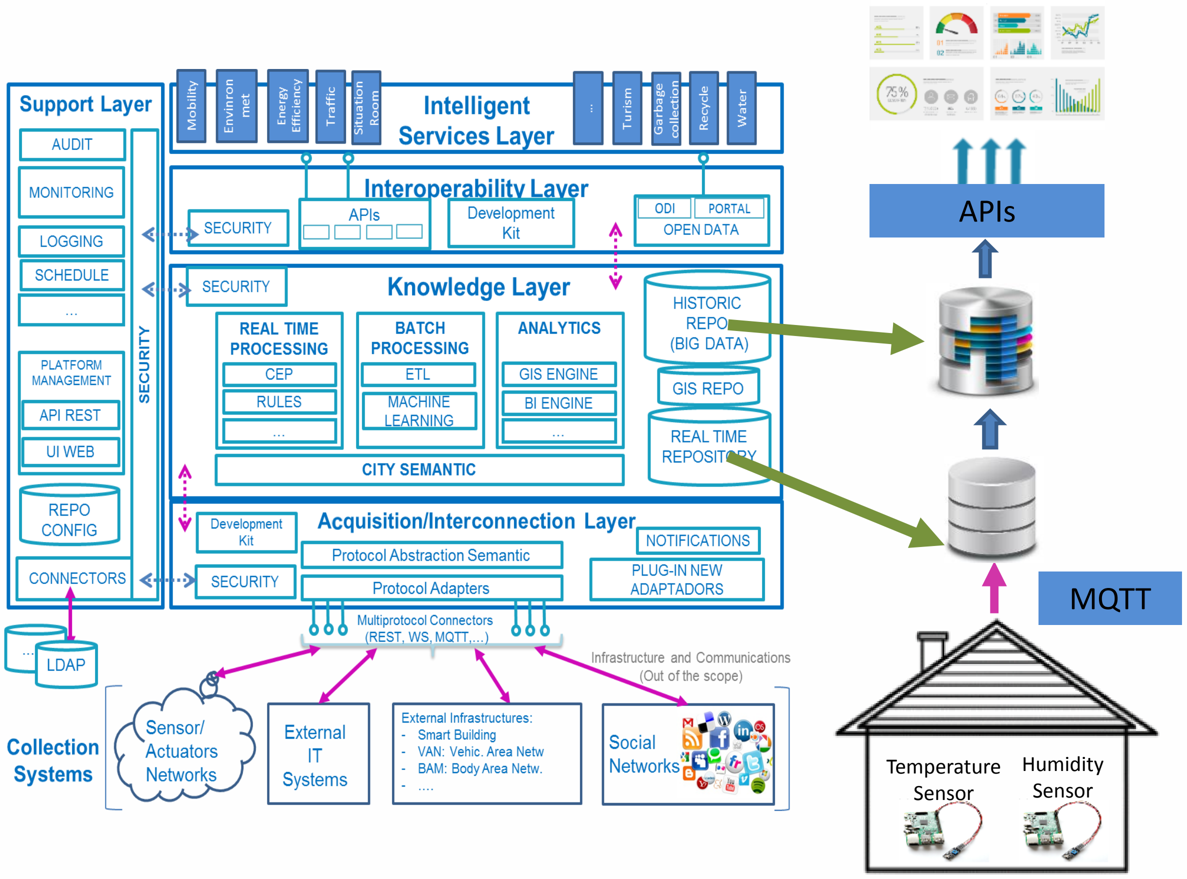


Figure 22 Implementation of Energy Efficiency demonstrator

The components in each layer are:

- Acquisition/Interconnection Layer

Information coming from different sensors as temperature sensors, humidity sensors and energy consumption meters is collected on time series.

