COSMOS

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement Nº 609043

D7.7.3 Integration of Results (Year 3)

WP7: Use cases Adaptation, Integration and Experimentation

Version: 1.0
Due Date: 31/08/2016
Delivery Date: 31/08/2016
Nature: Report
Dissemination Level: Public

Lead partner: ICCS/NTUA
Authors: George Kousiouris, Achilleas Marinakis, Panagiotis Bourelos, Orfefs Voutyras, Kyriaki-Dafni Galetza (ICCS/NTUA), Juan Sancho (ATOS), Shelly Garion ,Paula Ta-Shma (IBM), Francois Carrez, Adnan Akbar (UniS), Bogdan Târnaucă, Leonard Pitu (SIEMENS), Andres Recio (EMT)

Internal reviewers: Juan Rico (ATOS), Alexandros Psychas, (ICCS/NTUA)

www.iot-cosmos.eu

The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement n° 609043

Version Control:

Version	Date	Author	Author's Organization	Changes
0.10	10/06/2016	George Kousiouris	ICCS/NTUA	Update of previous version ToC based on new version of Integration Plan
0.15	16/06/2016	George Kousiouris	ICCS/NTUA	Initial Definition of System and Integration Group testing, Definition of subchapter template
0.20	25/06/2016	George Kousiouris	ICCS/NTUA	Finalization of Integration Group testing and ToC assignments
0.25	5/07/2016	George Kousiouris	ICCS/NTUA	Adding contributions on Twitter data feed, Fab/Lab UC, Marketplace implementation
0.30	12/07/2016	Juan Sancho	ATOS	Updated Madrid Scenario Y3
0.35	20/07/2016	George Kousiouris	ICCS/NTUA	Editing of Introduction, Conclusions
0.40	21/07/2016	Guy Gerson	IBM	Updated Secor Health Check
0.45	25/07/2016	Adnan Akbar	Univ. Surrey	Updated Madrid Events Combination
0.50	10/8/2016	George Kousiouris, Bogdan Tarnauca, Shelly Garion	ICCS, Siemens, IBM	Inputs on VE registry and Privacy and Consent Integration
0.55	17/8/2016	G. Kousiouris, K. Galetza	ICCS/NTUA	Inclusion of sections on Sound analysis requirements, Editing of various sections
0.60	20/8/2016	Achilleas Marinakis, Guy Sharon	ICCS/NTUA, IBM	Additions in privelets & registry cooperation, Data Mapper health checks
0.65	22/08/2016	Juan Sancho	ATOS	Completed Secor Health Check
0.70	22/07/2016	Adnan Akbar	Univ. Surrey	Updated Taipei Scenario Y3
0.75	25/08/2016	Orfefs Voutyras	ICCS/NTUA	Updated Camden Scenario Application, Sound Detection Application and Camden Feedback Conclusions
0.80	25/08/2016	Andres Recio	EMT	Madrid Feedback WebPanel
0.85	26/08/2016	George Kousiouris, Juan Sancho	ICCS, ATOS	Final edits and checks, ready for QA
0.90	29/08/2016	Juan Rico, Alex Psychas	ATOS, ICCS	Internal Review
1.0	31/08/2016	George Kousiouris	ICCS	Final Version Ready

Table of Contents

Table of Figures

List of Tables

Acronyms

Acronym	Meaning
AD	Application Developer
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CB	Case Base
CBR	Case Based Reasoning
CE	Complex Event
CEP	Complex Event Processing
µCEP	micro (lightweight) CEP
CG	Care Giver
CRS	Coordinate Reference System
CSV	Comma Separated Values
D	Deliverable
DDP	Distributed Data Protocol
DSO	Domain Specific Ontology
FC	Functional Component
GPS	Global Positioning System
GUI	Graphical User Interface
GVE	Group Virtual Entity
HSB	Hardware Security Board
HTTP	Hyper Text Transfer Protocol
HVAC	Heating, Ventilating, and Air Conditioning
ID	Identifier
IoT	Internet of Things
IP	Integration Point
JSON	Java-Script Object Notation
JVM	Java Virtual Machine
KNN	K-Nearest Neighbours
MB	Message Bus

ML	Machine Learning
PM	Person Month
QoL	Quality of Life
REST	Representational State Transfer
SAw	Situational Awareness
SD	Sequence Diagram
SIoT	Social Internet of Things
SP	Special Person
SQL	Simple Query Language
SVM	Support Vector Machine
UC	Use Case
UI	User Interface
UR	User Requirement
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTM	Universal Transverse Mercator
VE	Virtual Entity
VM	Virtual Machine
VPN	Virtual Private Network
WP	Work Package
XML	Extensible Markup Language
XP	Experience

Executive Summary

This report presents the results of the integration process carried out in year 3 of the COSMOS project. We continued the process from Y2, following the details and requirements, as these are expressed in D7.6.2 “Integration Plan (updated)”, and finalized in D7.6.3 “Integration Plan(Final)”, with relation to further integration between the COSMOS components and especially between the latter and the UCs, especially for the goals set out in the latter with relation to subsystems and functionalities. It is necessary to stress that we have retained the material from D7.7.2 and the previous integration periods (included in Chapter 2 and 3), mainly in an effort to have one centralized document in which all integration related information will be kept for future reference. Thus the new content in this version is centered around Chapter 4 and the related integration periods that fall under the scope of this document.

Following, and by receiving input from the Integration Plan for periods PM26-34 and PM35-36, the subsystems and integration points to be tested have been defined. The tests have been centered around 9 integration groups/scenarios, deriving from either the aforementioned subsystems or from the COSMOS UCs. It is necessary to stress that all the technical groups (derived from the relevant subsystems combinations) were instructed to work directly to the related UCs specified per case. Relative sequence diagrams have been extracted, indicating the testing process, along with deployment diagrams per case, message formats configuration, test cases and scenarios. Through the groups, we aim to highlight and test the COSMOS processes, functionalities and UC integration processes such as:

- Incorporation of end user feedback in the technical outcomes. While the details of this feedback is included in D7.3.3 and D7.2.3, their final conclusions and how these have affected backend technical implementations are also sketched in this document
- VE Registration API process, aiming to incorporate VE registry integration in aspects such as fuzzification of a specific feature or participation in friend recommendation processes.
- Incorporation of the privacy and consent management mechanism in the COSMOS environment and its cooperation with historical data retrieval of the Planner, as well as incorporation of failsafes and user preferences in the process
- Data flow from different types of sources, adaptively annotated, grouped and stored on a Cloud based object storage, extending from Y2 to include new sources (e.g. Twitter and weather data) available through necessary Node-RED flows and bridges
- Finalization of VE side component integration process, in order to produce one final software image that can be downloaded and used directly on the VE gateway device
- Integration actions and necessary points that are needed for the application scenarios that have been defined for Y3 for the UCs, including bridging and adaptation points with the COSMOS components
- Packaging and abstraction processes to enhance artefact reusability

This new set of features comes to complement or abstract the already achieved ones at Y1 and Y2, leading to a feature-rich and flexible solution that the COSMOS environment envisions to be. Furthermore, a set of extra use cases have been identified and implemented in Y3 in an effort to check COSMOS applicability in new areas or enhance the sustainability of the project outcomes. These actions and necessary integration/adaptation processes with the COSMOS environment revolve around the following aspects:

- An exploratory application of the Planning capabilities of COSMOS in a Sound detection and analysis use case, in which the Planner identifies sounds detected in a Smart home environment for enabling people with hearing impairment to strengthen their environmental awareness
- A collaboration in terms of data feeds to provide to a Fabrication Lab (www.formdecode.com) the specified data set for the exploration of a fabrication workshop scenario
- The creation of a marketplace entity (Eventflows.com) through which the Events on events archetype can be instantiated and the COSMOS technical outcomes in the form of events may be described and used, along with the further enhancement of these through combinatorial event orchestration, including also the reusability of produced COSMOS flows and functionalities.

For each of these cases, the defined testing scenario implementations are portrayed and information on their realization is provided. Where applicable, we highlight the process followed, so that it can be adapted by external audiences following the completion of the project. Data and message formats are portrayed for relaying and processing information. For the involved subsystems, the relevant integration points are identified and analyzed for the specific scenario case.

As a conclusion, the goals identified for this final integration period have been met, resulting in an extended and more feature-rich running instance of a COSMOS Platform Provider, offering increased applied functionalities that were tested through direct UC scenarios. 3 further scenarios have been also tested, in which COSMOS functionalities have been applied with very satisfactory results in terms of ease of usage and outcome. In relation to the integration plan, we have achieved a very satisfactory percentage of maturity levels, with all of the subgoals achieved, while exploring 3 additional external cases that were mentioned (Fab/Lab collaboration, Sound Detection and Events Marketplace). From the Initial versions of the applications in Y2, significant extensions have been implemented on all available UCs at the COSMOS side, enhancing the logic and complexity in Y3, in many cases with reduced need for integration due to the usage of Node-RED, for incorporating either new features or user desires in the flows.

1. Introduction

1.1. Purpose of the Document

D7.7.3 has the purpose of consolidating the results from the work performed in the technical tasks, aiming to provide the integrated view of COSMOS outcomes, with relation also to the UC systems and necessary interconnection. In order to achieve this goal, a process has been already defined in the context of D7.6.3 [7], that is concretized in specific actions in this document, describing the concrete integration points and necessary functionalities in them.

This includes the definition of subsystem based testing, along with determination of the integrated scenarios that are designed to drive inter-component cooperation and demonstrate the added value in the project development and in the context of the UCs. This process goes hand in hand with the work performed in WP2, regarding the architectural structure of the COSMOS framework as well as the requirements definition. In addition, the aim is to validate this approach based on the generic functionalities that the COSMOS platform should provide, and indicate how these can be instantiated or used. The test scenarios have been chosen so that they are directly included in the project UC scenarios, initial versions of which are available following the work presented in this document.

For each of these scenarios, the relevant parts of the platform are identified and the performed tests and results are portrayed, in terms of needed configuration and presentation of the results.

1.2. Integration Periods within the scope of this document-Differences with relation to the previous documents

The integration periods (as defined in D7.6.3) corresponding to this document are:

- Integration period 4 (PM26-34) and
- Integration period 5 (PM35-36).

Both of which are included **in Chapter 4**, in order not to break actions necessary for one flow in many chapters, given that most integration goals span across both time intervals.

In the document we have also maintained Chapters 2 and 3 from D7.7.2 (previous version of this document), in order to have an integrated view and a single point of reference for the overall integration process.

1.3. Document Structure

The document is structured as follows:

- In Chapter 2, the previous integration period (PM10-16) results are maintained, only for completeness purposes as mentioned above
- In Chapter 3, the previous integration periods (PM17-22 and PM23-25) results are maintained, only for completeness purposes as mentioned above
- Chapter 4 contains the **new** additions for **Y3** and with relation to the goals of these periods (PM26-34 and PM35-36) as identified by the template in the integration plan, and the various integrated tests are portrayed, along with the developed integration points, necessary configurations and results for each subsystem or application. Sequence diagrams are also included where applicable, to indicate the concrete interactions between the components while an evaluation of the integration goals (summarized in Section 4.1) with relation to the achieved results is portrayed (in

Section 4.3). Specific sections with relation to end user involvement have also been included to indicate the high level results of the user surveys performed in the context of the UCs and how these have affected the technical and integration processes (Section 4.2.9). Furthermore, extended UC scenarios have been explored based on our involvement in the IERC activities and various other activities such as the organized Hackathons in the context of WP8 (Sections 4.2.14 and 4.2.8).

- In Chapter 5, conclusions are portrayed with relation to this period and its outcomes.
- In the Annex we have maintained the unit tests on a component level, for the baseline components that have been available from Y1.

2. Integration Stage 1 (M10-M16) from D7.7.1

2.1. Overview from Integration plan

As identified in the Integration plan, the main goals for this period include:

- COSMOS Platform Setup
- Data Management and Analytics (Subgoal)
  - Data Feed, annotation and storage (Subsystem)
  - Storage and Analytics which can be divided into
    - Metadata search Storlet (Subsystem)
    - Modelling and Storage Analytics (Subsystem)
  - Security, Privacy and Storage with Analytics (Subsystem)
- Autonomous VE Behaviour (Subgoal)
  - Autonomous Behaviour with minimal integration to the platform (Subsystem)
  - Autonomous Behaviour with Platform involvement and automated event detection (Subsystem)

The main outline on how to create a COSMOS Platform provider is included in D7.6.1. In the following figure, Figure 1, we demonstrate how the components were deployed in the internal COSMOS Platform for the purpose of the experiments. More detailed deployment diagrams are included also in the individual subsystem cases.

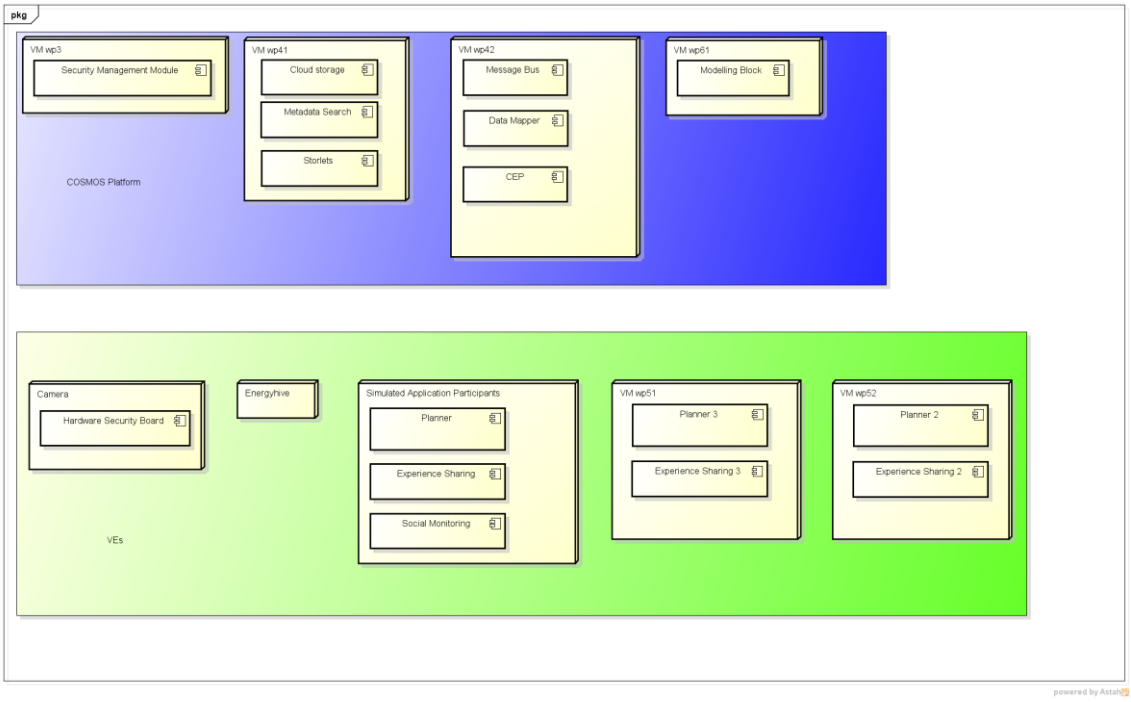


Figure 1: COSMOS Platform Overall Deployment Diagram

2.2. Performed Tests

In order to implement the performed tests on the defined subsystems originating from D7.6.1, the following process was followed:

- For each subsystem, a relevant aspect was investigated that would be of potential future use to the UC scenarios, in terms of practical aspects;
- For each subsystem, a generic end-to-end sequence diagram was created to drive the testing process;
- For each subsystem, a responsible person was appointed, that would drive the actual integration and testing. This person was responsible for organizing remote integration workshops (usually through Skype), with almost daily frequency, during which specific aspects would be tested, along with the developers whose components participated in the flow, towards the final goal of the scenario.

Following, we present details on each subsystem examined, including specific deployment diagrams, subsystem test cases (where applicable) and results screenshots, along with information on the configuration details.

2.2.1. Subgoal: Data Management and Analytics Tests

This subgoal involves the following components:

Message Bus

For Y1 demonstration we plan to have one statically defined topic for each data source.

Data Mapper

For Y1 demonstration the Data Mapper will be configurable regarding the size of the objects stored in the cloud and their metadata.

Swift Cloud Storage

The Data Mapper will collect data from the Message Bus and create Swift objects in the Cloud Storage. We envisage a single object to contain data gathered for a certain Virtual Entity over a period of time. For example, the Virtual Entity could be an apartment in Camden or a bus line in Madrid. A period of time could be a day or a week.

Metadata Indexing

The Data Mapper will associate Swift data objects with metadata. When this happens the Metadata Search component indexes this metadata automatically to make it available for subsequent search. Example metadata could be the time period (minimum and maximum timestamps) of data in an associated object.

Storlets

Storlets can be applied to data when they are created and/or when they are accessed. In this context, Storlets applied on data creation could be relevant. For example, when data from Camden housing are uploaded we could apply a Storlet to downsample the data or to calculate certain statistics about the data. Storlets can be also more generic, having a wide range of applications (e.g. image blurring, data preprocessing, etc.).

Analytics

Different data mining algorithms including Machine Learning and Time Series analysis can be applied on the data to achieve higher-level knowledge that refers to an event, a pattern or an abstract concept.

Security Framework

The cooperation between the security framework (H/W and S/W end) and the platform (especially the Cloud Storage) is an aspect that will ensure secure transfer between the IoT platform and the COSMOS platform.

The aforementioned components are combined in the following scenarios.

2.2.1.1 Data Feed, Annotation and Storage

In this scenario we aim to demonstrate the flow of data from their generation to their persistent storage in the cloud and test that an end-to-end data flow scenario works as expected using a live data feed. Specifically, we focus on:

- Integrating Camden UC data with the Message Bus
- Testing that the Data Mapper can collect data from the Message Bus
- Testing that the Data Mapper can generate objects in the Cloud Storage with associated metadata

Figure 2 describes the scenario while Figure 3 indicates the corresponding subsystem from the Integration plan [1].

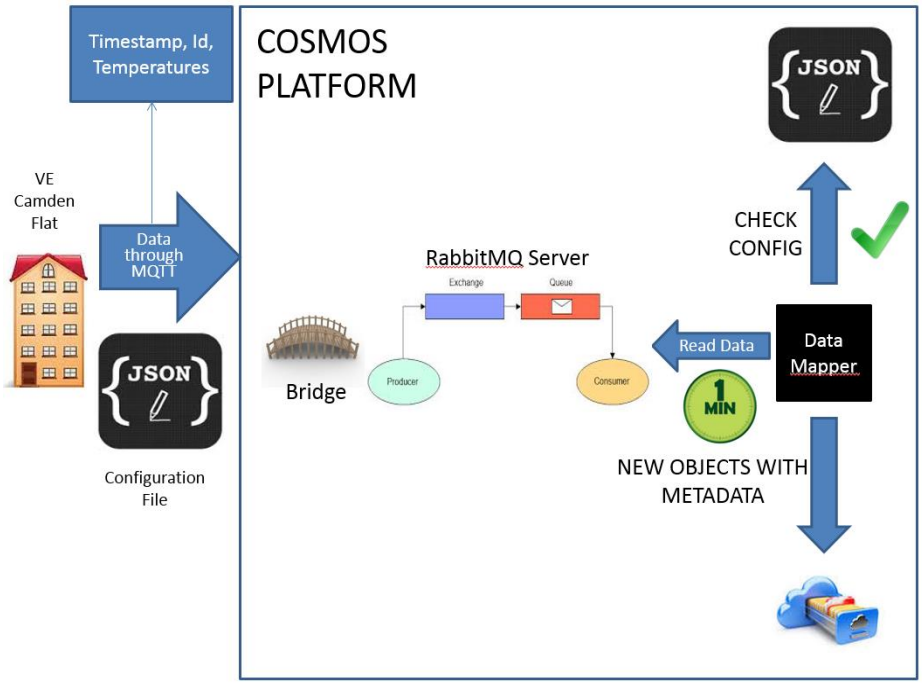


Figure 2: Data Feed, Annotation and Storage Scenario

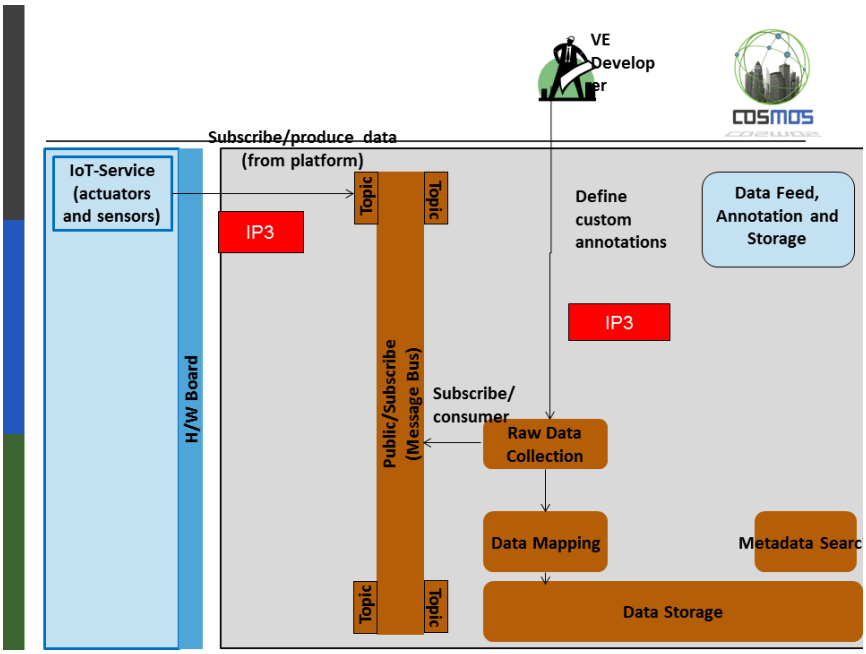


Figure 3: Data Feed, Annotation and Storage Subsystem

Link with COSMOS Message Bus

In order to address the first IP3, the VE Developer must install a RabbitMQ client to be able to publish data to the Message Bus, which is deployed in the COSMOS platform. For this purpose, a relevant exchange (topic), where the data will be published, needs to be created. In case the VE uses another messaging protocol (like MQTT), a bridge between this and the RabbitMQ must be implemented. The bridge will be running inside the COSMOS platform and will redirect the data from the source (VE) to the aforementioned exchange. The process is included in Figure 4.

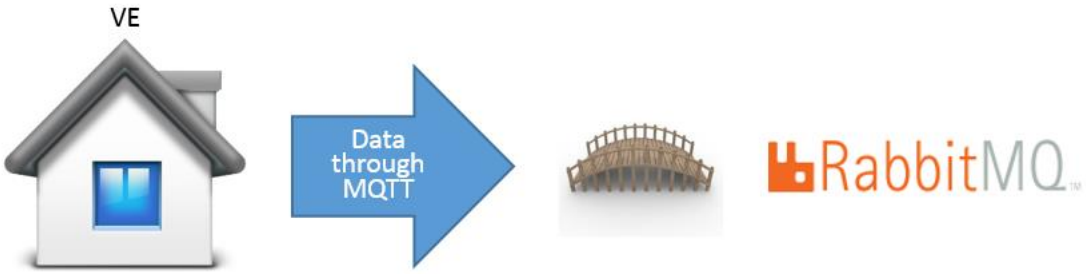


Figure 4: Subscribe/produce data adaptation

Figure 5 shows the configuration of the Bridge that was implemented for the year 1 scenario, in order to incorporate data coming from the Energyhive platform of Camden UC. “sourceHost” is the VE’s endpoint and “targetHost” is the IP address of the VM where COSMOS Message Bus is deployed. Figure 6 indicates the topic we have created to collect the data from

the Camden Use Case. In order to monitor the topic and its configuration, we installed the RabbitMQ Management plugin, which provides an HTTP-based API. Through this web interface, we can also export statistics like queue length, message rates globally and per channel, data rates per connection, etc.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
<appSettings>
                <add key="sourceHost" value="host" />
                <add key="sourceUser" value="user" />
                <add key="sourcePassword" value="password" />
                <add key="sourceTopic" value="topic" />

                <add key="targetHost" value="localhost" />
                <add key="targetUser" value="" />
                <add key="targetPassword" value="" />
                <add key="targetTopic" value="FromMosquitto" />
        </appSettings>
</configuration>
```

Figure 5: Bridge Configuration

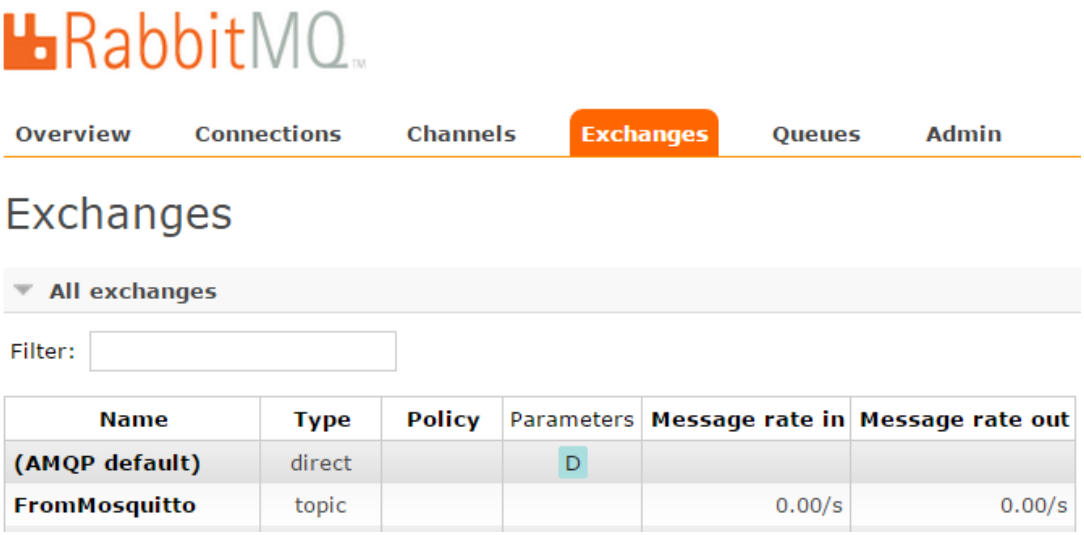


Figure 6: RabbitMQ HTTP-based API - Topic "FromMosquitto"

Data Format

Regarding the data format, JSON has been adopted as the one to be used in the COSMOS platform, but if a VE uses another format, a JSON adapter is needed for the data to be converted in JSON. Figure 7 describes an example of the Camden JSON data, used in the year 1 scenario.

```
{
    "estate":"Dalehead",
    "hid":"cPGKKhiI4q6Y",
    "ts":"1405578854",
    "instant":"226",
    "returnTemp":"72.3",
    "flowTemp":"72.4",
    "flowRate":"742",
    "cumulative":"15703"
}
```


Figure 7: JSON Message from the Camden UC

Metadata Annotation

With regard to the second IP3, the VE Developer must fill in the Data Mapping's configuration file, structured in JSON format. Thus, the data will be associated with enriched metadata in the COSMOS cloud storage and therefore they can be easily retrieved.

The configuration file, used in the year 1 scenario, is the following:


