ALMANAC

RELIABLE SMART SECURE
INTERNET OF THINGS FOR SMART CITIES

(FP7 609081)

D7.3.1 Cloud based APIs for Smart City applications - Developers Guide 1

Submission Date August 31 2015 – Version 1.0

Published by the ALMANAC Consortium

Dissemination Level: Public

Project co-funded by the European Commission within the 7th Framework Programme
Objective ICT-2013.1.4: A reliable, smart and secure Internet of Things for Smart Cities

Document control page

Document file: D7.3.1 Cloud based APIs Developers Guide 1.docx
Document version: 1.0
Document owner: Matts Ahlsén (CNET)

Work package: WP7 – Platform Integration
Task: T7.4
Deliverable type: R
Document status: ☒ approved by the document owner for internal review
☒ approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Matts Ahlsén (CNET)	2015-05-05	Initial structure
0.4	Matts Ahlsén, Mathias Axling (CNET)	2015-06-20	API structure and tools draft
0.5	Dario Bonino (ISMB), José Ángel Carvajal Soto, Jaroslav Pullman (FIT)	2015-08-03	API details for data fusion, event processing and semantic resources
0.6	Mathias Axling, Matts Ahlsen (CNET)	2015-08-10	OGC details
0.7	Mathias Axling, Matts Ahlsen (CNET)	2015-08-17	API details: provisioning
0.8	Mathias Axling (CNET)	2015-08-20	Developers tutorial
0.9	Mathias Axling (CNET), Matts Ahlsen (CNET)	2015-08-31	Version for internal review
1.0	Mathias Axling (CNET), Matts Ahlsen (CNET)	2015-09-02	Final version for submission

Internal review history:

Reviewed by	Date	Summary of comments
Thomas Gilbert (ALEX)	2015-08-31	Accepted with comments
Marco Jahn (FIT)	2015-08-31	Accepted with minor comments

Index:

List of Figures

Terminology

ALMANAC Platform	The ALMANAC Platform comprises a set of software components, guidelines, constraints, best practices, etc. that allow the development of Internet of Things applications for smart cities.
ALMANAC Platform Instance	A deployment of the ALMANAC platform. Depending on the choice of deployment this may comprise only a subset of the platform components. E.g. in some cases it may be sufficient to run an instance of SCRAL while in other cases only the Virtualization Layer and the Cloud-based APIs may be needed.
Capillary Network	Capillary Networks are flexible and autonomous communication networks normally used to locally collect information from sensors and actuators in the smart city. Examples of capillary networks include short-range networks based on Wireless M-Bus or DLMS-Cosem e.g. for utility metering (gas, water, electricity), collection of waste management data, pollution and traffic control sensors, smart lighting sensor, heating control sensors, etc.
Cloud-based API	Cloud-based APIs are a set of services that provide access to the ALMANAC platform for developers of smart city applications. These services can be accessed over the network through REST interfaces.
ETSI M2M	A standard defining M2M communication and platforms provided by ETSI. The basic concept of the standard is the Store and Share of data coming from smart devices. Data is stored and then shared with the Apps by the M2M Platform with standard APIs (httpREST based).
Federation	Federation describes the inter-operation between different ALMANAC platform instances (and external nodes) through a shared communication infrastructure. Each node (federate) in such a distributed system is an autonomous instance of the ALMANAC platform implementing a minimal set of components to enable communication. Further, each node manages access to its resources and services through access control policies.
IoT-ARM	The IoT Architectural Reference Model (IoT ARM) provides a collection of generic architectural concepts and constructs considered applicable to IoT system architectures. The IoT ARM does not say how to build IoT systems, it is a tool box of concepts, models and recommendations for the domain of IoT systems and their architectures.
IoTWorld Gateway	An IoTWorld Gateway is a software component that provides a logical interface towards an IoTWorld (domain). The IoTWorld gateway exposes a number of (IoT) Entities and provides a high-level API for communicating with this part of the physical world.
Machine-2-Machine (M2M)	M2M describes the ability of two devices to communicate with each other without human intervention through wired or wireless network and often through a M2M Platform. M2M communication is a prerequisite for the Internet of Things.
OGC	The Open Geographical Consortium, a (de facto) standards consortium for the promotion of standards for geospatial data interoperability.
LinkSmart Middleware	The LinkSmart Open Source Middleware was originally developed within the Hydra EU project for Networked Embedded Systems, allowing developers to incorporate heterogeneous physical devices into their applications through easy-to-use web services for controlling any device.

Executive summary

This report describes the subset of ALMANAC platform APIs to be deployed for Cloud access. The Cloud-based APIs are a set of external APIs to be used by Smart City applications and the developers of such applications. The Cloud-Based APIs rely on the services exposed by the ALMANAC platform components – which are more generic in nature - and expose a view of the ALMANAC system suitable for development of Smart City applications. This deliverable provides short descriptions of the tools and interfaces avaliable to developers aiming to build smart city applications using the ALMANAC platform. The document is primarily a guide to developers; extensive descriptions of the interfaces and tools are provided are provided in other deliverables in the ALMANAC project.

1. Introduction

1.1 Purpose, context and scope of this deliverable

This deliverable describes the current state of the Cloud based APIs and provides short descriptions of the tools and interfaces avaliable to developers aiming to build smart city applications using the ALMANAC platform. The document is primarily a guide to developers; extensive descriptions of the interfaces and tools are provided elsewhere. The chapters on developer tools and developer APIs provide a background to the tutorial. Not all parts of the Cloud APIs are ready at this stage, an updated version will be provided in D7.3.2 ”Cloud based API Developers Guide 2”.

1.2 Background

The Cloud-based APIs are a set of external APIs to be used by Smart City applications and the developers of such applications. The Cloud-Based APIs rely on the services exposed by the ALMANAC platform components – which are more generic in nature - and expose a view of the ALMANAC system suitable for development of Smart City applications. It is accessible from any system in the LinkSmart middleware domain or from mobile or web applications, abstracting the distributed nature of the ALMANAC Platform and the interfaces of the specific ALMANAC components.

The functionality of the Cloud-Based APIs will have an emphasis on finding resources, requesting and subscribing to data, and aggregation of data. Most of the data related services exposed by the Cloud Based APIs are handled by by the Virtualization Layer, which is exposed directly to the external applications. It connects the components of an ALMANAC Platform as well as federated ALMANAC platforms, making ALMANAC components and federations appear as a unified service to external applications providing support in,

- Combining services of several ALMANAC Platform components to provide a single service, e.g. for querying.
- Finding registered services and resources.
- Requesting and subscribing to ALMANAC data.
- Transformation of payload formats and adaptation to different communication protocols

The Cloud-based APIs will also provide management services, such as:

- Provisioning resources and devices to be used in the Smart City applications.
- Managing metadata for domain entities and data derived from complex event processing.
- Handling access control, such as granting and revoking application or user access to a specific resource and granting access to delegate access to a specific resource.

The subset of Cloud Services implemented will be based on requirements derived from activities in WP8 Applications Definition, Development and Evaluation.

1.3 Method and tools

Existing frameworks and standards have been used where possible. The Cloud based APIs are designed to expose a comprehensive view of the platform for developers while minimizing the adaptation between the Cloud based APIs and the ALMANAC Platform components that realize the API functionality. The ALMANAC project has chosen to use the Swagger API design and documentation environment[1], which will allow developers to inspect the interfaces and create client code in different languages.

[1] http://swagger.io/specification/

2. ALMANAC Component view

This chapter gives an overview of the different components of the ALMANAC platform, including their functionality, interfaces, and interactions. This chapter is not necessary knowledge for an application developer, but may provide a better understanding of the platform. The overall component diagram of the ALMANAC SCP is outlined in Figure 1.