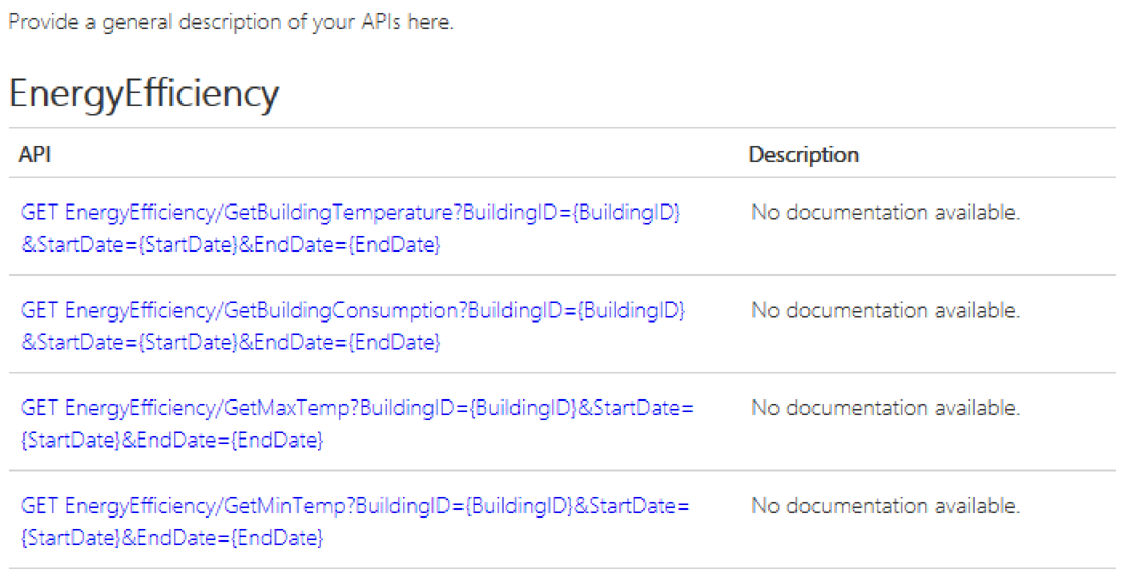
- Knowledge Layer

As mentioned before, ETL processes are generally part of the solution. At Energy Efficiency side, an ETL process transfers data from real-time to historical repository. There is another batch process, that is triggered every week in order to feed GIS repository with average values of temperature and energy consumptions.

- Interoperability Layer

As mentioned on the previous demonstrator, each bundle of API is secured by API-Keys to restrict the use of them. Diverse APIs are offered by the Energy Efficiency service:

  - GetBuildingTemperature
    - Returns an array of average temperature from the given building and a date range
  - GetBuildingConsumption
    - Returns an array of average consumption from the given building and a date range
  - GetMaxTemp
    - Returns an array of maximum temperature values from the given building and a date range
  - GetMinTemp
    - Returns an array of minimum temperature values from the given building and a date range



Provide a general description of your APIs here.

EnergyEfficiency

API	Description
GET EnergyEfficiency/GetBuildingTemperature?BuildingID={BuildingID}&StartDate={StartDate}&EndDate={EndDate}	No documentation available.
GET EnergyEfficiency/GetBuildingConsumption?BuildingID={BuildingID}&StartDate={StartDate}&EndDate={EndDate}	No documentation available.
GET EnergyEfficiency/GetMaxTemp?BuildingID={BuildingID}&StartDate={StartDate}&EndDate={EndDate}	No documentation available.
GET EnergyEfficiency/GetMinTemp?BuildingID={BuildingID}&StartDate={StartDate}&EndDate={EndDate}	No documentation available.

Figure 23 Available API

- Intelligence Service Layer

The Energy Efficiency Service can offer a dashboard of summary of the collected data from the households in an aggregated way. This application shows the average, maximum and minimum energy values per building displayed in charts over the time.

10.4 Demonstrator of Citizen's Engagement

The following draw shows the components of the SmartEnCity architecture implemented in the demonstrator of Citizen's Engagement.