```
{
   "period":"1",
   "size":"1",
   "mandatory_metadata":{
      "Timestamp":"ts",
      "Id":"hid"
   },
   "optional_metadata":{
    "Estate":"estate"
   }
}
```


Figure 8: Configuration File

The "period" key indicates how often the Data Mapping sends the data to the cloud storage. The unit of measurement is the minute and the value is an Integer.

The "size" key defines the lowest threshold of the message to be stored. If an aggregated message is smaller than the threshold, it has to wait for the next period before being stored. The unit of measurement is the kilobyte and the value is an Integer.

The mandatory metadata mean that the Data Mapping does not store any data that do not contain this kind of information, whereas for the optional ones, we give the VE Developer the capability to annotate the data with enriching metadata. Metadata checks (for the mandatory parts) are applied in all cases of testing.

The sequence diagram for this subsystem is depicted in Figure 9. The tested part in this scenario is depicted in the highlighted section.

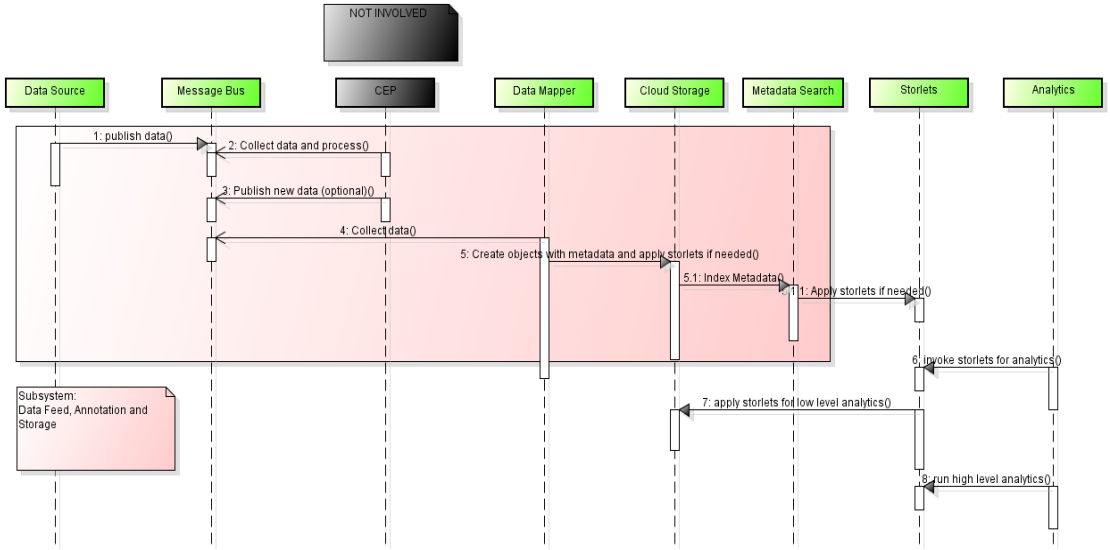


Figure 9: Sequence Diagram for Data Feed, Annotation and Storage Subsystem

Table 1 presents the results while executing the Subsystem Test Case.

Table 1: Data Feed, Annotation and Storage Test Case

Test Case Number Version	**Data Feed, Annotation and Storage_1**
Test Case Title	Storing live annotated data, coming from Camden flats through the COSMOS Message Bus, in the Cloud Storage.
Modules tested	Message Bus, Data Mapper, Cloud Storage, Metadata Search
Requirements addressed	4.1, 4.2, 4.3
Initial conditions	Camden data stream is available RabbitMQ service is installed and running in the COSMOS testbed Exchange (topic) “FromMosquitto” has been created in the Message Bus Bridge between Camden MQTT and COSMOS RabbitMQ is running Data Mapper’s configuration file is properly filled in
Expected results	Every 1 minute, data are stored as objects with Id, timestamps and other metadata. The size of the objects cannot be smaller than 1 kilobyte.
Owner	VE Developer
Steps	Execute: /NTUA/DataMapping/target java -jar DataMapping-Y1.jar Camden
Passed	Yes
Bug ID	None
Problems	We had to ensure that the bridge was up and running, before executing the Test Case
Required changes	The bridge needs to be run in daemon form, given the problem mentioned above

The following two figures depict the expected results that are written in the table above. Specifically, Figure 10 presents an aggregated JSON message that has been stored in the COSMOS Cloud Storage, whereas Figure 11 shows its metadata, fully aligned with the configuration file defined by the VE developer (when compared with Figure 8).

Figure 10: Data Object Body

Remark: All the messages have the same 'hid', which corresponds to the 'VE id' according to the configuration file. This means that the Data Mapping aggregates data coming from multiple sources but groups them in storage objects based on the specific field, defined in the configuration file (VE id).

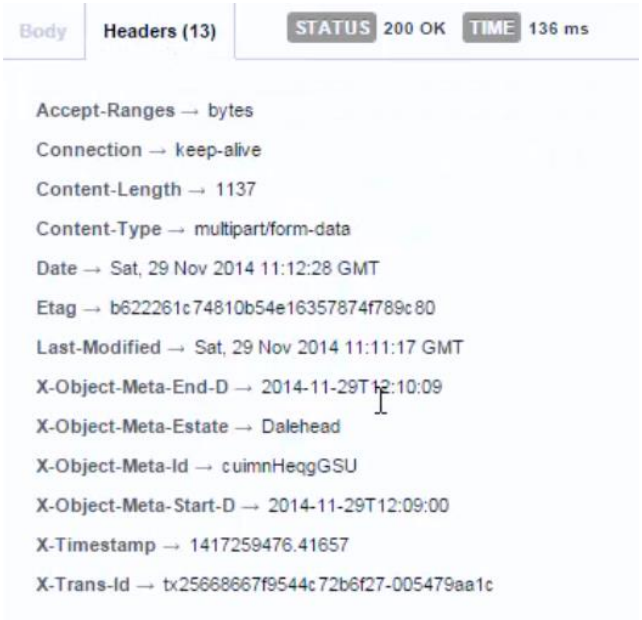


Figure 11: Data Object Metadata

Remark: The size of the object is 1137 bytes > 1 kilobyte, which is defined as the lowest threshold in the configuration file. Thus the Data Mapping abides by the specific constraint.

The Deployment Diagram of the scenario is shown in Figure 12.

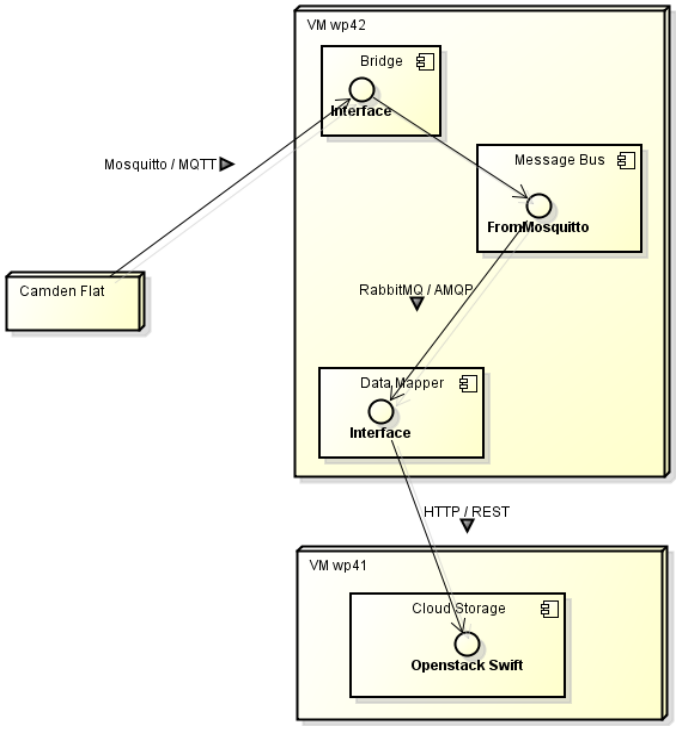


Figure 12: Data Feed, Annotation and Storage Deployment Diagram

2.2.1.2 Storage and Analytics on Metadata

The target subsystem in this case, as identified in D7.6.1, is included in Figure 13, identified by 3 integration points.

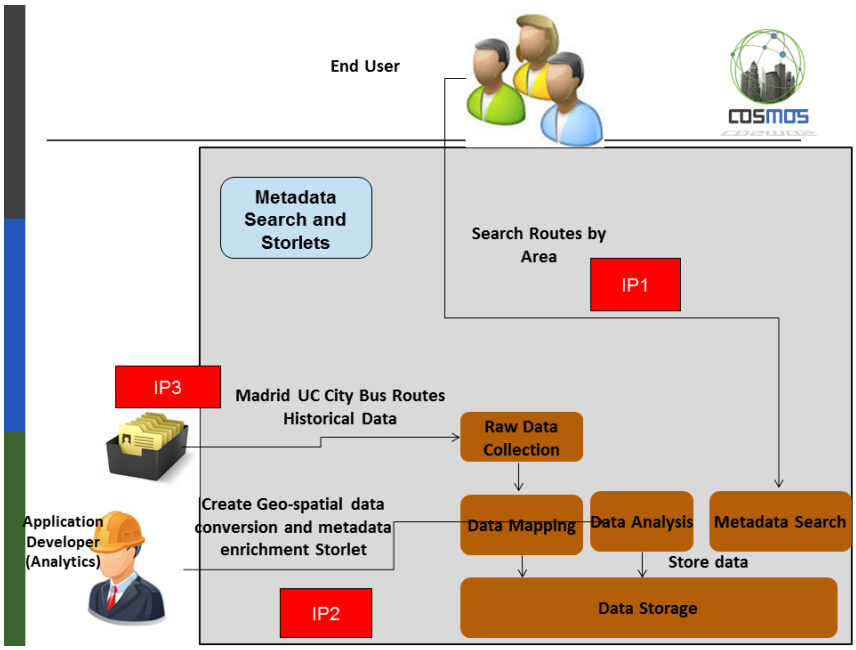


Figure 13: Metadata Search and Storlets Subsystem

The subsystem involves the integrated usage of object storage (OpenStack Swift) together with Storlets and geospatial metadata search for historical geospatial from the EMT Madrid bus transport use case. The aim was to show how Storlets and geospatial metadata search can be applied in an integrated fashion to geospatial data in order to locate data objects of interest. The demonstration involved a web front end with an intuitive GUI, which was developed using Javascript for this purpose, corresponding to IP1 (End user involvement and interface).

Scenario

The scenario involves data collected from EMT buses. Buses communicates data points approximately every 30 seconds and transmit data to the control centre including GPS location, odometer readings and the state of the door at the time the message is sent (Figure 14).

The Data Mapper is responsible for grouping these messages into objects and periodically uploading them to the cloud storage, as detailed in the previous scenario (Data Feed, Annotation and Storage), and corresponds to IP3. Since live feeds were not yet available from EMT with the data we needed, we uploaded excel files containing historical data for bus trips to the object storage. Each excel file contained data for a particular bus line during a particular time period.

The data generated by the EMT buses used the UTM coordinate reference system (CRS), whereas our metadata search uses the lat, long CRS. Therefore, we applied a Storlet which converts UTM coordinates to lat, long format. This was done using an open source Java library provided by IBM. This Storlet was applied when objects were created in the cloud storage (this is called a 'PUT' Storlet). In addition, this Storlet calculated a geospatial bounding box containing all of the data points in the given objects, and stored it as object metadata. An overview of the scenario is depicted in Figure 15. This corresponds to IP2. In order for a developer to create and deploy a new Storlet, he must follow the information available in [6].

Figure 14: Overview of EMT available data characteristics

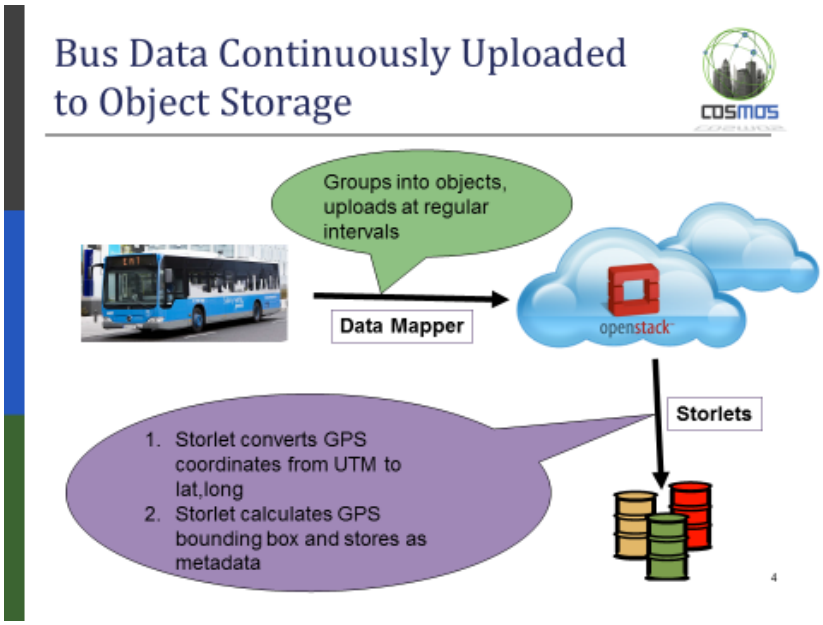


Figure 15: Overview of Metadata and Storlets Scenario

The resulting data and metadata can be viewed using the Postman REST client as shown below (Figure 16 and Figure 17). The data includes a line for each data point. The metadata includes certain system metadata fields as well as user defined metadata having the prefix X-Object-Meta (a Swift convention). The fields generated by the Storlet are X-Object-Meta-Line-Number-I (the bus line number), X-Object-Meta-Top-Left-G (top left geo-point), and X-Object-Meta-Bottom-Right-G (bottom right geo-point). Note that the suffix of the metadata field name indicates its type e.g. I for integer and G for geo-point. This is important for ensuring that the metadata is indexed and searched correctly.

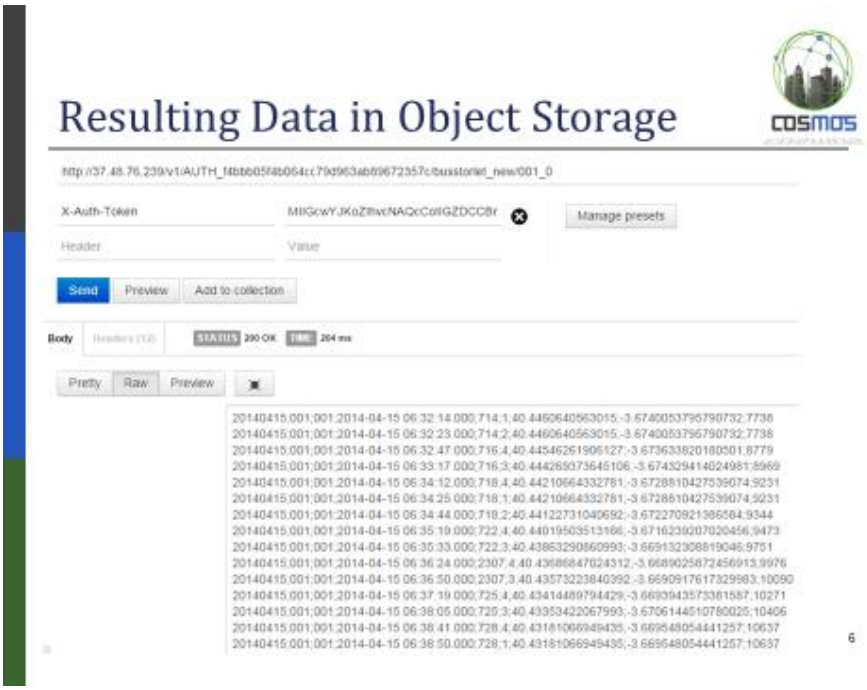


Figure 16: Data Format in Object Storage

In order to demonstrate how Storlets and metadata search work together, we developed a web interface which allows posing geo-spatial metadata search queries by drawing a bounding box on a map of Madrid, corresponding to IP1 (End user involvement and interface) in Figure 13. This triggers a metadata search query which searches for all data objects whose bounding boxes (in their metadata) are completely contained within the search bounding box. This interface was developed in Javascript using the open source Leaflet Javascript library. The following diagram (Figure 18) shows a metadata search bounding box drawn by the user in blue, together with all the object bounding boxes which match the given query.

Figure 17: Metadata insertion in Object Storage

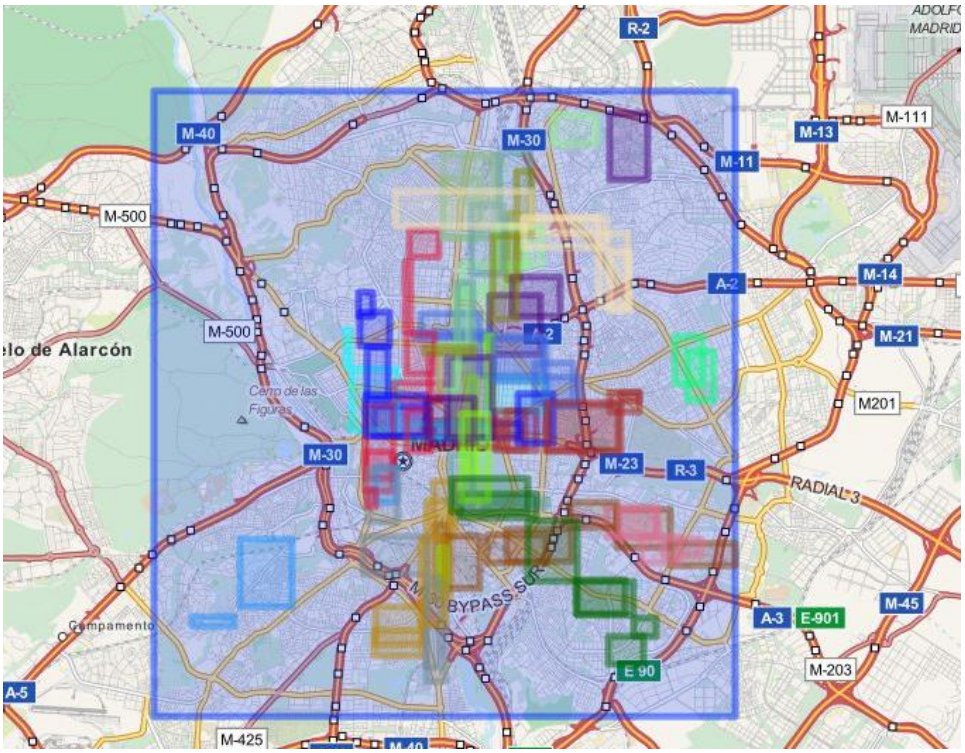


Figure 18: End user interface for the Metadata Search subsystem

Because this demonstration was driven by a GUI, we did not perform automated integration tests for the GUI part, but rather checked manually that systems were working correctly. Several bugs were discovered during this integration and successfully resolved, for example an issue arose when a Storlet increased the size of an object, and metadata search was returning only 100 search results by default and we changed this to be more configurable. The overall sequence diagram for this case appears in Figure 19 and the deployment diagram in Figure 20.

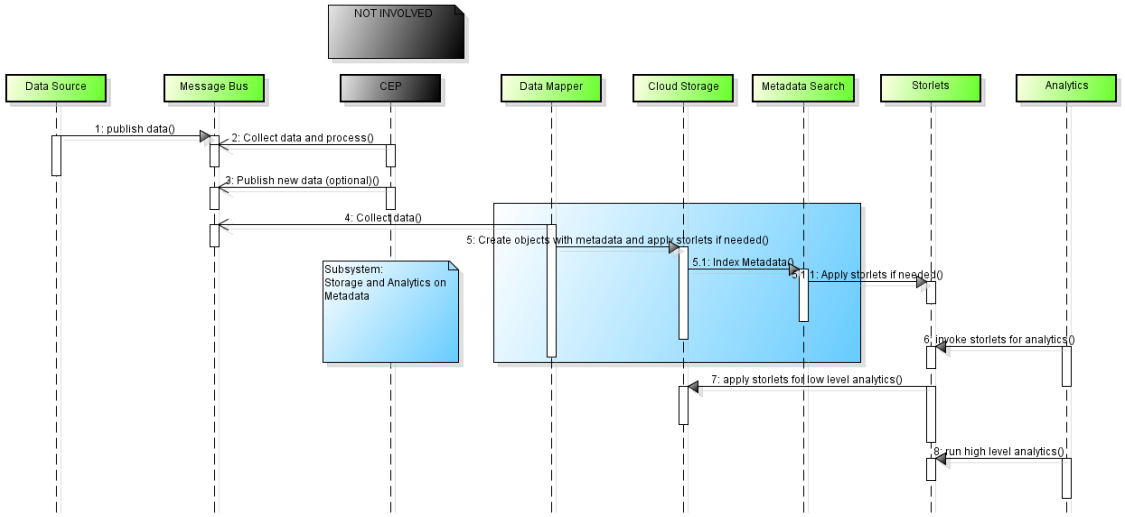


Figure 19: Sequence Diagram for Storage and Analytics on Metadata Subsystem

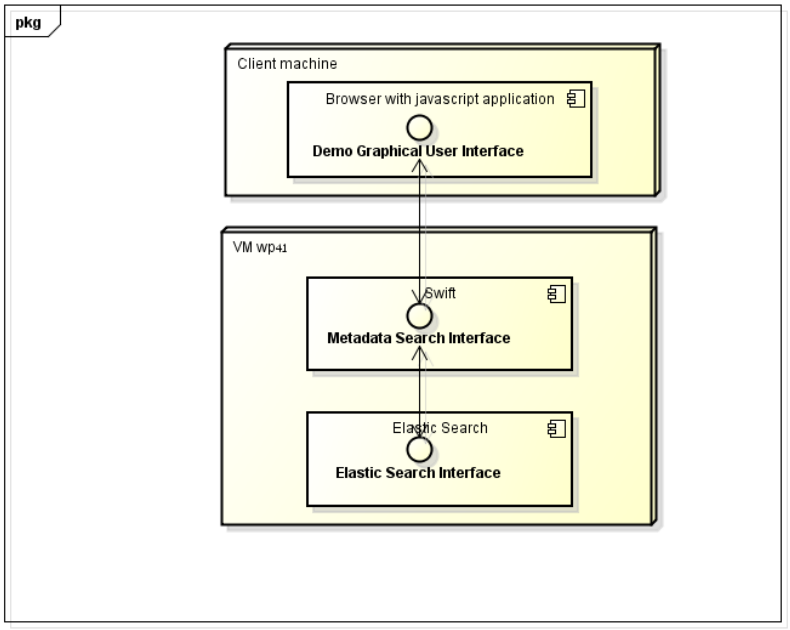


Figure 20: Deploymnet Diagram for the Metadata Search

2.2.1.3 Security, Privacy and Storage

We presented the integrated usage of the COSMOS security components together with the Storlets mechanism in the "blurred faces" demonstrator. The aim was to show how a "security flow" looks like for COSMOS and how security can integrate with the entire platform – in this particular case with the object storage. The test involved the Hardware Security Board and a web front-end which served as a GUI, displaying the results of the applied Storlet. The subsystem involved, as indicated in D7.6.1, appears in Figure 21 and consists of 3 IPs.

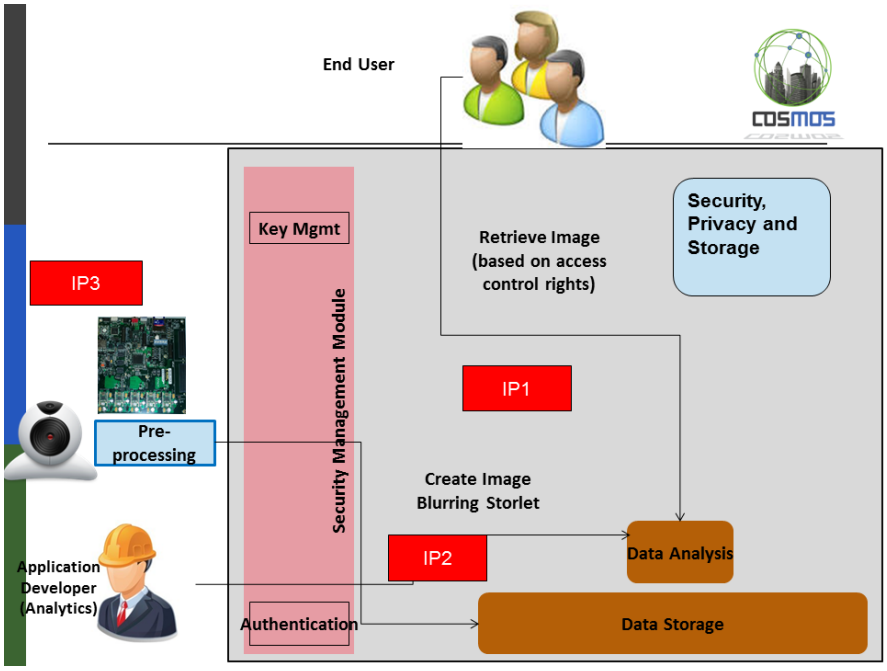


Figure 21: Security, Privacy and Storage

Scenario

The scenario is derived from the EMT use-case in which surveillance cameras take periodic pictures of the bus (Figure 22). As the pictures contain private information (e.g. faces of people) security is a must. Therefore, on the one hand side, the data transfer needs to be secured and on the other hand only authenticated users, which have the correct credentials, are allowed to view the images. The example is focused on demonstrating a "security path" between the camera and the end user. The joint demonstrator aims at showing:

- Secure key exchange: for each new hardware security board a key exchange session is used to enroll the board into COSMOS. The hardware security board uses asymmetric cryptography (ECDH) in order to securely fetch the symmetric encryption key, used to secure the data flow between the HSB and COSMOS. The HSB makes use of Keystone, which is already part of COSMOS, in order to generate and manage the keys.
- Secure data flow: each data package circulated between the HSB and COSMOS will be encrypted with the key previously exchanged. Both key exchange and data encryption/decryption make heavy usage of the hardware components within the HSB.
- Storlet based data access: Storlets are automatically generated upon each data request from COSMOS. Based on user rights, similar to the operating system approaches, COSMOS configures the Storlet with the user's rights. Therefore not all users will see the same images. The HSB pushes the pictures in full resolution to COSMOS but the Storlets only allow restricted usage of them. Users with restricted access will see the pictures with all faces blurred while unrestricted users will see the picture in full resolution. The mechanism will therefore keep, inside the cloud storage, the original pictures unchanged.

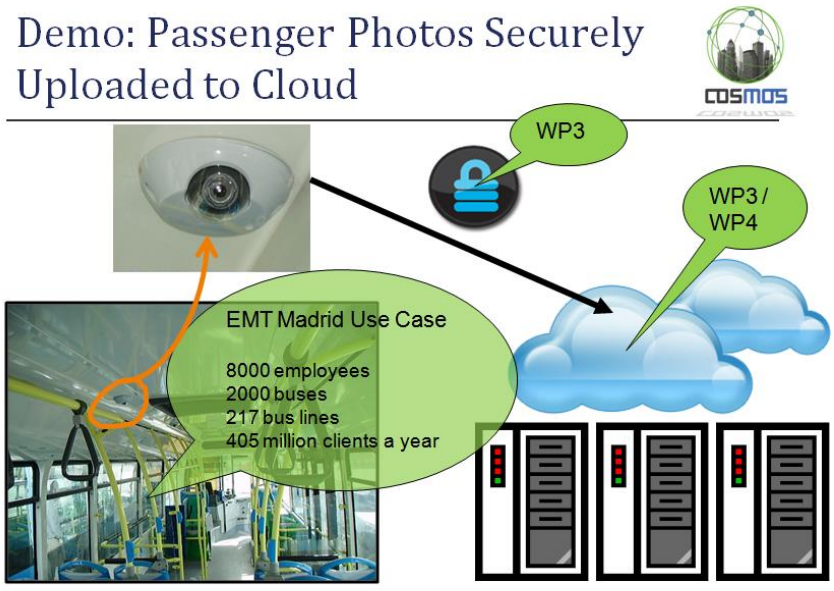


Figure 22: Security Demonstrator Show-Case

In order to simulate pictures from the EMT busses we have used a camera attached to the Hardware Security Board (Figure 23). The Hardware Security Board itself is connected to the internet and uploads the data to the cloud storage. When the data reaches the COSMOS platform it is first decrypted and then pushed to the cloud storage. The decryption task is

performed by a gateway which in this case was performed by the WP3 test-bed machine. This process corresponds to IP3, in order for the data to be adapted and reach the platform storage services. The deployment diagram for the scenario appears in Figure 24.

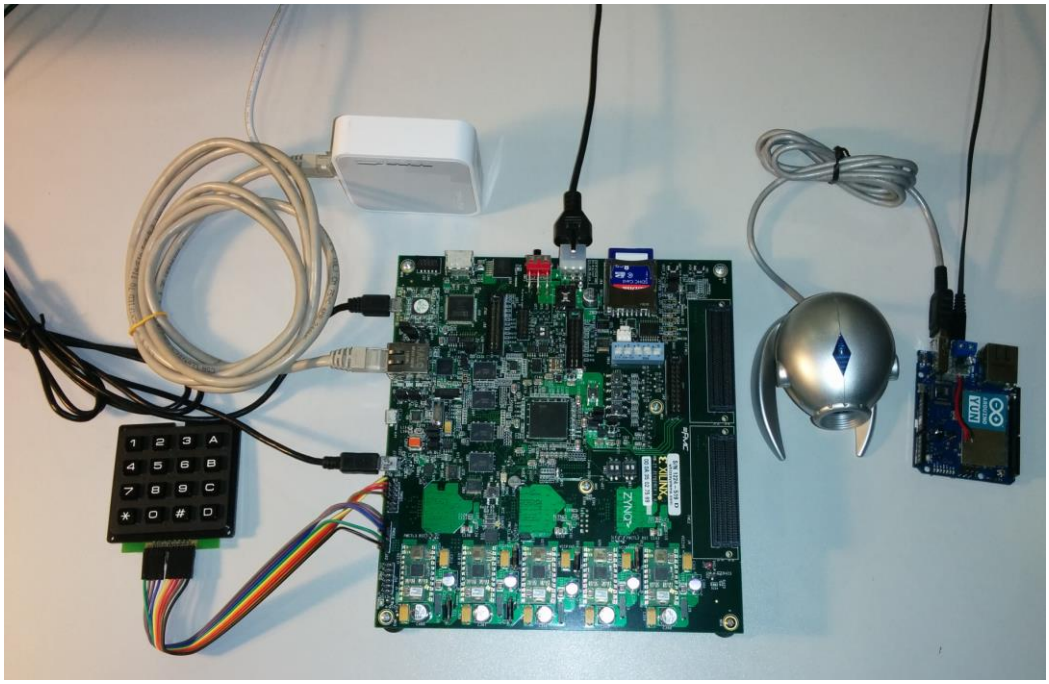


Figure 23: Hardware Security Board

On the cloud storage side, based on the users' rights, the facial blurring Storlet is invoked for non-trusted users. The generic Storlet creation process, which refers to IP2, is detailed in D4.1.1.

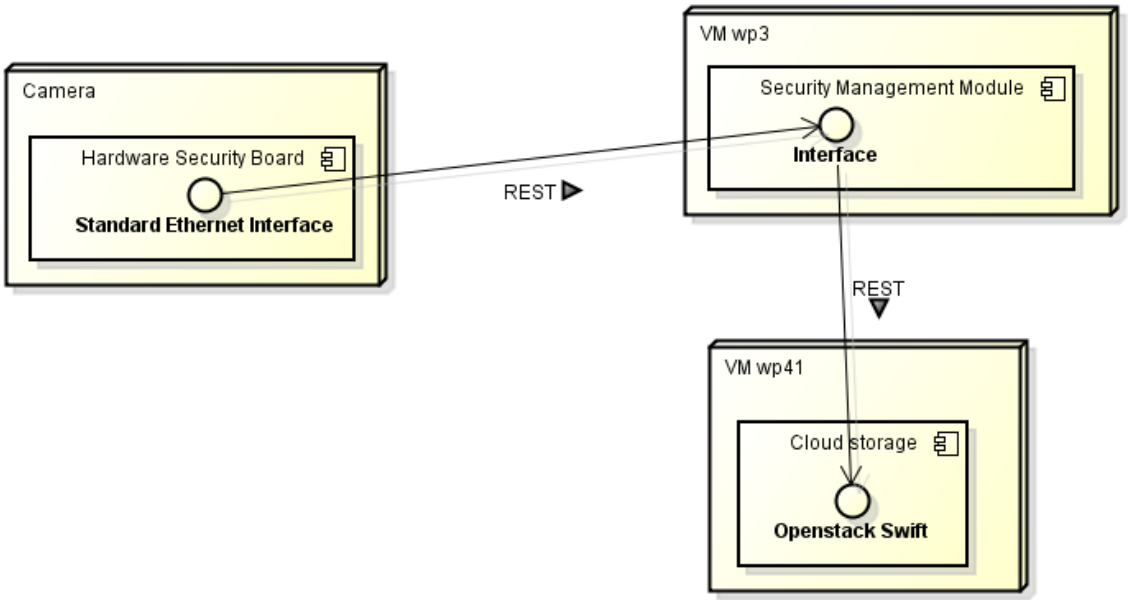


Figure 24: Security, Privacy and Storage Deployment Diagram

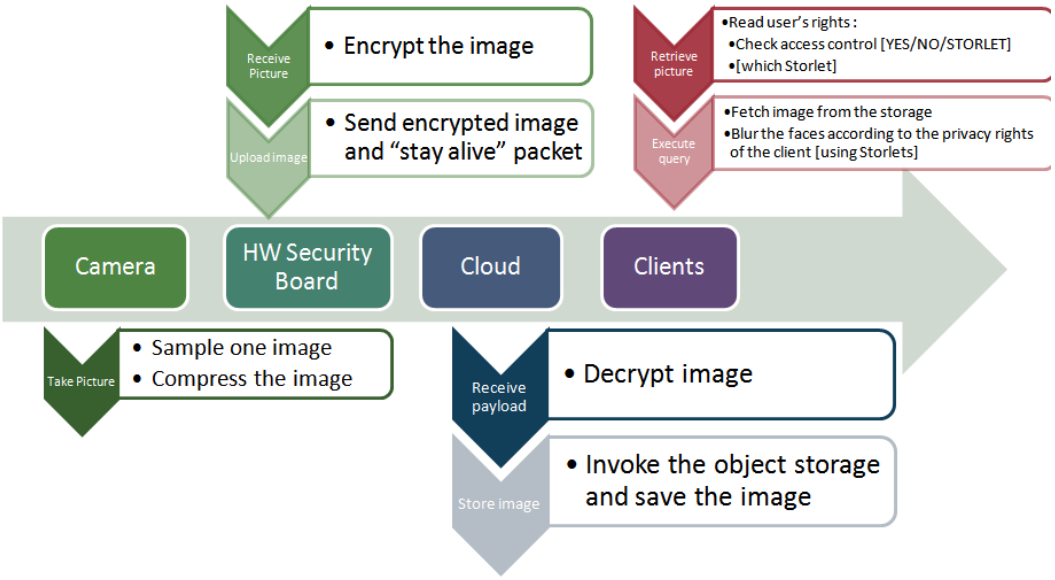


Figure 25: Demonstrator Flow

The Hardware Security Board performs the security tasks autonomously – each picture is automatically encrypted using the on-board stored, unique, encryption key. For this purpose AES128 is used as a cryptographic primitive. The flow is depicted in Figure 25, while the sequence diagram appears in Figure 26

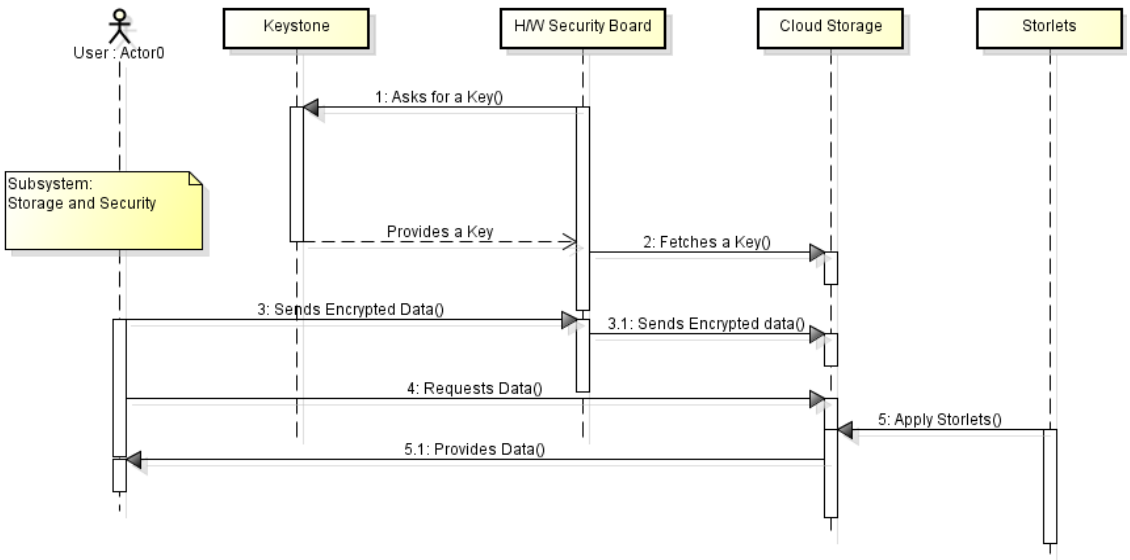


Figure 26: Sequence Diagram for Storage and Security Subsystem

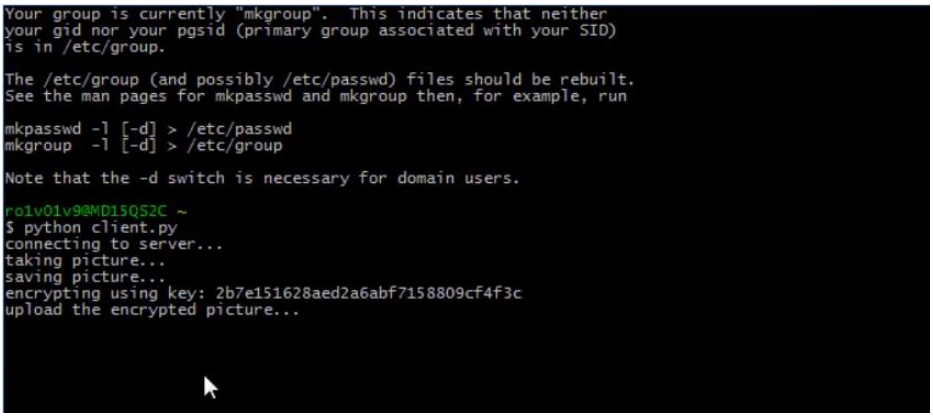


Figure 27: Client side image encryption

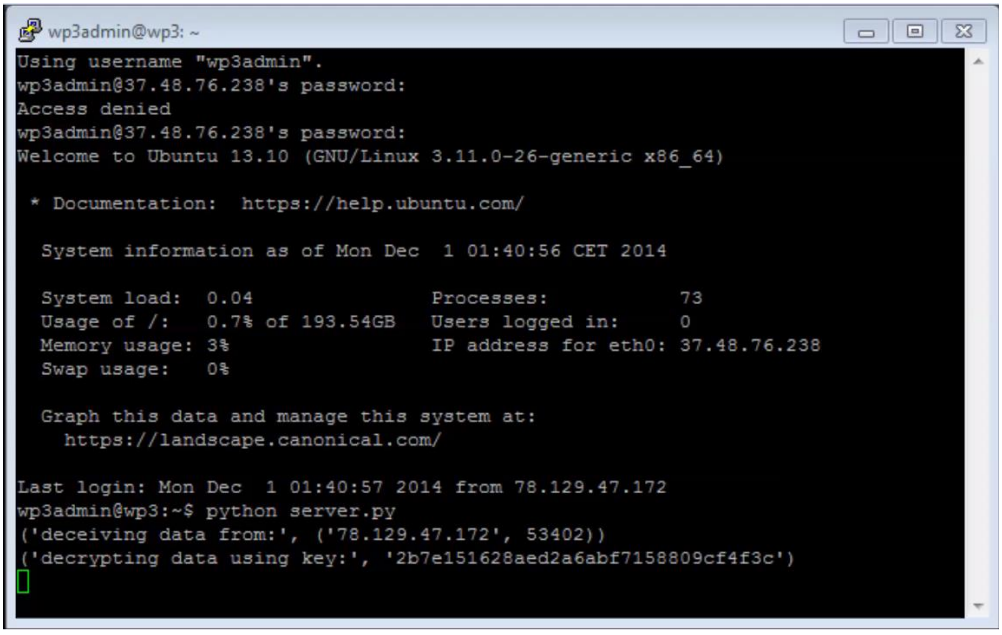


Figure 28: Platform Gateway for receiving and decrypting encrypted image

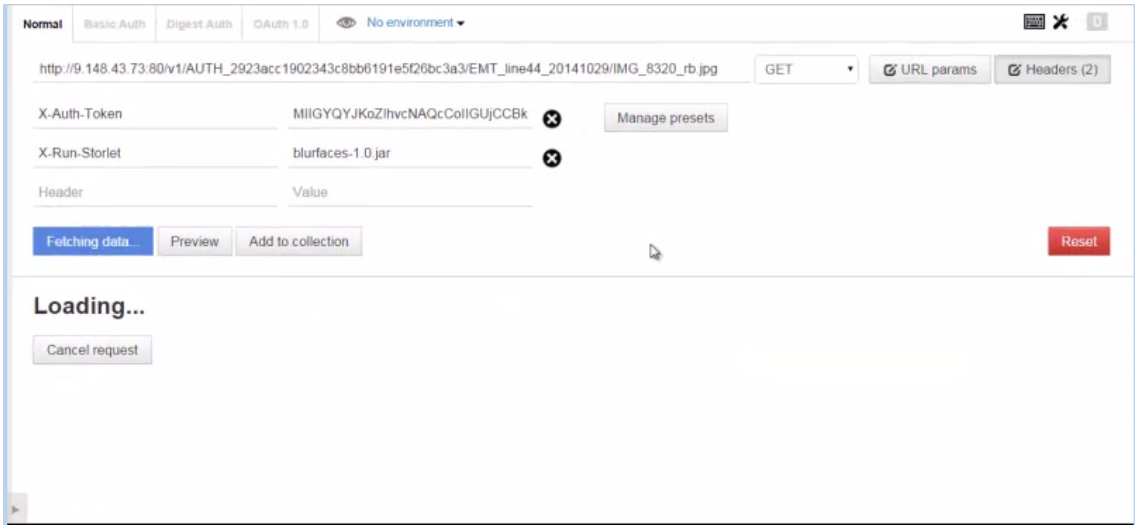


Figure 29: Facial Blurring Storlet Usage

Figure 30: Storlet Output (Blurred Image)

The Gateway logs show when and which Hardware Security Board has uploaded a picture. Each picture is decrypted, checked for validity (checksum verification) and then uploaded to the cloud storage. The uploaded pictures can be viewed using Postman REST Client. Depending on the users' rights they can:

- Not view the image;
- View the image but only blurred;
- View the image unblurred.

This corresponds to IP1, regarding end user involvement.

2.2.1.4 Modelling and Storage Analytics

Introduction

The aim of this scenario is to test and demonstrate the following:

1. Pre-processing Storlets: Machine Learning computations access data from the cloud storage after it has been pre-processed. This reduces the amount of data which need to be sent across the network and also offloads some of the computational load to the cloud storage. In order to have optimized performance, different Machine Learning algorithms involve several pre-processing tasks such as feature scaling and feature selection. We aim to perform those tasks in Storlets for fast processing when dealing with large data sets. We also aim to perform aggregation in order to reduce the total data across the network.
2. Data Analysis: Different variants of classification algorithms are implemented for pattern recognition from electricity consumption data. Classification is a supervised Machine Learning technique used widely for pattern recognition; it requires labelled data to learn and recognize the patterns. In our case, electricity consumption data with the ground truth reality acts as training data.

We integrated Apache Spark with object storage (openstack swift) for data analysis and modelling using historical data. Data Mapper is used to store data (both historical and live

data) in the form of objects. We successfully integrated Storlets for pre-processing of data which can be invoked from Apache Spark. The overall architecture of integrated components is shown in Figure 31 which is also explained in D7.6.1. We have demonstrated an Occupancy detection scenario to illustrate and validate our integration results (speicfic details for the accuracy of the model appear in Section 2.2.4.1). The application developer selects the model and define the input features and interested output. The modelling block has an access to smart storage of COSMOS in order to access historical data and train the models accordingly. The specific Storlets can be used for performing pre-processing to optimize the data analysis procedure.

Integration of New Models

The integration points for applcation developer (both Modelling and Analytics) are shown in Figure 31 as IP2. An application developer can either select the specific model or functionality from the COSMOS platform and then he will define the input features for the model. The application developer will also have to define the desired output quantity. The modelling block can be used for inferring high-level knowledge from raw data feeds or can also be used for prediction to get insight about the future depending on the application. The application developer (Modeling) can use generic Storlets from the library of Storlets provided or can write specific Storlets with the help of the application developer (Analytics).

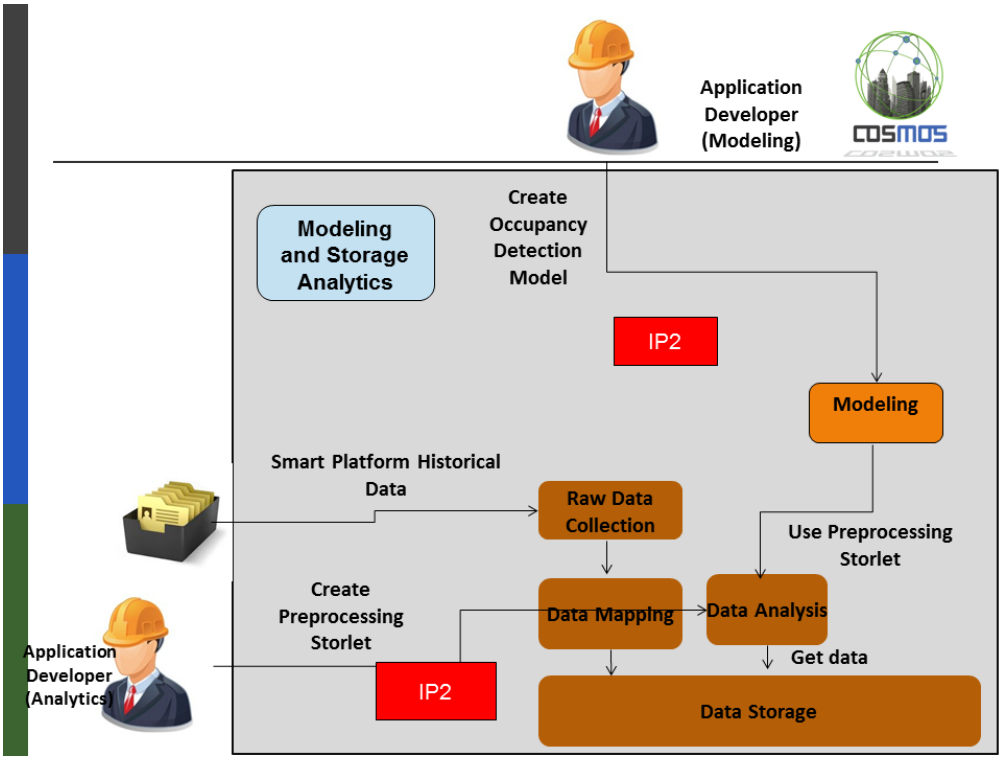


Figure 31: Storage and Modelling Subsystem

Data Flow

The overall architecture of the data flow is shown in Figure 33. Data Mapper stores the historical data (different formats of data have been tested) in openstack swift object storage in the form of objects. Object Storage is integrated with distributed Machine Learning platform (Apache Spark) for data analysis. The different functions of data analysis which were implemented in Spark is also shown in Figure 32. The Storlet creation, referring to IP2, is included as a generic process in [6].

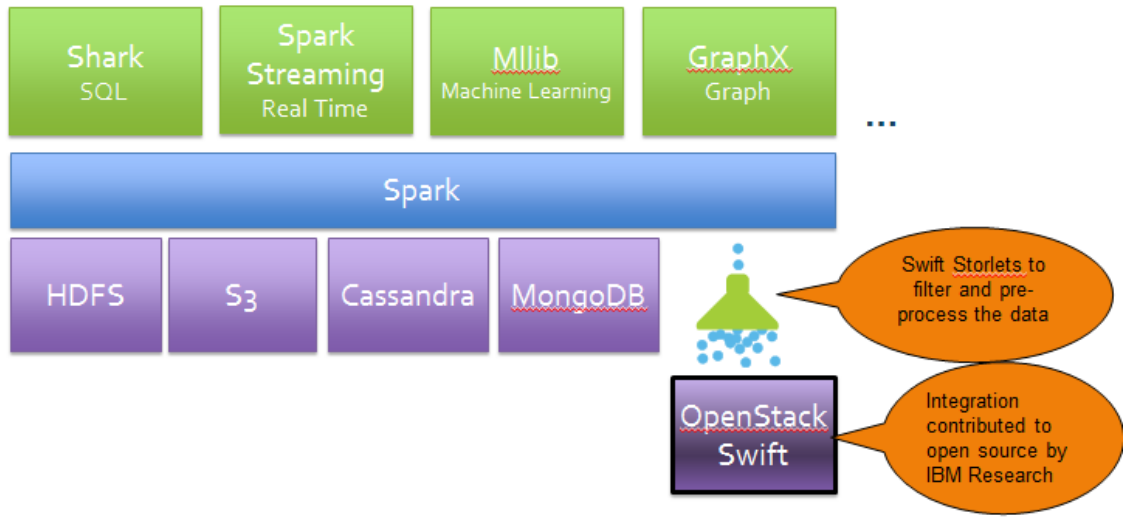


Figure 32: Overview of integration between Spark and Swift

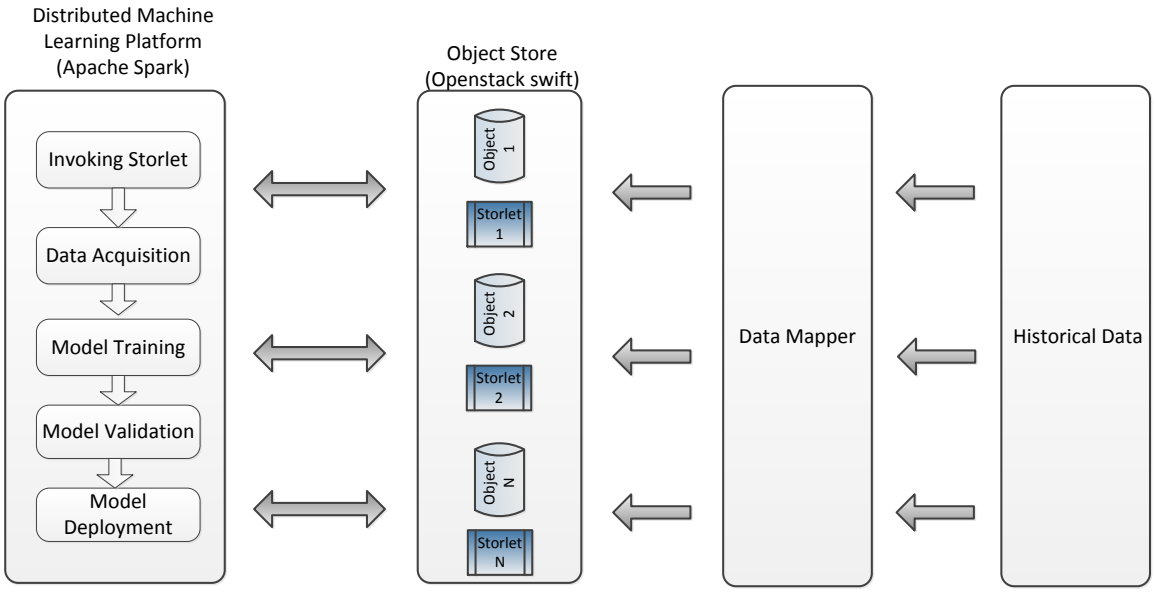


Figure 33: Data Flow across the Subsystem

The format of the data used appears in Figure 34 and contains information on the bulding workstation consumption of Surrey, including features such as power consumed, voltage, frequency, current, reactive power and phase angle. Based on these data, the model is trained (Figure 35) and validated. The deployed model then can be used for online occupancy detection (Figure 36). The sequence diagram of the subsystem appears in Figure 37 while the deployment diagram in Figure 38.