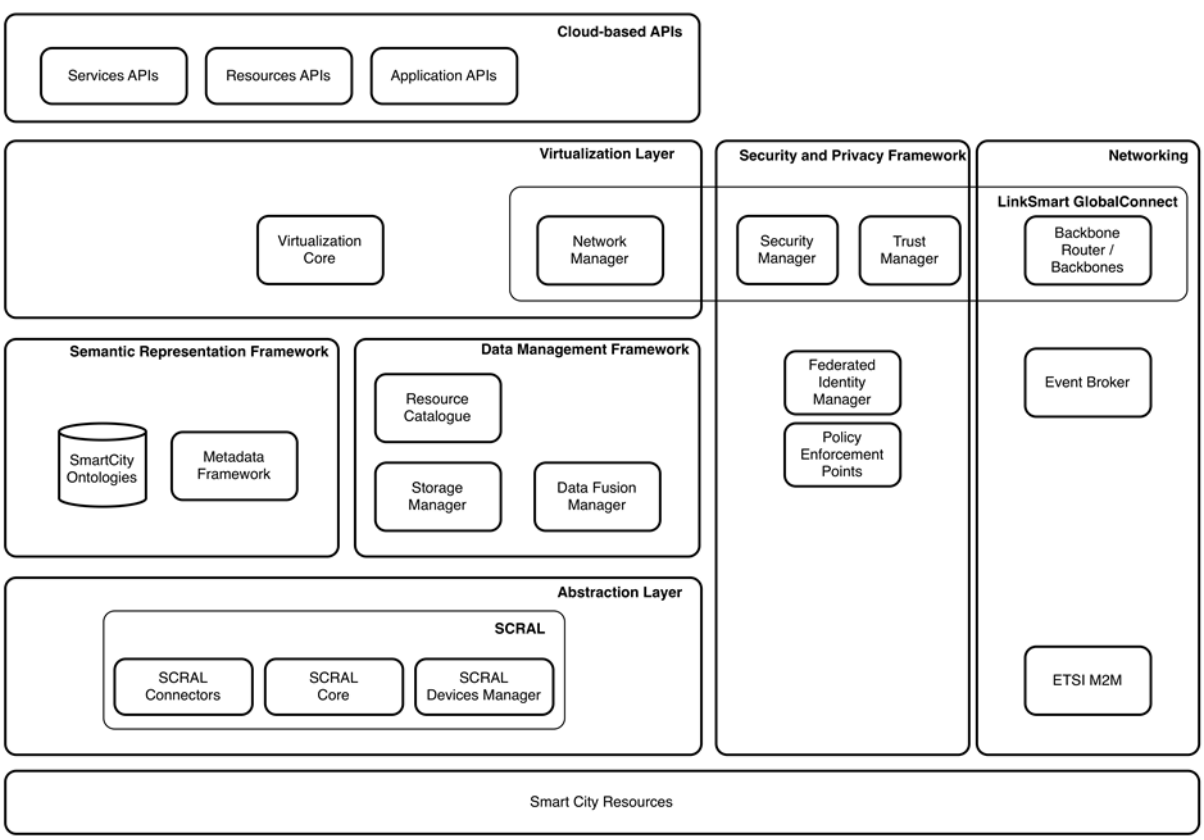

Figure 1: Component diagram of the ALMANAC platform

The ALMANAC platform is intended to be a middleware hosting multiple forms of applications; this function is also reflected in the list of its components. The main component subsets are:

- Cloud-based APIs, define a set of external APIs to be used by Smart City applications and by the developers of such applications.
- The Virtualization Layer: a set of components subset that abstracts from specific Smart City resources to virtual entities and APIs, easily accessible by Smart City applications,
- Semantic Representation Framework: exploits Smart City vocabularies, ontologies and metadata, supporting the virtualization and semantic processing of resources.
- The Data Management Framework: enables storage, caching and querying of collected Smart City resource data, as well as data fusion and event management.
- The Smart City Resource Adaptation Layer: the components that provide uniform access to heterogeneous devices, over multiple protocols while also enabling standards-based interoperability with M2M networks.
- Security and Policy Framework: the components that protect the privacy of stakeholders in a transparent way across the platform.

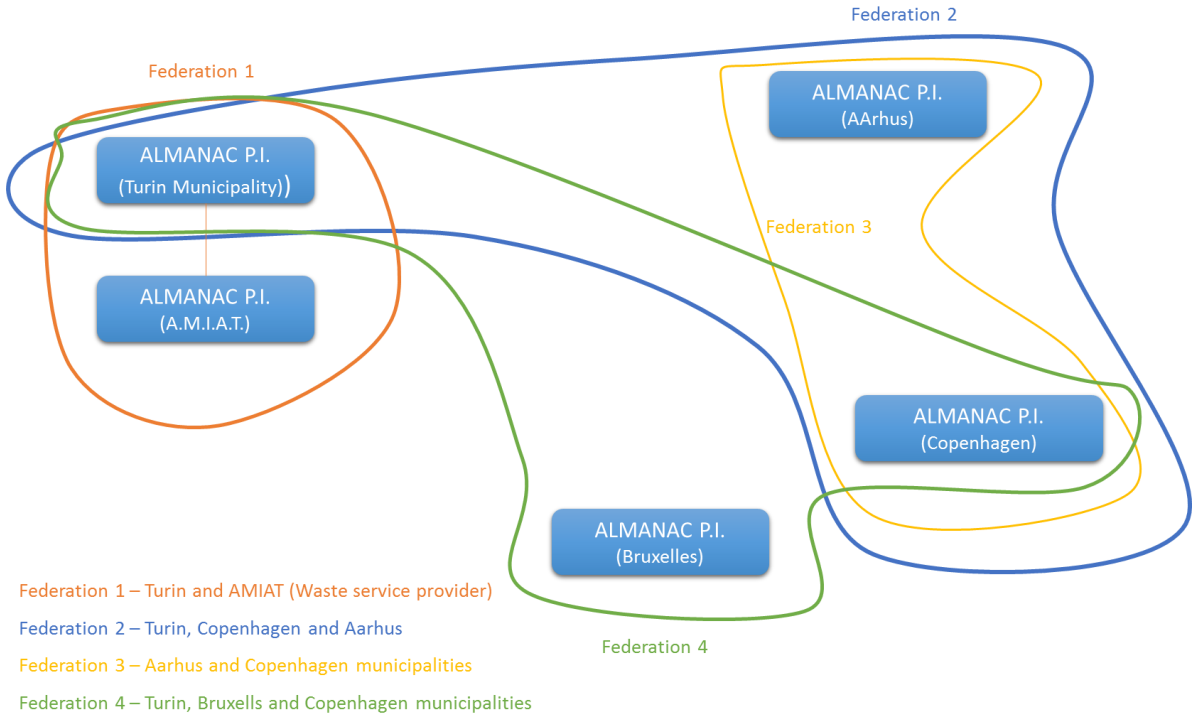

Figure 2: Multiple federations between ALMANAC Platform Instances.

An Almanac Platform Instance is a deployment of the ALMANAC platform. Depending on the choice of deployment this may comprise only a subset of the platform components. An ALMANAC Federation describes the inter-operation between different ALMANAC platform instances through a shared communication infrastructure. The Virtualization Layer delegates calls across platform instances in a federation, alleviating the need for users of the Cloud Based API to deal with the distributed nature of an ALMANAC Federation.

3. Developer tools

3.1 Almanac Data Fusion Language

The Data Fusion Language (DFL) is designed to replace the writing of data fusion queries in specific Complex Event Processing (CEP) languages. The DFL exploits the conceptual block-based representation of CEP queries defines in spChains (Bonino et al., 2012-1). Such a representation, and model, exploits the typical query pattern associated to complex stream operations, to decouple low-level CEP query writing from high-level definition of processing tasks. Query composition, in particular, is tackled with a block composition approach, in some aspects very similar to well-known block-based programming paradigms such as in Scratch (Maloney et al., 2010) or in well-established processing tools, for example, LabView,[2] Simulink.[3] In such a way, complex CEP query writing is mapped to simpler block interconnection while keeping the processing efficiency almost unchanged.