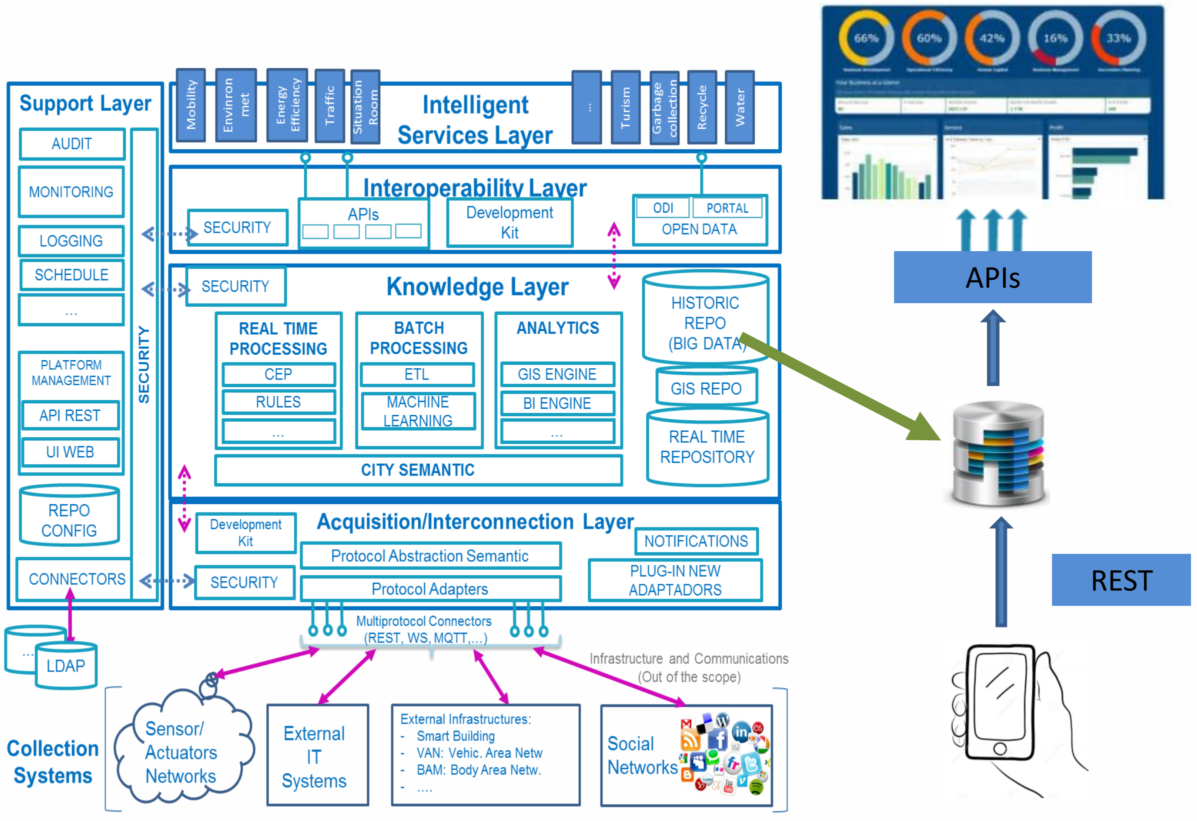


Figure 24 Implementation of Citizen's Engagement demonstrator

The components in each layer are:

- Acquisition/Interconnection Layer

In this case there is a phone application which collects data through a survey from citizens and is sent by REST protocol into this layer. Provided information through user's inquiries is represented in dashboard in order to analyse citizen's perception. Every time all data is checked ensuring that the origin is the official application to guarantee its veracity.

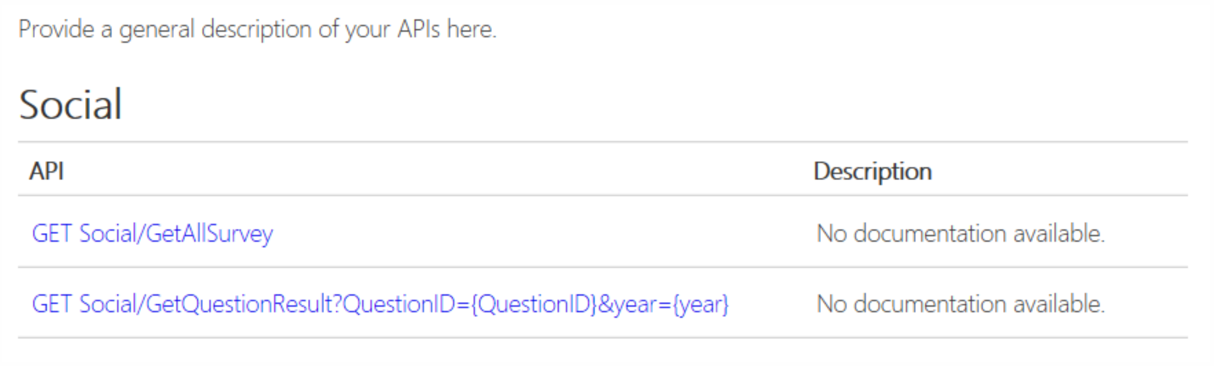
- Knowledge Layer

The results of the surveys are directly stored into the historic repository where they will be analysed and processed to obtain the KPIs of the social acceptance.

- Interoperability Layer

In order to feed different Intelligence Service Layers, there are some APIs to fulfill demanded information:

  - GetAllSurvey
    - Returns an array of all the questions stored on the platform
  - GetQuestionsResult
    - Returns an average of the answers from a specific question and year



Provide a general description of your APIs here.

Social

API	Description
GET Social/GetAllSurvey	No documentation available.
GET Social/GetQuestionResult?QuestionID={QuestionID}&year={year}	No documentation available.

Figure 25 Citizen's Engagement interoperability API

- Intelligence Service Layer

A dashboard summarizes anonymously obtained information on the mobile phone surveys on different type of graphical charts such as column charts, tachometers, pie charts, etc.