```
node-id,time_stamp,PIR,mic,temp,light,watts,frequency,RMS_voltage,RMS_current,reactive_power,phase_angle
108,"2014-06-18 00:00:15",2,6,1230,19,1.24,50.0,238.9,0.1,147507.04,273.0
108,"2014-06-18 00:00:25",0,15,1229,3,1.24,50.1,239.1,0.1,147507.04,273.0
108,"2014-06-18 00:00:35",0,13,1224,14,1.34,50.1,239.1,0.1,147506.94,273.0
108,"2014-06-18 00:00:45",3,2,1223,11,1.34,50.0,239.1,0.1,147507.04,273.0
108,"2014-06-18 00:00:55",0,0,1229,10,1.24,50.0,238.7,0.1,147506.94,273.0
108,"2014-06-18 00:01:05",1,13,1220,14,1.13,50.0,238.5,0.1,147506.94,273.0
108,"2014-06-18 00:01:15",2,6,1225,10,1.03,50.0,238.5,0.1,147506.94,273.0
108,"2014-06-18 00:01:25",0,0,1220,19,1.13,50.0,238.5,0.1,147506.83,273.0
108,"2014-06-18 00:01:35",1,5,1228,8,1.24,50.0,238.4,0.1,147506.83,273.0
108,"2014-06-18 00:01:45",2,2,1221,13,1.34,50.0,238.6,0.1,147506.94,273.0
108,"2014-06-18 00:01:55",1,2,1218,8,1.24,50.0,238.7,0.1,147506.94,273.0
108,"2014-06-18 00:02:05",1,2,1229,10,1.24,50.0,238.6,0.1,147506.83,273.0
108,"2014-06-18 00:02:15",1,5,1223,9,1.13,50.0,238.7,0.1,147506.94,273.0
108,"2014-06-18 00:02:25",2,18,1222,13,1.03,50.0,238.8,0.1,147506.94,273.0
108,"2014-06-18 00:02:35",1,16,1235,15,1.13,50.0,238.4,0.1,147506.83,273.0
108,"2014-06-18 00:02:45",1,1,1216,11,1.24,50.0,238.4,0.1,147506.83,273.0
108,"2014-06-18 00:02:55",1,7,1223,11,1.34,50.0,238.6,0.1,147506.94,273.0
108,"2014-06-18 00:03:05",2,1,1224,11,1.24,50.0,238.7,0.1,147506.94,273.0
108,"2014-06-18 00:03:15",1,6,1226,13,1.24,50.0,238.7,0.1,147506.94,273.0
108,"2014-06-18 00:03:25",1,1,1217,11,1.03,50.0,238.7,0.1,147506.94,273.0
108,"2014-06-18 00:03:35",2,11,1222,0,1.13,50.0,238.7,0.1,147506.94,273.0
108,"2014-06-18 00:03:45",1,9,1231,11,1.13,50.0,238.7,0.1,147506.94,273.0
108,"2014-06-18 00:03:55",0,1,1227,11,1.24,50.0,238.9,0.1,147506.94,273.0
108,"2014-06-18 00:04:05",1,5,1214,10,1.24,50.0,238.8,0.1,147506.94,273.0
108,"2014-06-18 00:04:15",1,2,1221,14,1.34,50.0,238.9,0.1,147506.94,273.0
108,"2014-06-18 00:04:25",1,5,1227,15,1.24,50.0,238.9,0.1,147506.94,273.0
108,"2014-06-18 00:04:35",0,4,1218,9,1.13,50.0,238.9,0.1,147506.94,273.0
108,"2014-06-18 00:04:45",1,17,1229,12,1.13,50.0,238.9,0.1,147506.94,273.0
108,"2014-06-18 00:04:55",1,1,1222,12,1.13,50.0,238.9,0.1,147507.04,273.0
108,"2014-06-18 00:05:05",2,15,1228,13,1.24,50.0,238.9,0.1,147506.94,273.0
108,"2014-06-18 00:05:15",1,15,1227,14,1.24,50.0,239.0,0.1,147506.94,273.0
108,"2014-06-18 00:05:25",2,13,1218,7,1.34,50.0,239.0,0.1,147506.94,273.0
```

Figure 34: Input data format for Modelling case

```
from pyspark.mllib.classification import LogisticRegressionWithSGD
from pyspark.mllib.regression import LabeledPoint
from numpy import array

def parsePoint(line):
    values = [float(x) for x in line.split(' ')]
    return LabeledPoint(values[0], values[1:])

# Load the data
print('invoking storlet.....')
data = sc.textFile("swift://SurreyProcessedData-x-run-aggregationstorlet.COSMOS/*")
print ("no of entries in data is "), data.count()

# Parse the data and build the model
parsedData = data.map(parsePoint)
model = LogisticRegressionWithSGD.train(parsedData)

# Evaluate the model on training data
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
print("Validation Error = " + str(trainErr*100)+" %")
```

Figure 35: Model Training using Spark MLlib, Storlets and Swift

```
wp6@wp6: ~/adnan/spark-1.1.0/bin
      /_/

Using Python version 2.7.5+ (default, Feb 27 2014 19:37:08)
SparkContext available as sc.
>>> execfile('ml_spark_storlet_online.py')
invoking storlet.....
no of entries in data is  3594
Model Training.....
Model Validating.....
Validation Error = 3.78408458542 %
Current State is detecting.....
User 1 is not at desk
User 2 is at desk
User 3 is not at desk


User 1 is not at desk
User 2 is at desk
User 3 is not at desk


User 1 is not at desk
User 2 is at desk
User 3 is not at desk
```

Figure 36: Runtime Prediction using the trained model

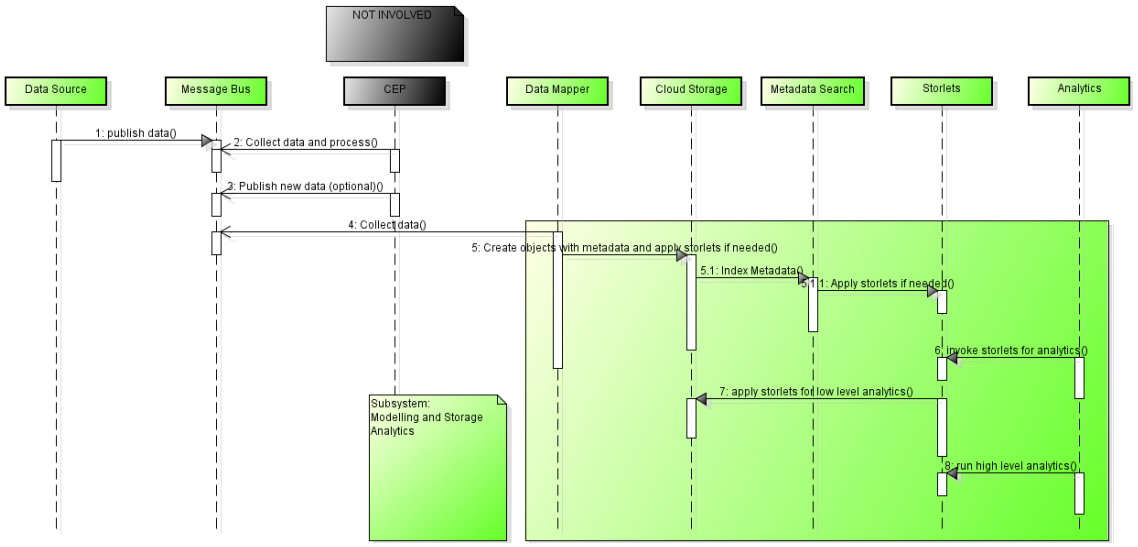


Figure 37: Sequence Diagram for Modelling and Storage Analytics Subsystem

Subsystem Test Case

The Subsystem test case, along with the covered requirements appears in the following table.

Test Case Number Version	**Data_Analysis_1**
Test Case Title	Data Analysis and Modelling for knowledge Inference
Module tested	Modelling Block along with the Data Mapper, Object Storage and Storlets
Requirements addressed	4.1, 4.2, 4.4, 4.6, 6.2, 6.6, 6.7, 6.47
Initial conditions	Access to VMs
Expected results	1) Data will be stored in the object storage in the form of objects 2) Storlets will be provoked and it will perform aggregation as a pre-processing step 3) Algorithms implemented in Apache Spark will access the pre-processed data from the object storage and Machine Learning models will be trained 4) The trained model will be validated using validation data set 5) The model will be deployed on real time data and it will detect the occupancy state of a user
Owner/Role	Application Developer
Steps	Provide the labelled data set Define input and output features Write or call the specific Storlet when running an application Run the application in VM wp61
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Deployment Diagram

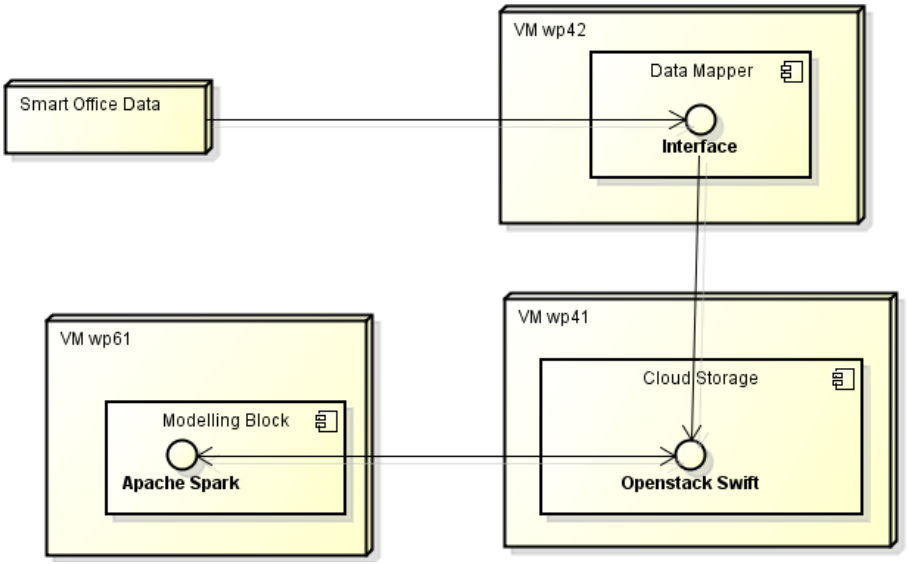


Figure 38: Modelling and Storage Analytics Deployment Diagram

2.2.2. Subgoal: Autonomous Behaviour of VEs

The subgoal includes the following componnets:

Message Bus

For the Y1 demonstration we plan to have one statically defined topic for the Camden data.

µCEP

The µCEP will draw data from the Message Bus and detect topic-appropriate events for publishing.

VE Planner

The VE Planner will have two basic functionalities. Firstly, it will use raw data to detect new Cases and store them appropriately inside the local CB. Also, depending on the scenario, the event detection will either be automated, courtesy of the µCEP, or will be manually provided, by user input (User: Actor).

Experience Sharing

Experience Sharing will be used together with the Planner, in order to find Solutions to Problems that are not available locally, by initiating communication with the remote Experience Sharing components. The remote Experience Sharing component requests a Solution from its Planner component and returns the provided answer to the origin VE.

Social Monitoring

Social Monitoring coordinates with the Planner for the evaluation of a shared Solution, whether positive or negative. The SM component calculates new metrics based on the specifics of the Experience Sharing mechanism feedback.

The subgoal is tested against two scenarios that are detailed in the following sections.

2.2.2.1 Planner with minimum integration with Platform

This scenario's overarching description is that the owner of a flat, while away from it, wants to set its internal temperature to a certain degree, before he/she arrives to it. Thus, he/she notifies the corresponding VE-flat by using a COSMOS-enabled application and provides as input:

i. the desired temperature
ii. the time needed before he/she arrives to the flat.

This is a problem regarding energy management where the desired temperature must be achieved right before the owner arrives to the flat. With that in mind, the following tests will establish the autonomous behaviour of the VEs. They will showcase their reaction to problems by using their own experience or this of others. VE2VE communication and social interaction will be demonstrated, as well as knowledge flow through experience sharing.

Initially we aim to present a more minimal version of the expected results in autonomous VE behaviour by demonstrating that the entire process is not reliant on the platform of COSMOS for initiation. By allowing the VE to react to user generated events, we present decentralised VE capabilities for managing their functions. In this scenario (Figure 39), we have removed the connectivity with any platform specific component, like the Message Bus or the µCEP and after receiving data for Case creation (simulating local observations), the event is triggered by user input, which correspons to the IP1 point and is performed through the GUI presented in Figure 40.

After that, the VE Planner proceeds in trying to locate a local Solution to the Problem, or to initiate Experience Sharing in order to retrieve a suitable Solution for actuation from a remote Friend VE. Regarding the second integration point of IP2 categorization, it describes the process by which a new CBR structure can be defined. One of the COSMOS project's targets is to allow application developers to enrich the CB used in VEs by adding their own Cases and therefore increasing the variety of contained knowledge in the system. In case the application developer, uses user input data as a way of signifying the existence of a Problem, the assumption is made that the application logic running on the client side will structure the relevant Case in terms of content and if storage or retrieval is needed then, the application will make use of the VEs Planner capabilities to do so. This can be achieved by using the Planner Java functions, which allow the updating of the CB with relevant properties and individuals.

The Knowledge Base structure appears in Figure 41 while the Case Base and Friend list structures appear in Figure 42. The scenario execution could go through several if not all stages described in Figure 43, depending on the End-User input. If the VE Planner locates the answer locally, then it will be returned directly to the user, without the need for Experience Sharing.

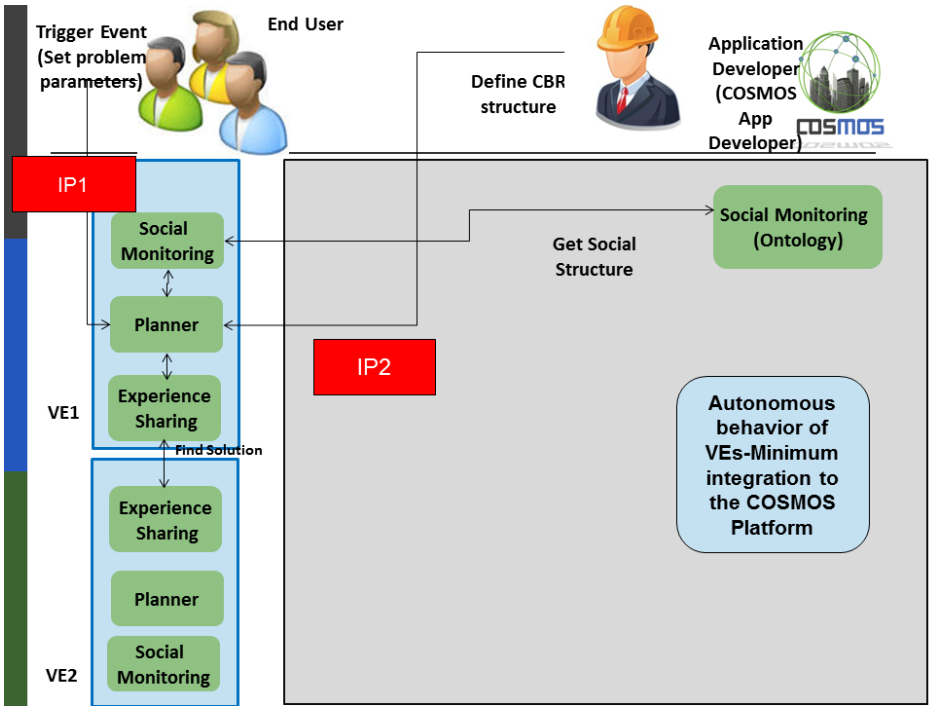


Figure 39: Autonomous Behavior of VEs with minimum platform integration subsystem

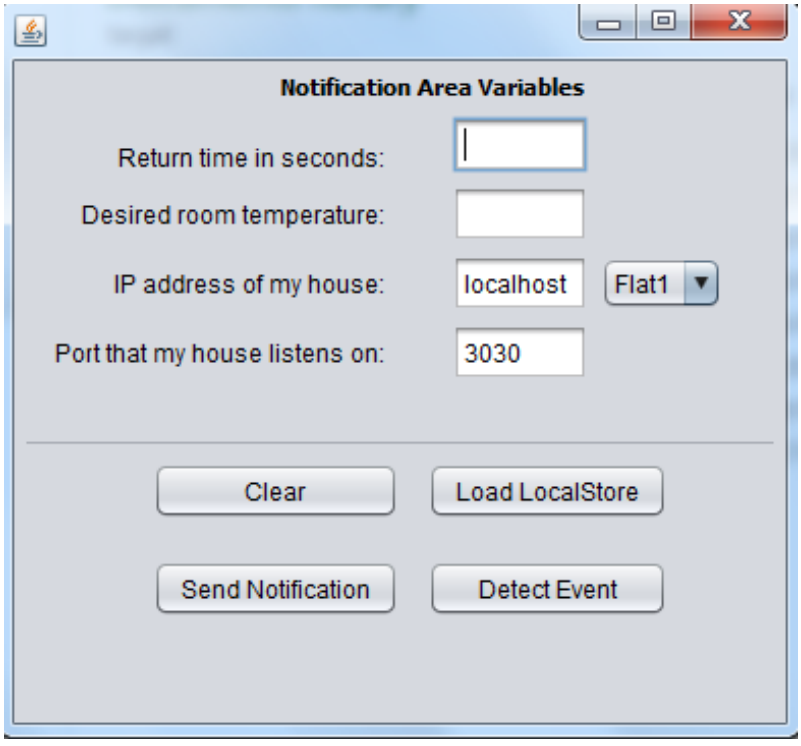


Figure 40: Autonomous Behavior of VEs application GUI

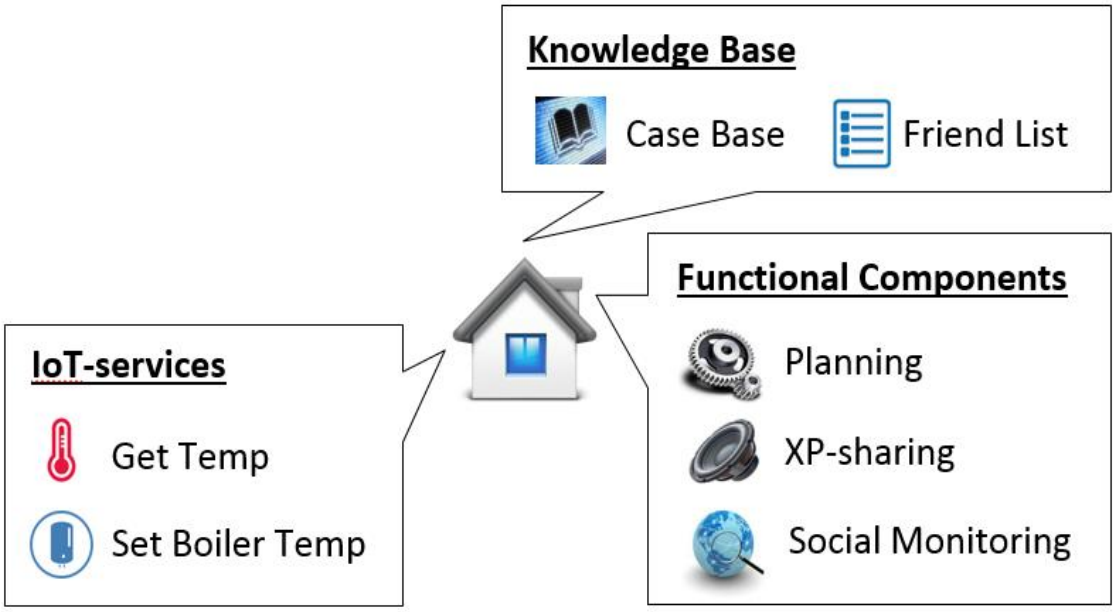


Figure 41: Components of the main flat-VE

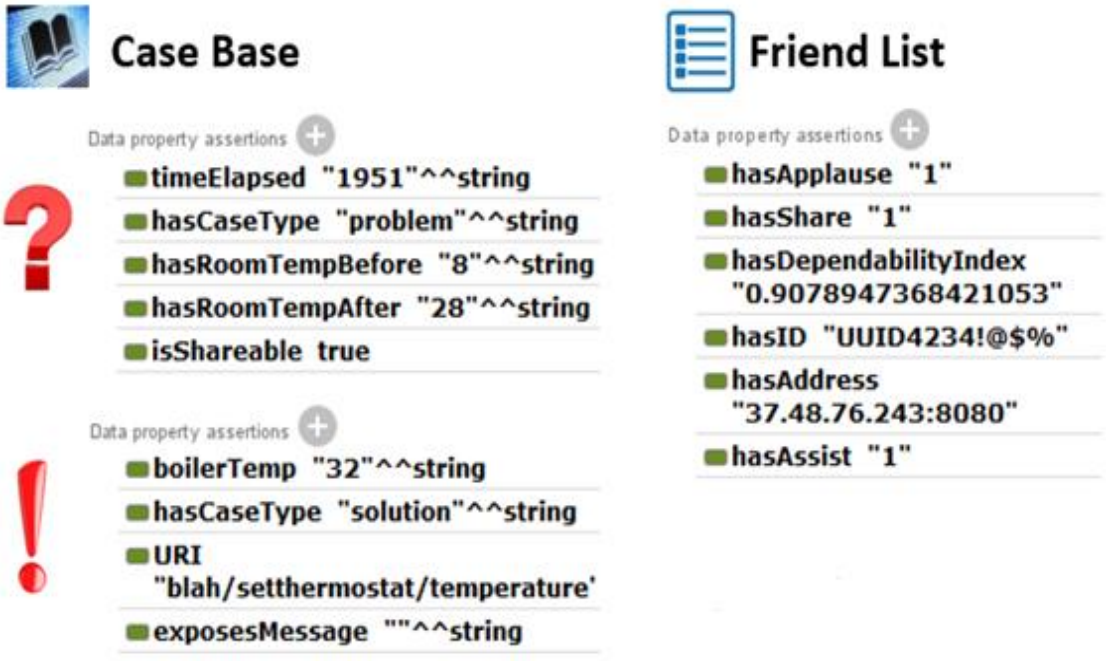


Figure 42: Case Base and Friend List examples

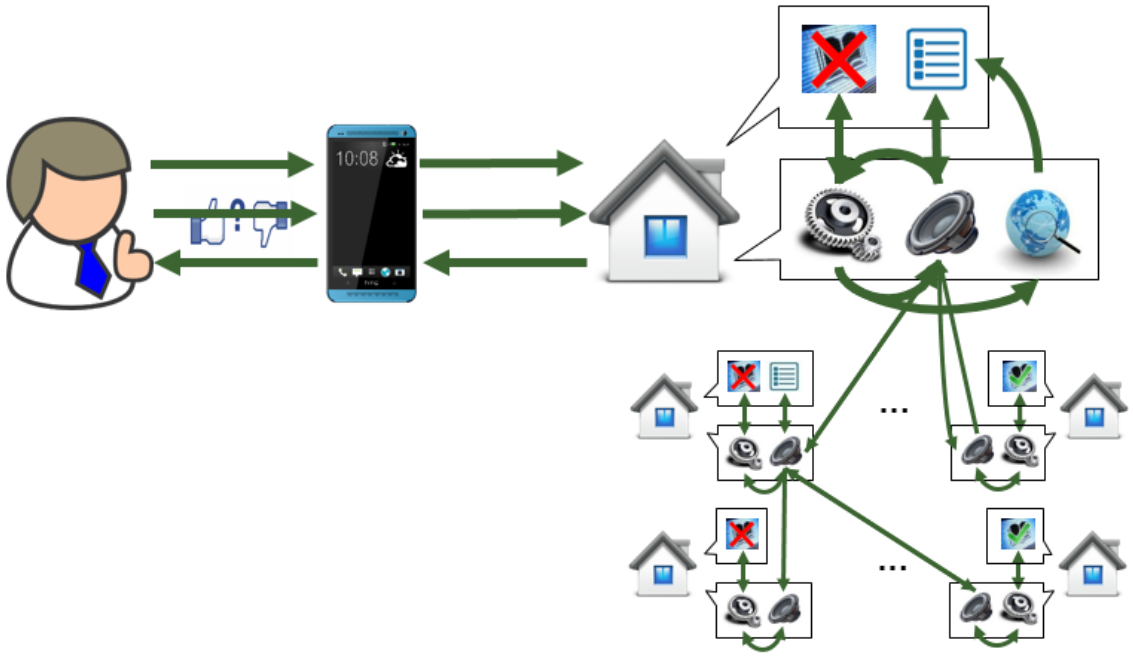


Figure 43: Visual representation of all scenario possibilities

If the user request cannot be resolved locally, then the flat will request help from its Friends (Followees). As such, the Friends that are communicated will search their own Case Bases for a suitable answer. If such an answer is located, then it will be send back to the original VE-flat, where the user gives feedback regarding the answer that was proposed. The "Shares" of the corresponding Friend (Figure 42) are increased by one. Depending on whether the user accepts the Solution, the new Case is stored and the "Applauses" are increased as well.

If an answer is not located on the immediate Friends, then they will recursively request the help of their own Friends. If the chosen answer belongs to an indirect Friend, then the "Assists" of the Friend that functioned as a Mediator are increased by one.

The test case for this scenario appears in Table 2. In future iterations, the Planner will have the capability to receive mappings of Cases, in order for the CB to acquire the necessary individual Case structure for storage.

Table 2: Planner with minimum integration to the COSMOS Platform Test Case

Test Case Number Version	**AB_1**
Test Case Title	Reaction to User Generated Event
Modules tested	Planner, Experience Sharing, Social Monitoring
Requirements addressed	UNI. 704, 706, 708, 715, 719, 5.9, UNI.251, 5.10, 5.11, 5.12, 5.29, UNI.010, 5.31, 6.8, 6.9, 6.18
Initial conditions	Have a CB and appropriate Friend lists inside the test bed Have the VE and application simulation jars inside the test bed
Expected results	The answer to the notification of the User Generated Event in the GUI A prompt for marking the helpfulness of the returned Solution, if Experience Sharing was used Changes in the CB and Friend list, depending on circumstances Command Line or Terminal Log with extra input, eg. requests

	made, similarities retrieved and queries used
Owner/Role	End User, Application Developer
Steps	Place the CB in the localstore folder of the home directory Open command line or terminal Navigate to the jar folder Execute “java –jar DemoProjectMaven-1.0-SNAPSHOT.jar” Keep the terminal or cmd open to view log info and then locate the GUI Enter a set of temperature and time Press “Send Notification” Grade the Solution as helpful or unhelpful if needed
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

2.2.2.2 Planner full integration with Platform

In this specific scenario our aim is to demonstrate that there is a clear chain of integration between the various developed components, especially in regards to “event detection” as a trigger from the Platform for further Planner actions. The expected chain of observed actions is the Data Source providing a stream of data in a topic of the Message Bus, the VE Planner to use this stream in order to extract Cases and at a later time the µCEP to begin using the stream as a means to detect events based on its specific rules. After the publishing of an event the VE Planner will initiate action on the specified Problem by using all means possible in detecting a Solution, which were presented in the previous section. The subsystem coming from D7.6.1 appears in Figure 44.

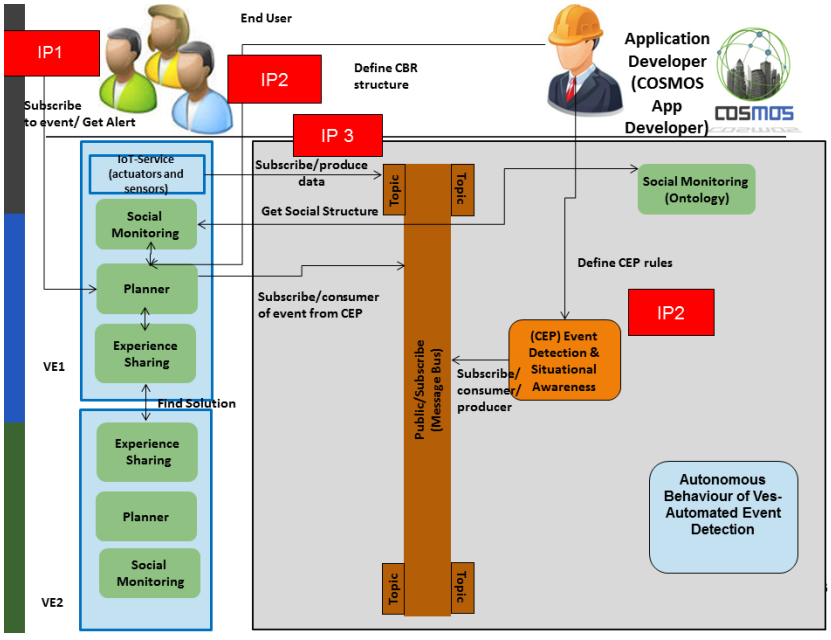


Figure 44: Autonomous Behavior of VEs- Automated Event Detection and Incorporation of COSMOS Platform

Many applications have to run continuously and without the intervention of the End-Users. Based on certain rules, the COSMOS µCEP engine is able to detect such an event. The VE-flat is notified through the COSMOS Message Bus and searches for an answer to this event. Following a similar approach with the previous scenario, this time the focus is on linking the VEs to the COSMOS Message Bus. The End-User does not have to intervene and greater autonomy is achieved.

Figure 45 shows the main components involved in this scenario:

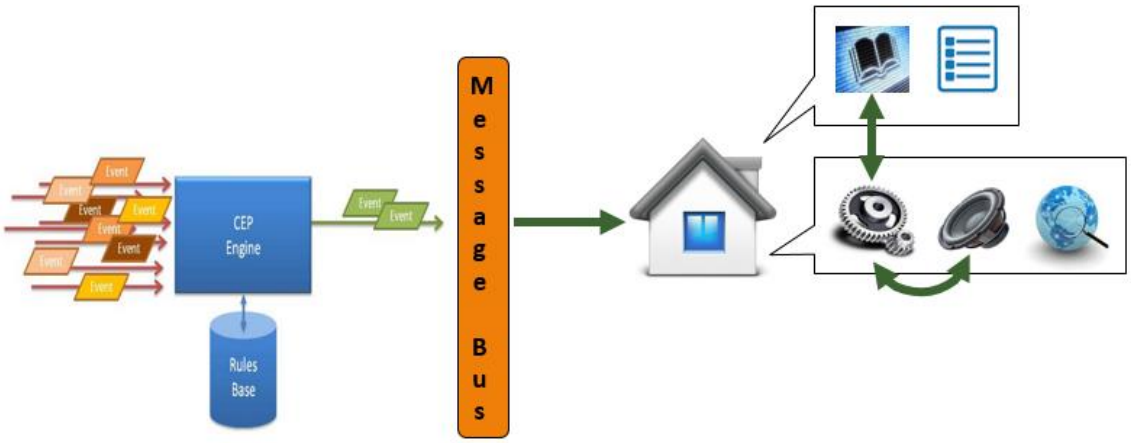


Figure 45: Visual representation of the platform integration scenario

The VE-flat is subscribed and listens to specific Topics of the Message Bus (based on live data) that are linked with specific types of events. Depending on the new events detected, Experience Sharing may be triggered or not. In case there is need to contact the Friends, the scenario proceeds as described in Figure 43, minus the evaluation of the answer.

Three integration points can be identified from Figure 44.

IP1 is once again the integration point which requires user input in the sense of demanding the use of the GUI for the VE to begin to listen on the Message Bus. This happens by pressing Detect Event in the Figure 40 GUI.

IP2, like the previous scenario, involves the creation of CBR structure and is mentioned previously, but now also refers to the creation of new CEP rules. If the application developer wishes to create an application which is not dependent on user input, then the starting point of the application will be a complex event detected by the platform's µCEP. In that case it is necessary to provide a way for the developer to inject detection rules, in the µCEP. Right now, the only way to add new rules to µCEP is by updating the rule's file which is coded in DOLCE.

Figure 46 shows an example of a DOLCE configuration file.

```
external int TEMP_ALERT = 39;
external duration T_WINDOW = 10 seconds;

/*
    System Test 1:
    Event filtration by constant value (sensor source)
*/

event temperature
{
    use
    {
        int sensor_id,
        int value
    };

    accept { sensor_id == 253 };
}


complex freezeThreshold
{
    detect temperature
    where value < 4;
}
```

Figure 46: Sample DOLCE configuration file of freezing event detection

In future iterations it may be feasible to create a UI where the developer will have greater ease in accessing and modifying rules of detection. Also in order to run the µCEP, the VM must be accessed directly and the µCEP script should be run manually. The tester has the ability to manually publish an event for detection by using the command **netcat -q 0 -u [ip] [port] < threshold.evt** where threshold.evt is a file containing a stream of data in the form of Figure 47.

```
1 temperature int sensor_id 253 int value 3
```

Figure 47: Example of a data stream as input to the µCEP

IP3 is a point of integration for the VE developers. There is the need for connecting the VE code, with the Message Bus, so that data can be published to the MB. At the moment, VEs only listen to the MB. Figure 48 is a code block of the functions each VE developer must actually implement with RabbitMQ, as this is the tool used for MB communication. The test case for this scenario appears in Table 3.