A stream processing block is "a (software) component taking one or more event streams in input and generating one or more event streams as output" (Bonino et al., 2013). The output and input streams are correlated by means of a processing function (i.e., a CEP query), which, in general, is not linear (e.g., threshold) and/or with memory (e.g., a moving average).

A stream processing block (see Figure 3) has a set of input ports, and a set of output ports, identified by unique port identifiers.

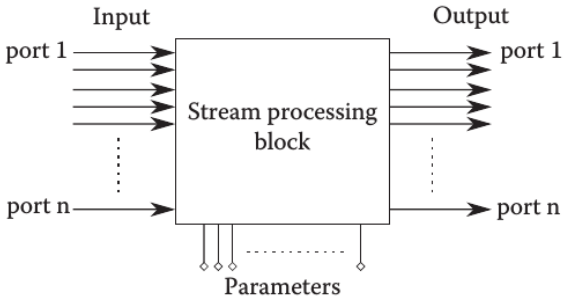

Figure 3. A generic stream processing block.

Every port can handle a specific type of event, that is, it has an associated data type that shall match the type of events received (generated) in input (output). In the Almanac DFL, these datatypes correspond to those allowed within the OGC SensorThings API candidate standard (Liang et al., 2015). A set of constant parameters can, furthermore, be defined to affect/tune the inner block functionality (i.e., the generated CEP query), for example, values, window lengths, operating modes (see the example given in Figure 4).

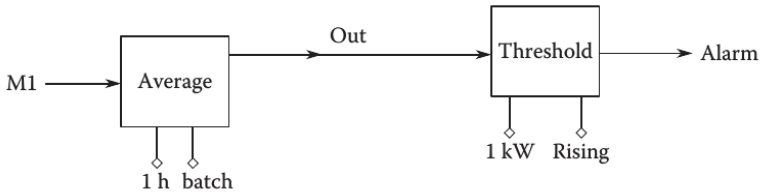

Figure 4. Sample chain (hourly average).

The ALMANAC Data Fusion Language is purposely designed to be easy to handle, and to manipulate, by system integrators and advanced users. As such, it exploits representation formats, which are

[2] http://www.ni.com/labview/i/
[3] http://www.mathworks.it/products/simulink/

gaining momentum in the ever-widening community of web-based mash-uppers, developers and integrators. In particular, chains in the DFL are defined exploiting a JavaScript Object Notation format, which can easily be exploited in graphical configuration UIs (planned in ALMANAC, too). Figure 5Figure 5 and Figure 6 Figure 6 report a very simple chain definition involving one block, one data source and one data drain.

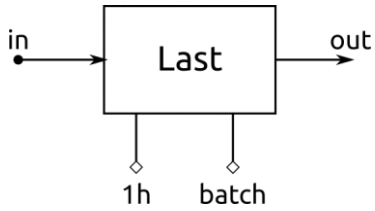

Figure 5. Single block chain.

```
{
  "chains" : [
    {
      "id": "fill_level_sub_sampling",
      "blocks" : [
          {
            "id" : "last_01af",
            "function" :"last",
            "params" : [
              {
                "name" : "window",
                "value" : "1",
                "uom" : "h"
              },
              {
                "name" : "mode",
                "value" : "batch"
              }
            ]
          }
      ],
      "input": {
              "blockId": "last_01af",
              "port": "in",
              "id": "in"
          },
      "output": {
          "blockId": "last_01af",
          "port": "out",
          "id": "out"
      }
    }
  ]
}
```

Figure 6. Single block chain notation.

The ALMANAC Data Fusion Language is described in detail in ID6.2.2 Data Fusion Language and Prototype 2.

3.2 Swagger

Swagger is a specification for documenting and defining REST APIs (URL, method, and representation) in a language-agnostic, implemetation-independent way. Similar to a WSDL description of a web service, the swagger specification of a REST API may be used to inspect the endpoints, methods and formats of the API and generate both server and client code.

The swagger specifiction is supported by an ecosystem of tools. Swagger definitions may be written or inspected using the Swagger UI editor (see Figure 7). The Swagger Codegen tool may used to generate code, or Swagger definitions may be generated from existing REST APIs.

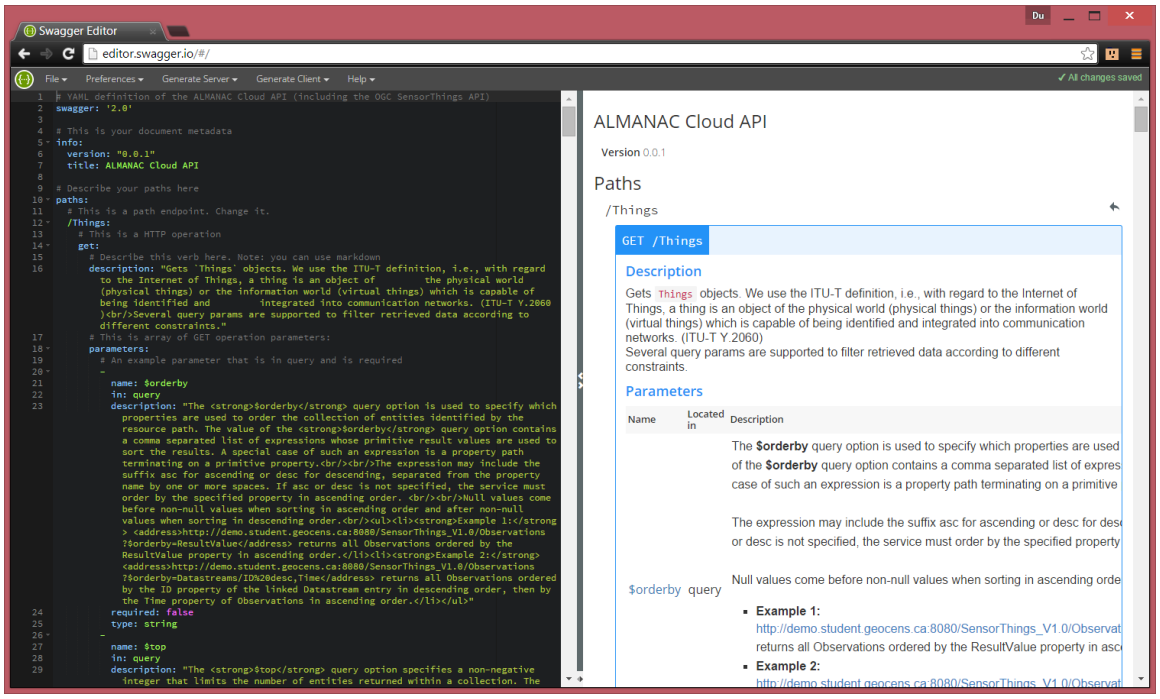

Figure 7: ALMANAC APIs in the Swagger UI editor

To start using the Cloud based APIs, the Swagger UI tool can be used to explore the APIs and use them in the design phase. Then use the Swagger Codegen tools to generate client libraries for the desired platform and language. The Swagger Codegen tools have support for a number of platforms, frameworks and languages including dotNET, Java, Python, Swift, Javascript and Nodejs. Swagger descriptions of the Cloud based APIs[4] will be made available for developers.

3.3 OGC SensorThings API

The OGC SensorThings API is an open and unified way to interconnect Internet of Things devices, data, and applications over the Web. In the ALAMANC project, this standard has been used by the platform instance components. Where the OGC SensorThings API Data Model is appropriate, the Cloud Based APIs also use the OGC SensorThings Data Model, JSON format, and the Sensing Profile API according to (Liang, et al. 2015).