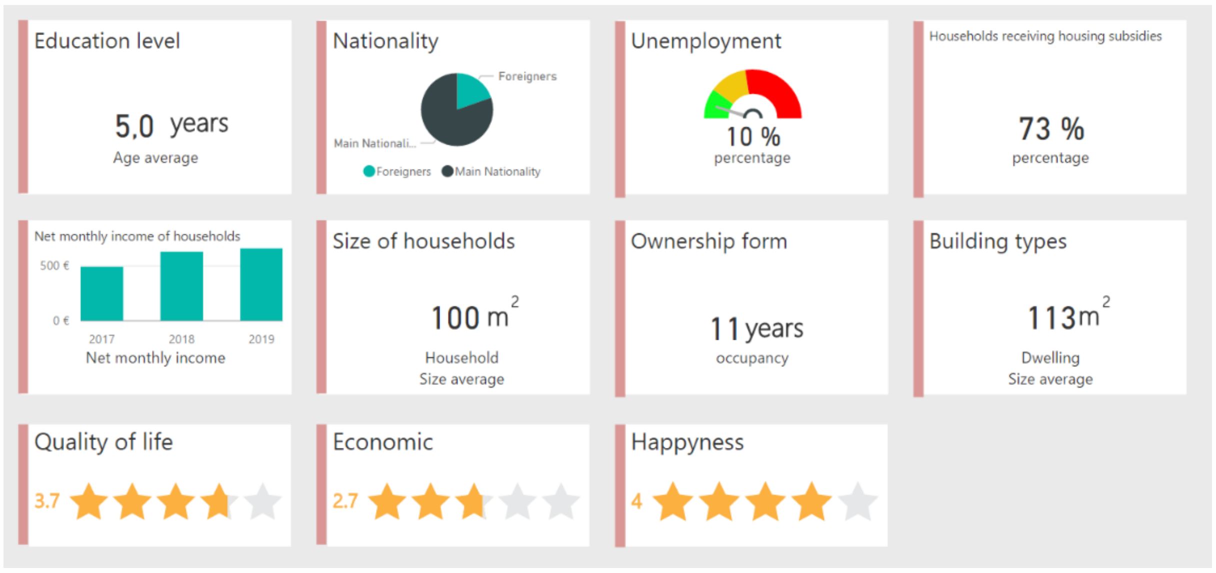


Figure 26 Survey summary mockup

10.5 Demonstrator complete

The following draw shows the components of the SmartEnCity architecture implemented in the whole demonstrator.

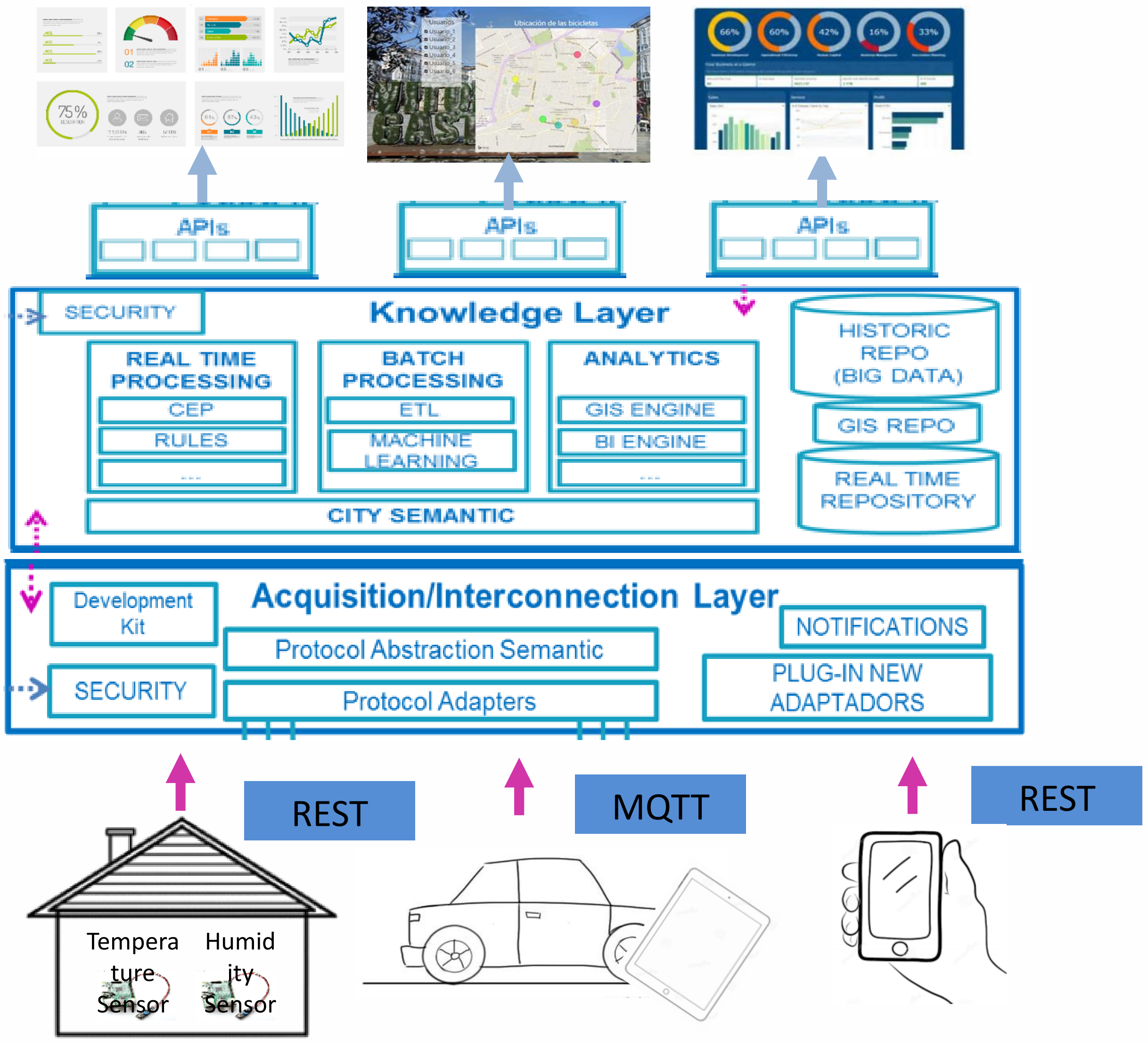


Figure 27 Implementation complete of the demonstrator

Beside the Mobility demonstrator, the Energy Efficiency demonstrator and the Citizen's Engagement demonstrator new applications will be created by combining and crossing data from previous demonstrators within the WP3, WP4 and WP5.