```
connection = factory.newConnection();
channel = connection.createChannel();
String exchangeName = "Situations";
String queueName = "DetectedEvents"; //a queue is a buffer that stores me
channel.exchangeDeclare(exchangeName, "topic"); //there are a few exchang
channel.queueDeclare(queueName, false, false, true, null);
String bindingKey = "#";
channel.queueBind(queueName, exchangeName, bindingKey);
//System.out.println("Waiting for messages to be published..." + "\n");
QueueingConsumer consumer = new QueueingConsumer(channel);
channel.basicConsume(queueName, true, consumer);
```

Figure 48: Example of code for MB subscription for listening.

Table 3: Planner full integration with COSMOS Platform

Test Case Number Version	**AB_2**
Test Case Title	Reaction to µCEP generated event
Module tested	µCEP, Message Bus, Planner, Experience Sharing, Social Monitoring
Requirements addressed	Same as **AB_1** 5.20
Initial conditions	Have a CB and appropriate Friend lists inside the test bed Have the VE and application simulation jars inside the test bed A functional µCEP in the test bed A functional Message Bus
Expected results	The answer to the µCEP event which will appear in the terminal/cmd A prompt for marking the helpfulness of the returned Solution, if Experience Sharing was used Changes in the CB and Friend list, depending on circumstances Command Line or Terminal Log with extra input, eg. requests made, similarities retrieved and queries used
Owner/Role	End User, Application Developer, VE developer
Steps	Place the CB in the localstore folder of the home directory Open command line or terminal Navigate to the jar folder Execute “java –jar DemoProjectMaven-1.0-SNAPSHOT.jar” Keep the terminal or cmd open to view log info and then locate the GUI Press button “Detect Event” The terminal/cmd will show whether the listening Planner detects an event send by the µCEP Grade the Solution as helpful or unhelpful if needed
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Following the tests of the scenarios, the deployment diagram for the testbed is produced in Figure 49. Finally the sequence diagram of Figure 50, describes the steps taken in the scenarios from an architectural point of view. The alt described as [via COSMOS platform] corresponds to the scenario described in this section and the alt described as [Decentralised] corresponds to the scenario described in section 2.2.2.1.

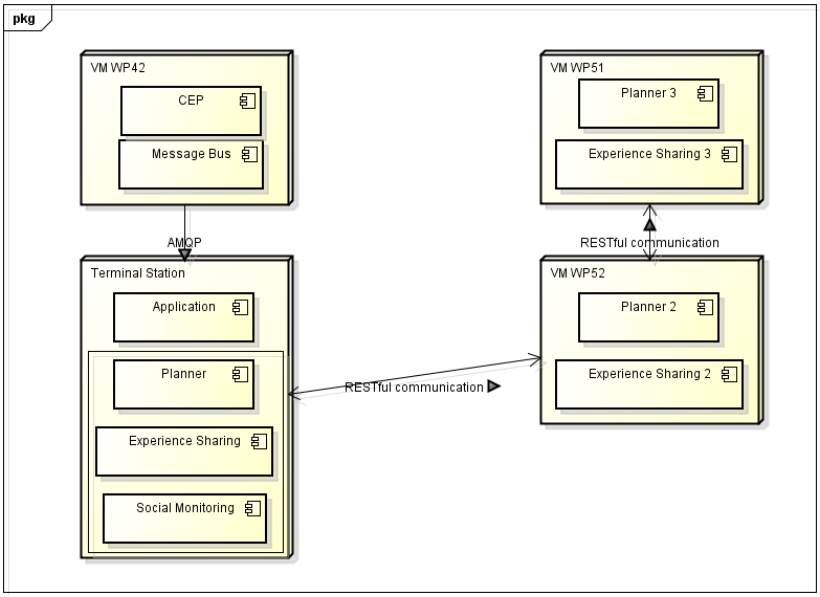


Figure 49: Autonomous Behaviour of VEs Deployment Diagram

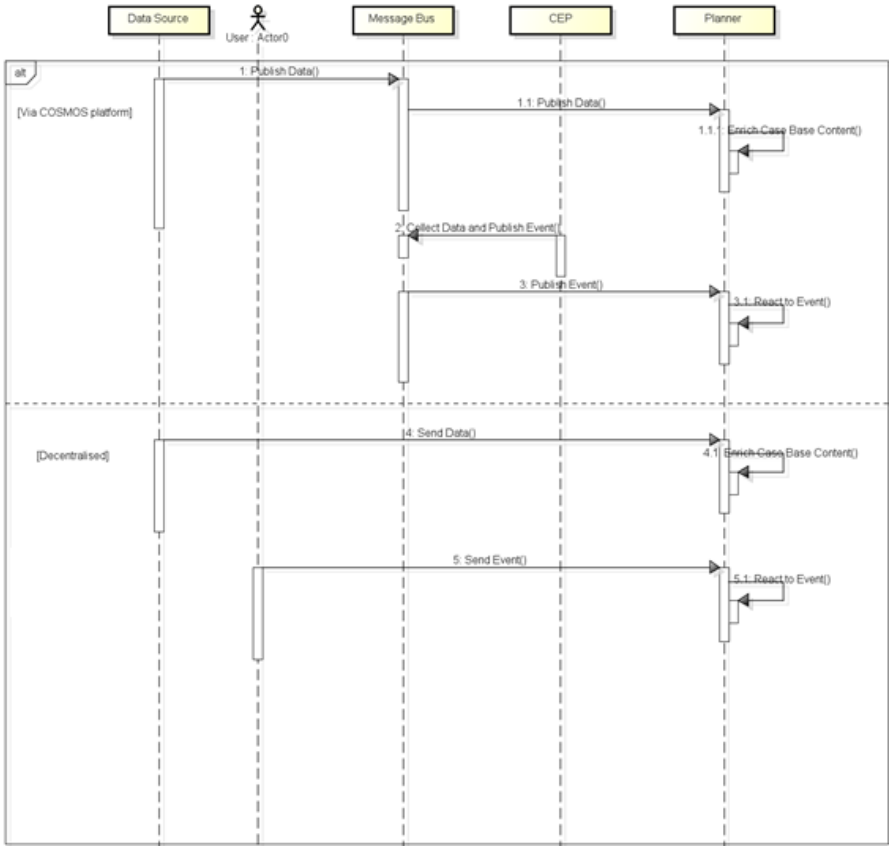


Figure 50: Autonomous Behaviour of VEs System Sequence Diagram

Enrich Case Base Content method in Figure 50 is detailed in Figure 51:

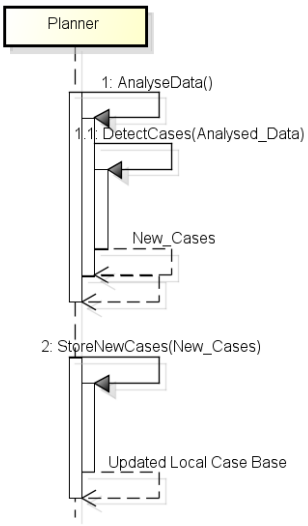


Figure 51: Enrich Case Base Content Sequence Diagram

React To Event in Figure 50 is detailed in Figure 52:

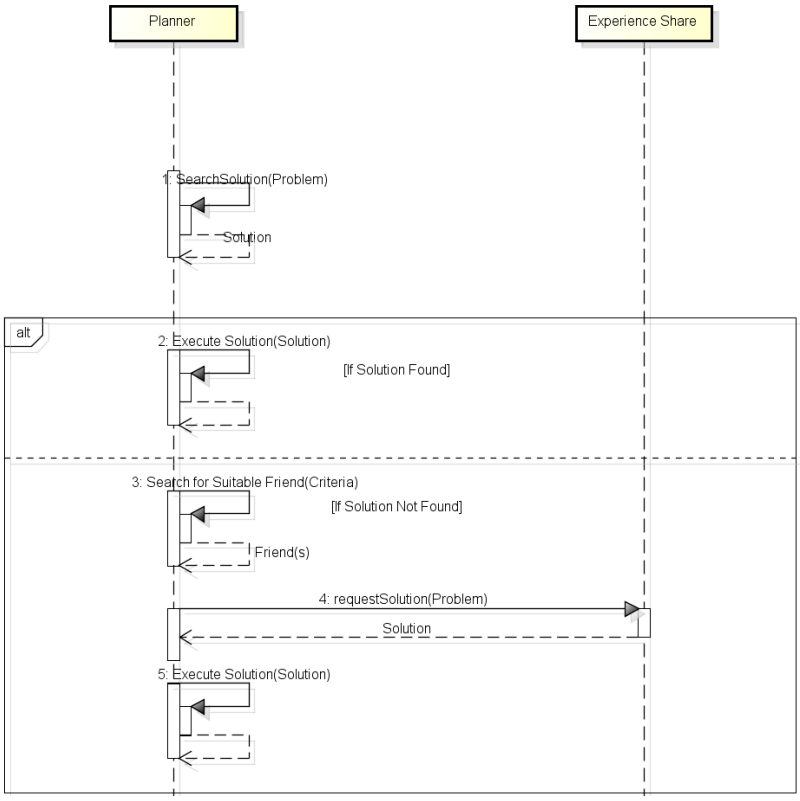


Figure 52: React to Event Sequence Diagram

RequestSolution(Problem) in Figure 52 is detailed in Figure 53:

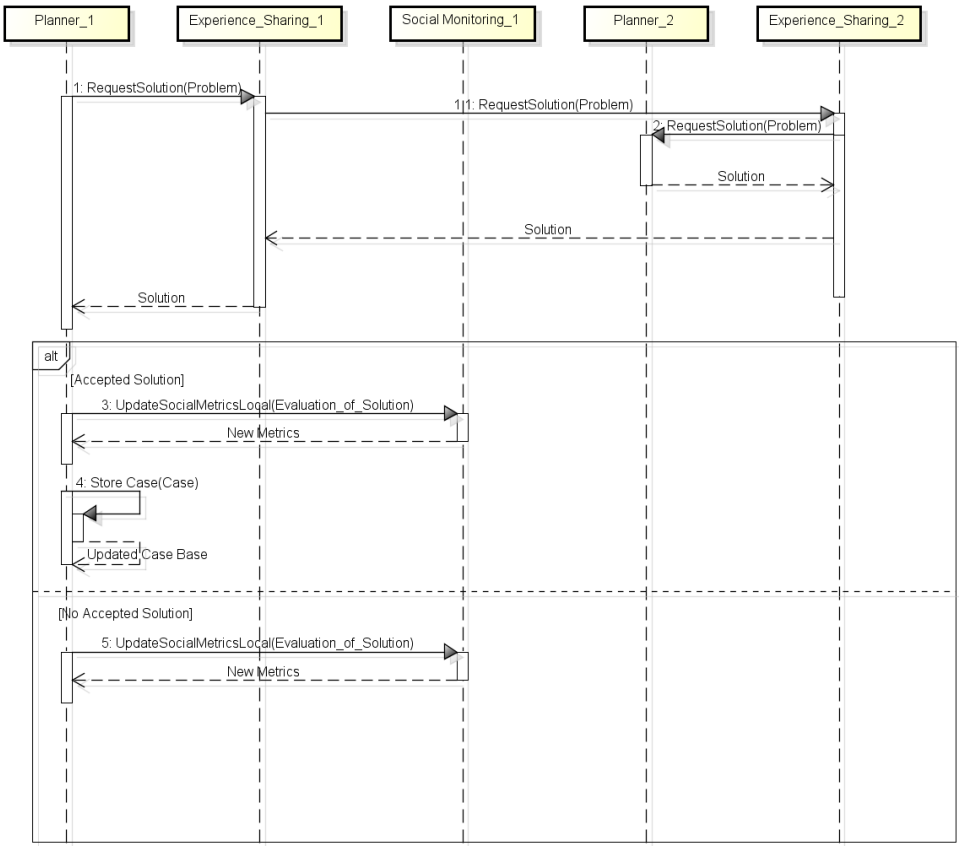


Figure 53: RequestSolution(Problem) Sequence Diagram

UpdateSocialMetricsLocal(Evaluation_of_Solution) in Figure 53 is detailed in Figure 54:

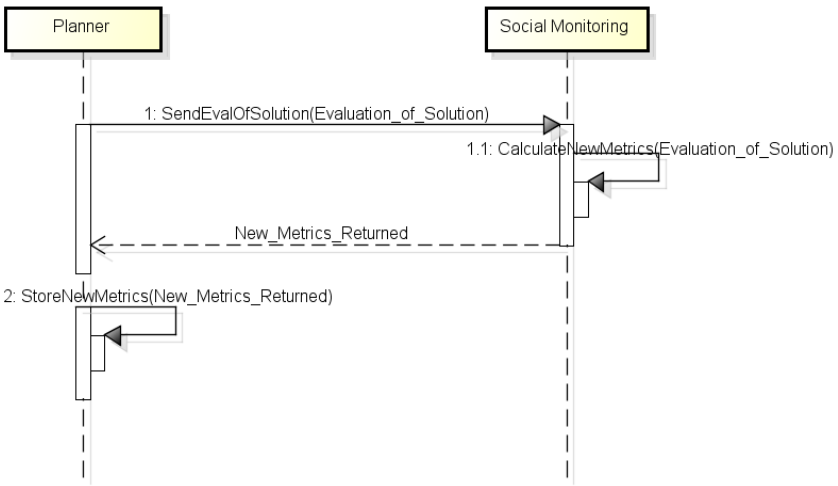


Figure 54: UpdateSocialMetricsLocal(Evaluation_of_Solution) Sequence Diagram

2.2.3. Component Tests

Before testing all the subsystems described above, we executed unit tests for each involved component. Because of the fact that some components participate in more than one subsystem, we present the unit tests results in the Annex, in order to avoid the repetition.

2.2.4. Occupancy detection model validation

As described in subchapter 2.2.1.4, we have used an occupancy detection scenario to demonstrate the application of pattern recognition techniques for inferring high level knowledge in IoT. Data was labelled with the help of users feedback and labelled data is used for training Machine Learning models. The total gathered data was divided into ratio 70:30. We have used the larger data set for training the model while other data set is used for validating the model. For the model creation we used the following techniques.

Support Vector Machine (SVM) [2] is an efficient algorithm which is widely used for classification because of their two main advantages: 1) its ability to generate nonlinear decision boundaries using kernel methods and 2) it gives a large margin boundary classifier. SVM requires a good knowledge and understanding about how they work for efficient implementation. The decisions about pre-processing of the data, choice of a kernel and setting parameters of SVM and kernel, greatly influence the performance of SVM and incorrect choices can severely reduce the performance of it. The choice of a proper kernel method for SVM is very important as is evident from the results in the next section. The SVM algorithm requires extensive time in training but, once the model is trained, it makes prediction on new data very fast.

On the other hand, K Nearest Neighbours (KNN) [3] is one of the simplest and instance techniques used for classification. It is a non-parametric algorithm which means that it does not make any prior assumptions on the data set. It works on the principle of finding predefined number of labelled samples nearest to the new point, and predict the class with the highest votes. KNN simply memorises all examples in the training set and then compares the new data features to them. For this reason, KNN is also called memory-based learning or instance-based learning. The advantage of KNN lies in simple implementation and reduced complexity. Despite its simplicity, it works quite good in situations where decision boundary is very irregular. Its performance is also very good when different classes do not overlap in feature space. KNN is also called lazy algorithm as it take zero effort for training but it requires full effort for the prediction of new data points.

2.2.4.1 Accuracy of Model

F-measure represents an accurate measure for evaluating the performance of multi-class classifiers (used for pattern recognition) and also one of the most commonly used metric to compare different classifiers. We have also used F-measure to compare the performance and validate the implemented algorithms. The selection of right features plays an important role for efficient implementation of Machine Learning algorithms. We have used three different feature sets for the evaluation of model which are shown in the following table:

No.	Features selected
Feature Set 1, F1	Active Power, Reactive Power
Feature Set 2, F2	Voltage, Current, Phase Angle
Feature Set, F3	Active Power, Reactive Power, Voltage, Current, Phase Angle

The first feature set, F1 consists of only power measurements and includes real power and reactive power. The second feature set, F2 consists of voltage and current measurements along with the phase angle between them. Finally, we have used all the five features for classification algorithms in F3. The complexity of algorithm increases with the number of features, and the selection of inappropriate features can result into complex decision boundary for classifiers affecting the performance of the algorithm as we discussed in the next section.

Figure 55 shows the F-measure plot of different classification algorithms implemented. From the figure, it is obvious that KNN performs best for F1 and F3 and achieves accuracy up to 94.01% while SVM-Rbf (SVM with Radial basis function as kernel) outperforms other algorithms for F2 with maximum efficiency of 86.99%. The reason for good performance of KNN for F1 and F3 is that the power features involved in F1 and F3 for different states are non-overlapping and distinct, and KNN performs very well in such situations. The overlapping nature of features in F2 resulted in the reduced performance of KNN, whereas SVM-Rbf performs better as compared to other variants of SVM. In general, SVM forms a hyper-plane as a decision boundary between different classes in feature space, and the shape of hyper-plane is governed by the kernel function chosen. The spherical nature of hyper-plane for Rbf kernel enables to classify simple and complex problems accurately. SVM-Poly (SVM with Polynomial kernel) performance is degraded for F1 as it tries to over fit the problem by forming complex decision surface. We have used feature set 3 for all further analysis in the paper.

The number of training samples plays an important role in the performance of a classifier. We have evaluated the performance of our algorithms against different number of training samples. As the training samples increase, the decision boundary becomes more accurate and the performance of classifier improves. Figure 56 shows how the F-measure of different classifiers improves as we increase the training samples. After a certain number of training samples, increasing the training data set does not have much effect on the performance of a classifier. SVM-Poly has the greatest effect on the performance with increasing training samples. SVM-Poly tries to differentiate all training samples by making complex and nonlinear decision boundary. For low data sets, the decision boundary is very specific but when the same model is validated against new data, the same decision boundary may not work accurately and result into degradation of performance. But as the training data set increases, the decision boundary becomes more general and more fitting for new data and hence performance of the classifier increases with increasing data set.

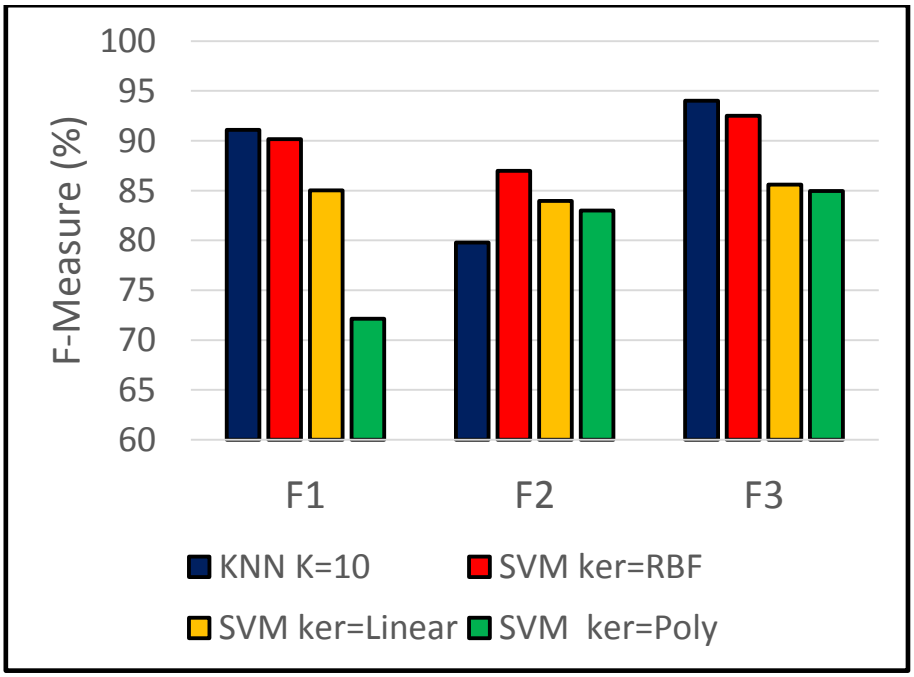


Figure 55: Performance of Classifiers

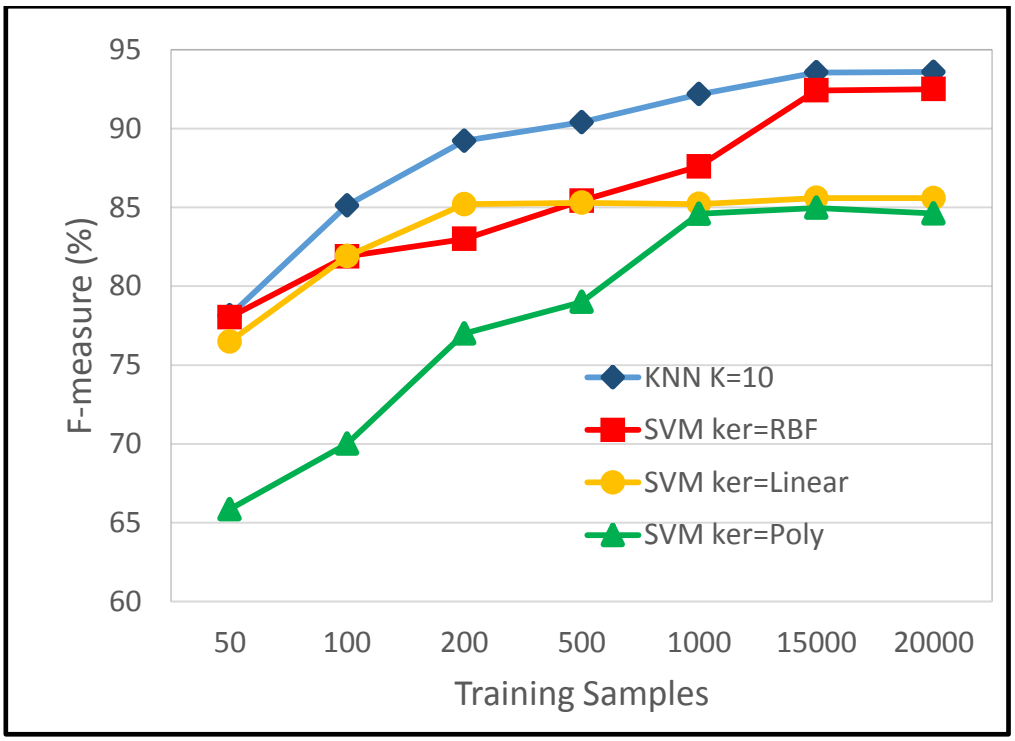


Figure 56: F-measure relation with training data

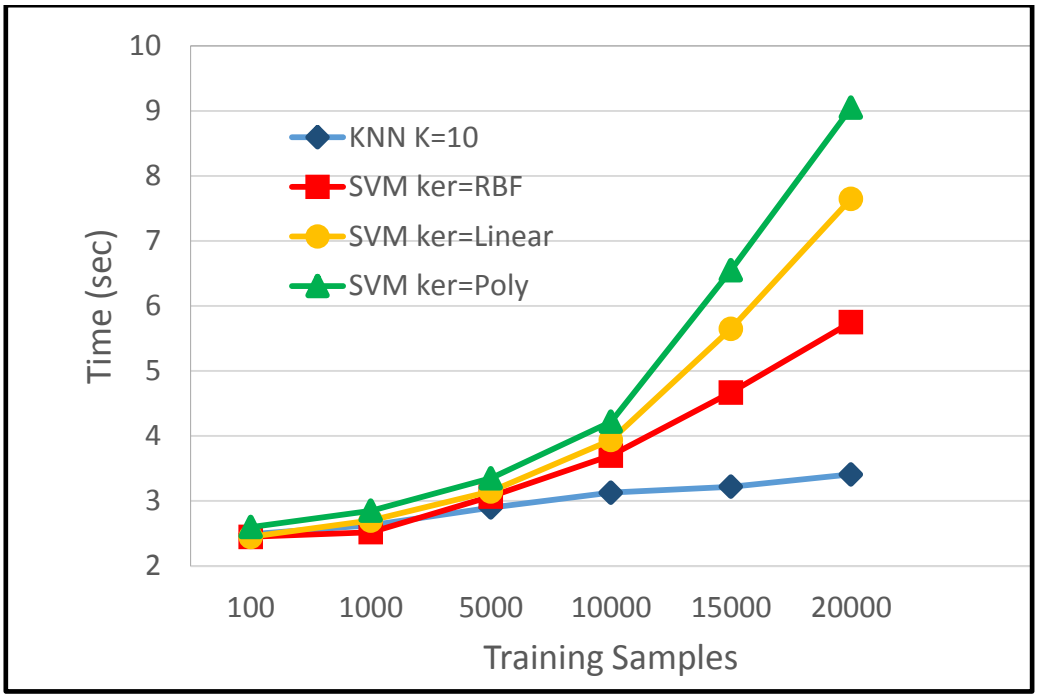


Figure 57: Time Complexity

F-measure is an accurate measure for reflecting the accuracy of a classifier. But as the data size increases, the complexity of the algorithm also plays an important role in choosing a particular technique: firstly, a large commercial building can have hundreds of nodes and the amount of data can be huge; and secondly, it is an important factor if we want to implement these algorithms on VE level to support the autonomous VEs. In order to give further insight about the performance of different algorithms, we compared the total time which includes training and validation for each algorithm. The amount of time taken by each algorithm for different number of training samples is plotted in Figure 57. The size of validation data remains the same for all cases. For small number of training samples, the performance of all algorithms remains almost the same. But as the data size increases, complexity and hence the time taken for all the variants of SVM increases rapidly. whereas, increase in time for KNN is significantly low. This is due to the instant learning nature of KNN. During training, KNN simply memorises all the examples in the training data set and used it for comparing and predicting new samples with highest nearby votes. In contrast, SVM implements gradient descent algorithm for finding optimum decision boundary (which is an iterative optimisation algorithm) and results in exponentially increasing time with increased number of training samples.

2.3. Observed Deviations and Applied Mitigation Strategies

With relation to observed deviations, a number of cases were identfied during this period. The real time data from EMT Madrid was not available so we loaded historical data files to the object storage. For this purpose the Data Mapper was adapted to enable adding data files to OpenStack Swift (as well as adding data subscribed to from the Message Bus). However this is also a valid and necessary case of data ingestion, especially for bridging existing platforms with the COSMOS ecosystem and using historical data to initialize the platform, thus minimizing the need for a preparatory stage before the platform goes online..

Furthermore, the EMT Madrid data was in UTM format, whereas metadata search required lat, long format. Therefore we introduced a Storlet to perform the conversion.

Another conclusion is that existing smart city platforms will most likely be varying in terms of protocols or data formats. Thus for linking them to the COSMOS platform, suitable bridging mechanisms must be deployed, as was performed in the case of Camden UC.

2.4. Future steps with regard to the advancements of this period

2.4.1. Security aspects

Future versions of will allow for an optimized usage of resources and processing time. Therefore the hardware security modules will be extended to make use of the ARM SoC interrupt controller. Supplementary the design will be optimized with respect to speed and power usage.

The software support (Linux drivers and supporting API's) will be extended to allow usage of ARM's sandboxing mechanism (TrustZone) in order to separate security critical tasks from the rest. This support will be baked into the entire platform therefore allowing a system-wide security approach. Also for the next years secure storage methods will be developed in order to allow safe storage of secret information such as encryption keys.

2.4.2. Data Analytics and Storage

In future versions we aim to allow more advanced forms of analysis (requirement 4.4). Currently we allow Storlets to be applied when an object is created or retrieved. In future we plan to allow analyses over sets of objects. Additionally, we intend to associate the objects with enriching social metadata depending on the VE's interactions with other VEs.

We intend to provide a situational knowledge acquisition service which is more suitable for dynamically changing environments. We will provide continuous changing of situation monitoring parameters and analysis and support for external queries.

2.4.3. Autonomous Behaviour of VEs

During Y1, regarding the main component of Autonomicity, the Planner, we managed to accomplish some of its main requirements and integrate it with the Experience Sharing component. More advanced requirements are going to be covered during the next two years as it has been described in D5.1.1 [5]. For example, the reasoning techniques used will be expanded in order to achieve GVE forming and management. The first steps for fulfilling the requirements under Task 5.3 (Network of things run-time adaptability) have already been taken.

As for the progress on the Social Monitoring and the Social Analysis components, we estimate that during Y2, the eventual introduction of a VE Registry will facilitate greater progress in the development of these components.

2.4.4. Modelling Aspects

We will focus on a solution that provides VEs with a good and powerful situation awareness support based on CEP. As far as Experience Sharing is concerned, the current solution developed during Year 1 will be improved and aligned along the development of the Case Base Reasoning part. It will also provide a preliminary solution to the Trust and Reputation mechanisms.

3. Integration Stages 2 (M17-M22) & 3 (M23-M25) from D7.7.2

3.1. Overview from Integration Plan

Following the results of the first integration period and the initial deployment and definition of the COSMOS platform, the second and third stages of integration (M17-M22 and M23-25) aim at further enhancing the linking in the platform and achieve better incorporation of the UC elements. Specifically and as derived from the Integration plan, the main goals that are foreseen for this period are:

- Definition of data models and fields of information across the various components in cooperation with the component needs and the UC data feeds. This implies mainly the identification of component combinations, data inputs and messaging needs. For each subsystem identified in Section 3.2, this has been included (where applicable) as a Message Formats subsection
- Initial version of the application scenarios, which implies work towards the following subgoals:
    - Application Definition Framework, abstracting the sources of data and enabling combinations to achieve the desired app logic in the context of a specific application, enriching the latter with the COSMOS features and initial Node-RED flows
    - Application Archetypes Definition relating to high level ways of combining subsystems, as these have been defined in D2.3.2, in order to create conceptual template applications that may be reused in similar scenarios and driven by the UC needs as these have been expressed in the related documentation (D7.1.1 and D7.1.2)
    - Application Scenarios concretization: in this case we expect to define how the application scenarios will be instantiated and implemented with the usage of the appropriate COSMOS services and components including aspects such as data flow from multiple sources of information, event definition and identification, specific models needed etc.
- Integration at the VE side, including VE resources specifications, and referring to the VE side components of COSMOS and how these can be deployed on the VE gateways
- COSMOS Platform and Cross Component integration was expected to carry on, mainly in order to incorporate any needed changes and new advancements, such as proactive experience sharing and the combination of ML and CEP approaches in the Data Management and Analytics domain, and direct CEP rules editing and update of the Case Base creation from historical data and the planner and privelets for direct VE2VE communication. Furthermore initial specifications for the Web UI were expected to be investigated.
- The VE Description and Linking to the platform, which aims at resulting in having semantically enriched VEs in the COSMOS platform, together with their annotations and endpoints description. This includes the subgoals of
    - COSMOS Ontology Definition covering aspects of the endpoints description through which data acquisition may be performed (e.g. through REST interfaces, through appropriate topics in the Message Bus etc.)
    - Domain Specific Ontology Definitions for Use Cases to decouple endpoints descriptions from the semantics of these endpoints, adapting to the UC

individual needs and enabling the mapping of the endpoint to the concept, with at least one being ready up to this point.
- Linking between the schema used in each case and a schema description stored in the context of the platform, so that it can be retrieved by the AD in order to get informed about the available data
- VE Instances Descriptions and Registry population to include concrete endpoints from the project UCs.

What was also performed was to include in all these cases the specific aspects of the UCs, either in terms of needs, necessary functionality or adaptation of the latter to a respective UC scenario. From the beginning of the process the UCs were directly involved. The work was organized in relevant technical groups, each of them having one or more (in case they were related) subgoals mentioned above. The outcomes of these groups are reflected in each one of the following subchapters. Furthermore, one group for each UC application was created, that was fed by the outcomes of these groups in order to adapt them to the application demo scenario.

3.2. Defined Subsystems and Tests

3.2.1. VE Instances Descriptions, Registry interface and Data Schema Storage/Retrieval

3.2.1.1 Scenario description

Adequate VE description is critical for future COSMOS application development as it allow VE discovery and reuse (Figure 58). During year two of the project we have implemented the core functionality of the VE registry including the semantic model, the registry API and the semantic annotation frontend.

The focus was put on the core VE description capabilities which spans from the actual physical entity being observed, passes through the IoT services and its endpoints and ends to the VE properties exposed by VEs. The semantic model is able to support the description of all these elements and to link them together in a consistent way. This consistency, as well as the adherence to a common description model, paves the way to the development of a complex, multi-criteria retrieval mechanism. Development of COSMOS enable applications involves multiple actors which are not necessarily in contact with each other. The VE registry acts therefore as a mediator or a proxy from a description point of view.

For instance, an IoT Service Developer which has developed the services which provide access to the sensors and actuators of a building will publish the semantic descriptions of these services so that other users can retrieve them.

These IoT services could be used directly but a VE developer will use the COSMOS approach in order to enrich these services and wrap them as VEs. Such extensions could include additional functionality such as CEP based raw data processing capabilities or even complex logic such as that for an HVAC system of a building. As a result, the VE developer will publish the description of the VEs which includes the VE identification (name, related physical entity, VE URI), its related physical entity location and the VE properties (which could linked directly to the plain IoT Service endpoint or to the extended VE services like the ones mentioned above) .

Once such VEs are published, COSMOS application developers will use the VE registry retrieval frontend to search for VEs which meet their requirements. Applications developers could use VEs owned by different VE developers in order to create integrated applications

Furthermore, we expect each use case to have data sources with their own data fields and associated data types. For example sensors from the Madrid Traffic use case record traffic speed and intensity, whereas sensors in the Taipei use case record active power and current. Therefore in COSMOS we allow associating a **schema** with each data source, and we expect each use case to have one or more data sources. This schema should also be included in the topic description of the registry.

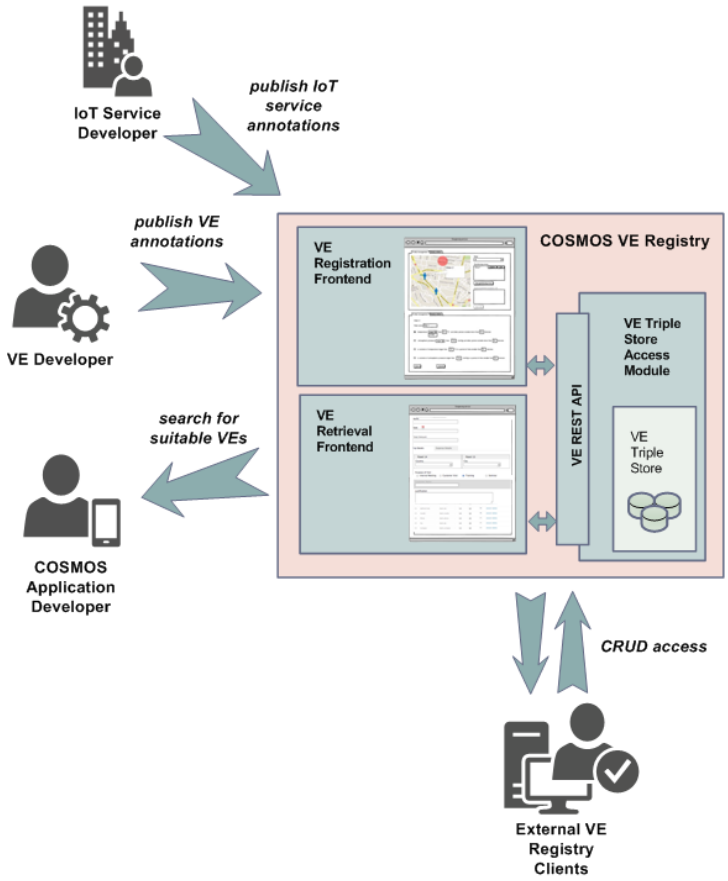


Figure 58: Generic usage scenarios for VE registry

3.2.1.2 Subsystem from D7.6.2 with integration points description

The subsystem derived from the D7.6.2 has been enriched with a new Integration Point (IP2) (Figure 59), which refers to the retrieval of VEs or their endpoints based on specific features requested. In this section we include information on IP4, regarding the declaration of VEs and their endpoints, while IP2 and IP3 will be pursued during Y3 of the project.

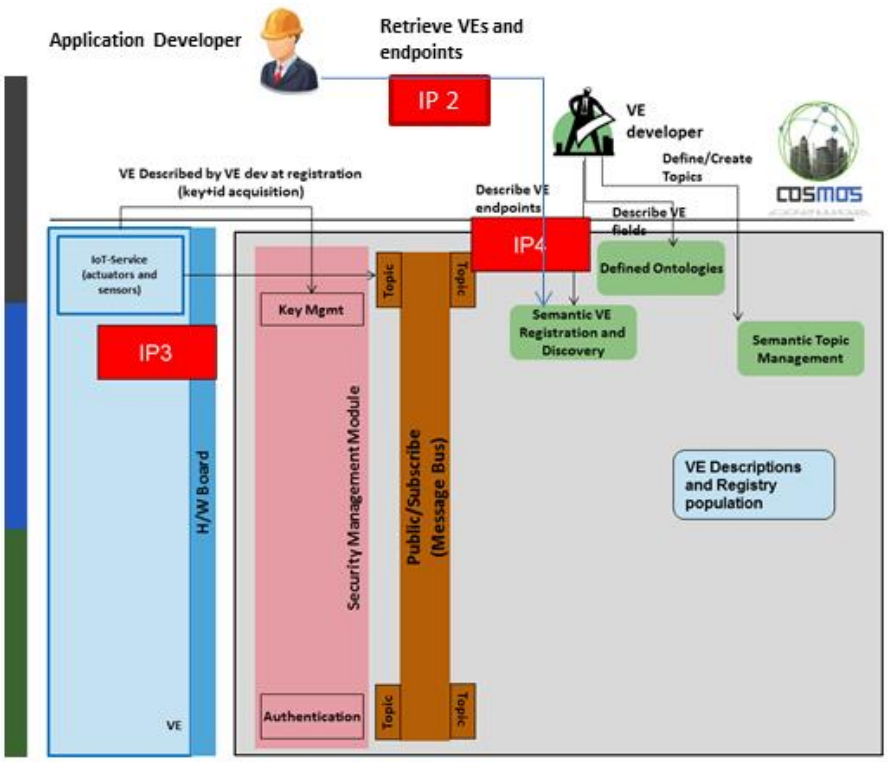


Figure 59: VE Registration Subsystem with identified IPs

Besides the API which exposes the access to the VE Registry, we have developed a web based frontend whose welcoming page is depicted in Figure 60. This tool facilitates the description of the VEs and of the associated elements (IoT services, interface endpoints, etc.) thus relating to IP2.

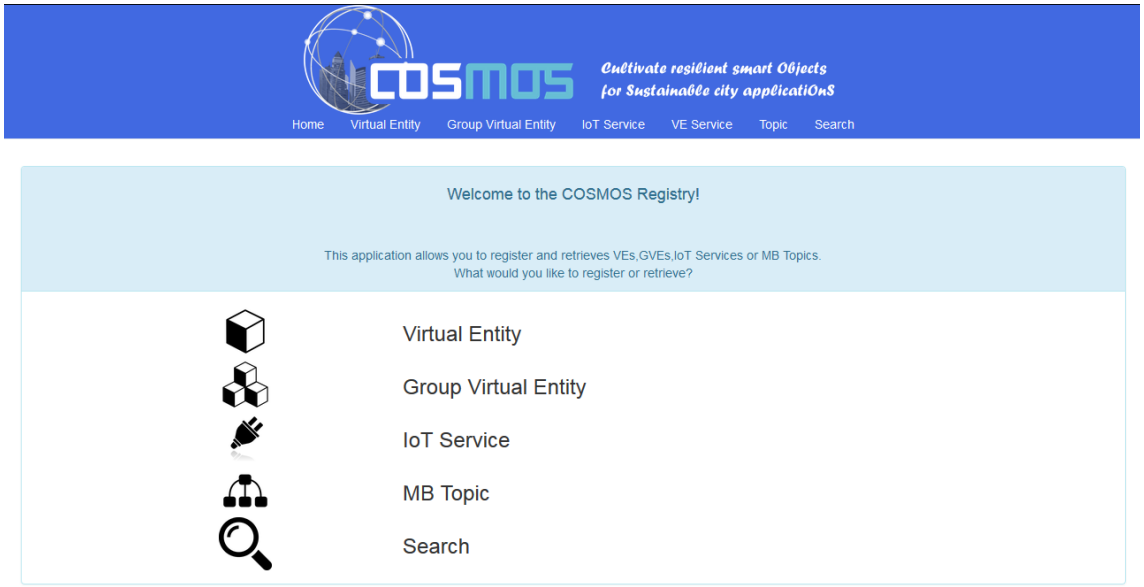


Figure 60: COSMOS VE semantic annotation welcoming webpage

The frontend has been developed after mockup versions have been created. This approach was taken in order to evaluate the annotation steps and how the whole process can be integrated into one tool. Figure 61 and Figure 62 depict two of the mockup and implementation versions of the tool pages. In the latter, the integration points with the DSOs are described, that are also part of IP4, mainly with relation to the defined ontologies. Semantic attributes from the DSOs can be included in the endpoint specifications so that semantic linking is achieved.

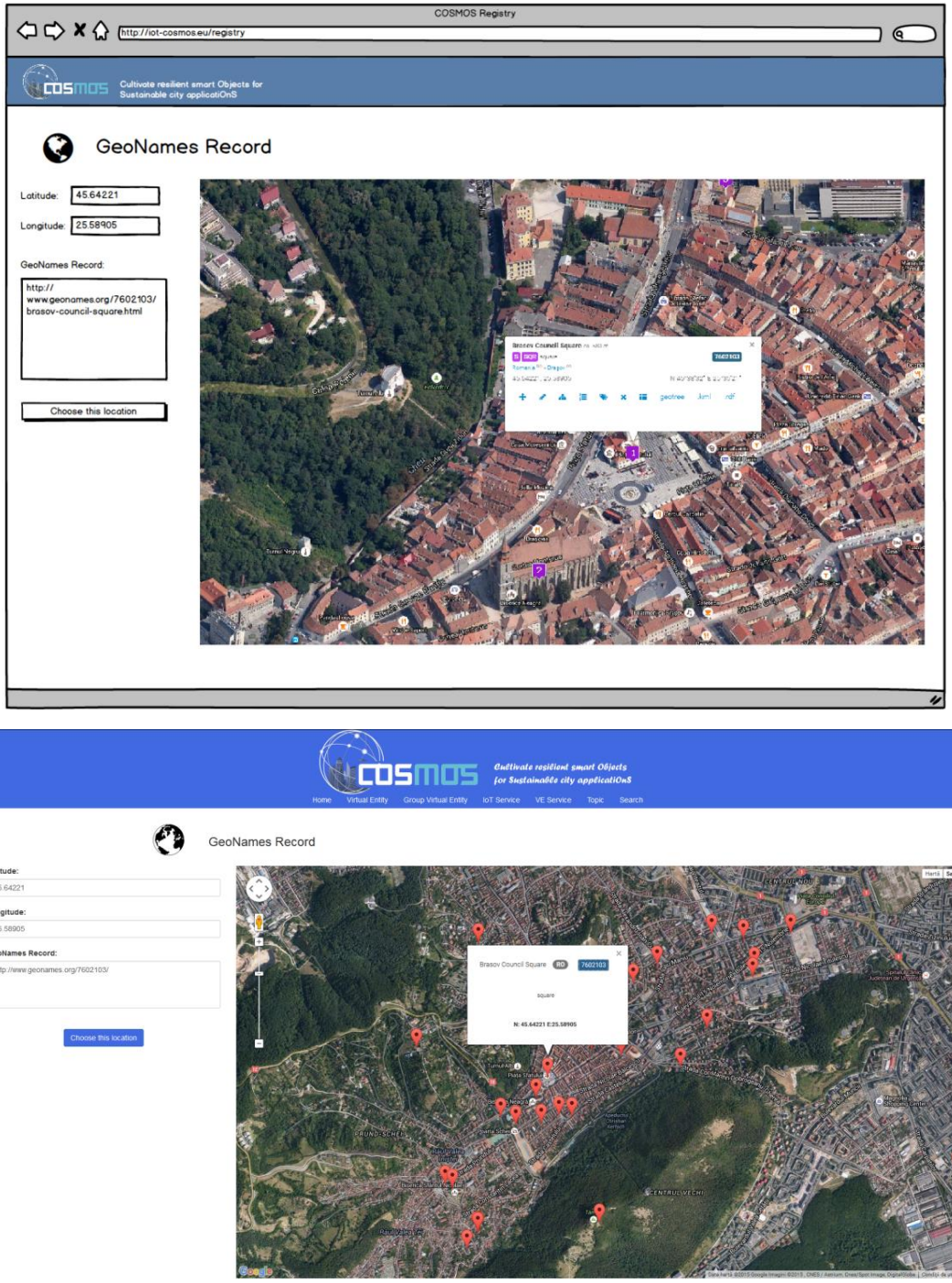


Figure 61: Mockup (top) and the actual implementation (bottom) for one of the location browser pages (in this case the GeoNames Record based location indicator)

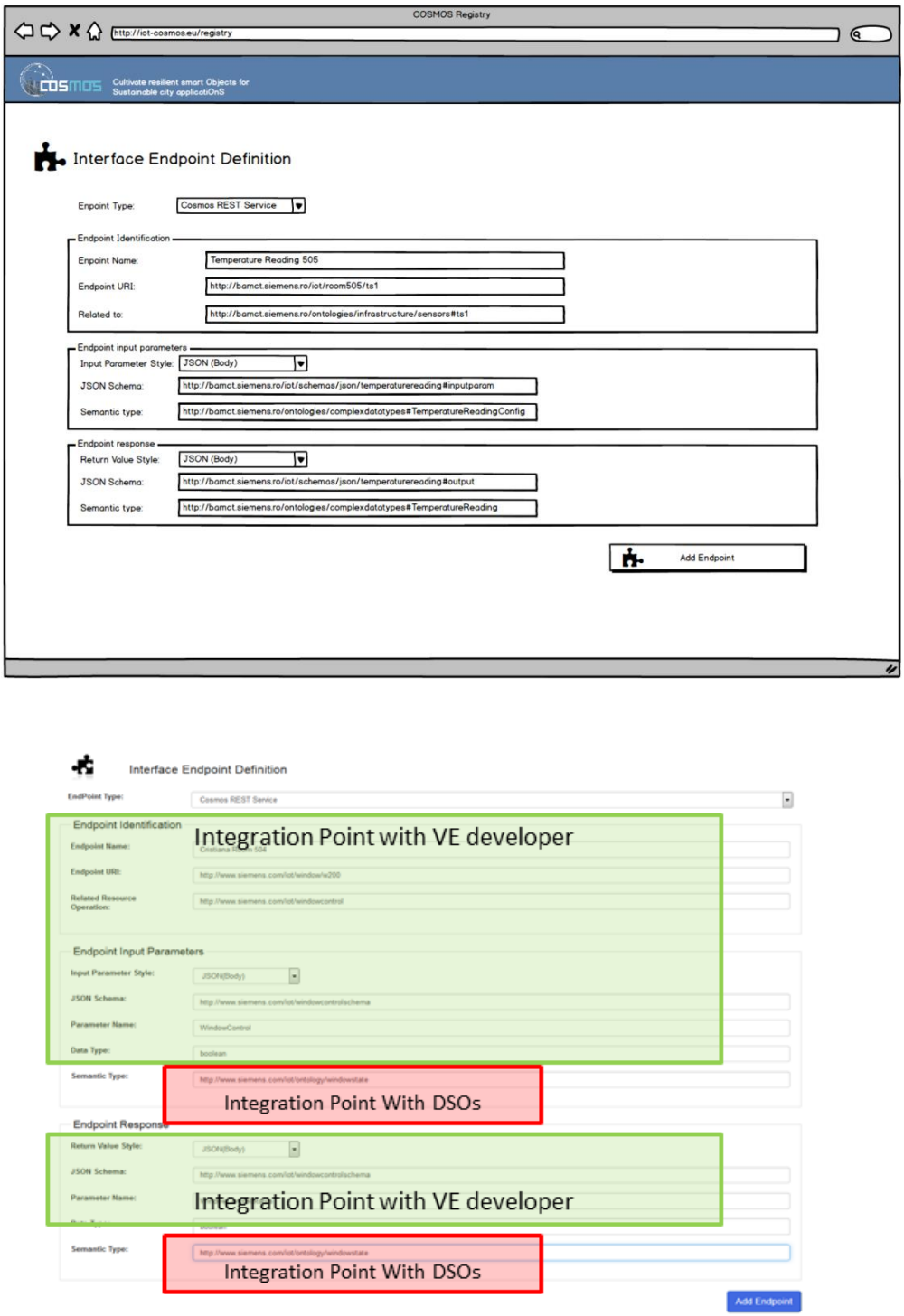


Figure 62: **Mockup (top) and the actual implementation (bottom) for the Interface Endpoint Definition page**

Despite the fact that most of the VE description is expected to be done through the VE registry frontend, the registry also exposes a web API which can be used by other external clients if required. The VE registry web API is based on REST services using the JSON format for the data transfers. A sample JSON object for a service endpoint request is depicted in Figure 63.