3.3.1 Introduction

The OGC SensorThings API provides an open and unified way to interconnect Internet of Things devices, data, and applications over the Web. The OGC SensorThings API is an open standard: non-proprietary, platform independent, and perpetual royalty-free. Although it is a new standard, it builds on a rich set of proven-working and widely-adopted open standards, such as Web protocols and the OGC Sensor Web Enablement (SWE) standards, including the ISO/OGC Observation and Measurement data model (OGC and ISO 19156:2011). As such, the OGC SensorThings API is extensible and can be applied to both simple and complex use cases.

The OGC SensorThings API data model consists of two parts: the Sensing profile and the Tasking profile. The Sensing profile provides functions similar to the OGC Sensor Observation Service (SOS) and the Tasking profile will provide functions similar to the OGC Sensor Planning Service (SPS). The main difference between the SensorThings API and the OGC SOS and SPS is that the SensorThings

[4] The current version can be found at https://fit-bscw.fit.fraunhofer.de/bscw/bscw.cgi/d44048537/20150519-cloud-api.yaml.

API is designed specifically for resource-constrained IoT devices and the Web developer community. As a result, the SensorThings API follows REST principles as well as the use of an efficient JSON encoding, and the use of the flexible OASIS OData protocol and URL conventions.

3.3.2 OGC SensorThings Data Model

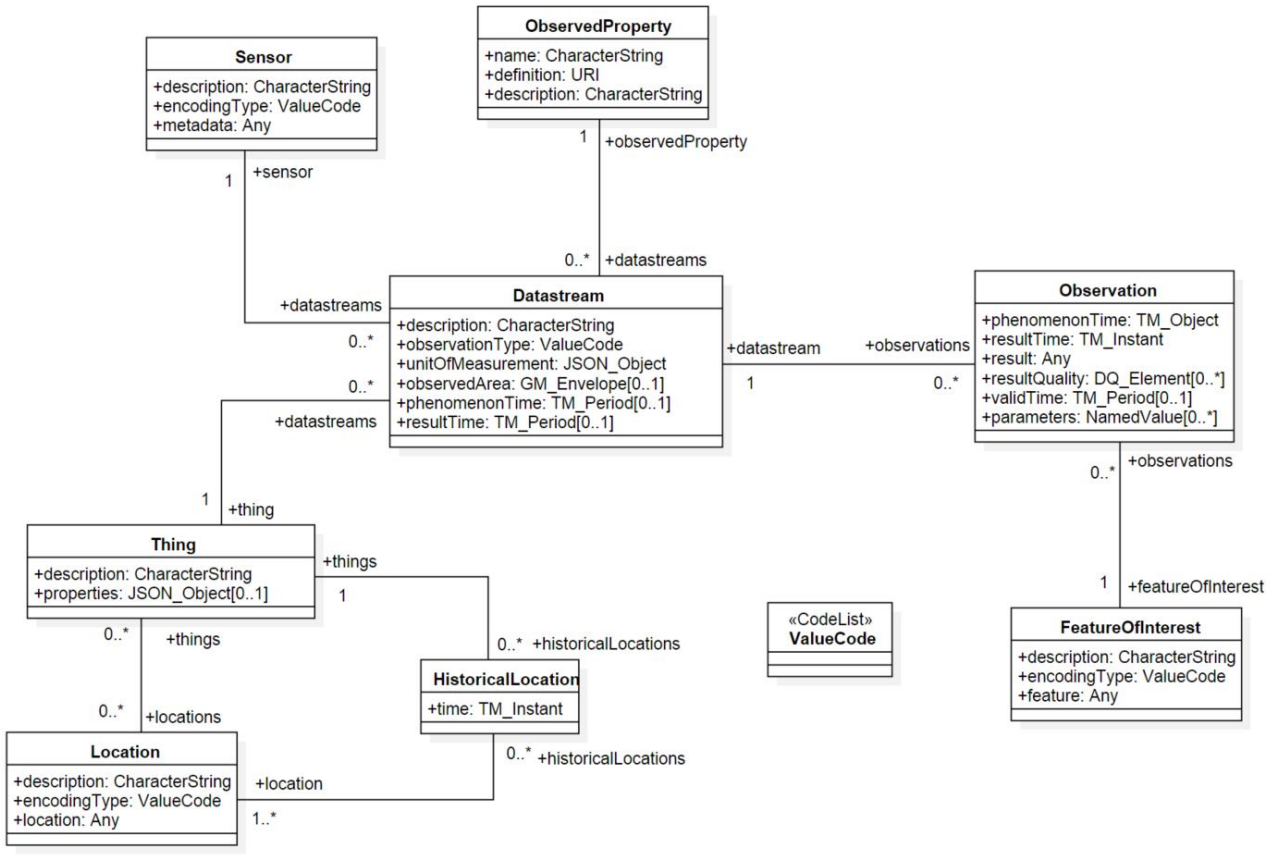

Figure 8: OGS SensorThings Static Model

As already mentioned, the OGC SensorThings API data model consists of the Sensing profile and the Tasking profile.

The Sensing profile allows IoT devices and applications to CREATE, READ, UPDATE, and DELETE (i.e., HTTP POST, GET, PATCH, and DELETE) IoT data and metadata in a SensorThings service. Managing and retrieving observations and metadata from IoT sensor systems is one of the most common use cases. As a result, the Sensing profile is designed based on the ISO/OGC Observation and Measurement (O&M) model (OGC and ISO 19156:2011).

The key to the model is that an Observation is modelled as an act that produces a result whose value is an estimate of a property of the observation target or FeatureOfInterest. An Observation instance is classified by its event time (e.g., resultTime and phenonmenonTime), FeatureOfInterest, ObservedProperty, and the procedure used (often a Sensor).

Moreover, Things are also modeled in the SensorThings API. Further the geographical Locations of Things are useful in almost every application and as a result they are included as well.

In the Sensing profile, a Thing has Locations and HistoricalLocations. A Thing also can have multiple Datastreams. A Datastream is a collection of Observations grouped by the same ObservedProperty and Sensor. An Observation is an event performed by a Sensor that produces a result whose value is an estimate of an ObservedProperty of the FeatureOfInterest.

3.3.3 Thing

The OGC SensorThings API follows the ITU-T definition, i.e., with regard to the Internet of Things, a thing is an object of the physical world (physical things) or the information world (virtual things) that is capable of being identified and integrated into communication networks (ITU-T Y.2060])

3.3.4 Location

The Location entity locates the Thing or the Things it is associated with. A Thing's Location entity is defined as the last known location of the Thing.

3.3.5 HistoricalLocation

A Thing's HistoricalLocation entity set provides the current (i.e. last known) and previous locations of the Thing with their time.

3.3.6 Datastream

A Datastream groups a collection of Observations and the Observations in a Datastream measure the same ObservedProperty and are produced by the same Sensor

3.3.7 Sensor

A Sensor is an instrument that observes a property or phenomenon with the goal of producing an estimate of the value of the property.

3.3.8 ObservedProperty

An ObservedProperty specifies the phenomenon of an Observation.

3.3.9 Observation

An Observation is an act of measuring or otherwise determining the value of a property (OGC and ISO 19156:2011).

3.3.10 FeatureOfInterest

An Observation results in a value being assigned to a phenomenon. The phenomenon is a property of a feature, the latter being the FeatureOfInterest of the Observation (OGC and ISO 19156:2001). In the context of the Internet of Things, many Observations' FeatureOfInterest can be the Location of the Thing. For example, the FeatureOfInterest of a wifi-connect thermostat can be the Location of the thermostat (i.e. the living room where the thermostat is located in). In the case of remote sensing, the FeatureOfInterest can be the geographical area or volume that is being sensed.