The aim of the SmartEnCity architecture and platform is to avoid the creation of vertical and isolated applications, separated in silos. The aim of the SmartEnCity architecture and platform is to facilitate the interconnection of components and elements of energy with the other of mobility or any other domain as security, traffic, etc.

The integration of aggregated information and the alignment of these data by means of time of the day, specific location…or other common conditions may allow the possibility of finding new knowledge and insights from the data.

10.6 User Guide

A visor guide for the demonstrator is available online. The guide presents the necessary support to show data from the example implemented.

During the development phase the visor for the lighthouse of Vitoria-Gasteiz demonstrator will be accessible through this URL: http://geoservergis.azurewebsites.net/gisviewer/viewer.do

For more detailed of the solution and to access specific repositories please contact project partners involved in this task.

Visor guide

The map viewer provides access, through OGC standards, to the set of geographic information and has been designed for viewing and querying that information and creating associated reports in an easy and simple way.

The main functionalities offered by the viewer include navigation (zoom in, zoom out and scroll through the map), enable and disable layers, get information about an item by clicking on it and creating KPI reports.

The browser in which geographic information is displayed occupies most of the screen and allows navigation in 2D. For optimal visualization it is recommended to use updated version of Firefox, Chrome or Internet Explorer.

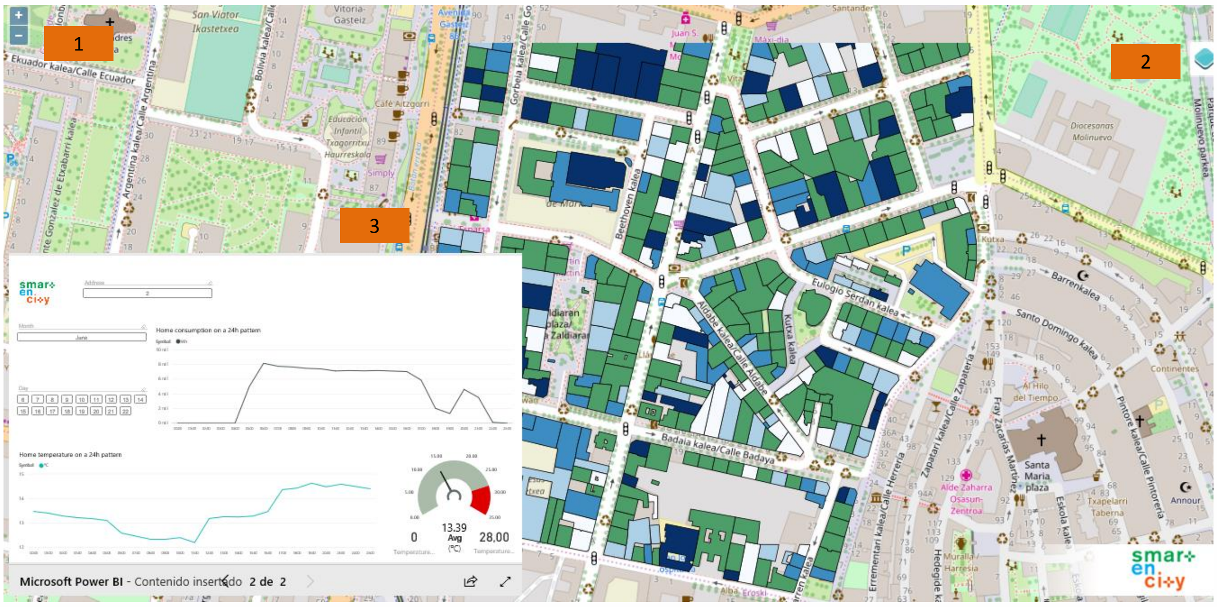

1
2
3
Home consumption on a 24h pattern
Home temperature on a 24h pattern
13.39
28.00
Microsoft Power BI - Contenido insertado 2 de 2

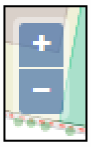
Figure 31 Viewer elements: zoom tools (1), map layers (2) and reports window (3)

The following details the viewer functionalities:

1) Navigation

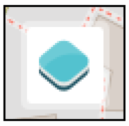
Navigation around the map can be done using one on the following options:

- Mouse navigation
  It is the most intuitive form of navigation. It includes:
  - Move the map: Click with the left mouse button and drag the map in the desired direction: up, down, right and left.
  - Zoom map at cursor location: Double click on the point of interest with the left mouse button
  - Zoom map at cursor location: Roll the mouse wheel forward to scale the map to the cursor location or roll the mouse wheel back to reduce map scale to the cursor location.
- Zoom tools
  Zoom tools appear in the top right if the viewer (No. 1). It includes:
  - Zoom in: Click the Plus (+) button to zoom in on the map.
  - Zoom out: Click the Minus (-) button to zoom out on the map.