```
{
    "EndpointIdentification":
    {
        "EndpointName":"Cristiana Room 504",
        "EndpointType":"Cosmos REST Service",
        "EndpointURI":"http://cosmos.eu/iot/window/w200",
        "RelatedResourceOperation":"http://cosmos.eu/iot/windowcontrol"
    },
    "EndpointInputParams":
    {
        "ParameterStyle":"JSON",
        "Schema":"http://cosmos.eu/iot/windowcontrolschema",
        "ParamName":"WindowControl",
        "DataType":"boolean",
        "SemanticType":"http://cosmos.eu/iot/ontology/windowstate"
    },
    "EndpointResponseParams":
    {
        "ParameterStyle":"JSON",
        "Schema":"http://cosmos.eu/iot/windowcontrolschema",
        "ParamName":"WindowControlStatus",
        "DataType":"boolean",
        "SemanticType":"http://cosmos.eu/iot/ontology/windowstate"
    }
}
```

Figure 63: Sample JSON object for service endpoint request

In Figure 62 there are also fields with relation to the schema used in a defined topic. These fields are the integration point IP4 that refers also to the JSON schema retrieval subsystem mentioned in D7.6.2 (and copied here in Figure 64) is important for the following components

- When a new topic is created for a data source in the **Message Bus** it needs to be associated with a schema
- This schema needs to be stored. We store it in **object storage** and associate it with the Message Bus topic using a simple naming convention
- Data flowing in the **Message Bus** should conform to the schema
- The schema is used by our Secor extensions in the **Data Mapper** in order to convert the data into Parquet format for storing in object storage
- When **applications** query the data they know what data format to expect according to the schema. **Applications** can retrieve the schema using the **object storage** REST interface.

Interfaces for storing the schema (i.e. IP 3 of Figure 64) are included in Section 3.2.1.3.

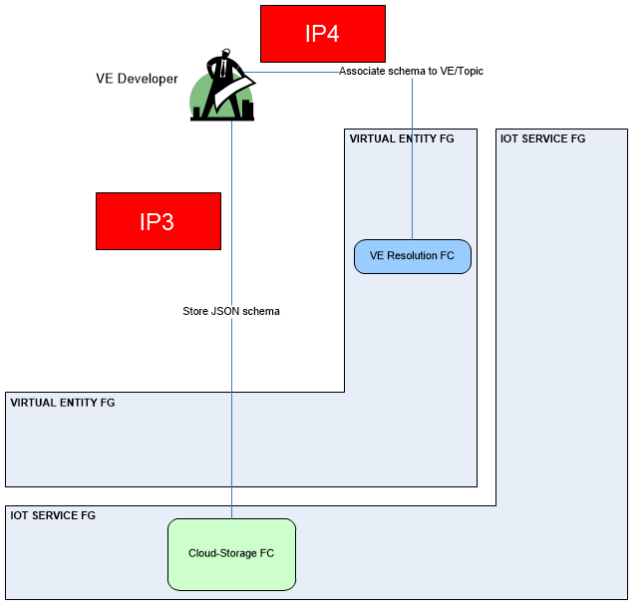


Figure 64: JSON schema storage and association

Relation to Domain Specific Ontologies

As mentioned in D7.6.2, in order to have a flexible and decoupled semantic model, the COSMOS ontology defined in D5.1.1 is focused on only protocols and endpoints specification, while the DSOs are focused on specific UCs, and cross-referenced in the main ontology. Furthermore, given that the COSMOS ontology supports REST and topic based endpoints, in the case of the description of the endpoints the relevant e.g. topics used in the COSMOS MB to redirect data from the UC platforms should be included, and not endpoints of other protocols used in the context of the UCs (such as DDP or AMQP used in the Madrid case).

Following, details on the developed Camden DSO are presented. The COSMOS DSO approach for the Camden Smart Home management is being developed on the basis of the Flat capability description that was provided by the Camden Housing Authority. This description includes a thorough list of all possible attributes a Flat can possess and includes not only technical but also structural information.

This ability to diversify the focus from the purely technical aspect has allowed us to work on a more general Domain Specific Ontology (DSO) for Smart Homes that has information on things like House Insulation as well as information on the type of sensors being provided.

In the context of the DSO for Camden we identify attributes that can be described in Boolean fashion (Exists) as well as those that can be summed up in a more enumerated way (Category) or via numerical values (Numerical) (Figure 65).

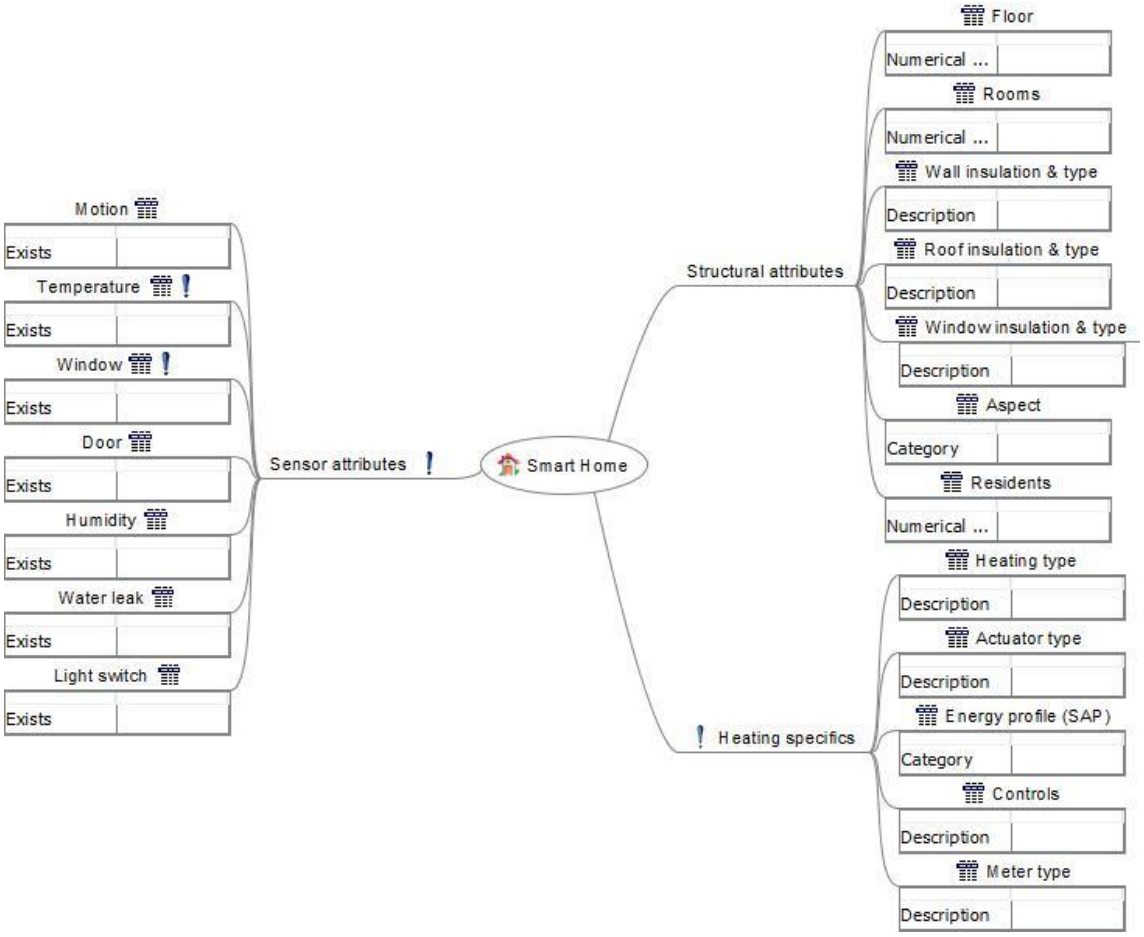


Figure 65: Camden DSO fields

Those identifiers of Boolean fashion as mentioned previously are the mainly the signifiers of individual sensor existence, as well as the existence of types of insulation, heat meters and heating controls. For the Madrid case, a relevant ontology has been defined by EMT and will be incorporated in Y3 of the project.

During Y3 of the project we will also integrate the functionality of the VE Registry with other components (mainly the Planner) through the API usage for dynamic retrieval, and enhance the semantic model to cope with the additional description requirements, while describing the existing VEs in the project. The VE query mechanism will be extended with an enhanced multi criteria search functionality and its associated API which will facilitate COSMOS based application development.

3.2.1.3 Message formats and configuration

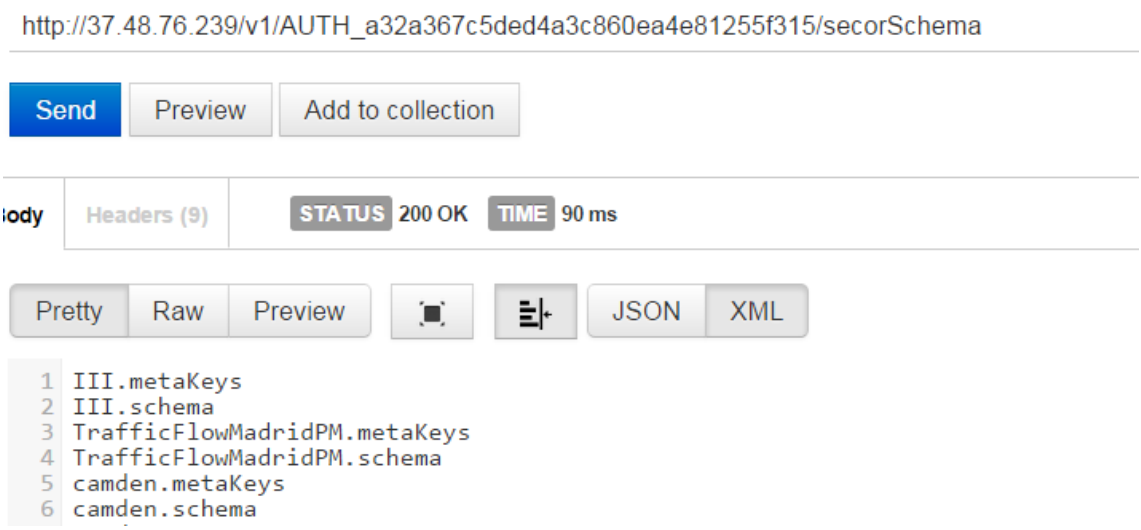
Driven by developments in the Camden Use Case, we have identified the need for receiving information about the endpoints of the sensor data in the form of a data stream being processed through the MQTT protocol. This is in connection to the way the Hildebrand Servers aggregate data from Flat Sensors and forward them to messaging topics based on the Estate names and the Flat identities (hid) assigned by them.

The correct way to ensure that we have knowledge of specific VE sensor endpoints comes from identifying both the address of the endpoint which is common for all VEs in the Camden scenario, but also pinpointing the specific topic that information is published on. Additional knowledge includes any authentication requirements that the MQTT connection requires.

Ergo, it must be mentioned that currently the format for storing individual VE MQTT endpoints for the Camden scenario is tcp://{{domain}}:{{port}} plus the topic description that is formatted thus: COSMOS/{{estate}}/{{hid}}. It is also being considered whether the storage of the endpoint credentials should be done for each individual endpoint, or at a higher level, considering the lack of individual VE credentials for accessing the data stream.

Work is also being done by the Camden Use Case partners into defining REST endpoints for the control of the actuators in each Flat. In this case depending on the format provided, there will be further updates of the endpoint storing structure and definitions accordingly.

With relation to the schema storage and retrieval, we store a schema for each Message Bus topic associated with a data source in the Object Storage (OpenStack Swift) in a container called secorSchema. The name of the object containing the schema is <topic_name>.schema. For example, currently the testbed contains 3 data sources so when listing the secorSchema container we see the following objects



Note that we also store metaKeys objects in the same container – these can be ignored here. Applications can retrieve the schema using the object storage (Swift) REST API. For example the following shows the GET command applied to TrafficFlowMadridPM.schema. The link to the schema should be included in the respective VE registry field mentioned in Figure 62.

We use Apache Avro to define schemas. Each schema needs to have a timestamp field called ts which uses epoch format (long) to represent the timestamp.

3.2.1.4 Sequence Diagram

In this case no sequence diagram is presented since the majority of the operations are manual through the GUI. In case of API usage (e.g. by the Planner), this will be a single query call.

3.2.2. Application Definition Framework through Node-RED Flows

As indicated in D7.6.2, Node-RED usage has been favored in the context of the project for a variety of purposes, ranging from data feed linking, to deployment, testing and service orchestration purposes. A variety of roles may interact with the Node-RED environment in order to create flows, exposing COSMOS services through nodes, integrating action sequences and data relay and enhancing the base functionality by the abstraction achieved through the tool.

3.2.2.1 Testing Scenario description

The envisioned scenario appears in Figure 66, derived from the Integration Plan. The main interacting roles may use the COSMOS repository in order to store, reuse, configure, combine specific flows. For this reason, we should provide them with the environment that will enable them to perform these operations and that consists of two parts:

a) The Node-RED framework, adapted for multiple users
b) The sharing functionalities between different entities expressed via repository structures

As well as the description of the process for the usage of these features.

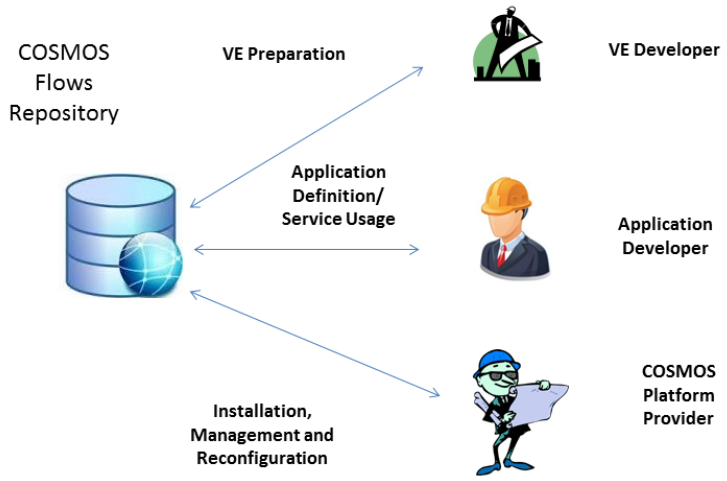


Figure 66: Reuse scenarios of Node-RED flows (Roles interactions Figure 17 from D7.6.2)

3.2.2.2 Subsystem from D7.6.2 with integration points description

The subsystem “Application Definition Framework Through Node-RED Flows” relates to this section (Figure 67). The repository functionality presented in this section applies to the Pattern Reusability FC whereas the Node-RED Environment setup and process relates to the Service Orchestration FC. The interfaces to COSMOS services are individual per component/service and are described in the subsequent sections of the document, either as flows or nodes.

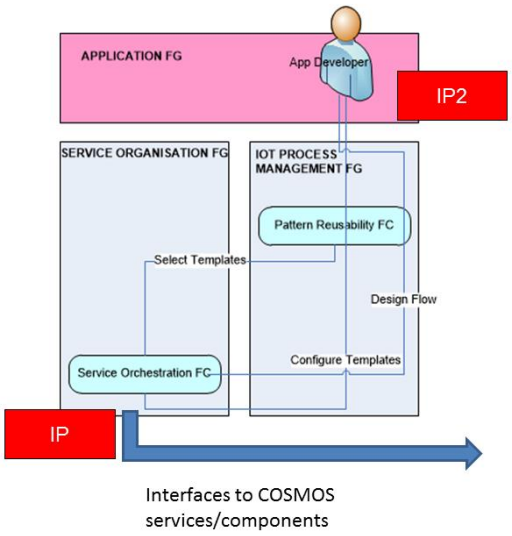


Figure 67: Generic Application and Flow Design Process (Figure 20 of D7.6.2 with identified IPs from Subsystem 9.3.6.1 of D2.3.2)

The main aspect described here is the IP2 for the Developer (App Developer but also COSMOS developers) in the repository of shared flows, since the usage of the Node-RED environment relates to its standard GUI. For the usage by multiple entities, we have created a multi-user setup that creates a separate environment for each individual user/entity/role to define their flows, described in detail in D3.2.2 (Section 5). This ensures that each partner using Node-RED does not intervene with other flows used in the project, for better separation but also for better visibility of the flows tab. Nodes used by more than one partners may be installed with the –g flag, thus being available to all the accounts. Each user is authenticated via the browser before entering in the Web based GUI. However given that flows may be reutilized, combined and shared, an according repository structure was created. This structure mentioned in this document is intended to be used inside the COSMOS project for enhanced manageability of the flows and faster development/integration. However potential linking to externally available repositories may be pursued in order to contribute the created flows to the Node-RED community, where applicable (e.g. http://flows.nodered.org/). Details regarding this integration point follow.

3.2.2.3 Message/Data formats and configuration

The main condition for a developer to share their flows is to notate them, including in the flow name the keyword “public”. This was inserted in order to differentiate between flows that may or may not be shared, based on the developer’s intentions. It was also used as an indicator to the developer that they should remove any sensitive information from any flow nodes, that is typically inserted during the node configuration. As an example, the Twitter node needs credentials for accessing the respective account. While this node, when copied, does not keep the credentials, this may not be the case for all node implementations. Furthermore, notations are necessary as comments in the flow where parametric input is needed during configuration. As an example, the notation of the Share_flows_public flow is presented in Figure 68. This flow is intended to be used by all partners in order to copy their public flows in the repo account. However configuration information is needed in terms of their account names, which is notated as a “INPUT NEEDED” comment. Furthermore, the condition based on which the

sharing is made is also commented (red boxes). In Figure 69, also the exact position of the node configuration input is highlighted (red box). A similar setup has been followed for the other major flow of this section, the Fetch_flows flow.

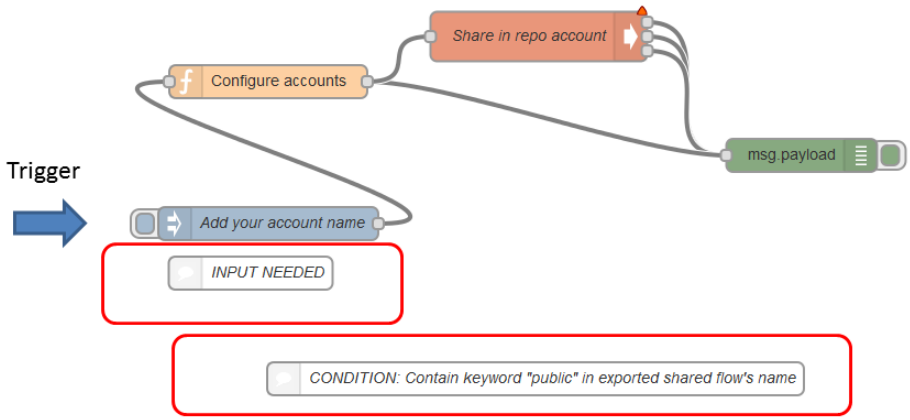


Figure 68: Share_flows_public flow notation example

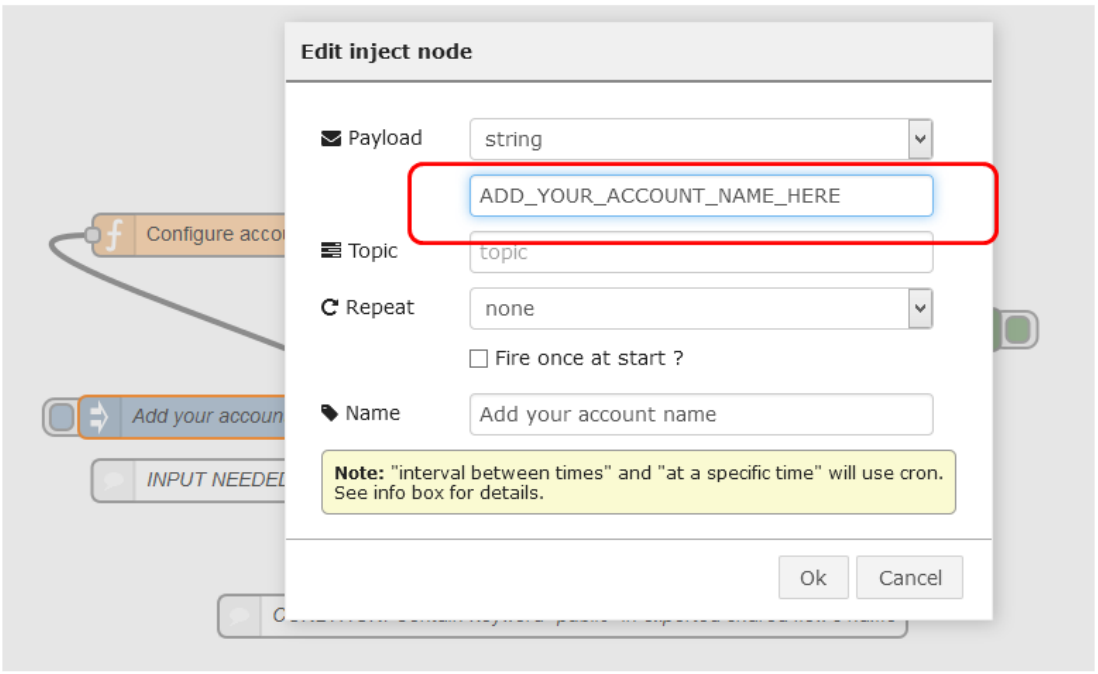


Figure 69: Internal node configuration for the Share flows functionality

3.2.2.4 Sequence Diagram

The Sequence diagram for the complete process appears in Figure 70. While this process is not automated, we chose to illustrate it with a sequence diagram in order to better highlight the

involved steps. The main requirement is that the Share and Fetch flows are included by default in the developer's initial account.

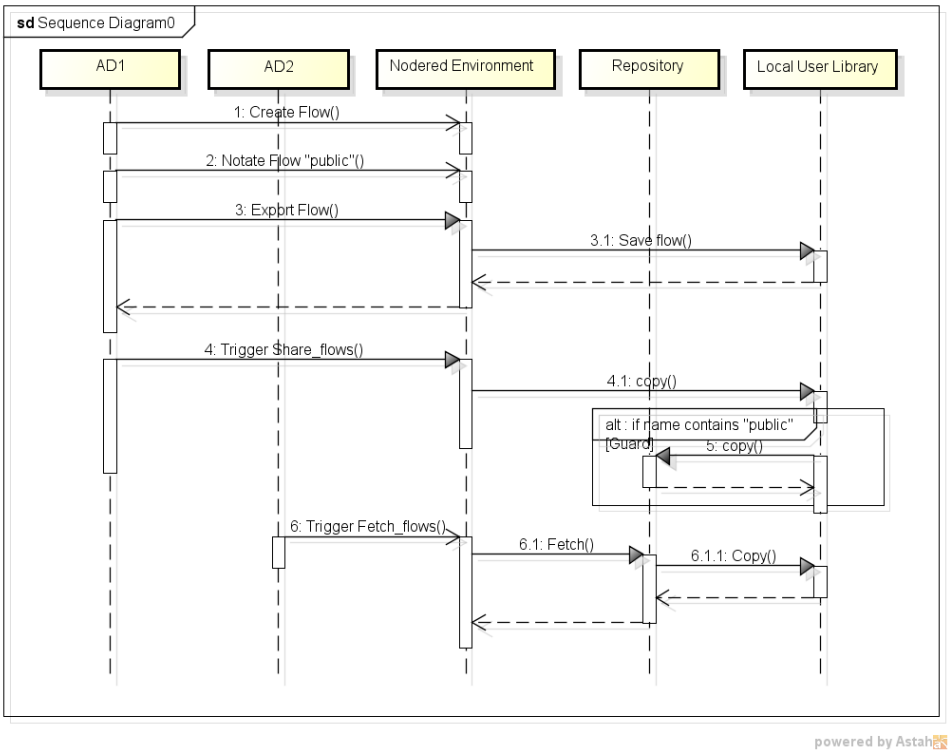


Figure 70: Sequence Diagram for sharing flows

3.2.2.5 Subsystem Test case table

The necessary test cases for the two operations (Share and Fetch flows) appear in Table 4 and Table 5 respectively.

Table 4: Flow sharing test case

Test Case Number Version	ADF_1
Test Case Title	Flow sharing between different Node-RED accounts
Module tested	Pattern Reusability FC (Repository flow), Service Orchestration FC
Requirements addressed	UNI.408, 5.31, UR1
Initial conditions	The developer has an account in Node-RED and has created a flow that they want to share inside the account environment. The developer has copied in their account the Share_flows_public functionality and has renamed it to Share_flows (so that it is not recopied). There is a "repo" account available in the Node-RED environment
Expected results	All local flows with the notation "public" are copied in the repo account.
Owner/Role	Developer
Steps	The developer has selected and exported the flow in the local

	library and has notated its name with the keyword “public” The developer has configured the Share_flows flow with the account name. The developer triggers the flow and the local flows are copied in the Repo account.
Passed	Yes
Bug ID	None
Problems	None
Required changes	Potential usage of the rsync functionality with the according options (or append of the original account name) may improve the flow especially in cases of overwrites.

Table 5: Flow Fetching test case

Test Case Number Version	**ADF_2**
Test Case Title	Flow Fetching from Repository
Module tested	Pattern Reusability FC (Repository flow), Service Orchestration FC
Requirements addressed	UNI.408, 5.31, UR1
Initial conditions	The developer has an account in Node-RED. There is a “repo” account available in the Node-RED environment
Expected results	All repo flows from the repo account are copied in the user’s account.
Owner/Role	Developer
Steps	The developer has configured the Fetch_flows flow with the account name. The developer triggers the flow and the local flows are copied in the Repo account.
Passed	Yes
Bug ID	None
Problems	None
Required changes	Potential usage of the rsync functionality with the according options (or append of the original account name) may improve the flow especially in cases of overwrites.

3.2.2.6 Inclusion of node.js popular functions in Node-RED

One of the issues that has been encountered during this period is the inclusion of common Node.js functions, for which there is no specific Node-RED node available, inside the Node-RED environment or external repositories. Node-RED nodes have some differences in packaging and implementation with relation to their node.js equivalents, thus direct porting is not feasible. Due to the fact that Node-RED is executed in a sandboxed environment, it does not have access to the overall features and libraries of the normal Node.js installation in the same server. Thus references to these libraries (mainly through the ‘include’ keyword) need to be enriched with the following process (we use as an example the amqplib of Node.js).

1. Install globally (for all Node-RED accounts) the required library through the relevant instruction:

sudo npm install –g amqplib

2. Follow the instructions in [8] and change the settings.js file global context tag to include the needed libraries (Figure 71), restarting afterwards the Node-RED daemon:

sudo nano /usr/lib/node_modules/node-red/settings.js

```
    functionGlobalContext: {
        os:require('os'),
        amqp:require('amqplib'),
        amqpcreds:require('amqplib/lib/credentials'),
        amqpconnect:require('amqplib/lib/connect'),
        assert:require('assert'),
//      amqputil:require('amqplib/test/util'),
        net:require('net'),
        url:require('url'),
        amqpcreds:require('amqplib/lib/credentials'),
        amqpCall:require('amqplib/callback_api')
        // bonescript:require('bonescript'),
        // arduino:require('duino')
```

Figure 71: Configuration of settings.js file in Node-RED for external Node.js libraries

3. Create a typical 'function' node in Node-RED and enter the client code exploiting the library (as indicated in the common examples of these libraries), by referencing the function using the global.context keyword followed by the designated name in the settings.js file (Figure 72). This definition needs to replace any typical 'include' instance of the external library.

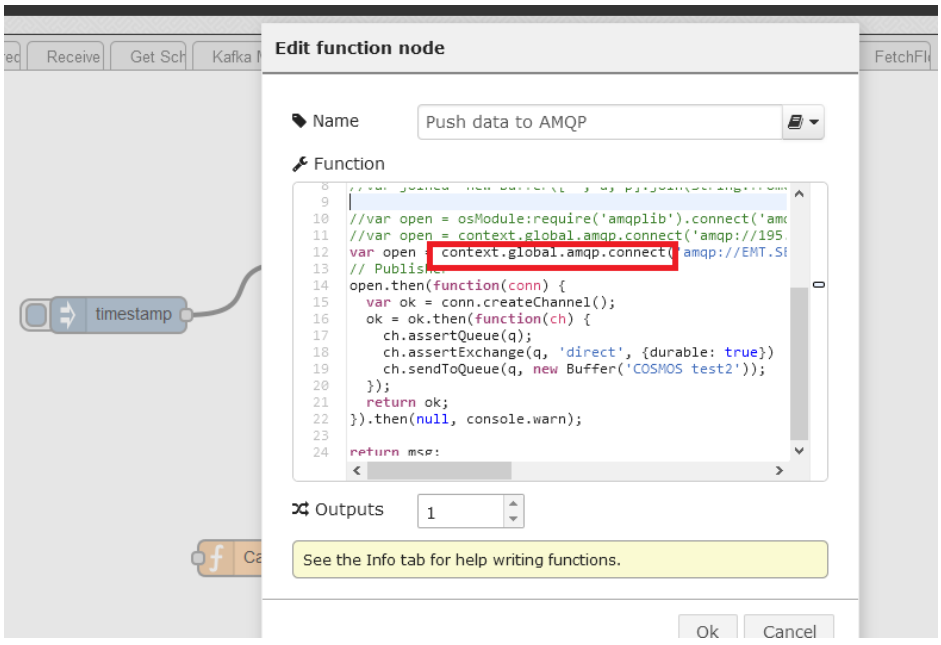


Figure 72: Node-RED function node code for inclusion of external libraries

This process was followed for example for the creation of DDP and AMQP publishing clients for Section 3.2.6.

3.2.3. VE side COSMOS Components integration/Installation

VEs are represented by different software modules running on top of a hardware platform. As security is provided/showcased by the Hardware Security Board it will also be used as an integration point for all the VE side components (Figure 73). A Hardware Security Board can host one or more VEs. The Hardware Security Board implies a high degree of security and a number of constraints for the user therefore the first step is to integrate all components using a more light-weight approach, e.g. a Raspberry Pi development board which provides the same hardware architecture (ARMv7) as the Hardware Security Board.

The VE integration is important for the following components:

- Privelets – every time new data is generated a privacy filter can be applied by the data owner. Privacy filters are generated applied and managed using the Privelets mechanism;
- Planner & Experience Sharing – are vital components of the VEs;
- µCEP – the µCEP engine provides the processing capabilities of the VEs. Each VE needs to have access to it without interfering with the other VE's;

Although not a direct component of VEs, security components such as cryptographic primitives and/or key exchange mechanisms are also available to all VEs.

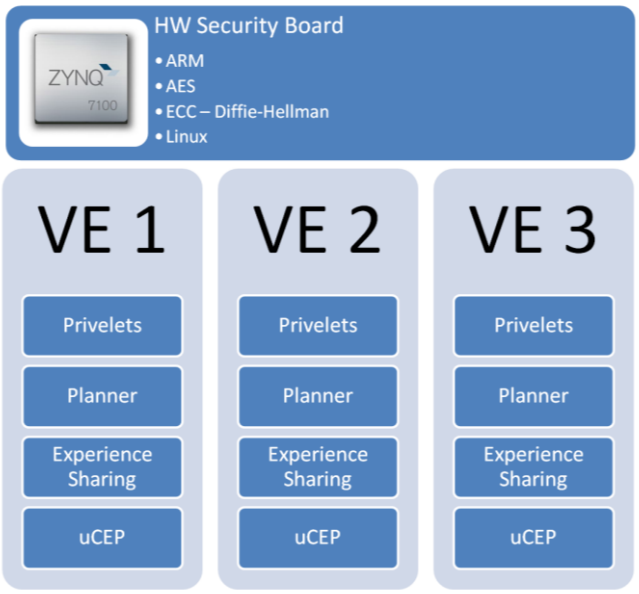


Figure 73: VE side COSMOS components

3.2.3.1 Testing Scenario description

While the functional integration of the components has been implemented in the testbed, we have deployed all the VE side components in a Raspberry Pi 2 board, which meets all the resources limitations, analytically described in section 4.2.4.1 of D7.6.2. In order to reach this goal, the first step was to collect the software dependencies of the components which are presented in Table 6.

Table 6: Software dependencies of VE side components

Name	Version	Component
Node.js	0.10.4	Node-RED Environment
Node-RED	0.10.36	Node-RED Environment
Java Runtime Environment	1.8	Node-RED Environment, Privelets, Planner, Experience Sharing
Jetty-Maven-Plugin	9.1.5.v20140505	Privelets, Experience Sharing
Apache-Jena	2.10.0	Privelets, Planner
Pellet-Jena	2.3.2	Privelets, Planner
FreeLan	1.1	Privelets
Rake	10.0.4	µCEP

Jetty-Maven-Plugin, Apache-Jena and Pellet-Jena are Java libraries so they are covered by installing JVM 1.8 in the raspberry pi. The next step was to install FreeLan and Node-RED which are key tools for the deployment. However, the major difficulty we faced during this process was the fact that up to now there is no available FreeLan package for ARM devices, so we had to compile it from source. To do so there was a need to install some packages/libraries which are prerequisite. These packages, alongside with the ones needed for Node-RED are presented in Table 7.

Table 7: Prerequisite packages for FreeLan and Node-RED

Freelan	Node-RED
python-setuptools	build-essential
scons	python-dev
libssl-dev	python-rpi.gpio
libcurl4-openssl-dev	nodejs
libboost-system-dev	
libboost-thread-dev	
libboost-program-options-dev	
libboost-filesystem-dev	
libboost-iostreams-dev	
nginx	
gcc-4.8	
g++-4.8	

3.2.3.2 Subsystem from D7.6.2 with integration points description

Figure 21 shows the Security, Privacy and Storage subsystem. In order to address the IP3, the VE developer needs to access the Hardware Security API. For Y2 the latter is a File I/O API, typical for Linux based operating system. Specifically, the cryptographic accelerator and the key agreement protocol are implemented as hardware modules on the Hardware Security Board. The modules are accessible through the main system bus at hardware level. At the operating system level a memory mapped driver model was developed, according to the

standard Linux driver model. The main control bits within the control and status registers as well as the data input and output registers have been exposed in Linux as files. Access to the modules is performed by classic File I/O operations from within the operating system. The drivers for the hardware security modules are loaded at boot. The File I/O driver model is exposed at user level therefore no superuser access is needed in order to access the security primitives.

An example of using the encryption/decryption API is:

```
#define AES_CMD_DECRYPT 0
#define AES_CMD_ENCRYPT 1

AES_DATA_T  data;

// Open file descriptor
aes_fd = open("/dev/aes", O_RDWR | O_SYNC);

// Call AES driver
ioctl(aes_fd, cmd, &data);     // cmd = AES_CMD_ENCRYPT or cmd = AES_CMD_DECRYPT
```

For Y3, the API will be implemented through Node-RED and will consist of three nodes:

- encryption node
- decryption node
- key agreement/exchange node

During Y3, we will investigate the capability to use a REST interface to call these nodes (e.g. POST operation to encrypt data).

The process demonstrated in Figure 21, refers to the communication between a VE and the COSMOS platform. However, in terms of security, the same API applies also to VE2VE communication depicted in Figure 100, which presents two different kind of intra VE communication; Friends Recommendation and Experience Sharing.

3.2.4. COSMOS components integration

3.2.4.1 CEP, Situational Awareness and Machine Learning Cooperation

3.2.4.1.1 Scenario description

In the city of Madrid, thousands of heterogeneous traffic sensors have been deployed on different locations across the city. These sensors provide real-time information about the traffic flow in the city such as average traffic speed, average traffic intensity, type of road, etc. Complex Event Processing has the potential to provide a distributed solution for analyzing, correlating and inferring high-level knowledge from this large amount of data in near real-time. In this sense, traffic speed can be analyzed using a simple rule as "if current speed is less than a threshold speed; generate slow speed event". However, the adjustment of these settings requires application developers and city administrators to have prior knowledge about the system, something which is not always possible, what poses a weak aspect in the solution. Additionally, every road segment has a different response: there might be some segments with speed restrictions while others are *free to ride*, so the threshold values will be different for both of them. In this way, at a high city level there will be hundreds of different road segments and it is almost impossible for application developers or administrators to understand the whole behavior of each individual road segment. Therefore, an IoT application with rules and conditions set with static thresholds will suffer severe performance degradations due to the dynamic nature of the analyzed environment. For example, in the event of bad weather or strong rain conditions, traffic will move slowly and those rules set for normal conditions may generate a false congestion alarm. In turn, threshold values should be different in such conditions. In addition, the response of the road is also changing with respect to time: a road might have a different behavior during morning rush hours as compared to quite night hours.

In order to address the aforementioned drawbacks, we propose to exploit historical data and use a novel method from Machine Learning domain in order to find optimized threshold values, which will be ingested in a specific µCEP Engine. Our proposed architecture is able to infer complex events from raw data streams in a distributed manner and is able to provide adaptive solutions at the same time. Figure 74 below shows the high-level architecture of our approach. Data collected from different traffic sensors is both analyzed in real-time and also stored efficiently in the Cloud Storage. Analytics on historical data generate optimized threshold values which are then fed into the µCEP Engine.

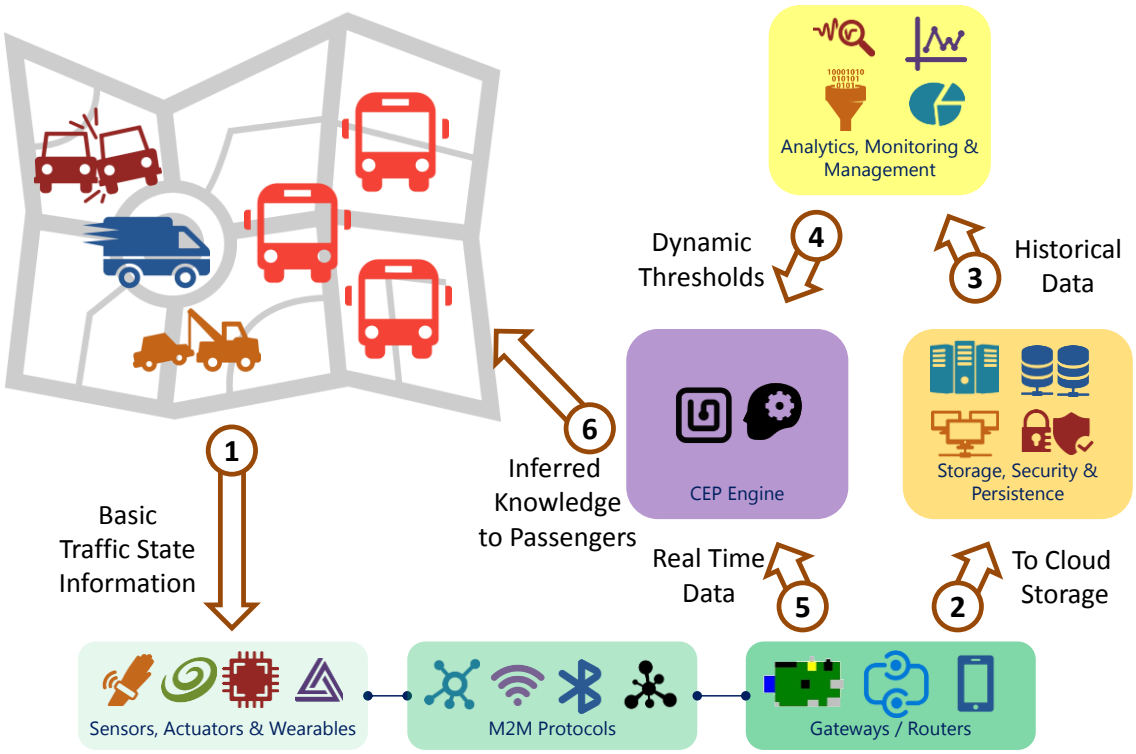


Figure 74: Madrid Traffic State Analysis Use Case

3.2.4.1.2 Subsystem from D7.6.2 with integration points description

The subsystem that relates to this scenario is shown in Figure 75.

The integration points require as a first step to provide the required rules with static thresholds that detect the evolution of the traffic state. This action has been reworked and is now implemented by means of the Situational Awareness guided wizard. Additionally, the application developer selects which of the thresholds are going to be dynamically updated, leveraging on the Machine Learning component its calculation. As with any other use case requiring the usage of a Complex Event Processing engine, specific configuration files are created by said wizard in order to collect data from datasources and provide value-added information out to the datasinks.

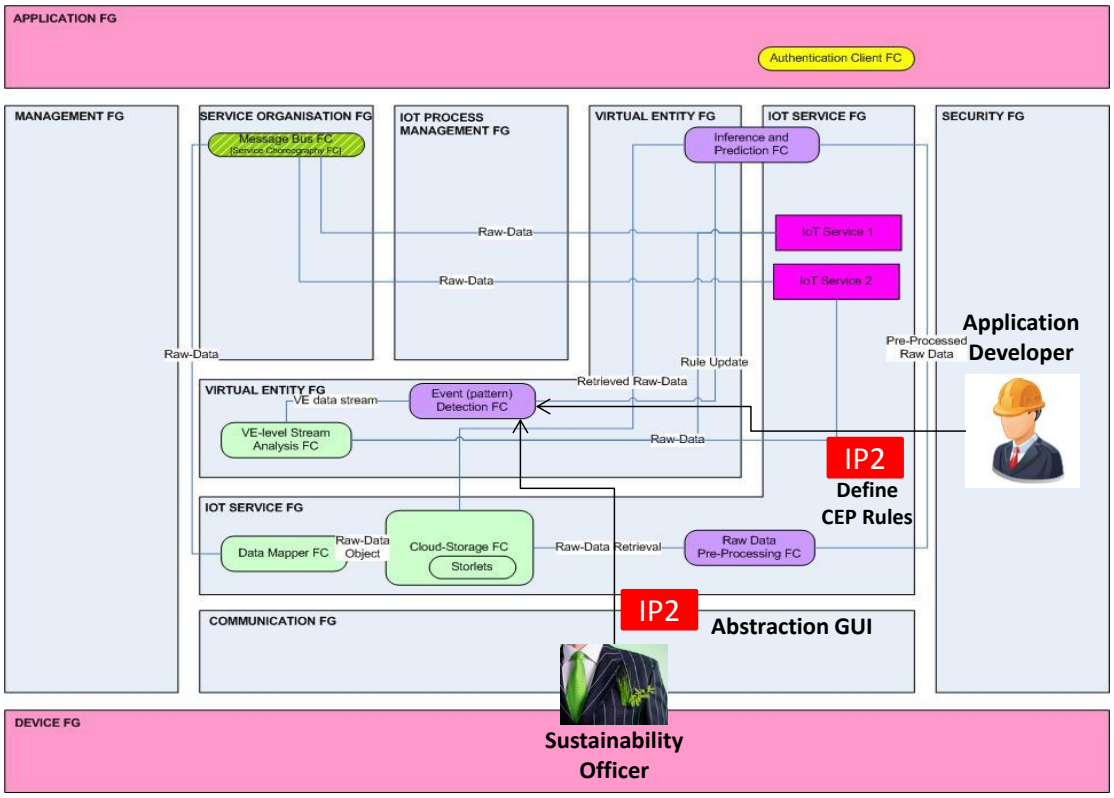


Figure 75: Traffic flow Subsystem Diagram

3.2.4.1.3 Message formats and configuration

Threshold values are calculated in Spark using Machine Learning libraries and published under a specific message topic in a JSON format. One instant of such message for new speed and intensity threshold values is shown below.

```
{
    "newThresholds": {
        "ruleID": "PM10005",
        "ThresholdIntensity": 130,
        "ThresholdSpeed": 45
    }
}
```

The information received in real-time by the µCEP Engine from the Message Bus is the same as the one being stored in the Cloud Storage, and therefore it has been already described in section 3.2.5.1.3.

The evaluation of the traffic state is being doing following the next rule:

```
detect TrafficInfo
where    diff(TrafficSpeed) < 0.0
      && diff(TrafficIntensity) < 0.0
      && (TrafficSpeed < ThresholdSpeed)
      && (TrafficIntensity < ThresholdIntensity)
      in [TUPLE_WINDOW];
```

Both *ThresholdSpeed* and *ThresholdIntensity* are dynamic variables that can be updated by the Machine Learning algorithm. When the above condition is evaluated to *true*, the following complex event is generated:

```
payload {
      int ValueSpeed = TrafficSpeed,
      int ValueIntensity = TrafficIntensity,
      float DiffSpeed = diff(TrafficSpeed),
      float DiffIntensity = diff(TrafficIntensity),
      string ts = ts,
      string tf = tf
};
```

An example of a real complex event is shown below:

```
{
    "TrafficState": {
        "tf": "14:17:50",
        "ts": "1442837870350",
        "DiffIntensity": "-89.583328",
        "DiffSpeed": "-59.722221",
        "ValueIntensity": "600",
        "ValueSpeed": "29"
    }
}
```

3.2.4.1.4 Activity Diagram

Historical data is accessed from COSMOS object storage using Spark. Data is extracted and filtered using Spark SQL queries. We implemented a Spark SQL driver which allows filtering the data close to the object storage, before it is sent to Spark. This significantly reduces the amount of data sent across the network. The driver uses metadata search to search for objects containing data relevant to a given Spark SQL query.

For applying Machine Learning algorithms on Spark SQL data, spark introduces a relatively new library called spark ML, which is still a work under progress offering only few basic algorithms. In order to use existing Machine Learning libraries from Spark MLlib, we need a wrapper in order to convert Spark SQL data frame into Resilient Distributed Datasets (RDD). Figure 72 shows different steps involved in the implementation of finding optimized threshold values.

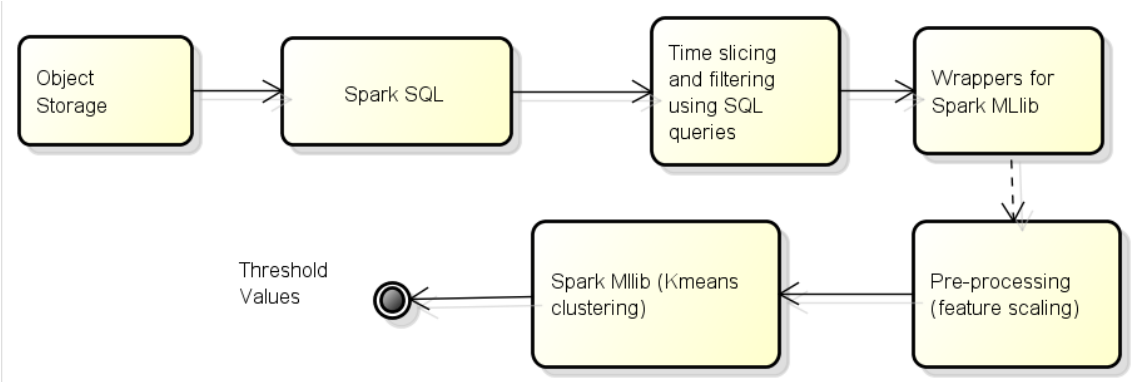


Figure 76: Steps for finding threshold values for µCEP rules

3.2.4.1.5 Subsystem Test case table

The subsystem test case appears in Table 8.

Table 8: Test case for the µCEP and ML cooperation

Test Case Number Version	MAD_02
Test Case Title	Traffic thresholds calculation
Module tested	Message Bus, Cloud Storage, Node-RED Flow, µCEP, Machine Learning
Requirements addressed	5.20, 5.21, 6.43, 6.5, 6.6
Initial conditions	Real-time traffic data feed Historical traffic information stored Node-RED rules with dynamic thresholds
Expected results	New thresholds calculated by Machine Learning µCEP traffic congestion detection rules are adjusted to actual conditions with new thresholds
Owner/Role	VE developer
Steps	Machine Learning (ML) retrieves data from Cloud Storage ML computes new thresholds µCEP receives new thresholds and updates rules
Passed	Yes
Bug ID	None
Problems	Lack of persistence of new thresholds upon µCEP Engine reboot
Required changes	ML component stores new thresholds in Cloud Storage µCEP sends bootstrap message after startup

3.2.4.1.6 Deployment Diagram

The following diagram (Figure 77) represents a particular deployment of the working components that build up this testing scenario. Alternatively, it is possible that certain components may be deployed close to the data source, for instance the µCEP instance.

3.2.4.1.7 Specific tests that may be needed

As the context of the application changes, threshold values might not be accurate and needs to be recalculated. We propose to use the silhouette index as a parameter to evaluate the quality of clusters. As new data arrive, we evaluate the quality of clusters and as the cluster quality drops we recalculate the threshold values. Also, we propose to give more weightage to most recent data so the clusters are more context-aware. We will implement these both functionalities in Year 3.

At this stage of development the µCEP Engine implements a way to update the thresholds on runtime, but it lacks the ability to persist these changes in its internal memory. In case of an unexpected reboot, the updated thresholds will be missed since the bootstrap process will read does ones stored in the DOLCE rule file. There are several ways of solving this issue: one is to modify the DOLCE rule file each time new thresholds are calculated; another one is to implement a 'bootstrap message' to be sent by the µCEP each time it starts, so a dedicated daemon may handle this message and send the latest calculated thresholds.

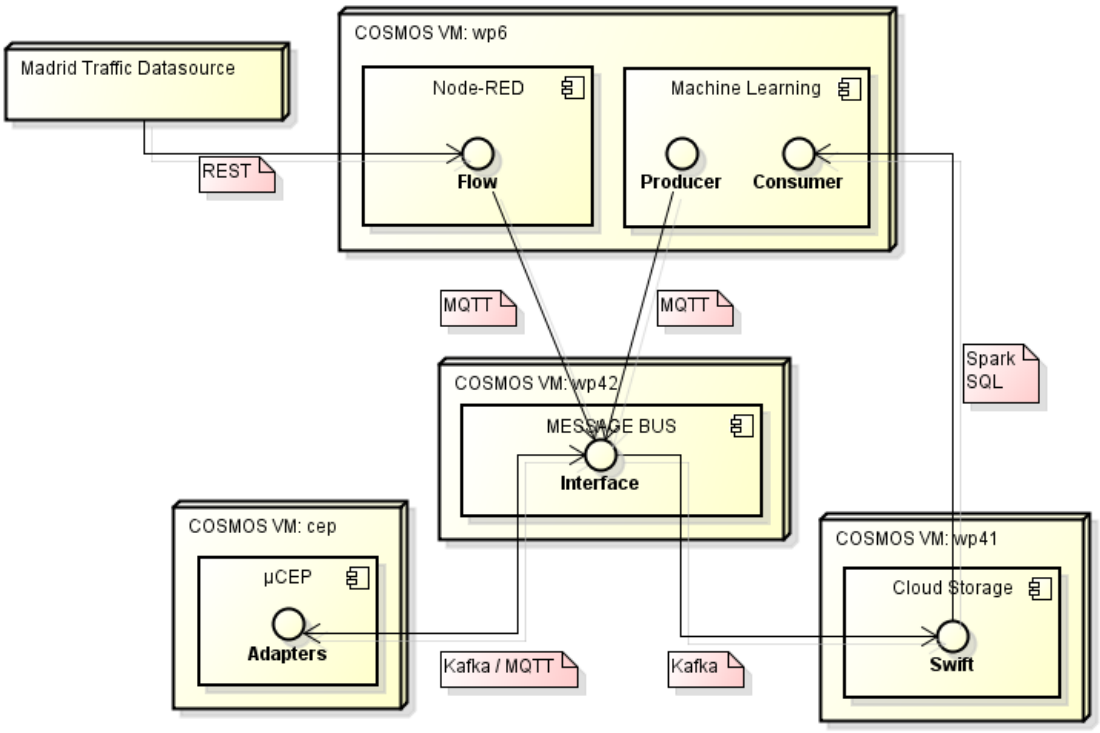


Figure 77: Madrid Traffic Analysis Use Case - Deployment Diagram

3.2.4.2 Proactive Experience Sharing

3.2.4.2.1 Scenario description

This scenario aims to demonstrate integration between the Functional Components of Experience Sharing, Situational Awareness and the Planner, in an environment of smart home management. The Use Case which has been chosen for the actual implementation is the Camden Flat ecosystem.

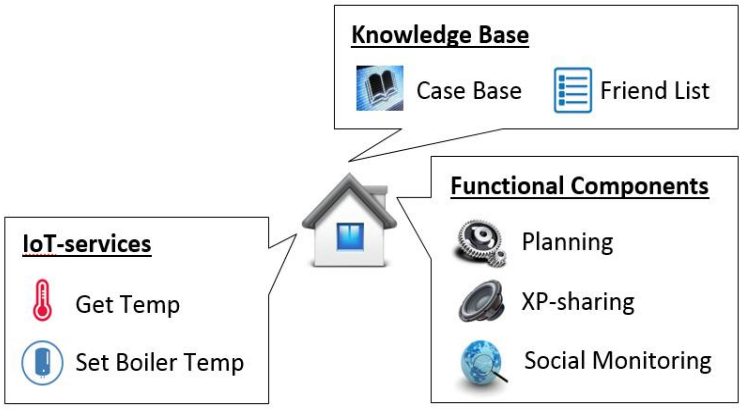


Figure 78: Camden Flat VE overview

Along with the components presented in Figure 78, the scenario makes use of the VE's Situational Awareness Functional Component, which contains the µCEP engine, implementing CEP techniques as presented in Figure 79.

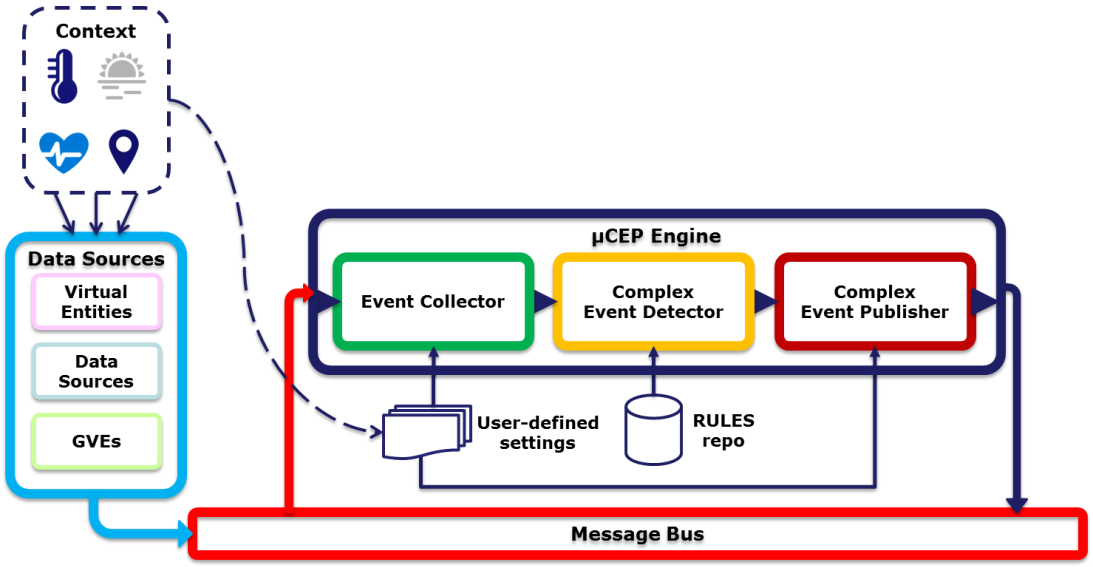


Figure 79: SAw FC and the µCEP Engine

The storyline chosen for this scenario is the deployment of COSMOS VE code on any given VE Flat, which makes use of existing data collection and aggregation mechanism framework presented in the EnergyHive site as it is maintained and provided by Hildebrand, in benefit of the Camden Housing Authority.

The incoming data is passed through a bridging overlay, which receives JSON files with relative sensor data through the use of MQTT technology. The functions act as a restructuring point for the information to be inserted into components such as the SAw FC through multiple exit points in this part of the flow. This restructuring will take place after specific configuration by the App developer.

The idea is to make use of an Application like logic which will detect sensor malfunctions, or aberrant data points in its simplest form, and by making use of more advanced Event Detection rules, it will also be possible to detect catastrophic events like a “house fire” and differentiate on them.

Afterwards, the detected Events will be analyzed on as per the methodology of the CEP technique as is implemented by the COSMOS µCEP engine instance running on the VE side. This analysis is what the µCEP Engine provides compared to other data stream analytic techniques, since it can generate an “output” (a Complex Event) containing some parameters that don’t appear in the original input (the Event), using the diff(), count(), average(), sum() functions.

Such analysis-provided Complex Events can be forwarded to the Planner FC of the VE, which will reason on them, by use of CBR techniques and decide on whether to proceed with internal corrective actions, disseminate the Event through Proactive Experience Sharing of the Experience Sharing FC, or both.

The use of CBR in this case, represents an enhancement of the original CBR implementation in the context of COSMOS, as the numerical Case handling of Y1 are not viable for Complex Event Handling. In this case the textual representation of Complex Events, in their entirety or at least in a much larger degree, necessitated the use of different CBR similarity functions for retrieved Case Base Events. Therefor it was decided to implement the Levenshtein Distance metric which can calculate distances between strings based on the number of steps it takes to turn

one string into another. This distance function calculates for example that the strings “COSMOS” and “COMSOS” have a distance of two, as there are two steps involved in turning “COSMOS into COMSOS”. In order to turn the distance into a percentage the length of the longer string is subtracted with the distance and the result is divided by the preceding length.

The Experience Sharing FC will receive the command to disseminate the Complex Event, into a different VE service as the one used for the classic Experience Sharing method. All VEs inside the Followers/Followees lists will be notified. The receiving VEs will have to check the Social Reputation of the originator VE in order to validate in a best effort fashion the truthfulness of the disseminated Complex Event.

The formulation of the configuration needed per Functional Component and per section of the above scenario, will be the responsibility of the Application Developer, as is demonstrated in the Sequence Diagram of Section 3.2.4.2.4. Pending the creation of a unified wizard, each configuration must be made individually.

A simple example to demonstrate the use of the above is that in the case of a detected sensor malfunction Complex Event, by the SAw FC’s µCEP engine, the Planner FC based on the Application Developer’s Cases, might decide to simply disregard the readings of the affected sensor by eliminating the abnormal values during incoming data processing, deactivate calls to affected Applications, send alerts to the End User or the maintenance authority, proactively share the Event with other VEs, or any combination of the above.

3.2.4.2.2 Subsystem from D7.6.2 with integration points

The basis for the creation of the Proactive Experience Sharing scenario is the Autonomous behavior of VEs as it was described in section 3.2.6 of D7.6.2. There have been, in the course of development, some modifications to the actual layout of actions described by Figure 80. Specifically, the person responsible for subscribing the Planner FC to the SAw FC’s Complex Event Publishing broker and the related topic management is henceforth performed by the Application Developer in the course of the configuration upload and management as is described in section 3.2.4.2.1.

So IP1 and IP2 in this case are unified as the configuration step of the scenario flow. Specifically they entail the uploading and usage of Apache Avro data schemas for data consumption and publishing inside the relevant MQTT brokers and their topics, as well as the actual topic concretization. Additionally as described in the figure, IP2 also entails the introduction of new Case structure in the Case Base of the Planner FC so as to support the new Event Reasoning concept.

The Integration Point 3 (IP3), contains the new bridge of the data from Camden Flats to the actual VE. The bridge itself is created in Node-RED, and receives MQTT data from the Hildebrand mosquito Broker, acts on them and republishes the modified data stream to the local mosquito Broker of the VE again through MQTT so as to forward them to any FCs interested.

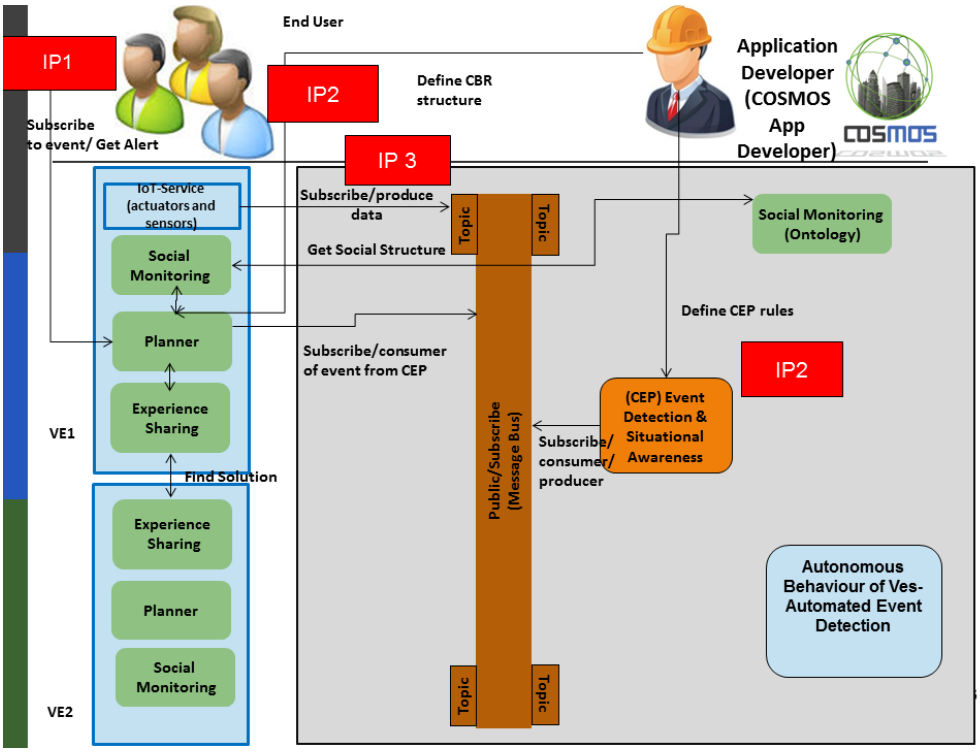


Figure 80: Autonomous Behaviour of VEs, basis for Proactive XP Sharing

In Figure 81, there is a clear demonstration of the primary flow of Complex Events between the µCEP engine's Publisher and other VE FCs like the Planner and the Experience Sharing. Additionally the Social Monitoring FC will also be involved in supplementing the process. In the following image the µCEP engine receives data that signify events, as the rules of event detection act upon them, and after analysis it publishes, through its Publisher parts, Complex Events which course through the internal MQTT broker. After that the Planner FC reasons on the Complex Events through the use of event specific CBR functions and implements action through the Solution retrieval, whether that is sharing the Complex Events, or any other actuation defined by the Application Developer.

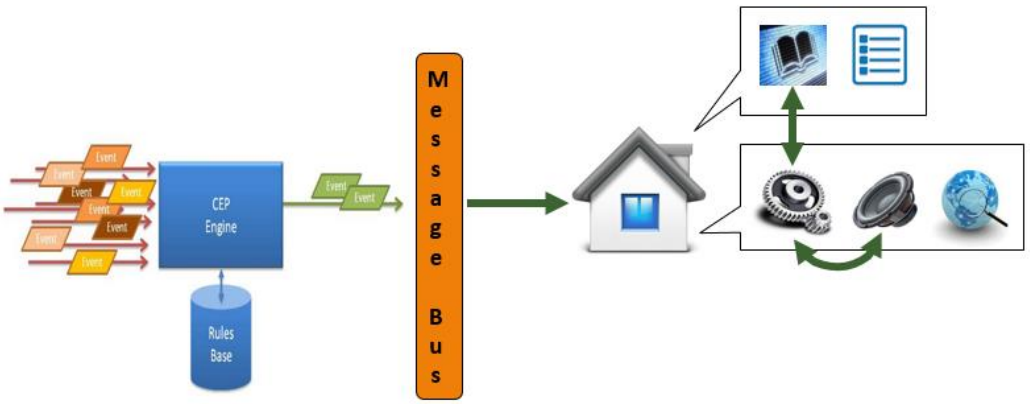


Figure 81: Flow of extracted Complex Events into other VE sided FCs

3.2.4.2.3 Message formats and configuration

This section marks the attempt to iterate through all the possible messages and communiques between the various components involved in this group of FC integration. Starting at the beginning of the scenario's flow, with the incoming VE data, there is the following input:

```
{
      "estate": "Oxenholm",
      "servertime": 1441362105,
      "hid": "aaaabbbbcccddd",
      "heatmeter": {
            "instant": 0,
            "flowTemp": 12,
            "returnTemp": 8,
            "flowRate": 222,
            "cumulative": 8888888
      },
      "sensors": [
      {
            "type": "window",
            "state": "open",
            "ts": 1441362105,
            "sid": 123
      },
      {
            "type": "window",
            "state": "closed",
            "ts": 1441362105,
            "sid": 321
      } ]
}
```

Based on the scenario of sensor malfunction the data input of the µCEP engine could be the entirety of the JSON Object, or a trimmed version containing only sensor information which corresponds to an Avro Schema of:

```
{
       "namespace": "cosmos.CEP.ComplexEvent.Input",
       "type": "record",
       "name": "VEData",
       "fields":[
              {
                     "name": "servertime",
                     "type": "string"
              },
              {
                     "name": "sensors",
                     "type":{
                            "type": "array",
                            "items":{
                                   "name":"sensor",
                                   "type":"record",
                                   "fields":[
                                          {"name": "type", "type": "string"},
                                          {"name": "state", "type": "string"},
                                          {"name": "ts", "type": "long"},
                                          {"name": "since", "type": "long"}
                                   ]
                            }
                     }
              }
       ]
}
```

From the above schema as it is defined in Apache Avro in order to provide integration with the Apache Spark framework used by the Cloud storage FCs, it is evident that the bridge removes superfluous information about heating data and merely passes on sensor information. Then the bridge can publish the data to a specific topic of the local VE MQTT broker (inter-VE component communication is performed through MQTT).

Topic management, in the sense of receiving and publishing data on them, is also the responsibility of the Application Developer in charge of configuration. However at this point, all configuration attempts are made a priori to the development of the scenario of integration.

After the reception of the data by the SAw FC, the µCEP engine will make use of the specific rules uploaded to it by the Application developer.

If and when the µCEP engine detects an Event which after analysis can lead to the desired Complex Event it can publish in the local VE MQTT broker a JSON message which describes the aforementioned Complex Event. A proposal for the message is the following Avro schema:

```
{
        "namespace": "cosmos.CEP.ComplexEvent.Publish",
        "type": "record",
        "name": "SensorMalfunction",
        "fields": [
                {"name": "hasComplexEventName", "type": "string"},
                {"name": "hasSensorType", "type": "string"},
                {"name": "hasSid", "type": "int"},
                {"name": "since", "type": "long"},
                {"name": "ts", "type": "long"}
        ]
}
```

Which validates an example event of:

```
{
        "hasComplexEventName": "SensorMalfunction",
        "hasSensorType": "temperature",
        "hasSid": 132,
        "since": 1433838989,
        "ts": 1441362105
}
```

The first three fields are the event name, the sensor type and the sensor id and the last two are the timestamp that the sensor last gave out a signal (from the data feed) and the timestamp of the Complex Event generation.

3.2.4.2.4 Sequence Diagram

In this section the sequence diagram corresponding to the scenario of Proactive Experience Sharing is presented. It is worthy to note that the sequence of actions described in Figure 82, are not specifying any kind of Complex Event. They are the agnostic set of steps any Application Developer must take along with all the actions performed by key FCs in the flow. The actual contents of the configuration input in the first three actions are what differentiate detected and handled events. The rest of the Sequence actions are as described in the previous subsections of 3.2.4.2

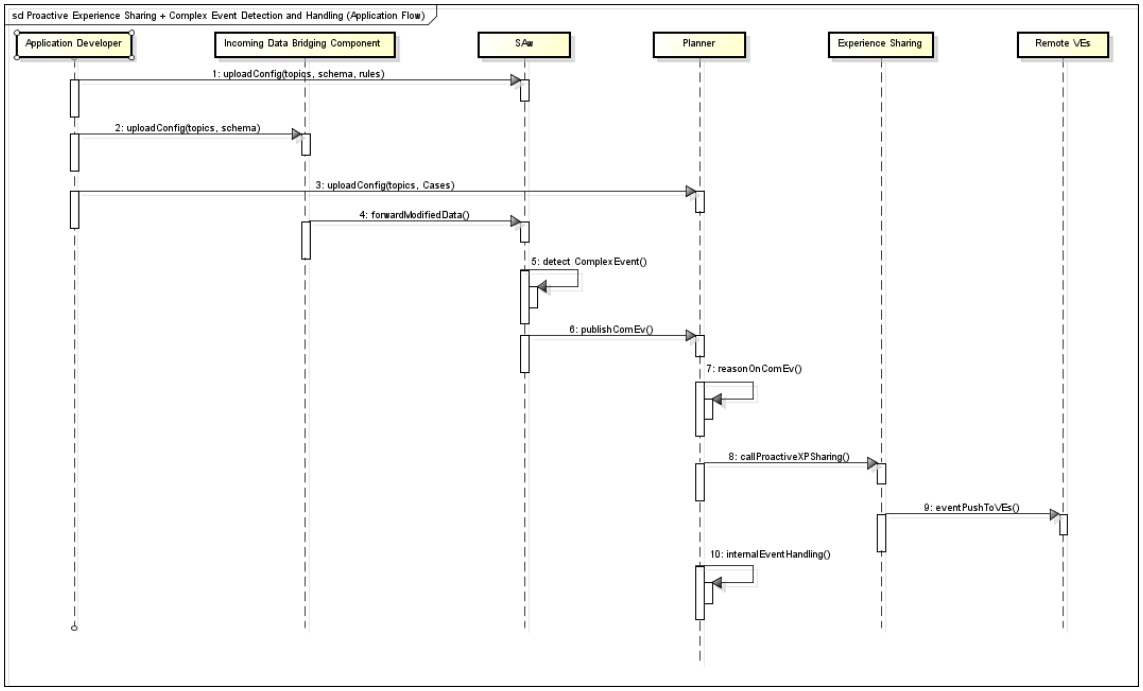


Figure 82: Proactive Experience Sharing Sequence Diagram

3.2.4.2.5 Subsystem Test case table

The test case for the Proactive Experience Sharing scenario appears in Table 9.

Table 9: Test case for Proactive Experience Sharing

Test Case Number Version	**PRO_01**
Test Case Title	Proactive Experience Sharing
Module tested	Data Bridging, SAw FC's µCEP engine, Planner FC, Experience Sharing FC, Social Monitoring FC
Requirements addressed	4.7, 4.9, 4.13, 4.14 5.10, 5.14, 5.15, 5.20, [5.22,5.23], [5.28, UNI. 015, UNI. 100, UNI.508], [5.29, UNI. 010, UNI. 704, UNI.706, UNI.708, UNI.715, UNI.719]
Initial conditions	Uploaded Configurations to Planner, SAw FCs and the Bridging Component, Connectivity between all Components
Expected results	Identify and handle Complex Event through Sharing it to other VEs, or handling it internally, on the basis of CBR Solution retrieval
Owner/Role	Application Developer VE developer
Steps	Start node red by executing command "node-red" in Linux console (Flows Begin) Start VE code by executing "java –jar OriginalFlatVE.jar" in cmd console Start reading data from stream (output in node.js console) Input events into SAw FC will be processed based on rules

	If a Complex Event is Detected, send µCEP engine publisher data to Planner FC (output in cmd console) Planner FC acts with CBR on the CE, finds a plan of action Based on Solution (output in cmd console) Share CE with Friend VEs (output in cmd console) VEs receive sharing of the CE (output in cmd console)
Passed	Yes
Bug ID	N/A
Problems	None
Required changes	None

3.2.4.2.6 Deployment Diagram

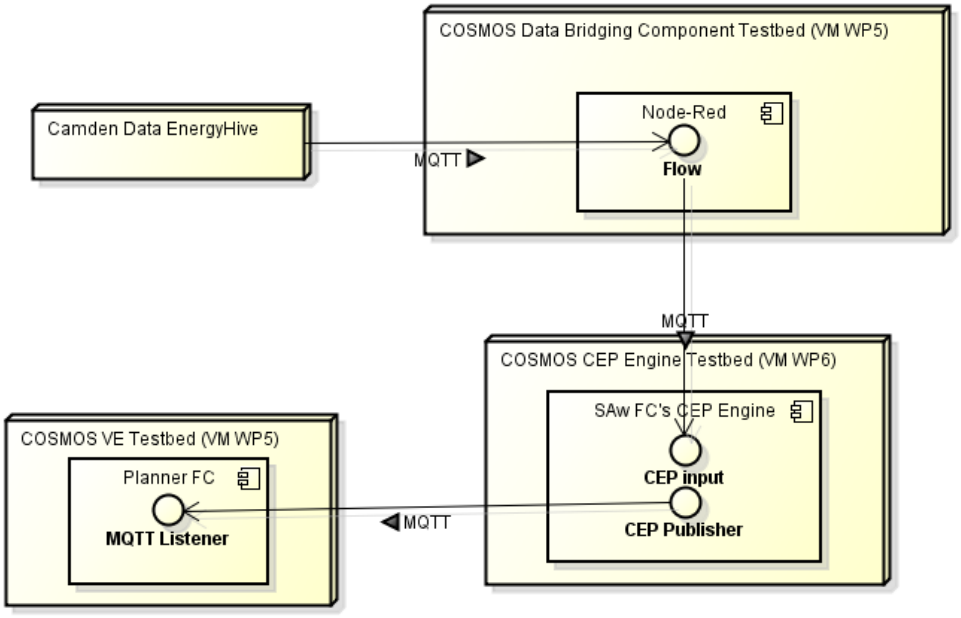


Figure 83: Deployment Diagram for the Proactive Experience Sharing Subgroup

In the Figure 83 deployment diagram, the EnergyHive Node represents the Camden Data input node from the Hildebrand Servers. The Node-RED flow is running in a COSMOS specific VM which NTUA can access and the VE code runs locally in the NTUA test bed. Additionally the µCEP Engine of the SAw FC is running on the WP6 VM of ATOS.

3.2.5. Data Feeds integration

3.2.5.1 Madrid data feed

3.2.5.1.1 Data feed description

From the amount of data feeds provided by Madrid City Council we have selected the one that provides traffic sensing data in real-time[1]. The sensorization of the traffic is done my means of a set of equipment installed in the streets, what allows counting the number of vehicles, obtaining the speed and calculating the intensity, among others. From a range of more than 10.000 traffic sensors, we have extracted and study a subset of 253 Measurement Points (MP) in order to showcase our work during Y2.

In this sense, for Interurban Traffic the following fields are provided in the feed:

Field	Description
ID	Unique identifier of the Measurement Point
Intensity	Intensity in number of vehicles per hour (a.k.a. traffic flow)
Occupancy	Percentage of vehicle occupancy in the measurement point
Load	Calculated from intensity, occupancy and characteristics of the street, this parameters informs about the traffic load level
Service Level	Not implemented
Speed	Average speed of the vehicles detected in the last "integration period" (usually 5 minutes)
Error	-1 means the values are not valid

An example of the received Measurement Point message (Punto de Medida in Spanish; 'pm') is the following:

```
<pm>
    <ID>PM10005</ID>
    <intensity>1740</intensity>
    <occupancy>15</occupancy>
    <load>74</load>
    <serviceLevel>0</serviceLevel>
    <speed>70</speed>
    <error>N</error>
</pm>
```

[1] Traffic intensity in Real-Time – Madrid Opendata Portal: http://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=02f2c23866b93410VgnVCM1000000b205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD

3.2.5.1.2 Subsystem from D7.6.2 with integration points

The following picture is referenced from Section 3.2.1 - Data Feed, Annotation and Storage Subsystem of Deliverable D7.6.2.

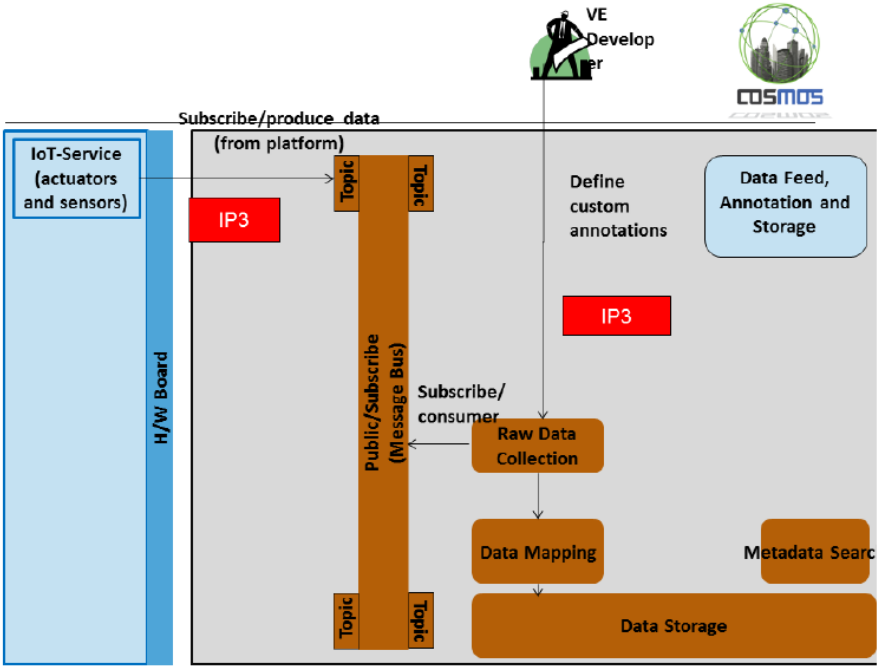


Figure 84: Data Feed, Annotation and Storage Subsystem

3.2.5.1.3 Message formats and configuration

Given that the Opendata portal does not need any credentials to retrieve this specific feed, the operation to get the information of the Measurement Points in the city is as simple as triggering an HTTP Request to http://datos.madrid.es/egob/catalogo/202087-0-trafico-intensidad.xml.

The original, raw payload received from the OpenData portal comes in the form of a single XML file grouping all the Measurement Points. In order to make them more usable by the rest of COSMOS components, a Node-RED flow reformat them to a JSON object, besides inserting the timestamp (ts and tf) of the measurement, what aids later processing.

```
{
    "ID": "PM10001",
    "intensity": "3780",
    "occupancy": "29",
    "load": "65",
    "serviceLevel": "0",
    "speed": "32",
    "error": "N",
    "ts": 1441368443937,
    "tf": "14:07:23"
}
```

After that, each Measurement Point is served, as a JSON Object, to the Message Bus, one by one, to the topic */cosmos/Madrid/TrafficFlow/MP.*

Finally, an Apache AVRO schema is used to store this data feed into the Cloud Storage. The following schema includes additional fields for other kind of messages that may be received in the future:

```
{"namespace": "cosmos",
  "type": "record",
  "name": "TrafficFlowMadridPM",
  "fields": [
    {"name": "codigo", "type": "string"},
    {"name": "descripcion",  "type": ["null","string"]},
    {"name": "accesoAsociado", "type": ["null","long"]},
    {"name": "intensidad", "type": "int"},
    {"name": "ocupacion",  "type": "int"},
    {"name": "carga", "type": "int"},
    {"name": "nivelServicio", "type": "int"},
    {"name": "velocidad",  "type": ["null","int"]},
    {"name": "intensidadSat",  "type": ["null","int"]},
    {"name": "error", "type": "string"},
    {"name": "subarea",  "type": ["null","int"]},
    {"name": "ts", "type": "long"},
    {"name": "tf", "type": "string"}
  ]
}
```

The Data Mapper (Secor with our extensions) collects many json objects conforming to such a schema and aggregates them into a single object. It annotates these objects with metadata such as minimum and maximum values, which is used to later optimize Spark SQL queries. The list of fields for which this metadata should be collected is stored in the secorSchema container in an object with name <topic_name>.metaKeys. For example, if we GET the TrafficFlowMadridPM.metaKeys object we see four fields for which metadata is collected.

Figure 85: Example of Data Mapper request

3.2.5.1.4 Activity Diagram and Node-RED flow

The following diagram and Node-RED flow represent the lifecycle of the information from the Madrid Opendata feed to the MessageBus, and indirectly to the Cloud Storage.

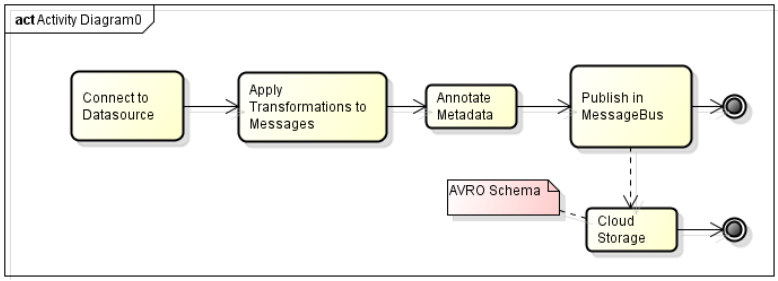


Figure 86: Madrid data feed Activity Diagram

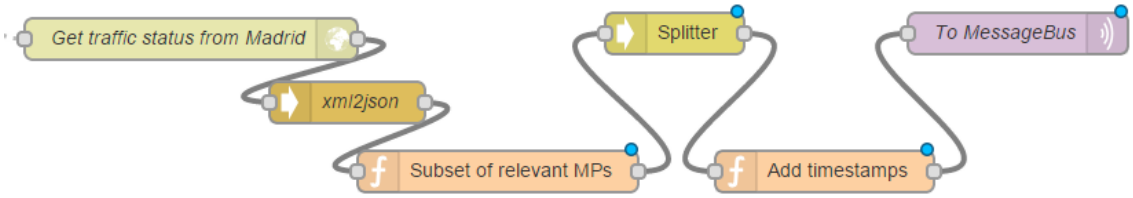


Figure 87: Madrid data feed Flow

After this, interested components would be able to access this information in two ways:

- Real-time data can be received by subscribing to the appropriate MB topic.
- Historical data can be retrieved on-demand using the appropriate Spark methods.

3.2.5.1.5 Subsystem Test case table

The test case for this subsystem appears in Table 10.

Table 10: Test Case for the Madrid Data Feed incorporation

Test Case Number Version	**MAD_01**
Test Case Title	Traffic feed storage
Module tested	Message Bus, Cloud Storage, Node-RED Flow
Requirements addressed	4.1, 4.2, 4.9, 6.1, 6.5
Initial conditions	A functional Message Bus AVRO Schema shared with Cloud Storage Node-RED instance
Expected results	Traffic data made available in real-time through Message Bus Traffic data stored for later analysis in Cloud Storage
Owner/Role	VE developer
Steps	Retrieve data from Madrid datasource every 5 minutes Parse data, include timestamps Publish into Message Bus Swift receives data and stores it Message Bus re-publish latest data to any subscriber
Passed	Yes
Bug ID	None
Problems	Desynchronization of datasource causes data corruption
Required changes	Check data corruption to retrieve another request

3.2.5.1.6 Deployment Diagram

The instantiation of the involved components spans across several Virtual Machines as can be seen in the following figure.

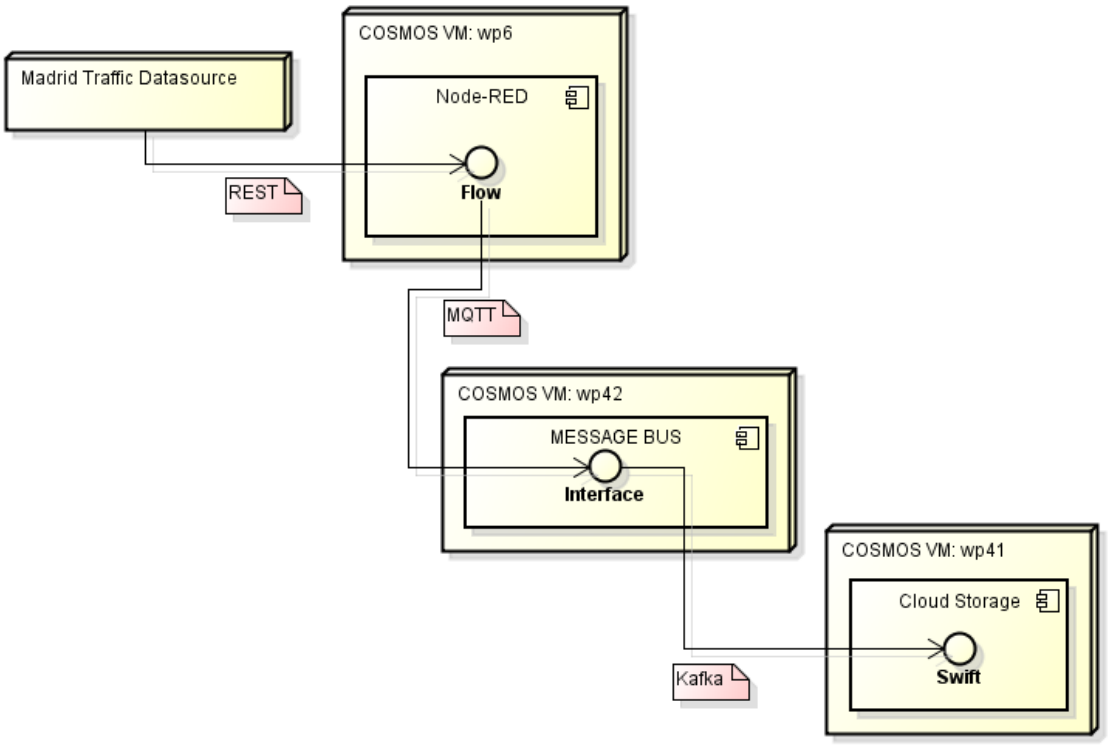


Figure 88: Madrid Data Feed Deployment Diagram

3.2.5.1.7 Specific tests that may be needed

It turns out that the XML file containing the Measurement Point data is updated by the Madrid OpenData system every 5 minutes, thus in case the Node-RED flow requests it at the same time in which is being updated it will get a malformed response. In this sense, certain mechanisms are being used in order to synchronize flow requests and XML updates. Although it seems the XML file is updated following a regular pattern, from time to time those updates occur unexpectedly. In such sense, in order to validate that data retrieved from the OpenData portal is valid, a specific parsing must be done every time to verify the integrity of the requested XML file and retrieve another request in case the data is corrupted.

3.2.5.2 Camden data feed

3.2.5.2.1 Data feed description

As described previously in section 3.2.4.2, the data feed from the Camden housing authority passes through the Hildebrand Servers where they are processed and transmitted out on an MQTT feed in a readable form. The JSON structure of the data being sent out on MQTT has been designed so that COSMOS services and components can interact with it easily.

A sample of streamed data can be found in subsection 3.2.4.2.3. The topics used for the publishing of the data are of the form of: “cosmos/{{estate}}/{{flat_hid}}”, where estate is one of the three estates that Camden provides data on (Oxenholm, Dalehead, Gillfoot) and flat_hids are the flat unique ids.

The actual MQTT endpoint for receiving the data is “tcp://mqtt.energyhive.com:1883”, with a set of credentials which are updated in relatively frequent intervals. This allows use of specific input MQTT nodes in the flow based programming tool Node-RED, which can connect to the endpoint and receive the data, or any other input method, such as the one used by the Planner FC through the use of the Eclipse Paho library for MQTT communication.

Even though the Camden Data feed was also integrated during Y1, the process was adapted further in Y2 in order to take under consideration the new feed with the enriched sensors available for Y2.

3.2.5.2.2 Subsystem from D7.6.2 with integration points

Same as in the previous section (0)

3.2.5.2.3 Message formats and configuration

The description of the message format can be read in subsection 3.2.4.2.3, as well as section 5.1.2 of D7.2.2. Each sensor’s data will appear at most once in each publication of Hildebrand Server data on a specific topic. This means that messages may contain info from as many sensors as are updated since the previous publish. Updated sensors do not indicate a change in value or state. In fact, updating and maintaining a similar as previous reading is encouraged as too much time between updates in sensors might lead into detecting a false positive Sensor Malfunction Complex Event by COSMOS.

3.2.5.2.4 Activity Diagram and Node-RED flow

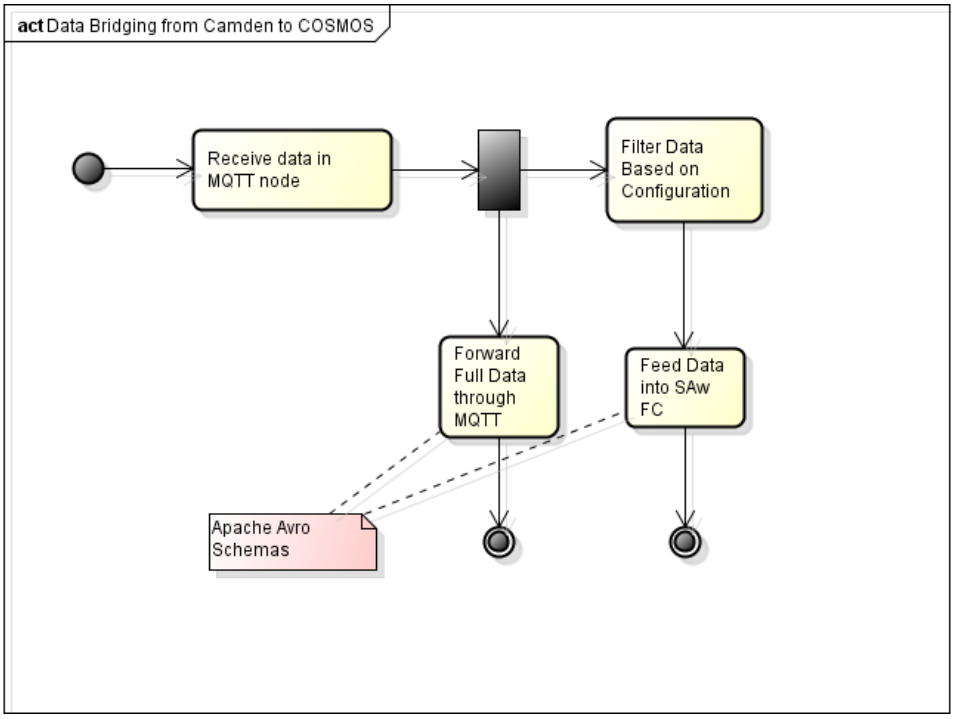


Figure 89: Data Bridging Activity Diagram

The diagram in Figure 89 demonstrates the way data is routed and acted on by the Camden specific Bridging Component of COSMOS. The flow described above is implemented using Node-RED and is represented in Figure 90.

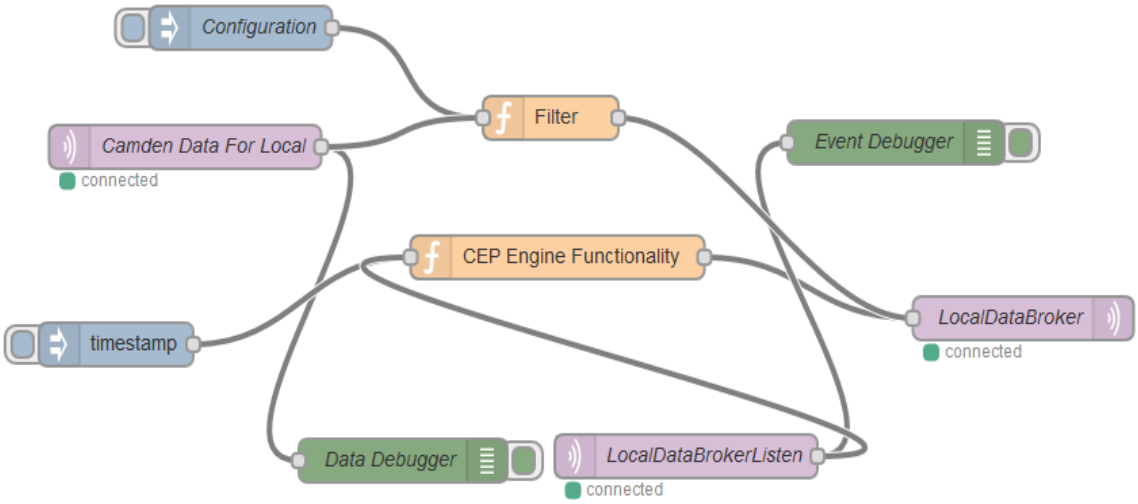


Figure 90: Data Bridging Flow and Connectivity

3.2.5.2.5 Subsystem Test case table

The test case for the Camden Data feed appears in Table 11.

Table 11: Test Case for the Camden Data Feed

Test Case Number Version	**CAM_01**
Test Case Title	Camden Data Extraction
Module tested	Node-RED flow, Java MQTT libraries
Requirements addressed	4.1, 4.2, 4.9, 6.1, 6.5
Initial conditions	Node-RED flow, Hildebrand Server MQTT Broker, VE code
Expected results	Receive JSON formatted data in Node-RED flow Successfully filter fields Forward Message to Planner FC and µCEP representing function Succeed in reading data from Planner FC
Owner/Role	VE developer
Steps	Start node red by executing command “node-red” in node.js console Start VE code by executing “java –jar OriginalFlatVE.jar” in cmd console Retrieve data from Camden data source every 10 seconds in each topic (one topic per flat) (node-red MQTT node, automatic) Parse data, extract estate, hid, ts, cumulative (function node automatic, output in node.js console) Publish into VE Message Bus, maintaining topic structure (MQTT node output) Subscribe into MB by the VE’s Planner FC (output in cmd console) Retrieve Data successfully for future use (output in cmd console)

Passed	Yes
Bug ID	None
Problems	None
Required changes	None

3.2.5.2.6 Deployment Diagram

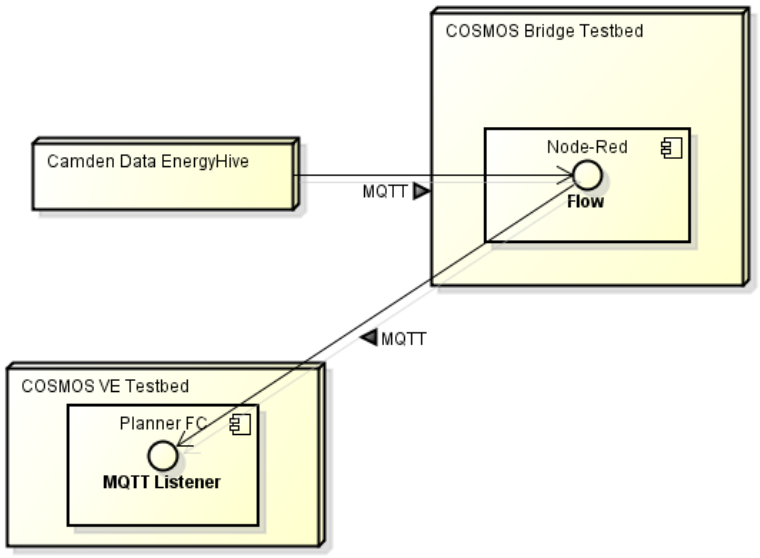


Figure 91: Deployment Diagram for Camden Data Feed

In Figure 91 deployment diagram, the EnergyHive Node represents the Camden Data input node from the Hildebrand Servers. The Node-RED flow is running in a COSMOS specific VM which NTUA can access and the VE code runs locally in the NTUA test bed.

3.2.5.2.7 Specific tests that may be needed

Further work needed in the Node-RED flow to implement more agnostic data filtering methods. Alternatively a more flexible schema can prevent possible data losses. No data corruption noticed regardless of the frequency of message reception.

3.2.5.3 Taipei data feed

3.2.5.3.1 Data feed description

Institute of Information Industry (III) have formed a Smart Network System Institute which provides solutions for energy management to hundreds of houses in Taipei. They provide the users with smart sockets and smart strips which measure real time electricity consumption information. The real time energy data is available online with the help of smart gateways. It includes different characteristics such as active power, current, connection status and time stamp. It also includes other data fields as well such as meterCapability and NetType which are redundant for our work in COSMOS and hence filtered out with the help of Node-RED flow. A summary of data fields which we used for COSMOS platform is given below.

Field	Description
dev_id	Unique identifier for the socket
gw_dev_id	Unique identifier of the gateway with which socket is connected
Info_id	Identifier for the data feature (such as “3” for Active Power)
Info_value	Actual measure of the data feature being reported
Info_desc	Name of the data feature
Report_time	Time for reporting the value to gateway

3.2.5.3.2 Subsystem from D7.6.2 with integration points

Same as section 0.

3.2.5.3.3 Message formats and configuration

Online data from III is available in JSON format which provides real-time information of different fields of electricity data. A small part of online data is shown below which measures the active power as “24.85”for the device with id “RS02000D6F00008E9D70”.