3.4 The Smart City Ontology

The focus of the OGC SensorThings data model is observation data, sensors and IoT devices. The Thing entity is "an object of the physical world or the information world that is capable of being identified and integrated into communication networks" (REF ITU-T Y.2060). This corresponds to the "Virtual Entity" concept of the IoT-A Domain Model, which may represent a "Physical Entity" in the

digital world. However, the information provided by the Thing entity of OGC SensorThings is limited to identity, location and a collection of key-value pairs. To represent the Smart City domain, and be able to explore type information and relationships between concepts in the Smart City, additional functionality and power of expression is needed. The smart city domain is represented in the Smart City Ontology.

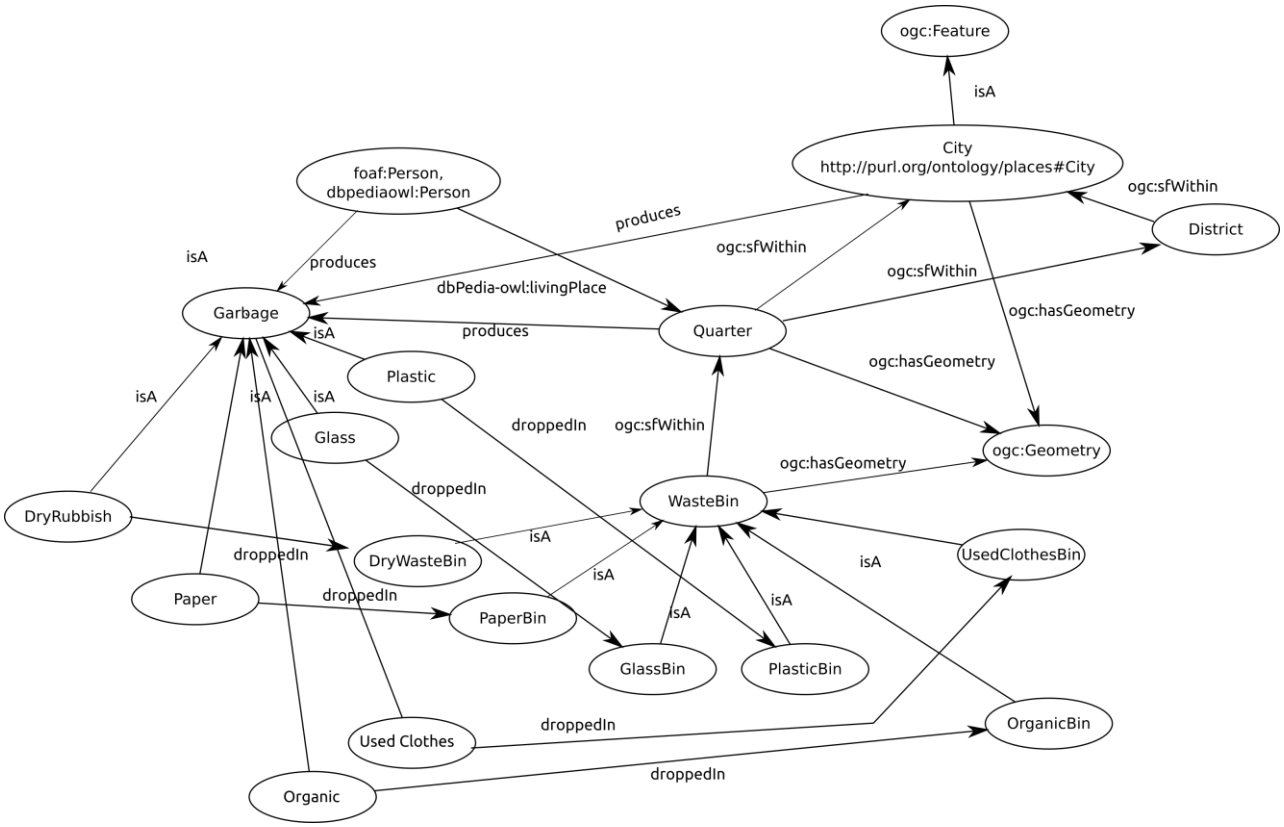

Figure 9: Ontology for the waste management use case

The Semantic Representation Framework (SRF) currently provides a REST API to perform queries and type inferencing on the graph data of the Smart City Ontology. (This is described in ID5.4.) The SRL will be extended with project specific queries to meet the needs of client applications.

If the application being developed uses static type information, e.g. in a configuration file, and does not need to perform type inference or queries over the ontology graph, the Smart City Ontology may be used during development only and not at run-time.

4. Developer APIs

4.1 Cloud API organization

The Cloud based APIs are structured in logical groups: the Smart City Resource Library API, the Historical Data API, the Live Data API, the Data Fusion Services API, the Provisioning API and the Management API. Since not all tasks in the ALAMANAC projects are finished or have started, the design and implementation status of the APIs and the components realizing the functionality varies.

Name	Format	API	Status	Realizing components
Smart City Resource Library API	OGC SensorThings, SPARQL 1.1. HTTP formats	OGC SensorThings API, SPARQL 1.1 HTTP	First version implemented	Virtualization Layer, Semantic Resource Framework, Resource Catalogue
Historical Data API	OGC SensorThings	OGC SensorThings API	First version implemented	Virtualization Layer, Storage Manager
Live Data API	OGC SensorThings	Custom	In development	Virtualization Layer, Data Fusion Manager
Data Fusion Services API	ALMANAC Data Fusion Language (spChains)	ALMANAC Data Fusion Language (spChains)	First version implemented	Virtualization Layer, Data Fusion Language, Data Fusion Manager
Provisioning API	OGC SensorThings + JSON-LD	OGC SensorThings API	In development	Virtualization Layer, Resource Catalogue, SCRAL
Management API	-	-	In design	Virtualization Layer, Security and Privacy Framework

4.2 Smart City Resource Library Services API

The Smart City Resource Library Services API (SCRLS API) s used to query for IoT Resources and Things based on metadata, e.g. resource type, the feature of interest being observed or observable property.

For type inference and queries on classes the smart city domain, the SCRLS API use the SPARQL 1.1. HTTP API described in (ID5.4), which provides easy programmatic access to the graph based domain models of the Smart City Ontology.

The instance and device queries use the OGC SensorThings API. This may be used to search instances of a specific type, location, or Datastream values, e.g. for all Things that are WasteBins, all Things that are within 100 meters of a geolocation, all Things on a specific address or Things that fulfil some restriction on the latest value of a Datastream.

The following OGC SensorThings API URI Patterns are part of the Smart City Resources Library API:

- SERVICE_ROOT_URI/ENTITY_SET_NAME
- SERVICE_ROOT_URI/ENTITY_SET_NAME(ID_OF_THE_ENTITY)

Resource	HTTP verbs	Query Options	Example
cloud-api/ld/v0_1/Things	GET	$filter, $skip, $top	/Things? $filter=type eq WasteBin /Things?$filter=location geo.distance 100 45.07248713 7.69348914 100=meter 45.07248713=latitude 7.69348914=longitude /Things?$filter=address startswith Via Bologna http://linksmart.cnet.se:44441/ogc/Things?$filter=location geo.distance 100 45.07248713 7.69348914&$xpath=//IoT:fillLevel[.>90]
cloud-api/ld/v0_1/Things(ENTITY_SET_NAME)	GET		/Things(FE6494DE)

4.3 Historical Data API

The Historical Data API provides access to stored observations in data streams (time series data) from sensors or data fusion queries. The observations may be filtered by time stamps or other properties of the observations using a subset of the OGC SensorThings API Sensing Profile. To find Observations based on type or properties of the Thing a Datastream belongs to, multiple query steps will have to be performed.

1. Optionally query the Smart City Resources Library API to find the resource types matching the query (e.g., inferring that the WasteBin means PaperBin, GlassBin, OrganicBin).
2. Find the resources (Things) that match the resource type and other constraints using the Smart City Resources Library API. E.g., ask for the ID of an OGC Thing with the type OrganicBin within 100 meters of a specified Location.
3. Use the IDs of the Datastreams of the Thing to query the Historical Data API for Observations.