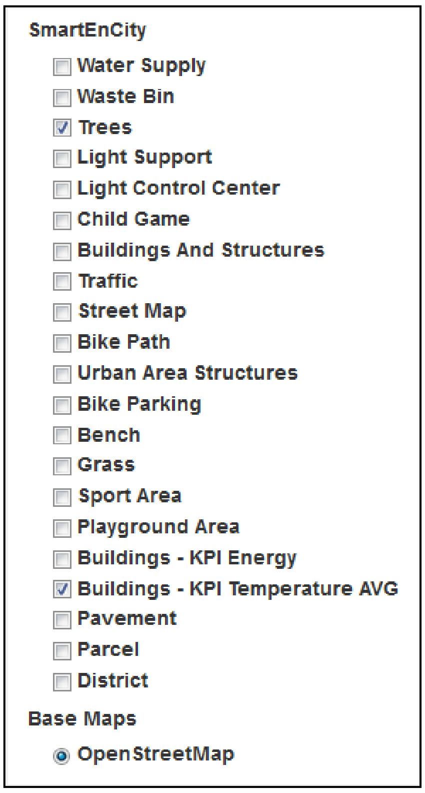


2) Map layers

Map layers menu allows seeing what layers are available in order to select them and add them to the map. This menu is accessed by placing the cursor over the icon on the top left of the viewer (No.2).



The list of layers available for visualization is displayed immediately. Beside each layer, there is a check box that is used to turn a layer on ☑ or off ☐.



SmartEnCity

- [ ] Water Supply
- [ ] Waste Bin
- [✓] Trees
- [ ] Light Support
- [ ] Light Control Center
- [ ] Child Game
- [ ] Buildings And Structures
- [ ] Traffic
- [ ] Street Map
- [ ] Bike Path
- [ ] Urban Area Structures
- [ ] Bike Parking
- [ ] Bench
- [ ] Grass
- [ ] Sport Area
- [ ] Playground Area
- [ ] Buildings - KPI Energy
- [✓] Buildings - KPI Temperature AVG
- [ ] Pavement
- [ ] Parcel
- [ ] District

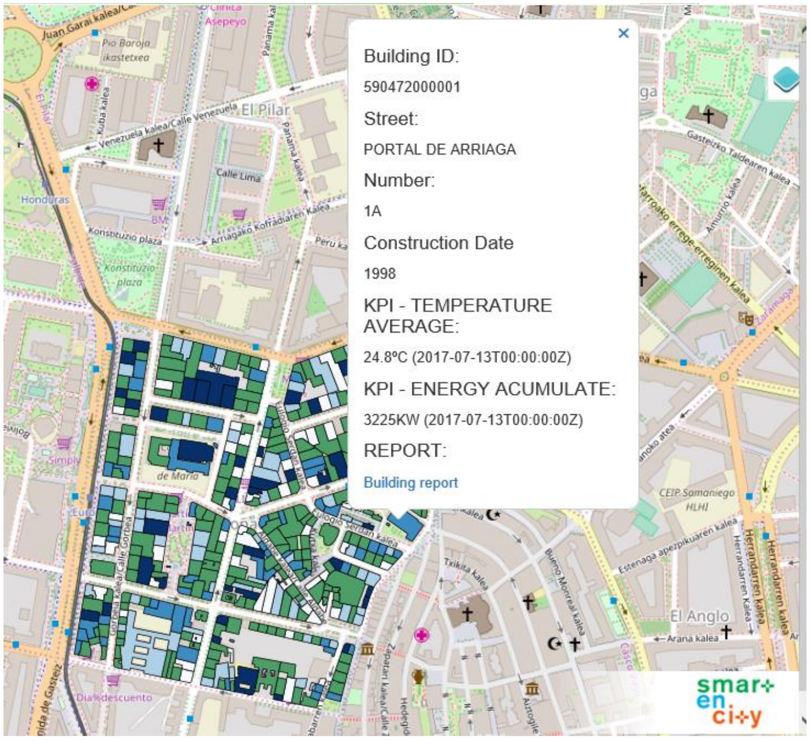
Base Maps

● OpenStreetMap

In addition to the thematic layers, the viewer provides a base map as a background map by default (OpenStreetMap). This map serves as support for locating information related to the territory and cannot be deactivated.