```
{
dev_id: "RS02000D6F00008E9D70"
gw_dev_id: "RS90000D6F000174532C"
-2:{
      info_id: "3"
      info_value: "24.85"
      info_desc: "ActivePower"
      report_time: "1442398227000"
}
```

3.2.5.3.4 Activity Diagram

The following diagram represents the lifecycle of the information from the III Taipei endpoint to the MessageBus, and indirectly to the Cloud Storage.

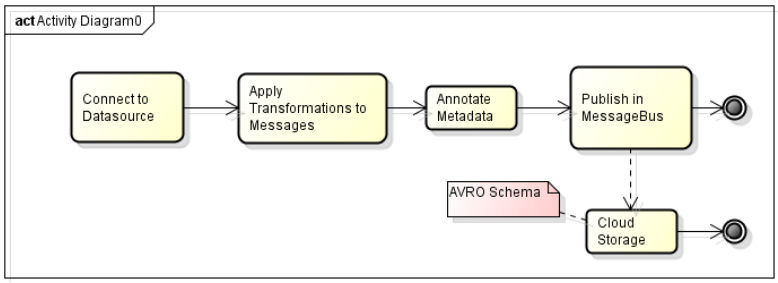


Figure 92: Taipei data feed Activity Diagram

3.2.5.3.5 Subsystem Test case table

The test case for the Taipei data feed appears in Table 12.

Table 12: Test Case for the Taipei Data Feed

Test Case Number Version	III_01
Test Case Title	III feed storage
Module tested	Message Bus, Cloud Storage, Node-RED Flow
Requirements addressed	4.1, 4.2, 4.9, 6.1, 6.5
Initial conditions	A functional Message Bus AVRO Schema shared with Cloud Storage Node-RED instance
Expected results	Electricity consumption data made available in real-time through Message Bus Electricity consumption data stored for later analysis in Cloud Storage
Owner/Role	VE developer
Steps	Retrieve data from III URL every 3 seconds Publish into Message Bus Swift receives data and stores it Message Bus re-publish latest data to any subscriber
Passed	Yes
Bug ID	None
Problems	Desynchronization of data source causes data corruption
Required changes	Check data corruption to retrieve another request

3.2.5.3.6 Deployment Diagram

The instantiation of the involved components spans across several Virtual Machines as can be seen in the following Figure 93.

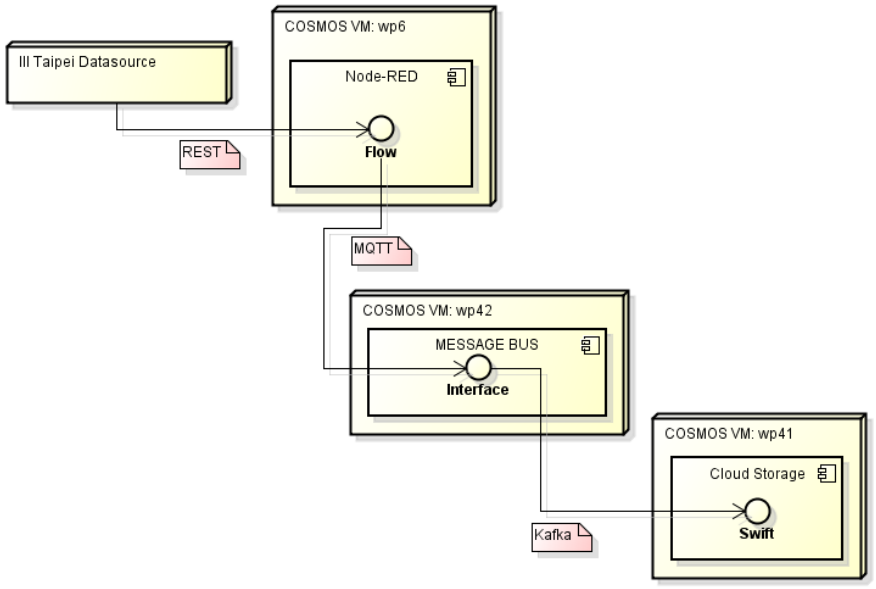


Figure 93: Deployment diagram for III data feed

3.2.6. Madrid Scenario Application

The purpose of this section is to highlight how the integration is performed between the main EMT services, as they are described in D7.3.2, and the COSMOS technical components side. The overall sequence is intended to indicate how the two sides can be combined, enable the functionality range that is envisioned for the Madrid Scenario Application and serve as a template for future additions.

The involved goals and subgoals, as these are defined in D7.6.2, for this section are the following:

- COSMOS Platform integration and especially the Data Management and Analytics subgoal, with relation to the Data Feed inputs and the µCEP, SA and ML process defined in 3.2.4.1, given that this provides the main functionality in terms of predictions and event identification
- Data Model template and especially the subgoal of Data Fields definition, mainly with relation to exchanged messages with the EMT platform and potential values
- Application Definition, Creation and Deployment and especially the Application Archetypes Definition, in relation to how the flow can be abstracted and then extended with other functionalities, and Application Scenario Concretization, with relation to the concrete predictions, events and component instantiations that are needed for the specific UC.

3.2.6.1 Scenario description

The envisaged scenario for this case appears in Figure 94. By utilizing the information provided in D7.3.2, COSMOS needs to build the Data Feed and Data Output layers, corresponding to the defined interfaces from EMT. These are generic interfaces and may be used in any case of performed I/O towards the EMT services. Furthermore, COSMOS functionalities need to be bridged to the data provided by the Data Feed layer e.g. in terms of the MB component, from which on the functionalities described in the previous chapters(e.g. 3.2.4.1) can kick in. Events identified from these internal COSMOS sequences can then be directed towards the Data Output layer so that they can be portrayed in the SP portal or mobile app front ends. For each different event/notification, a specific flow may be needed (or adjustments to the basic flow) that will inject the specificities of that feature (e.g. specific µCEP rules, specific ML models etc). The specific app logic may be used in order to filter events, create appropriate messages or adapt notifications for users.

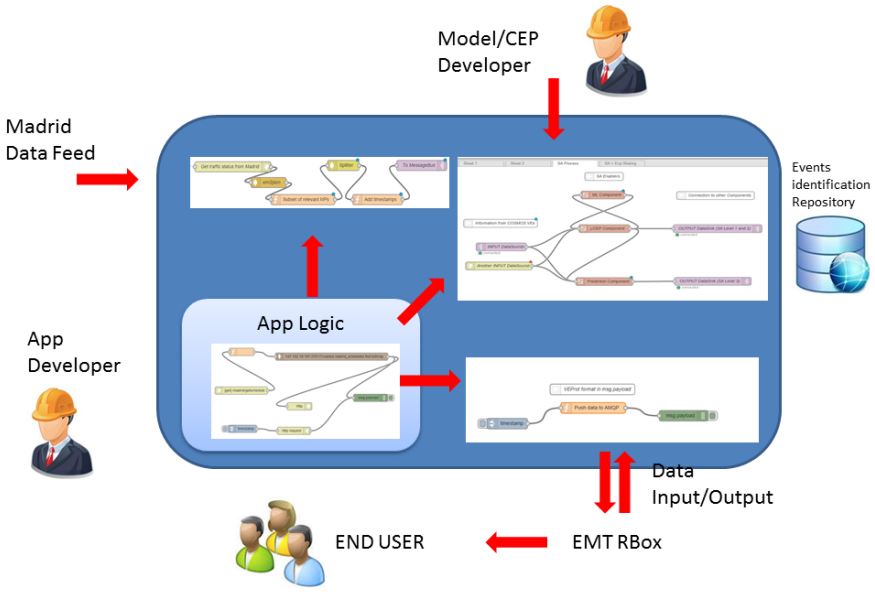


Figure 94: Madrid Scenario Integration

The basic scenario for Y2 includes the following concrete steps, in agreement with the generic steps identified in Section 5.3.1 of D7.6.2:

1) The SP or CG logs in the EMT platform and creates the planned route. This is stored inside the RB server.

2) COSMOS goal is to provide functionalities which will be responsible for identifying various events with relation to a route and a SP id. In order to do so, we need to expose this functionality that will take as argument the SP id under consideration and following this input to retrieve the user related data from RB through DDP. Indicative tables include:

- ROUTESMad table of planned routes
- ROUTESMAD.bustrack includes all bus positions which pass through planned routes
- ROUTESMAD.usertrack includes all user positions when the user is on route (if the user is into the bus, the event contains data related to bus)
- ROUTESMAD.chkpoint includes user pass through by check point of route plan.

Information from the EMT side is populated in real time using DDP through observers from the Control Fleet system. EMT feeds usertrack and chkpoint when an SP (through the mobile app) sends events to the RB system using amqp. The DDP Interface[9] for getting info may be used. With regard to message formats for registering, these are included in Section 3.2.6.3.

3) Once we have the available information, specific processing per event case may be performed in the COSMOS environment, either inside the COSMOS app logic or through feeding to the MB (e.g. in case of identification that the user is off route). For example, in the case of traffic events identification, we need to decide if the user tracked routes (as reported by the ROUTESMAD.usrtrack and published through the Data input layer to the MB) go past a number of Measurement points of Madrid traffic data (e.g. through the in-area function of µCEP), for which we have identified prediction models and CEP rules boundaries, as mentioned in Section 3.2.4.1. For these MPs we need then to register to the COSMOS MB and listen on events related to them.

4) Once an alarm is identified from Step 3, we need to push it to MsgOut (OUTPUT Layer). For sending data into RB we use a Rabbit MQ Server (amqp.emtmadrid.es), targeting at the ROUTES.alarms collection. For the message a suitable format has been defined (Section 3.2.6.3). The format includes the notification text shown to the user, as well as other related information (e.g. could include GPS location of an identified traffic jam point). An alternative process is to display all traffic points for which there is an identified problem, so that the user takes them under consideration during route planning. In this case the Rbox component needs to have registered to the COSMOS platform and receive notifications on all available MPs.

5) EMT subscribes using Meteor DDP to the ROUTEMAD.alarms and triggers messages to CG portal and SP app. Notification is shown in the Web interface of the SP app.

It must be stressed that the AMQP functionality is only for sending data from external systems to the EMT RB layers. For getting data the DDP system exposed at rbmobility.emtmadrid.es:3333 must be used.

3.2.6.2 Subsystem from D7.6.2 with integration points description

As mentioned in Section 3.2.6, a large number of subgoals (and their related subsystems) are included in this case. In order not to repeat information, given that the associated subsystems and their integration points have been already described in the related chapters, we include only the generic centralized archetype application in this case (Figure 95). There are two integration points of type 2 (Application developer interactions), mainly with relation to how the App Developer defines the app logic internally in the COSMOS environment (described in detail in Section 0) and how they communicate to the Application client the information. The latter hides also an indirect IP1 point (interface with end users), however from the platform's point of view this is dealt with from the external system layer. In the specific case the two main integration points refer to how information is exchanged with the EMT Rbox, either to retrieve data (Data Input) or to push data (Data Output). These functionalities are expected to be given as Node-RED sublfows, which implies their iterated usage in generic and arbitrary application scenarios. As ax example, we illustrate the subflow of the publication of data from COSMOS to EMT Rbox, for showing notification in the end user GUIs. This subflow may be used in any case that the Application Developer needs to redirect information towards the RBox system, by introducing the relevant message (specifications of the message format are included in 3.2.6.3) in the msg.payload input of the "Push to AMQP" node (Figure 96). This of course may be combined with previous application logic inside Node-RED that will prepare the message, based on cooperations also with already available flows for linking with the MB, combining CEP and ML etc.

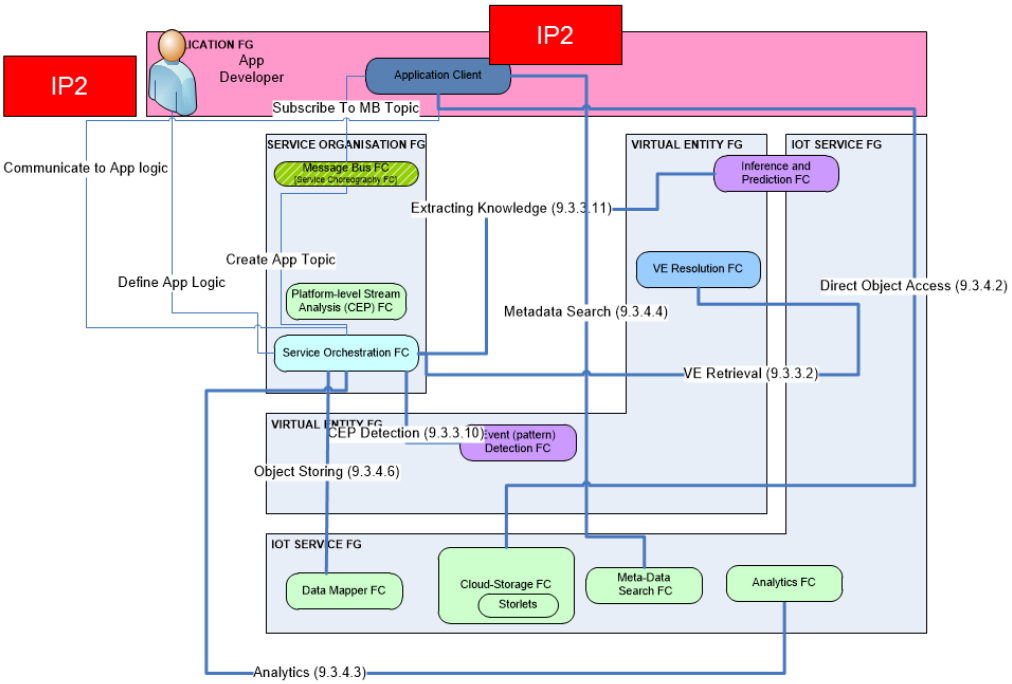


Figure 95: Centralized archetype subsystem as used in the Madrid Scenario Application

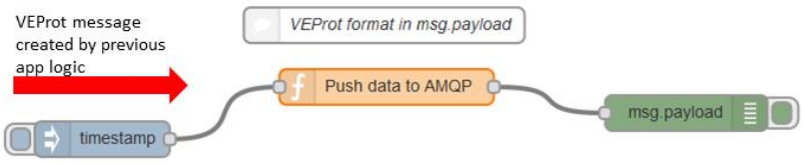


Figure 96: Generic Data Output Flow towards the RBox system

3.2.6.3 Message formats and configuration

The message formats for this section relate to the Data Input and Output layers from COSMOS to the EMT system.

For the Data input layer, a login must be performed through DDP with specific credentials. This subscription is either on an individual user or overall users retrieval basis. If the initial login is performed as the individual user and a registration is performed to an according table (e.g. through a command similar to ¨client.subscribe('ROUTESMAD.usrtrack.user'), only the new events from this user will be redirected, following the format defined in Figure 97. Alternatively, if the login is as a user with higher access rights and use a variation of the command (¨client.subscribe('ROUTESMAD.usrtrack.all') , events from all users will be forwarded. Thus filtering of the retrieved json objects based on SP id for a specific user should be performed based on the needed ids. At this stage the data from ROUTESMAD.usrtrack and ROUTESMAD.userplans are expected to be used. Also direct subscriptions using filters may be achieved. For example for subscribing to a set of documents by id: client. subscribe('ROUTESMAD.eventpos.custom'",[id1,id2,id3...]) or for subscribing a document by filter (subscribe('ROUTESMAD.eventpos.custom',{'nameRouteUser':'jmendez'})).

```
{
    "_id" : "5602a883886d5e19ec918613",
    "instant" : "2015-09-23 01:13:41.425951",
    "idRoute" : "560275d32c2826228c0349a5",
    "geometry" : {
        "type" : "Point",
        "coordinates" : [
            -3.70462989807129,
            40.4125099182129
        ]
    },
    "idSesion" : "840591758",
    "busData" : null,
    "dayWeek" : "X",
    "dayType" : "LA",
    "nameRouteUser" : "jmendez",
    "order" : "0"
}
```


Figure 97: Data format for ROUTESMAD.usertrack responses

For the Data Output layer, the message content format for pushing notifications in the RB server (through the AMQP protocol) appears in Figure 98.