The following OGC SensorThings API URI Patterns are part of the Historical Data API:

- SERVICE_ROOT_URI/ENTITY_SET_NAME
- SERVICE_ROOT_URI/ENTITY_SET_NAME(ID_OF_THE_ENTITY)

Resource	HTTP verbs	Query Options	Example
cloud-api/hd/v0_1/Datastreams	GET	$filter, $orderby, $count, $skip, $top	/Datastreams(AB6494CB) /Datastreams?$top=10&$skip=10
cloud-api/hd/v0_1/Datastreams(ID_OF_THE_ENTITY)/Observations	GET	$filter, $orderby, $count, $skip, $top	/Datastreams(AB6494CB)/Observations?$top=10&$skip=10
cloud-api/hd/v0_1/Observations	GET	$filter, $orderby, $count, $skip, $top	/Observations(3458) /Observations? $filter=Datastream/id eq 'AB6494CB' and resultTime ge 2010-0601T00:00:00Z

4.4 Live Data API

The Live Data API is used to query resources for data directly and subscribe to events from sensors or data fusion queries using web sockets. As with the Historical Data API, the Smart City Resources Library API is used to find the id of the resources, e.g. the ID of a Thing that is an instance of the class OrganicBin within 100 meters of a specified Location. The resource identifiers in the result may be used to subscribe for events via web sockets or to query the resources for data.

4.4.1 Direct querying

The direct querying or control/actuation of connected physical devices using the Live Data API is done via a subset of the OGC SensorThings API Sensing Profile, like the Historical Data API is used to query for historical data. Note that the Observations collection will only contain one observation, the latest/current value.

The following OGC SensorThings API URI Patterns are part of the Historical Data API:

- SERVICE_ROOT_URI/ENTITY_SET_NAME(ID_OF_THE_ENTITY)/ ENTITY_SET_NAME(ID_OF_THE_ENTITY)/ENTITY_SET_NAME

Resource	HTTP verbs	Query Options	Example
cloud-api/ld/v0_1/ Things(ID_OF_THE_ENTITY)/ Datastreams(ID_OF_THE_ENTITY)/ Observations	GET	-	Things(FE6494DE)/Datastreams(AB6494CB)/Observations

4.4.2 Subscription to events

The Live Data API for subscribing to events is not yet specified and will be further developed and detailed in the second iteration of this document (D7.3.2).

However, an implementation of the mechanism in the Virtualization Layer exists. It allows the application to subscribe to events from Datastreams (using MQTT topics, see D3.1.2) and receive data on web sockets. This mechanism has already been used in the ALMANAC prototypes used in demonstrations are detailed in (D5.1.2).

4.5 Data Fusion Services API

In the ALMANAC DFL specification (ID6.3.2), a set of REST APIs has been designed to allow users to Create, Read, Update and Delete (CRUD) data fusion queries defined using the DFL and implemented by the ALMANAC Data Fusion Manager. These APIs are still preliminary and might undergo several reviews during the project lifetime. These resources will be integrated into the full ALMANAC Cloud APIs.

The Data Fusion Services API is defined using the Swagger tool and syntax, which enables automatic generation of client and server-side code.

The main REST resources offered by the DF layer, for what concerns the definition of the stream processing chains are described in the table below.

Resource	Description	Allowed HTTP verbs
/dfs/v0_5/data-fusion/chains	The collection of all chains registered in the DFM, using the DFL language.	GET (read), POST (create)
/dfs/v0_5/data-fusion/chains/{chain-id}	The processing chain identified by the given {chain-id}. Chain definition is intended in a wide sense and includes also the specification of source and drains connected to the actual chain	GET (read), PUT (update),
/dfs/v0_5/data-fusion/sources	The sources currently registered in the DFM	GET (read), POST(create)
/dfs/v0_5/data-fusion/sources/{source-id}	The source identified by the given {source-id}	GET (read), PUT (update),
/dfs/v0_5/data-fusion/drains	The drains currently registered in the DFM	GET (read), POST(create)
/dfs/v0_5/data-fusion/drains/{drain-id}	The drain identified by the given {drain-id}	GET (read), PUT (update),
/dfs/v0_5/data-fusion/templates	The templates currently registered in the DFM	GET (read), POST(create)
/dfs/v0_5/data-fusion/templates/{template-id}	The source identified by the given {template-id}	GET (read), PUT (update),

The full documentation of available calls and formats is still subject to changes and a preliminary version of it is reported in Appendix A of the internal deliverable ID6.2.2.

4.6 Provisioning API

To define new types of resources, these types will have to be added to the Smart City Ontology using the Smart City Resource Library API. New instances of these types are then added by the SCRAL when devices, e.g. temperature or fill-level sensors, start reporting data. However, the SCRAL will not add resources (Things) that are not devices. To add new instances of e.g. a city Quarter or a PaperBin from an application at design time or run-time, the Provisioning API is used.

The details of the Provisioning API will be further developed and detailed in D7.3.2, the second iteration of this document, but the current design is outlined below.

To connect the OGC SensorThings API with semantic information in the Smart City Ontology, JSON-LD[5] will be used. E.g., to create a new instance of PaperBin, JSON-LD is used to annotate the representation of a Thing with class information from the Smart City Ontology.

```
{
  "@context": {
     "smartcity": "http://almanac-
project.eu/ontologies/smartcity.owl#"
  },

  "@id":
"fa26ae9fe5199c9330b14c92bd4a70099d4
35fba23aaac5dcebd5943d40e2fe9",
  "@type": "smartcity:PaperBin",
  "DataStreams": [{
    "@id":
"ba77ae9fe5199c9330b14c92bd4a70099d
435fba23aaac5dcebd5943d40eef45",
"ObservedProperty":{"@id":"e5c959aa911
9dddff08d9239452bbe77710c28617e4da1
9127885a4edd88d0a8"}
  }]
}
```

Figure 10: A Thing representation with JSON-LD annotation.

The semantic information in the JSON-LD format may be extracted as triples as shown in Figure 11.

```
<http:// almanac-project.eu
/fa26ae9fe5199c9330b14c92bd4a70099d435fba23aaac5dcebd5943d40e2fe9>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://almanac-
project.eu/ontologies/smartcity.owl#PaperBin>
```

Figure 11: Triples extracted from the JSON-LD Thing representation.

The following OGC SensorThings API URI Patterns are part of the Provisioning API:

- SERVICE_ROOT_URI/ENTITY_SET_NAME
- SERVICE_ROOT_URI/ENTITY_SET_NAME(ID_OF_THE_ENTITY)

[5] http://json-ld.org/

Resource	HTTP Verbs	Example
/cloud-api/pm/v0_1/Things	GET (read), POST (create), PUT (update), DELETE (delete)	/Things /Things(AB6494CB)

4.7 Management API

The Security and Privacy Framework realizing the Management API is still under design. The basc security framework is outlined in (D3.1.2) and will be further developed in the next iteration of the API design. Therefore we can only provide a cursory overview of the scope and purpose the Management API. It will handle:

- Users, roles and access control to devices and data – possibly offering the possibility to set access control for historical and live data differently - for ALMANAC platform instances and federations.
- Authentication mechanisms and access control for developers and end user applications.
- Also, the Management API will handle the necessary credentials and trust management for devices using the automatic provisioning in an ALMANAC platform instance performed by the SCRAL component.

5. Development Tutorial

This section will exemplify how the Cloud based APIs may be used by a developer building a small application on the ALAMANAC platform. We will use an example from the waste bin domain to illustrate the use of the Cloud based APIs. Sections for the Cloud based APIs that are still in development at this time will be added in D7.3.2, e.g. subscriptions to live data, resource provisioning and security management.