3) Map tip

The viewer allows identifying and visualizing the alphanumeric information of the layers loaded on the map. In order to identify an object displayed, place the cursor on the map and click the left mouse button at the point where we want to get information. A window with the information of the geographic object will be displayed. In order to close the window, click on the icon ✕ on the top left.

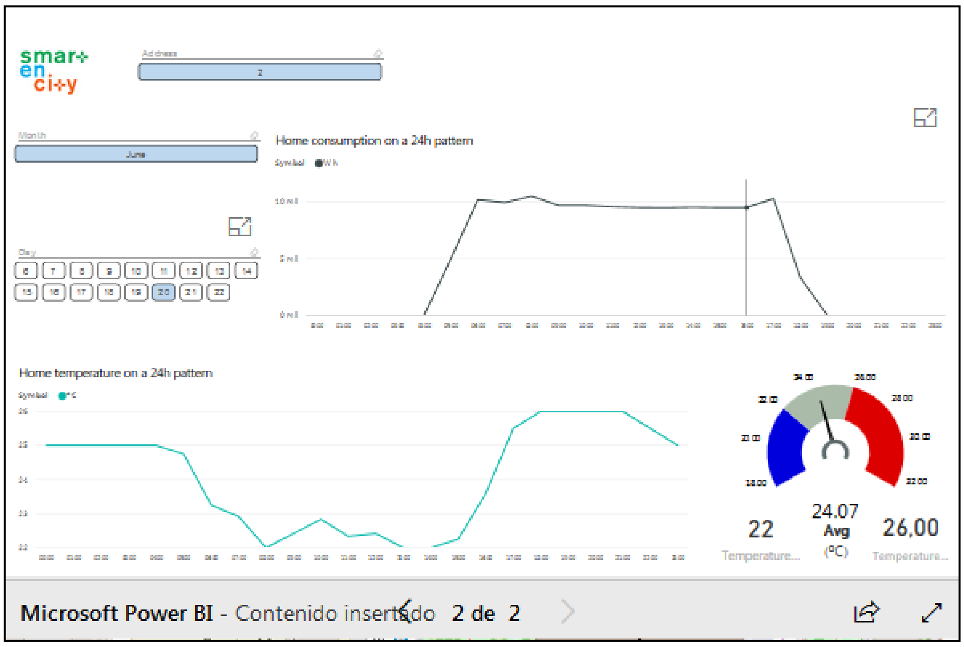


Building ID:
590472000001
Street:
PORTAL DE ARRIAGA
Number:
1A
Construction Date
1998
KPI - TEMPERATURE AVERAGE:
24.8ºC (2017-07-13T00:00:00Z)
KPI - ENERGY ACUMULATE:
3225KW (2017-07-13T00:00:00Z)
REPORT:
Building report

Figure 32 Map tip

4) Reports windows

The reports window appears on the bottom right-hand corner of the viewer. This window displays graphic and alphanumeric data of the KPIs in an easy-to-use interface. The information is displayed according to the data selected in the different fields (address, date, etc.). In order to see information about an specific time once the graphs are displayed with the criteria, run the mouse over the graph. A pop-up window will be displayed immediately with the related alphanumeric information.



Besides, there are some icons displayed in the window with additional functions:

- Access the full-screen mode.
- Display in a larger size each part of the report.
- Share the URL.

10.7 RA Demonstrator Functionality map

This section indicates which technologies, tools and mechanisms are used to build the demonstrator. The matrix in the following Table includes summary of the modules/layers from the RA and indicating the technologies/tools/mechanisms used.

Layer	Module	Platform Technology, tool or mechanism
Acquisition/Interconnection layer	Protocol Adapters	Bluetooh, Wifi, SigFox, MQTT, REST API, Modbus, etc
	Development kit	C/C++ SDK, C#, Java SDK,
	Protocol Abstraction Semantic	Users can define custom data models to create abstract devices.
	Notifications	Support of notifications out-of-range and no-data
	Security	HTTPS, REST API keys, two-factor authentication, access control based on users/passwords, user groups and authorities, device/gateway specific
	Plug-in New Adapters	New protocols can be supported implemented as a middle layer on top of the REST APIs.
Knowledge layer	Historic Repo	Historical data can be stored in different ways (databases, file systems, distributed file systems for big data…)
	City Semantic	Semantic meaning can be added to data by means of ontology deployment and using APIs
	Real Time Repository	Real time data can be stored in different ways (databases, file systems, distributed file systems for big data…)
	GIS Repository	SQL Server with GIS data
	Real Time Processing	Some ETL process can manage the real time data and move them to other databases.
	Batch Processing	Some ETL process can be executed without direct user interaction.
	Analytics	Data Analytics can be done over all the data stored in Historic Repo.
Interoperability layer	Open Data	Open Data can be integrated by using REST APIs and also Open Data can be provided by