5.1 Finding types in the Smart City Ontology

The domain model of the smart city is represented by the Smart City Ontology. There, we may find the classes of Things relevant to the application, which in turn may have Datastreams with Observations (measurements) associated to them. We may query the Smart City Resource Library Services API (SCRLS API) using the SPARQL 1.1. HTTP interface (see D5.4). This way we can find the class identifers for all possible classes of wastebins and the ObservableProperties they may have. Using these classes, we may query other parts of the Cloud based APIs for instances and their data.

In this example, we are interested to find out the different sub classes of WasteBin, so that we may find the type of bins used for organic waste.

HTTP Metod	Endpoint	Request Content Type	Response Type
POST	/cloud-api/scrl/v0_1/resource/semantic/	application/ sparql-query	application/ sparqlresults+xml

The request body is a SPARQL query as shown in Figure 12.

```
SELECT ?subClass WHERE { ?subClass <http://www.w3.org/2000/01/rdf-
schema#subClassOf> <http://www.ismb.it/ontologies/wastebin#WasteBin> . }
```

Figure 12: SPARQL Query for types of waste bins.

The response can be seen in Figure 13 in SPARQL Query Results XML Format
(SPARQL 1.1 Query Results JSON Format and other response formats are also possible, described in ID5.4.1).

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="subClass" />
  </head>
  <results>
    <result>
      <binding name="subClass">
        <uri>http://www.ismb.it/ontologies/wastebin#UsedClothesBin</uri>
      </binding>
    </result>
    <result>
      <binding name="subClass">
        <uri>http://www.ismb.it/ontologies/wastebin#OrganicBin</uri>
      </binding>
    </result>
    <result>
      <binding name="subClass">
        <uri>http://www.ismb.it/ontologies/wastebin#DryWasteBin</uri>
      </binding>
    </result>
    <result>
      <binding name="subClass">
        <uri>http://www.ismb.it/ontologies/wastebin#PaperBin</uri>
      </binding>
    </result>
    <result>
      <binding name="subClass">
        <uri>http://www.ismb.it/ontologies/wastebin#PlasticBin</uri>
      </binding>
    </result>
    <result>
      <binding name="subClass">
        <uri>http://www.ismb.it/ontologies/wastebin#GlassBin</uri>
      </binding>
    </result>
  </results>
</sparql>
```

Figure 13: SPARQL Query response.

In many applications, the set of classes handled in an application may be fixed. Then we may use an ontology editor like WebProtege[6] to browse the Smart City Ontology at design time and store the type identifiers in the application's configuration file instead of fetching them at run-time.

[6] http://greencom.fit.fraunhofer.de:8080/webprotege/

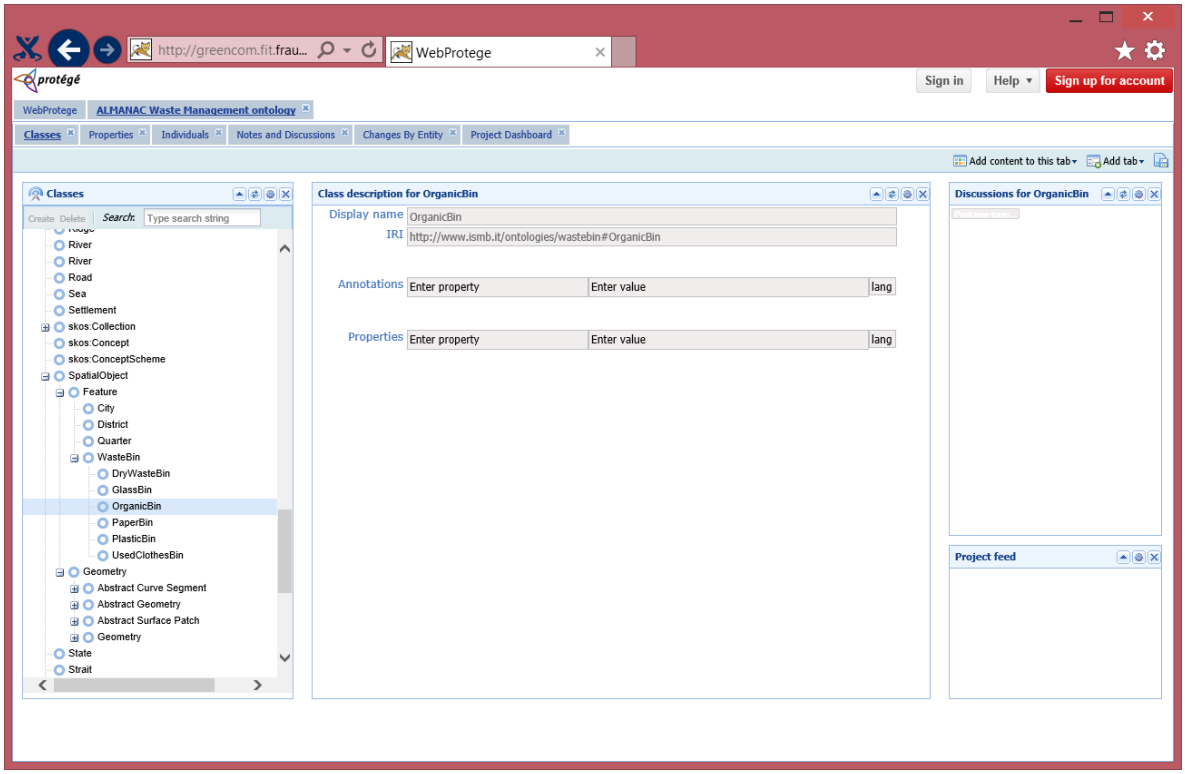

Figure 14: View of the WasteBin class hierarchy in WebProtege.

In Figure 14, we see how the WebProtege Classes view is used to browse the subclasses of WasteBin, and we see that OrganicBin has the URI:

"http://www.ismb.it/ontologies/wastebin#OrganicBin".

5.2 Finding resources

To find a resource, e.g. a device or Smart City Resource Library Services API, The Smart City Resource Library Services API is used. Using the type identifier we retreived from the SCRLS API, we may ask for all waste bins of the class OrganicBin within a 100 meters of a specific location.

HTTP Metod	Endpoint	Query String	Request Content Type	Response Content Type
GET	/cloud-api/scrl/v0_1/Things	filter=location geo.distance 100 45.07248713 7.69348914 100=meter 45.07248713=latitude 7.69348914=longitude	application/ json	application/json

In the reponse from this query, a part of which is shown in Figure 15, we can find the identifiers of Things, Datastreams and ObservableProperties we need to e.g. query for further data, perform actuation, or create DFL queries.


```
{
    "Thing": [
        {
            "id": "41ee87c585fc7b031e31cc10093d59f02cd09091df106ef49860cc5308861abf",
            "Description": "The WasteBin connected to the WasteBinSimulator network.",
            "Metadata": "http://almanac-project.eu/ontologies/smartcity.owl#WasteBin",
            "Locations": [
                {
                    "Time": "2015-06-10T08:14:57.497Z",
                    "Geometry": {
                        "type": "Point",
                        "coordinates": [
                            7.69279983,
                            45.07283598
                        ]
                    }
                }
            ],
            "Datastreams": [
                {
                    "id":
"a29444551b4c8f4919d8690b9a18167e83a9a32f0ea7356a00aa4c45f10407a1",
                    "ObservedProperty": {
                        "id":
"1a6df8a833eb841d860c3d61ea20fab5542e42361ca4ef9bb3024ae4f9596521",
                        "UnitOfMeasurement": "unknown"
                    }
                },
                {
                    "id":
"9b9e27f9a6c3ab90e29d9648f449b7377f82d28e46b7bdb34973cefa6730d6f4",
                    "ObservedProperty": {
                        "id":
"a79cdca3780f898368bb0477d684ef7af588f8a2889be867d7319c2cacdded55",
                        "URI": "http://almanac-
project.eu/ontologies/smartcity.owl#TemperatureState",
                        "UnitOfMeasurement": "C"
                    }
                },
                {
                    "id":
"1d989552561d30afcdcc4f5b89a8fcd68d2887f763a958fae74e582c47437fc7",
                    "ObservedProperty": {
                        "id":
"53e27d119786139435b85d9dcf7e30783c423640abd945c5d2388c92d1b12a05",
                        "URI": "http://almanac-
project.eu/ontologies/smartcity.owl#FillLevelState",
                        "UnitOfMeasurement": "%"
                    }
                }
            ]
        }
    ]
}
```


Figure 15: The response from a SCRLS API query.

5.3 Querying resources

The current and historical values of Datastreams associated with Things may be accessed through the Historical Data API. Using the information retrieved from the SCRLS API, we query the Historical Data API for the Observations from the Datastream we saw in the response from the previous query. In the table below is an example of a query for the Observations of a specified Datastream for the week starting 2015-08-17.

HTTP Metod	Endpoint	Query String	Request Content Type	Response Content Type
GET	/cloud-api/hd/v0_1/ Observations	$filter= Datastream/id eq 'ab1db42dea1bcdcb03f61b2a47ada8a77955715abe9bbed6611d82ba5ffa3570' and resultTime ge 2015-0817T00:00:00Z and resultTime le 2015-0824T00:00:00Z	application/ json	application/json

The response from the Historical Data API is a set of Observations from the specified time period.