		the platform.
	Development Kit	Not SDK will be provided at the moment
	APIs	All data processed in the Knowledge Layer can be offered by REST APIs to Intelligent Services Layer. Security issues will be managed.
	Security	HTTPS, two-factor authentication, no global password/key, sessionless REST API, access control based on realms, users, user groups and authorities
Intelligent Services layer	Verticals applications	Added value services can be constructed by using the APIs of the Interoperability Layer. Among the verticals applications are the Energy Efficiency Mobility and Citizen's Engagement.
Support layer	Audit	Not Audit will be provided at the moment
	Monitoring	Some monitoring tools will be provided in order to evaluate the platform itself.
	Logging	Logging web interface will be provided. In each layer there will be different logging access.
	Schedule	Not General Schedule will be provided at the moment
	Platform Management	The Platform will be managed by consortium partners during the duration of the project.
	Repo Config	Repo Config will be managed by Real Repo owners.
	Connectors	The needed Connectors for the three domains will be created.

Table 4 RA functionality – Platform functionality matching

11 Conclusions, deviations and outputs for other WPs

In this deliverable some of the interoperability mechanisms that follow the Reference Architecture are showed and explained. Of course there are other more ones that can fulfil with the requirements of the RA and they are not included here. It will depend on the specific needs of the city infrastructure, devices, sensors and components of the specific implementation of the RA in each lighthouse.

A demonstrator or prototype consists of a technological solution that fulfils the requirements of a RA and provides the modules and functionality specific for the domain it represents. Several demonstrators built with different technologies and frameworks can agree with a common Reference Architecture and consequently be valid instantiations or implementations of that architecture. The development and deployment of the reference architecture could vary among different demonstrators.

The demonstrator or prototype developed within this task is an implementation following the Reference Architecture described in this document and in previous documents of WP6. This demonstrator is attached to the implementation of the RA in the lighthouse of Vitoria-Gasteiz but this demonstrator also offers the functionality necessary to build the CIOP in other lighthouses. The demonstrator resultant of this task is aligned with the ones from tasks 6.2 and 6.3.

Later on, this demonstrator will be enhanced by integrating the results of tasks 6.5 and 6.6 including technologies for HMI and added value services.

No deviations have been produced according to the dates and content of the deliverable with respect to the proposed plan.

The outputs produced in this deliverable will have effects mainly on other activities of the WP6 and on activities related with the deployment of the CIOP platform in the three lighthouse cities (Vitoria-Gasteiz WP3, Tartu WP4 and Sonderborg WP5).

12 References

AENOR. ((2015)). *UNE 17804:2015. https://www.aenor.es/aenor/normas/ctn/fichactn.asp?codigonorm=AEN/CTN%20178#.WG58xBvhC71.* AENOR CTN-178.

Berners-Lee, T., Hendler, J., & al., O. L. (2001). The semantic web. *Scientific american, vol. 284, no. 5,*, pp. 28–37.

Bizer, C., Heath, T., & and Berners-Lee, T. (2009). Linked data-the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pp. 205–227.

D. Brickley, D., & Guha, R. V. (2004). *RDF vocabulary description language 1.0: RDF schema.*

Foundation, S. (December de 2016). *HDFS Architecture. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html*.

Ghemawat, S., Howard, G., & Shun-Tak, L. (2015). *Google Patents, US Patente nº US Patent 9,047,307.*

Klyne, G., & Carroll, J. J. (2006). *Resource description framework (rdf): Concepts and abstract syntax.*

Leavitt, N. (2010). Will NoSQL databases live up to their promise? *Computer, 43(2)*, 12-14.

McGuinness, D. L., & Van Harmelen, F. (2004). Owl web ontology language overview. *W3C recommendation, vol. 10, no. 10*, p. 2004.

Nokia, C. (December de 2016). *The Disco Project. Disco Distributed Filesystem. https://disco.readthedocs.org/en/latest/howto/ddfs.html*.

OneM2M. (December de 2016). *http://www.onem2m.org/images/files/deliverables/Release2/TR-0007-Study_on_Abstraction_and_Semantics_Enablement-V2_11_1.pdf*.

Prud'Hommeaux, E., & Seaborne, A. e. (2008). Sparql query language for rdf. *W3C recommendation, vol. 15,*.

SmartEnCityD6.1. (2016). *SmartEnCity Deliverable 6.1: CIOP Functional and Non-Functional Specifications.*

SmartEnCityD6.2. (2017). *SmartEnCity D6.2 "CIOP architecture generic implementation".*

SmartEnCityD7.2. (2017). *SmartEnCityD7.9 "KPIs definitions".*

W3C. (December de 2016). *https://www.w3.org/RDF/*.

W3C. (December de 2016). *https://www.w3.org/TR/owl2-overview/*.

W3C. (December de 2016). *https://www.w3.org/XML/Core/#Publications*.

W3C. (December de 2016). *https://www.w3.org/XML/Schema*.

Wikipedia. (December de 2016). *https://en.wikipedia.org/wiki/Reference_architecture*.

13 Annex

If applicable