```
{
    "Observations": [
        {
            "ID": "1",
            "Datastream": {
                "ID":
"ab1db42dea1bcdcb03f61b2a47ada8a77955715abe9bbed6611d82ba5ffa3570"
            },
            "ResultValue": "100.0",
            "Time": "2015-08-17T13:35:43.6900000Z"
        },
        {
            "ID": "2",
            "Datastream": {
                "ID":
"ab1db42dea1bcdcb03f61b2a47ada8a77955715abe9bbed6611d82ba5ffa3570"
            },
            "ResultValue": "100.0",
            "Time": "2015-08-18T11:05:43.6920000Z"
        },
        {
            "ID": "3",
            "Datastream": {
                "ID":
"ab1db42dea1bcdcb03f61b2a47ada8a77955715abe9bbed6611d82ba5ffa3570"
            },
            "ResultValue": "100.0",
            "Time": "2015-08-19T10:35:43.6580000Z"
        }
    ]
}
```

Figure 16: The response from the Historical Data API

5.4 Defining data fusion queries

To create new data fusion queries, the Data Fusion Services API is used. An example of combining data from existing sources using the Data Fusion Services API is the "bad smell detection" query. Whenever the weather becomes hot, e.g., in summer, and OrganicBin waste bins are almost full the chance to have unpleasant odour spreading in the neighbourhood of the rubbish collection isle increases dramatically. In a smart city scenario, this situation can easily be overcome by implementing "bad smell" detection as result of a Data Fusion process merging information from the current fill-level

of monitored waste bins and the current air temperature. Such detection process can be easily described using the ALMANAC DFL as shown in Figure 17.Figure 17

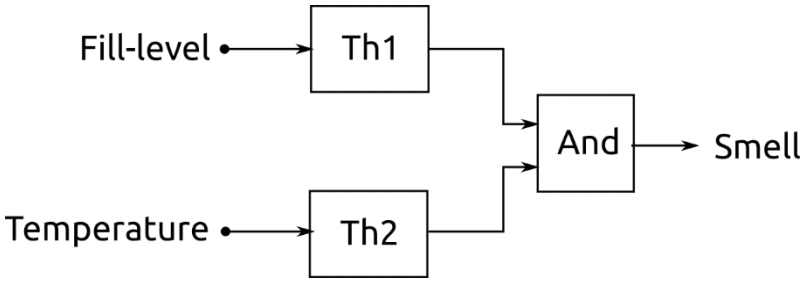

Figure 17. Bad smell detection query with event streams and threshold filters

The corresponding DFL construct, in JSON notation is reported in Figure 18Figure 18. Is it easy to notice how simple is the definition of such a chain and, the degree of customization that can be supported, even in this very simple scenario. By posting the chain definition to the Data Fusion Services API, we have created a data fusion query that can be used by the application to detect bad smell from waste bins.

HTTP Metod	**Endpoint**	**Request Content Type**	**Response Content Type**
POST	/dfs/v0_5/data-fusion/chains	application/ json	application/json

```
{
  "chains": [
    {
      "id": "smell_detection",
      "blocks": [
        {
          "id": "th1",
          "function": "threshold",
          "params": [
            {
              "name": "threshold",
              "value": "80",
              "uom": "%"
            },
            {
              "name": "mode",
              "value": "rising"
            }
          ]
        },
        {
          "id": "th2",
          "function": "threshold",
          "params": [
            {
              "name": "threshold",
              "value": "27",
              "uom": "°C"
            },
            {
              "name": "mode",
              "value": "rising"
            }
          ]
        },
        {
          "id": "and",
          "function": "and",
          "params": [
            {
              "name": "window",
              "value": "1",
              "uom": "h"
            }
          ]
        }
      ],
      "connections": [
        {
          "from": {
            "blockId": "th1",
            "port": "out"
          },
          "to": {
            "blockId": "and",
            "port": "in1"
          }
        },
        {
          "from": {
            "blockId": "th2",
            "port": "out"
          },
          "to": {
            "blockId": "and",
            "port": "in2"
          }
        }
      ],
      "inputs": [
        {
          "blockId": "th1",
          "port": "in",
          "id": "fill_level"
        },
        {
          "blockId": "th2",
          "port": "in",
          "id": "temperature"
        }
      ],
      "outputs": [
        {
          "blockId": "and",
          "port": "out",
          "id": "out"
        }
      ]
    }
  ]
}
```


Figure 18. Odour detection chain definition in JSON.

6. Future work

The parts of the Cloud based APIs that are still in development will be specified in the next iteration of this deliverable: the Live Data API, the Provisioning API and the Management API. Swagger descriptions of the Cloud based APIs will be provided for these APIs.

The capabilities of the Cloud API categories will thus evolve, among these are enhanced semantic entity query capabilities, providing an integration of the OGC data and Smart City Ontology resources with external services for business systems data. The progress of this work will be recorded in the Cloud Based API online resource page linking the tutorial examples to an ALMANAC Platform Instance "sandbox" for experimentation with and test of the APIs.

The possibility of adding graphical interfaces for parts of the Cloud based APIs will be investigated in order to improve the quality and ease of use. The Data Fusion Language in particular is considered suitable for a graphical representation. For the final version of the Cloud based development APIs we foresee a Web IDE including the collection of ALMANAC tools as well as hyperlinked versions of the developer tutorials.

7. References

(Bonino et al., 2012-1)	D. Bonino and F. Corno. spchains: A declarative framework for data stream processing in pervasive applications. Procedia Computer Science, 10(0):316–323, 2012. ANT 2012 and MobiWIS 2012.
(Bonino et al., 2013)	Bonino, D., F. Corno, and L. De Russis, "Real-time Big Data Processing for Domain Experts, An Application to Smart Buildings", Big Data Computing: CRC Press, pp. 415-447, 2013.
(ID6.2.2)	ALMANAC ID6.2.2 Data Fusion Language and Prototype 2
(ID6.3.2)	ALMANAC ID6.3.2 Event Processing Agent for Smart City Applications
(ID5.4)	ALMANAC ID5.4 Ontologies and Semantic Representation Layer Prototype 1
(D3.1.2)	ALMANAC D3.1.2 System Architecture Analysis and Design Specification 2
(D5.1.2)	D5.1.2 Design of the abstraction framework and models 2
(Liang et al., 2015)	Liang, S., Huang, C., Khalafbeigi, T. (2015) OGC® SensorThings API, Part 1: Sensing
(OGC and ISO 19156:2011)	OGC Abstract Specification, Geographic information Observations and measurements
(ITU-T Y.2060)	Overview of the Internet of things