```
{
"target":"datagramServer",
  "vep_data": [
    {
      "dataLayer": {
        "layer": "ROUTESMAD.alarms",
        "idRoute": "55d1801ffb954f08f8f1489f",
        "idSesion": "21d1701fab914f08c8e1438e",
        "instant": "2015-08-10 14:53:02.462053",
        "codeAlarm": "10",
        "textAlarm": "Traveler outside the planned route",
        "levelAlarm": "W",
        "geometry": {
          "type": "Point",
          "coordinates": [
            "-3.69138338267",
            "40.4211505276"
          ]
        }
      }
    }
  ]
}
```


Figure 98: Push notification to RB server datagram format

The description of the individual fields follows:

1."target":"datagramServer" -> this section is mandatory and our AMQP system is using this data for sending the data to the msgout layer (layer towards SP portal or mobile app).

2. “Vep_data”: this object is mandatory and contains the data sent to ROUTESMAD.alarms.

3. “Instant”-> Mandatory timestamp in UTC format

3. “idRoute”-> the idRoute of the planned route for a specific user.

4. “Geometry”-> in GEO-JSON format. Mandatory.

5. “idSesion”->Using usrtrack idsesion. Mandatory.

6.”busData”->Optional. Using usrtrack (when the user is into the bus, the busData item contains information about the bus activity).

7. “levelAlarm”-> Warning, Fatal, Information. Mandatory.

8.”codeAlarm”-> A consensual code. Mandatory (pending of definition)

9.”textAlarm”-> Mandatory (extra information, e.g. “Anticipated traffic at”)

An indicative set of code alarms necessary for the fields 7,8,9 is included in Table 13.

Table 13: Set of code alarms for the Madrid events exchange

CODE	LEVEL	DESCRIPTION
00	I	OK
10	W	USER MOVING AWAY ROUTE (WALKING) (LESS THAN 100 METERS)
12	F	USER MOVING AWAY ROUTE (WALKING) (MORE THAN 100 METERS)
20	F	USER LEFT BUS BEFORE THE BUS STOP
21	F	USER TAKING A WRONG LINE
22	F	LINE CANCELED. WILL NOT STOP PLANNED BY
23	W	BUS DELAYED
24	W	TRAFFIC JAM MAY AFFECT THE LINE
30	F	USER NOT LOCATED
31	W	BATTERY TOO LOW IN USER DEVICE
32	W	GPS NO ACTIVE
40	W	NO INPUT TRACKS IN LAST 60 SECONDS
41	F	NO INPUT TRACKS IN LAST 180 SECONDS
50	F	ALARM BUTTOM RECEIVED FROM USER

3.2.6.4 Sequence Diagram

The sequence diagram for the overall scenario appears in Figure 99. The overall operations are highlighted in the respective boxes. If these are presented in the previous chapters we have limited the number of visualized steps for better display, as is the case in the CEP+ML cooperation part (Section 3.2.4.1) the Madrid Data feed ingestion (Section 3.2.5.1). The App Definition is portrayed in detail since the respective section (0) is generic and not focused on a specific logic. The red boxes refer to the message formats that are needed in steps 7.1 (Data Input) and 10 (Data output) and were defined in Section 3.2.6.3. It is necessary to stress that the sequence is generic and can be extended for other events, if the necessary modifications are performed (e.g. according CEP rules defined, according filtering in app logic etc.). These are also the points of extension for the Y3 prototype, following Y2’s identification of traffic events case.

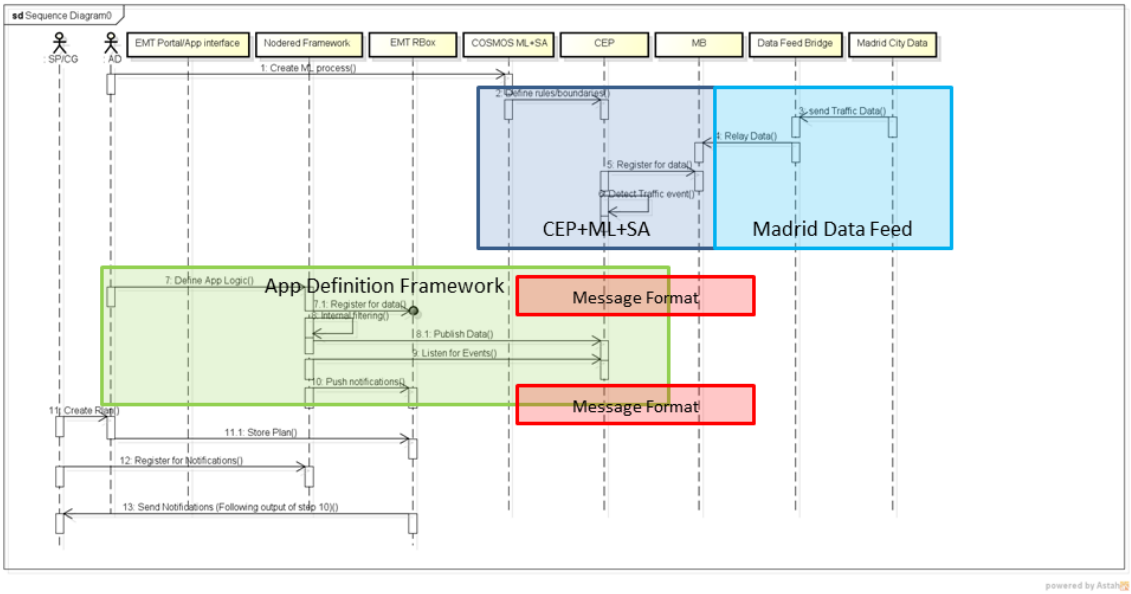


Figure 99: Sequence diagram for Madrid Scenario Application

3.2.6.5 Subsystem Test case table

Given that the majority of functionalities have been tested in their respective sections, in this chapter we define only the overall end to end test case that is needed, i.e. the creation of a plan by a CG/SP and the receipt of notifications regarding traffic incidents at their front end, which includes the key interaction between the COSMOS platform and the EMT system.

Test Case Number Version	**MAD_1**
Test Case Title	Madrid Application Scenario Event identification 1
Module tested	End to End functionality producing notiifcations of events for the SP portal
Requirements addressed	4.1, 4.2, 4.9, 6.1, 6.5, 6.21, 6.41, 6.43
Initial conditions	CEP rules have been defined for the MPs inside a route. Madrid data feed has been established. Relevant flows are connected (especially Data Output flow)
Expected results	Notification is sent and displayed to the SP portal if an event (e.g. anticipated traffic in the planned route of the user) is identified. Alternative display may include the visualization of intense traffic points regardless of the route for the CG to take under consideration during route planning
Owner/Role	SP/CG
Steps	1. The actor logs in the EMT portal and creates the route plan 2. The actor logs in the respective flow and inserts the SP id (this is triggered manually for testing purposes) 3. The events identified from the COSMOS µCEP are relayed through the MB to the App logic and from there to the

	SP/CG interface (portal or mobile app)
Passed	
Bug ID	N/A
Problems	None
Required changes	None

3.2.6.6 Deployment Diagram

A deployment diagram is not repeated for this case, since all the participating components are already available and described in the previous sections in terms of deployment.

3.2.6.7 Specific tests that may be needed

Scalability tests may be performed in Y3 with relation to how many notifications can be identified and sent to the respective clients (and how many of them). However this is also covered by the baseline technologies used within the COSMOS environment, and that have proven themselves in operational backgrounds. Other cases of testing (e.g. accuracy of model predictions in terms of future trends, accurate boundaries for CEP rules etc.) should be dealt with in the respective sections.

3.2.7. Camden Scenario Application

The Camden scenario Application's main target is the integration of all groups pertaining to the Camden scenario's targets such as autonomous VE self-management, Knowledge (Experience) acquisition or proactive dissemination, data flow integration and making use of COSMOS specific Services such as those of Machine Learning.

Based on the description of the targets detailed in the subgroups mentioned, there can be a truly realistic application of capabilities such as they have been developed up to this point in a real world environment. Additionally with what is already being worked on, the Camden scenario also implements the need for a more efficient Heating Schedule, in the face of energy waste and excessive carbon production as is described as an issue in D7.2.2.

This will be achieved through the use of a Heating Scheduling Management Application being developed by the COSMOS NTUA team in collaboration with the Camden housing authority and Hildebrand partners.

3.2.7.1 Scenario description

The proposed scenario takes into account that the End User desires an increased amount of cost efficiency, without having to be manually acting in order to provide feedback or actuation to a heating schedule of their flat, as is the case with Smart Meters that imply the monitoring of their readings by the End User and the need for continuous modification of settings. Such an approach is time consuming and will eventually alienate users even if the data is provided in understandable monetary terms and not in consumption metrics. For the purposes of the scenario, each flat possesses a management and monitoring tablet which can act as the Gateway to the entire network. The VE code will be located on this tablet, as well as all COSMOS applications the End User may choose to install.

The Heating Scheduling application will provide a Graphical User Interface for ease of access with a minimal of complexity and required options. The End User is only to be engaged during the early phase of the scenario actions. The first step is to plan a program, stating the desired temperature value for their flat, for specific time intervals of the planning period. Additionally the End User must input their desired budget. The application will then form the Problem by combining user input with the predicted temperatures during the programming period (provided by the Platform or third party Apps) and will use the VE Services offered by COSMOS implementation of VE functionality, in order to locate a similar Problem as the one described by the End User and return its Solution.

The Solution is structured as the actuation to be undertaken and the consumption per time period. This process involves the use of Case Base Reasoning on the internal VE Knowledge Base (Case Base). Given the possibility that the VE itself may not possess suitable Knowledge (Experience), it will initiate its own Experience Sharing mechanism, which targets suitable remote VEs the flat VE has knowledge of. These VEs will, in turn, search their own Case Bases for a suitable Solution and return their answers to the original VE. At this point the application will evaluate the monetary requirements of the returned Solution and actuate the Schedule or modify the input if the End User's budget is overshot.

The creation of the Problem part of the Case is described as a process which takes part every half hour sub interval and creates a vector with the values of:

- Inside temperature
- Desired temperature inside (provided by the End User)
- Temperature outside (predicted by a weather website)

A Solution has as properties:

- The URI of the IoT-service for setting the valve.
- The energy consumption that corresponds to the problem.

By executing the URIs at the corresponding time intervals, the heating schedule is executed.

Additionally use of Social Networking techniques in the context of IoT (SIoT) is made, in order to simulate relations between Social Nodes, which can aid the process of Knowledge diffusion, through Social associations of similarity. Therefore our Experience Sharing mechanism, which is enabled by the Network of Socially active Things, can act efficiently in locating suitable answers, irrelevant of location. The VEs themselves are creating associations in a decentralized manner, in order to avoid centralized approaches with limited scalability.

Finally a target of demonstration for this year is the implementation of the Proactive Experience Sharing as presented in section 3.2.4.2, with the actual Complex Event being detected, that of Sensor Malfunction / Sensor non Responsiveness.

This part of the Camden scenario implementation, will be running in parallel to the Heating Management scenario, and will be triggered by a series of Events (consecutive sensor readings) for each Sensor inside the physical Flat, that trigger CEP detection from the µCEP engine of the SAw FC based on a preloaded set of rules. This means that during the successful run of the deployed VE code, the data from the Sensors will not only be used by the Planner FC but will also be forwarded as segmented Events based on the desired input by the µCEP engine implementation. This is the basis for the real time self-monitoring capabilities that a VE has to possess.

3.2.7.2 Subsystem from D7.6.2 with integration points description

Figure 100 is the main illustration of a generic Application that involves VE2VE communication and makes use of the autonomous VE behaviour that the COSMOS Project is aiming for. In this specific scenario the roles that are implied by the IPs 1 and 2 are being handled by the Application Developer and the User input. The Apllication Developer has direct responsibility for implementing CBR Case schemas defining the structure of Problem Solutions, as well as defining a way for the User's input data in IP1 to be merged and unified into a Problem structure itself understood by the Planner. So it is evident by section 3.2.7.1, that the End User will input the data, into a Graphical User Interface for interaction with the VE Services that handle the Application actions and the Application developer is responsible for creating a flow that uses, enriches and forms the data, into the end result of providing a Heating Schedule, either through local CBR, on the historical data created Cases (IP2) or through Experience Sharing (Reactive) with other VEs.

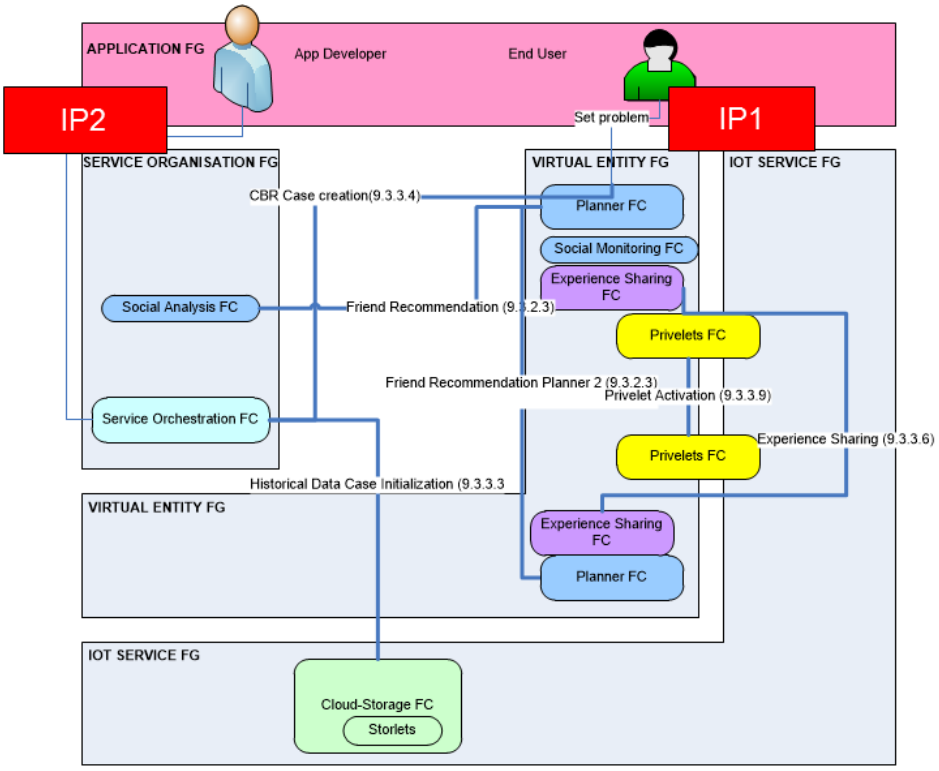


Figure 100: VE2VE application archetype based on defined system cases of D2.3.2

The above description of these IPs is also evident in Figure 101, where we explain in a different and clearer illustration the role of each member of the scenario flow, focusing this time not on the specific FCs involved so much as their place in the flow (VE side, platform side, etc.).

Both of the above images therefor demonstrate that the main points or integration and initialization of actions, are in the case of the Heating Schedule Management Application, the setting of a CBR Case structure, the way historical data are used for Case creation and finally the handling of User input as laid out in 3.2.7.1.

The historical data itself is stored concurrently with its production and consumption by VE side components into the Cloud Storage FC. The initial Case Base will be extracted from historical data derived from tracking the energy behavior of the End User.

Over a period of time (e.g. six months) the Cloud Storage FC will accumulate readings for Tin, Tout, consumption and flow rates (for fixed intervals). Further on, data retrieval may be performed and extract the relevant information which can be used in Case creation. This process is used strictly for the initial creation of Cases in a VE which possesses none. Later acquisition of Knowledge will be through the mechanism of Experience Sharing.

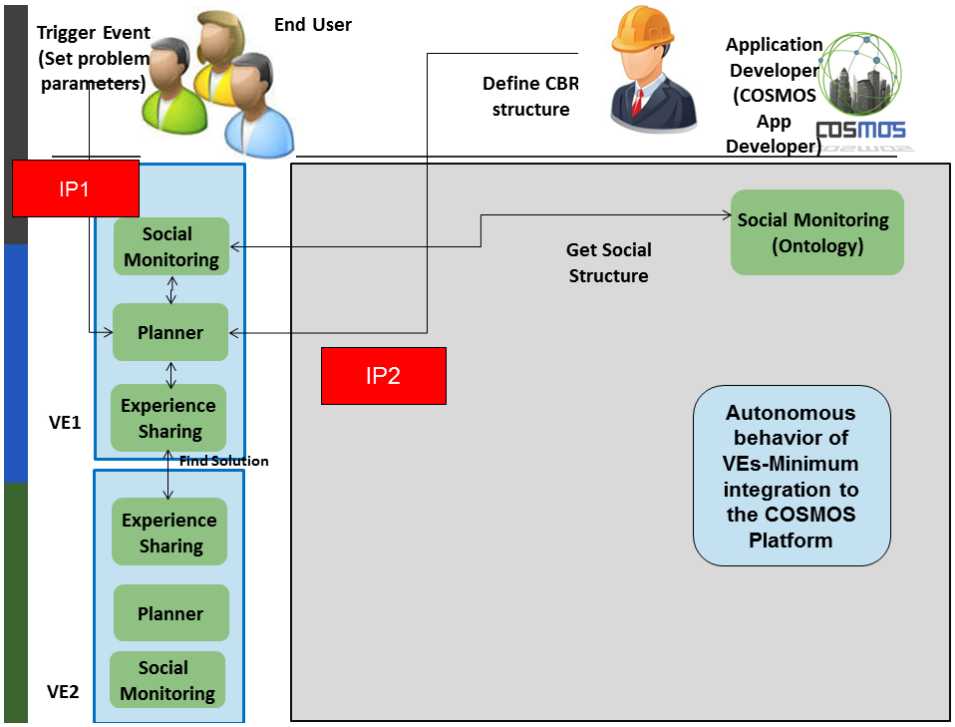


Figure 101: Autonomous Behavior of VEs with minimum platform integration subsystem

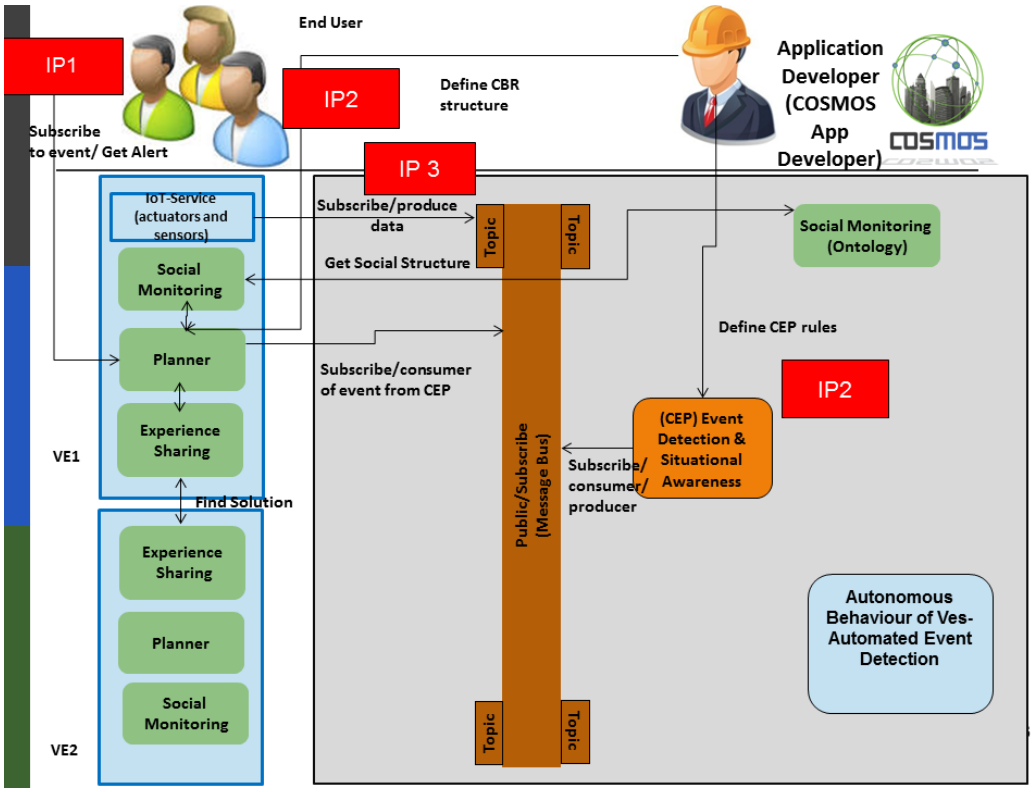


Figure 102: Autonomous Behavior of VEs- Automated Event Detection and Incorporation of COSMOS Platform

Apart from the Application which will handle the Heating Schedule, we aim to demonstrate the use of the Proactive Experience Sharing mechanism, in light of detected Complex Events based on singular Sensor inputs (henceforth called Events for the purposes of the µCEP engine functionality. In Figure 102 the main IPs are again being showcased, with the difference being that in our current implementation it is the Application Developer which has the responsibility for setting up the connections between cause and effect, in the sense of schema use and data flow interconnection.

This means practically that the AD must provide the DOLCE rule file as input to the µCEP engine, having in mind the Event structure that is being posted on the SAw input endpoint (MQTT connection). Additionally the AD must also provide the schema for the output that will be produced by the µCEP engine publisher component (the actual Complex Event) and also connect it with the schema used for the Case structure, as the handling of the detected Complex Event will be done in the Planner FC through CBR, specifically chosen for CE handling.

As described in section 3.2.4.2.1, the CBR of CEs uses the Levenshtein Distance metric which can calculate distances between strings based on the number of steps it takes to turn one string into another. So in receiving the textual representation of CEs and their accompanying data, we can use this method to implement a new similarity mechanism for retrieval of Solutions. The specific Complex Event used in this version of the Camden scenario is the Sensor Malfunction / Sensor non-Responsiveness.

When such a Complex Event is detected, the Planner will retrieve from the Solution an attribute demonstrating the need to handle the CE by sharing or internally with this Case aiming for both. So when the retrieval of the Solution is completed the Planner FC will forward the CE to the Proactive Experience Sharing Service for sharing with Friend VEs and will also call on an additional VE Service (part of the Solution), in order to instruct the Data Bridging Component to ignore the values produced by that Sensor. At this stage, the reintegration of the Sensor data in the general flow will be based on a time out mechanism, making a new check on the Sensor readings to detect whether the fault persists.

3.2.7.3 Message formats and configuration

The process of configuring the Data Bridging component and the message format has already been described in great detail in section 3.2.5.2 along with visual examples and certain clarifications in section 3.2.5.2.3.

The process by which the Application Developer will update the CBR structure to reflect the new Cases used by the Heating Schedule Management Application, will be performed in the context of the historical data Case creation (initialization of Cases), as it makes use of specific Planner functions which safely access the CB and make the necessary additions in Data type and Object type properties need.

This means that the initialization of Cases, also provides the necessary schema for further use of the CBR method by the Application and the Planner retrieval, similarity based, functions. It is only necessary for the AD to have a clear outline of the needed structure of each Case into Problem-Solution object pairs.

The End Users will be able to enter their preferences into the Application through a simple GUI as mentioned in 3.2.7.1. An example of an early draft follows.

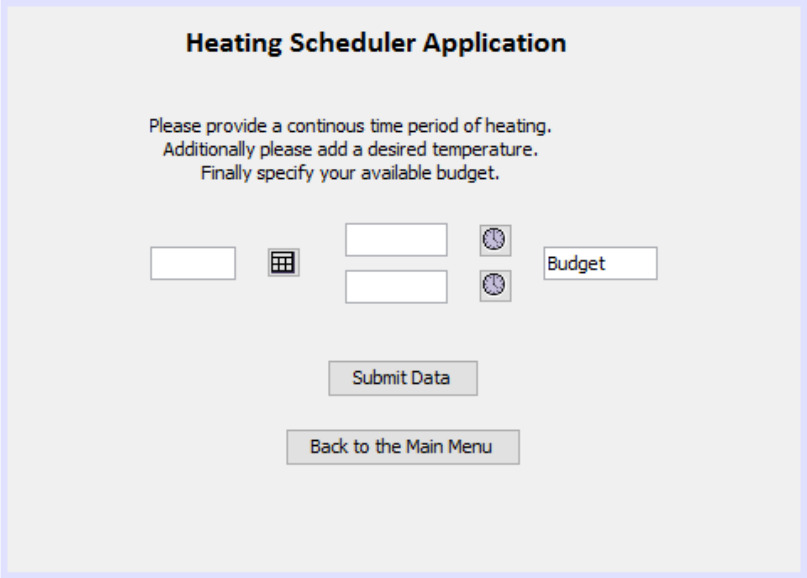


Figure 103: Heating Scheduling App GUI

3.2.7.4 Sequence Diagram

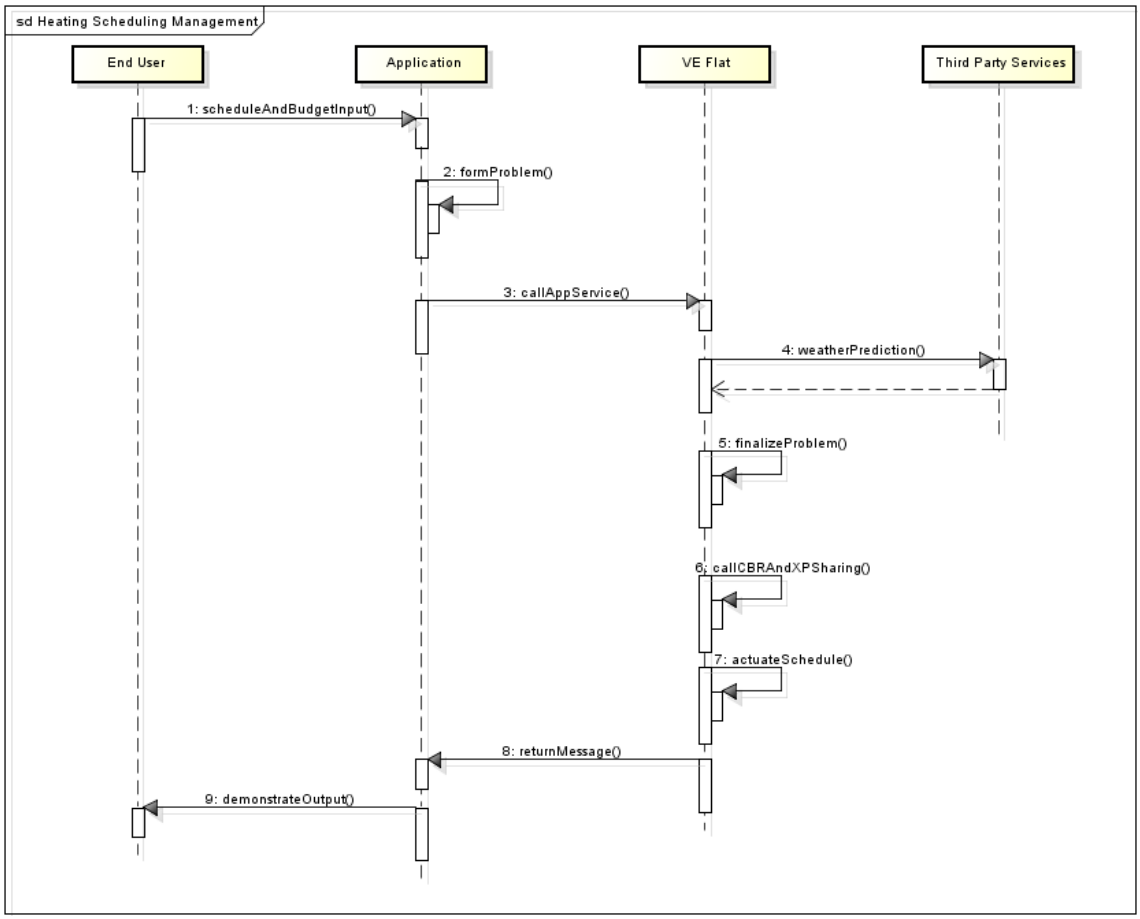


Figure 104: Steps for the Heating Schedule Management Application (General View)

The Sequence Diagram of Figure 104 demonstrates the steps to be taken for the implementation of the Heating Schedule sub scenario. Initially the End Users will use the GUI of Figure 103 and input a date and time for the schedule to be set on. Also they must also provide with a desired budget to aid in the refining of the CBR retrieval, taking back only the most efficient Heating Schedules for the Heating Valve.

Once the input is committed, the Application code will form the time period in half hour intervals, encode it in string with the date provided and pass that encoding and the budget as a REST communique to the Application Service endpoint in the VE.

From that moment forward, the Application code is responsible for retrieving a weather prediction for the period of time provided by the user so as to identify the external temperatures at the given location. The use of third party weather services which provide public APIs is recommended (eg. DarkSky Forecast API) with special care to the type of input being provided. In this case the example input is given as a JSON object. After this step the Application calls on the Planner to provide the starting inside temperature for the Flat so as to form the first half hour vector Problem and continue on with the expected inside temperatures for the further vectors. Each Problem will be returning after the application of CBR a Solution on whether to actuate on or off to the Heater, thus forming the schedule. Between each application of CBR, there might also be the need for Experience Sharing between Friends in order to locate suitable Solutions.

The process described in the SD of Figure 105 is the Experience Sharing flow as described in the previous year.

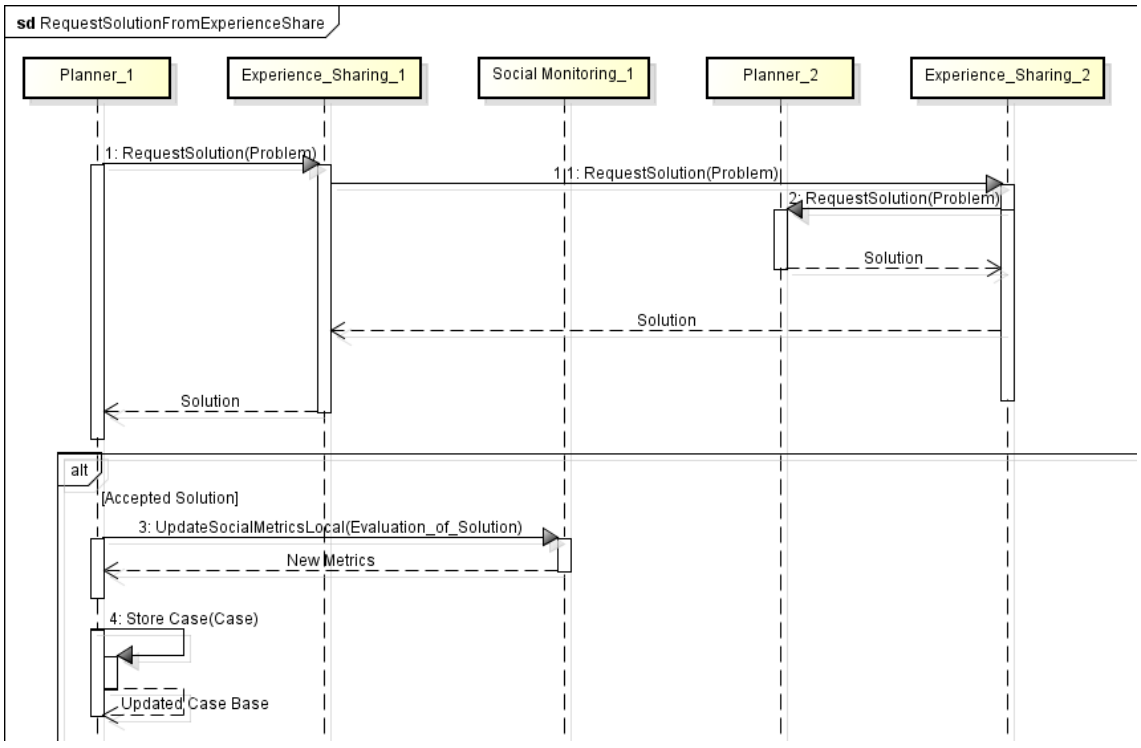


Figure 105: Steps for the Heating Schedule Management Application (CBR and XP Sharing parts)

In parallel to this entire process the VE is also going to be monitoring its sensors for any malfunctions or failures to update values. The process used is that described in section 3.2.4.2.4, with the added steps of how to handle internally such a detection. If indeed a temperature sensor fails for any detectable reason, then the Planner will retrieve in the

Solution of the Complex Event the URI of a Service which, given a specific input (JSON structured file), will update a filtering mechanism currently present in the Data Bridging Component. The filter update REST Service will instruct the flow to ignore values in the specific sensor, thus signaling the invalidation of any VE Apps requiring that specific sensor. As described previously, the reintegration of the Sensor data in the general flow will be based on a time out mechanism, making a new check on the Sensor readings to detect whether the fault persists.

3.2.7.5 Subsystem Test case table

The test case for this scenario appears in Table 14.

Table 14: Camden Scenario Application Complex Event Handling

Test Case Number Version	**CAM_UNI_01**
Test Case Title	Complex Event handling
Module tested	Data Bridging, SAw FC's µCEP engine, Planner FC, Experience Sharing FC, Social Monitoring FC
Requirements addressed	5.10, 5.14, 5.15, [5.22,5.23], [5.28, UNI. 015, UNI. 100, UNI.508], [5.29, UNI. 010, UNI. 704, UNI.706, UNI.708, UNI.715, UNI.719]
Initial conditions	Uploaded Configurations to Planner and the Bridging Component Connectivity between all Components Run VE code by using the command "java –jar OriginalFlatVE.jar>logflat.txt" for producing a log file of the System outputs Installing node-red flows in testbed and running them by using the command "node-red" in node.js console (flows accessed by web browser)
Expected results	Handle irregularities in sensor operation by sharing alerts in remote VEs and by filtering affected sensors
Owner/Role	Application Developer
Steps	Complex Event Generator simulation gives out Complex Event, this is achieved by using the trigger node of the node red flow which the tester started (output in node.js console) Planner listens on CE MQTT topic and receives message (log file) CBR finds similar Solution to Problem (formed by the Complex Event) (log file) Solution contains URI of Filter Service and body of message to be send (log file) Planner calling the Service and updating Filter (log file and node.js console as well as FilterDetails.cfg file) Data Streams have removed values for affected sensors (node.js console)
Passed	Yes
Bug ID	N/A
Problems	None
Required changes	None

Above are the results for the first test of the Camden Scenario. The reason we used a Complex Event Simulator is that the Camden data do not trigger the Sensor Malfunction Rule. Therefor we decided to trigger it manually for testing purposes.

Table 15: Heating Schedule Test Case

Test Case Number Version	**CAM_UNI_02**
Test Case Title	Heating Schedule
Module tested	Data Bridging, Planner FC, Experience Sharing FC, Social Monitoring FC
Requirements addressed	[5.9, UNI. 251], 5.10, 5.11, 5.12, [5.29, UNI. 010, UNI. 704, UNI.706, UNI.708, UNI.715, UNI.719], 5.31, 6.8, 6.9, 6.15, 6.18
Initial conditions	Initialized CB with Cases Install VE code in the testbed Install and run node-red flows by running “node-red” command in node.js console Installing App in the VE side testbed Installing GUI in testbed Run VE code and GUI by using the commands “java –jar OriginalFlatVE.jar>logflat.txt” and “java –jar NewAppGUI.jar>loggui.txt” in the command line
Expected results	Form a Problem and associate a Solution Remote Solution acquisition (Experience Sharing) Alert User with text box pop up
Owner/Role	Application Developer
Steps	The User enters schedule duration and budget in GUI (output in loggui file) The App contacts weather service (output in logflat file) The App forms Problem by also retrieving internal VE data (sensor temperature values) (output in logflat file) The Planner performs CBR and retrieves Solution either internally or through Experience Sharing (output in logflat file) Textual pop up window alert to user for the schedule retrieved (on-off states) User rates the schedule suggestion positive or negative in new pop up window
Passed	Yes
Bug ID	N/A
Problems	None
Required changes	None

The above test table (Table 15) demonstrates the use of VE side components in forming a Problem from the User input and returning a Solution based on the Heating needs.

The lack of historical data lead us to prepopulating the CB with Cases for the Solution retrieval and therefore in this phase the Cloud Storage FC for historical data Case creation was not used.

Additionally the lack of flat heating actuators necessitates the textual alerting of the user for the recommended schedule.

3.2.7.6 Deployment Diagram

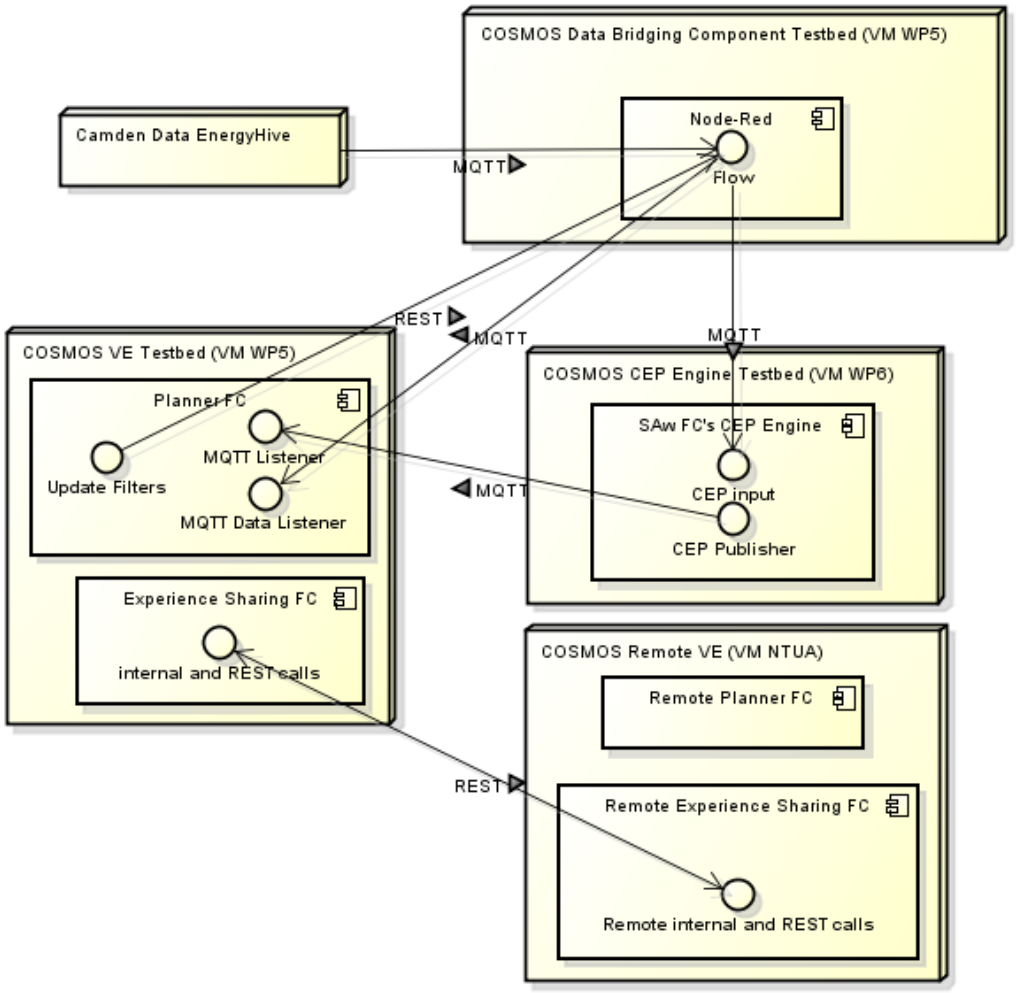


Figure 106: Camden Scenario Deployment Diagram

The above deployment diagram illustrates the initial testbed used for the component testing and the integration between them. The aim for the eventual deployment in a real flat is to take both the VE side and node-red based components and have them run on a tablet at the flat which will receive necessary connections from the Hildebrand Servers and the Cloud Storage FC.

As per the hardware constraints of the above deployment diagram we foresee the additional modification of the Privelet deployment process into performing their functionality in an ARM architecture compatible way.

3.2.7.7 Specific tests that may be needed

We identify the need for further enrichment of the number of Data Objects being stored in the Cloud Storage FC, so as to make the historical Case creation process more inclusive and reliable.

Additionally we also discern the need for validating the effects of the produced schedule in a real world application of the aforementioned approach. We must devise better ways to assess impact, which is being planned for as soon as we receive access to void or event real Flats for

testing. Steps have been taken to ensure override access so as not to adversely affect QoL for dwellers in the event of testing mishaps.

We have also identified that the weather prediction service used for the testing requires registration of each VE so as to receive a token of access for the API use. There is a limit currently of 1000 calls per day, per account. Though the limit is large enough to allay concerns of a future throttling in the available data, nevertheless it must be taken into account.

A true issue that was researched this past year in terms of future scalability is the amount of strain, the Experience Sharing Service will impose on VEs. Therefor by planning a list of tests, we approached the issue by testing the previous year's scenario incorporating XP Sharing use.

Our tests focused on the ability of the testbed as limited as a Raspberry Pi 2 to provide a consistently stable environment for remote VE components being used by an Application to communicate and Share their Experiences as well as reason on their stored Knowledge, to provide answers to each other's queries. Our tests focused on demonstrating how differentiated workloads of queries may affect VE performance in resource constraint environments. Thus our approach is divided into 3 categories of incoming traffic. The Raspberry running VE Service is contacted and attempts to locate a Solution to the incoming Problem first locally (circular points), if not possible by contacting the first VM (square points) and if needed the second one, through the recursive Experience Sharing mechanism (rhombus points). The VMs aiding the process are not Raspberries. The tool used for the simulation is Apache JMeter®

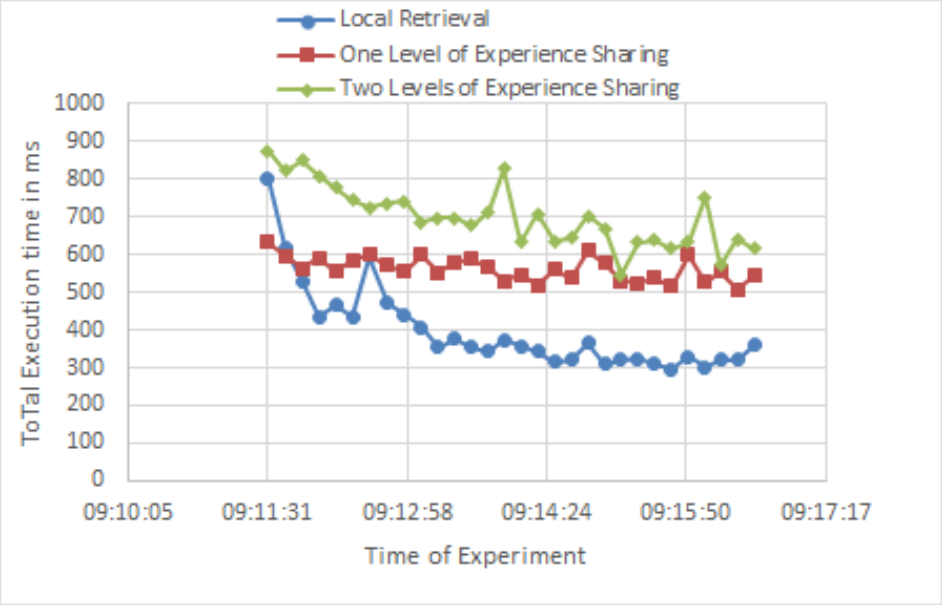


Figure 107: Low volume simulations

The first category is the Low Volume category which is characterized by a single Query Thread running infinite loops, with a constant 10 second delay between calls.

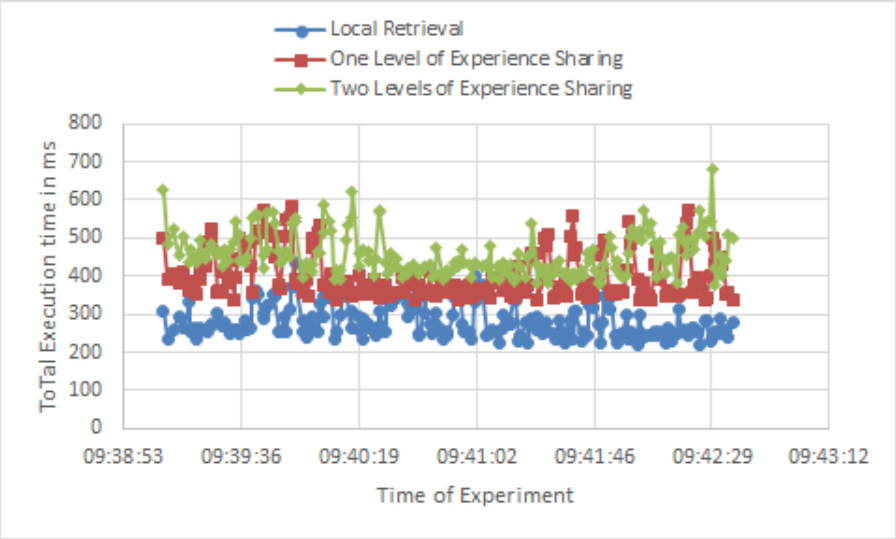


Figure 108: Medium volume simulations

The second category is the Medium Volume category, which we deem to be the normal volume of communications expected by the VE. This included the existence of ten Query Threads, starting operation in intervals of two seconds until all operational, with infinite loops for each one and a Poisson timer of query delay of a lambda value of 10000 ms. This leads to an increased load of queries which, due to the revised nature of the Experience Sharing code in handling incoming HTTP requests, were serviced in less time than the serial arrivals of the Low Volume category. This was true in all three versions of the simulation.

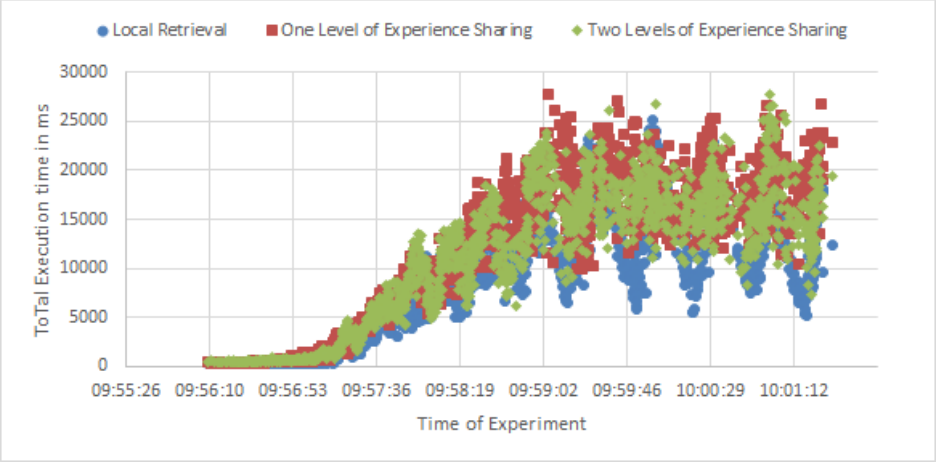


Figure 109: High volume simulations

The third category is the High Volume category which is described by the use of one hundred Query Threads operational in two second intervals, infinite loops and a Poisson timed query delay of 5000 ms lambda. This category initially worked in similar times to the previous two but as more Query Threads came into operation, times increased, especially after fifty concurrently running Threads until stabilizing at averages of 12, 16 and 17 seconds for each version of the simulation. In this category deviation was also increased with results in the third version reaching as high as 28 and as low as 8 seconds.

Therefore we conclude that the Experience Sharing used in the context of any Application will have wide scalability if the volume of inter-VE communications remains at most medium-high. This is not trivial as the High load that we tested is unlikely to happen in a sustained fashion in any phase of the Project's development and deployment.

3.2.8. Taipei Scenario Application

In COSMOS, one of our intended objectives is to a bridge the gap between historical data analysis and real-time data analysis solutions. In this regard, we have explored and developed a platform based on OpenStack Swift for storage and using multiple Spark libraries on the top of it for analyzing historical data at one end and developing a micro Complex Event Processing (µCEP) engines for analyzing real-time data stream in a distributed manner.

We have used these components successfully on III data in order to detect anomalies automatically in real-time. In Taipei, III is providing services to hundreds of houses with thousands of devices connected to their smart sockets. Smart sockets provide the users about real-time energy usage in order to make users more aware of their energy consumption.

3.2.8.1 Anomaly detection

There are many applications of outlier or anomaly detection. Both terms are used interchangeable in the literature. Anomaly can be described as anything unusual and which does not confine to the normal behaviour. For example, malfunctioning of any electronic device can result into excessive amount of power dissipation which needs to be detected and reported as soon as possible. Similarly, a temperature of 50 C reported by a sensor during winters is an anomaly which might be due to sensor fault or fire. A traffic accident on the road might result into very low average traffic speed which is also an example of anomaly. Another example of anomaly is the detection of electricity usage during midnight which might indicate an intrusion.

There are many methods from Machine Learning domain which are available for anomaly detection which can be used for monitoring applications. But they are not suitable for large-scale real-time applications. In contrast to these methods, event processing methods based on Event Driven Architecture (EDA) which are optimized for real-time analysis can be used. It involves analysing the data stream on run using rule-based inference. For monitoring applications, it involves setting of rules using threshold values in order to make a decision. For example, if we want to monitor power measurements a rule can be set as if the current measurement is greater than the threshold power; generate the event. For a single house, there might be many appliances and each appliance may have a different behaviour. The normal working range of every appliance is specific and it is almost impossible to use the human knowledge to set optimized threshold values. Also, the behaviour of devices change with respect to time. The use of microwave in daytime is not an anomaly but if we detect the use of microwave at night, it is an example of anomaly. Hence, the threshold values should evolve with the time. This is just a single example of monitoring. Another application may involve the monitoring of sensor readings such as temperature or humidity in order to detect malfunctioning of a sensor or any unusual event such as fire.

We propose to explore the historical data of devices in order to model their normal behaviour and find the threshold values automatically. We propose to have threshold values with respect to the different contexts such as morning or evening, weekday or weekend, summers or winters etc. Threshold values will be adaptive as the behaviour of devices evolve and the rules for CEP can be updated on the run in order to provide a generic solution.

3.2.8.2 Scenario description

Figure 110 below shows the high-level architecture of Taipei scenario. Different appliances are connected to smart gateway with the help of smart plugs. Smart plugs monitor real-time electricity consumption data which is published in real-time on the specific topic in apache Kafka. Real-time data is being monitored with the help of µCEP in order to detect anomalies. While historical data is being analyzed in Apache spark in order to calculate statistical properties of the data which is used to calculate the normal working range of appliances. The normal working range is used as threshold values for CEP rules which will be updated with respect to time. Anomalies for a specific appliance are highly dependent on time as well. For example, switching on the TV in an evening is a normal behavior whereas switching it on during midnight can be an anomaly. Analytics on historical data enables to have specific characteristics with respect to particular user or appliance. It helps to understand the behavior of users in a better way.

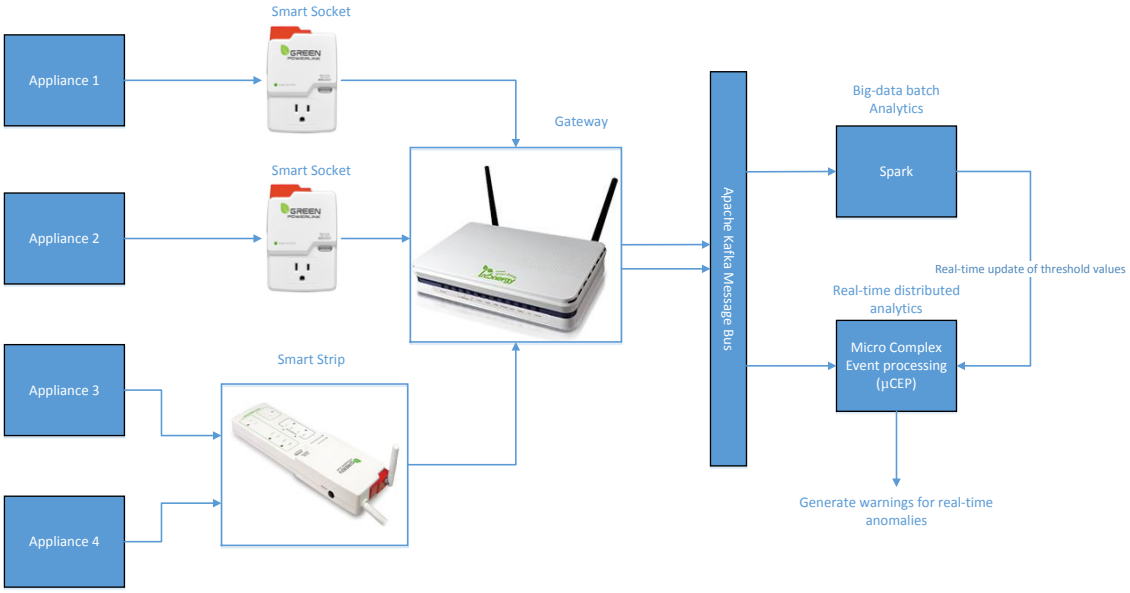


Figure 110: Real-time anomaly detection

3.2.8.3 Subsystem from D7.6.2 with integration points description

Following subsystems are involved in this scenario and have been included in the previous sections.

1) Data feed, Annotation and Storage for Taipei case (3.2.5.3)
2) µCEP, Situational awareness and Machine Learning (3.2.4.1)

3.2.8.4 Message formats and configuration

Complex Event Anomaly is detected by µCEP using Dolce language specification. A snippet of a dolce code for detecting Anomaly is shown below where initial value of threshold current is given as 2. But once the application will be run, it will update the threshold values calculated by analyzing and modelling historical data.

```
external int TUPLE_WINDOW = 3;
external float ThresholdCurrent = 2.0;

event Anomaly {
       use {
            string id,
            float Current,
            string tf
       };
}

complex ExampleAnomaly {
       payload {
              float ValueCurrent = Current,
              string tf = tf
      };
      detect Anomaly
      where Current < ThresholdCurrent in [TUPLE_WINDOW];
```

3.2.8.5 Sequence Diagram

Sequence diagram for the Taipei scenario is shown below in Figure 111. Application developer registers to III smart gateway for the notifications of warning messages once an anomaly is detected. COSMOS platform obtains data from III APIs and publish into COSMOS message bus after processing through data feed bridge. Historical is being stored in the cloud storage and analytics is run through Apache Spark in order to calculate optimized threshold values which are passed to µCEP. µCEP applies the pre-defined rules and detects an anomaly which is notified to the application developer through III gateway.

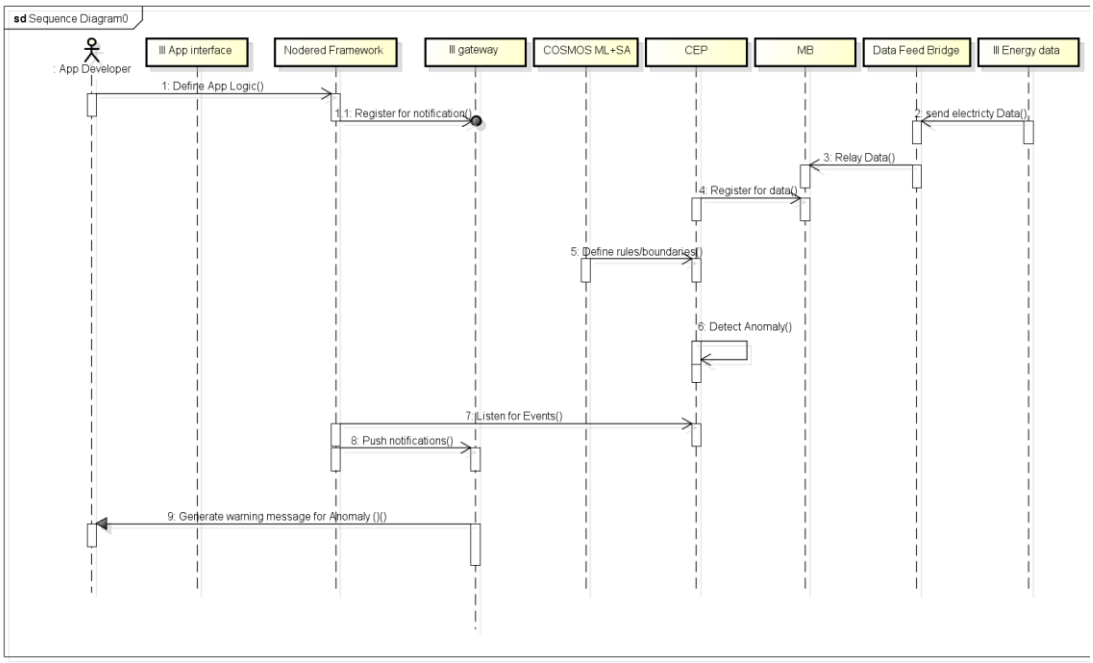


Figure 111: Sequence diagram for Taipei Scenario

3.2.8.6 Subsystem Test case table

Table 16: Test case table for the Taipei application scenario

Test Case Number Version	III_1
Test Case Title	Taipei Application Scenario Anomaly identification 1
Module tested	End to End functionality producing notifications of Anomalies for Electricity data
Requirements addressed	4.1, 4.2, 4.9, 6.1, 6.5, 6.21, 6.41, 6.43
Initial conditions	CEP rules have been defined for the anomaly. III data feed has been established. Access to historical data through Spark and cloud storage enables
Expected results	A notification of warning message is generated whenever current or power readings are more than normal usage indicating some unusual behavior
Owner/Role	User
Steps	1. User connects the appliance to a smart plug 2. The user registered for anomalies or unusual behavior 3. The events identified from the COSMOS µCEP are relayed through the MB to the App logic and from there to the III gateway.
Bug ID	N/A

3.2.8.7 Deployment Diagram

Deployment diagram of the overall scenario is shown below in the Figure 112. It uses COSMOS functionalities for storing, managing and analyzing large historical data in an optimized manner using object storage and Spark. Real-time data is analyzed using µCEP. Node-RED provides an interface connect III Taipei data source to COSMOS message bus.

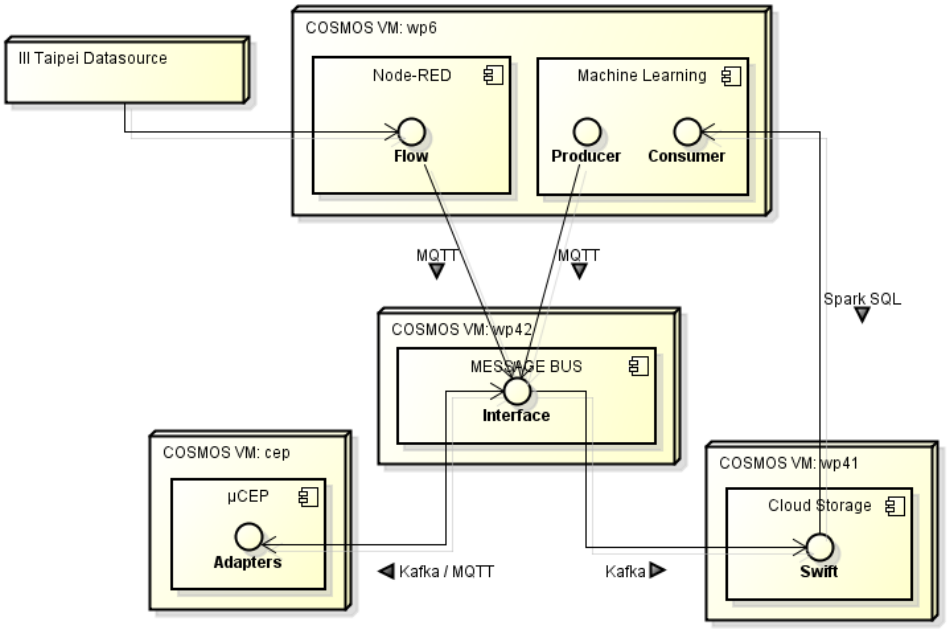


Figure 112: Deployment diagram for III Scenario

3.3. Evaluation of Plan Goals, observed deviations and applied mitigation strategies

In order to evaluate the implementation of the plan, we have created the overall Table 17, with the goals of this period and the accomplished results. This analysis aims at identifying also gaps, deviations or extensions to the planned goals per period. From the overall table, we can conclude that from the 14 subgoals, 12 have reached the anticipated level of maturity, while 2 will be needed to be continued in the following periods. Also from the 12 achieved subgoals 4 have gone beyond the anticipated level. The two cases that need to be continued are listed below.

One aspect in which a delay has been presented refers to the description of concrete endpoints through the registry. However this does not affect the remaining testing scenarios, given that the endpoints are known and concrete to the project and the functionalities that may need this feature (e.g. dynamic VE replacement or retrieval) are scheduled for Y3. Furthermore, the key component functionality that is needed for this to be accomplished is already available. The other case of deviation refers to the overall Web UI process, for which mockups have been initially created, however and given that many components are anticipating producing Node-RED interfaces or further enhancing their individual GUIs, this process has been put on hold until these aspects are finalized. Again this delay is not considered severe given that only the initial specifications had been planned for M25.

With relation to the 4 subgoals that have progressed beyond expectation, these refer mainly to the DSOs produced, for which 1 was anticipated to be ready and we have 2, the available Node-RED flows that include all data feeds, repository functionality and flows for the majority of operations in the centralized archetype. Furthermore, testing has been extended for the VE2VE applications archetype with the inclusion of a Raspberry Pi testbed.

With regard to corrections to the designed timeline of the integration plan, the two main goals of the COSMOS platform integration (Data Management and Analytics and Autonomous VE Behaviour), these were originally planned to go up to M25, but they need to go past that checkpoint given that in Y3 a number of new features (e.g. Trust and Reputation, VE reconfiguration etc.) are expected to be delivered or integrated (e.g VE side components with the hardware security API). With regard to the Data Fields definition, it must be noted that given that this process is done per case and may include specific needs per case (e.g. specific events names definition), this action has been performed in all cases examined so far, but is expected to continue in Y3 (e.g. if new events are identified for the Madrid case). Thus the relevant time line in the integration plan should be expanded. In all cases this is a minor step, that should however be kept in mind.

With regard to subgoals defined, the VE endpoint specification part of the VE side COSMOS Components integration goal seems to overlap with the VE Description goal, therefore this subgoal should be removed with a more concrete one (integration of VE side components with the security API) taking its place.

Table 17: Evaluation of goals over achieved results in M25

Goal	Subgoal	Anticipated Status (M25)	Actual Status (M25)
VE Description and Linking	VE Registry and Linking to JSON schema	Available	Available
	COSMOS Ontology Definition	Available	Available
	DSOs	1 Available	2 Available
	VE instances	Available	In progress
COSMOS Platform Integration	Data Management and Analytics (CEP+ML cooperation)	Available	Available
	Web UI Integrated Environment	Initial specifications	Initial Mockups , pending to check linking with Node-RED and other GUIs
	Autonomous VE behavior (Proactive Experience Sharing)	Available	Available
Application Definition and Creation	Scenarios Concretization	Initial for all UCs	Available, with extended functionalities in some cases (Camden)
	Application Definition Framework	Initial	Available and extended with numerous flows and interfaces available and the repository functionality
	Conceptual Archetypes Definition	Available	Available and extended (included abstraction through Node-RED flows for Madrid)
VE side components integration	Endpoint Definition	Available	Available through Registry
	Installation process	Initial	Initial
Data Model	Data Fields Definition	Available	Available
	JSON schema retrieval	Available	Available

3.4. Future steps with regard to the advancements of this period

With regard to future steps from actions of this period, we can highlight the following aspects. With relation to the VE incorporation phase, the creation of concrete VE instances and descriptions is one of the goals, so that this part of the platform may be used in the overall sequences or during runtime operation.

For the VE2VE archetype case, we need to further generalize the process by offering the functionality of CBR creation from the historical data and enable the exposure of the VE side components as Node-RED flows. The availability of historical data from Camden in Y3 will significantly enhance this process. Furthermore, this update will enable the implementation of heating schedules and the evaluation of their effect on actual users.

For the centralized archetype case, the list of identified events needs to be extended for the cases of the Madrid and Taipei UC, while the usage of an external repository of flows available to the community will be investigated.

Based on the work of this period and the current versions of the sequences and operations, we can perform further abstractions during Y3 that will enable the functionalities to be applied in different contexts.

4. Integration Stages 4 (M26-M34) & 5 (M35-M36) in Y3

4.1. Overview from Integration Plan

Following the results of the previous integration periods, the fourth and fifth stages of integration (M26-M34 and M35-36) aim at further enhancing the linking in the platform and achieve better incorporation of the UC elements, while finalizing the COSMOS outcomes and artefacts. The involved integration goals for this stage include the continuation of the COSMOS platform setup and cross component integration, especially with relation to the following subgoals:

- Integration with relation to new functionalities that emerge from the platform side, pending the update in Y3 architecture and components capabilities such as Privacy and Consent management, new fuzzification mechanism of Privelets, ingestion of external data flows etc.
- Extension of data management and analytics in order to include new data sources as indicated by the respective UC scenarios and applications. This includes also the definition of data models and fields of information across the various components in cooperation with the component needs and the UC data feeds for the extended data sources used during Y3 that is part of the Data Model/Template Integration goal as indicated in D7.6.3. Extended data flows include social network data (Twitter), weather data and past historical data from Madrid city traffic monitoring.
- Autonomous VE behavior in terms of easier and unified installation, integration in the H/W side and adaptation of the Planner applications in new data flows retrieved from the Cloud storage
- Packaging process in order to more abstractively expose COSMOS artefacts

With relation to the VE description and linking to the platform the following subgoal will be completed:

- VE Instances Descriptions and Registry population, based on the concrete examples from the UCs
- Investigation of applying fuzzification to properties after VE registration

With relation to the VE side components integration, the process will continue for the finalization of the subgoal:

- Components installation to VE instances: in this case we will pursue the deployment of all VE side components, potentially through more automated ways (including Node-RED flows) through the existence of one Raspberry Pi image that includes the respective components

For the Application Definition, Creation and Deployment goal we anticipate extensions with regard to the following subgoals:

- Complex applications creation with reusable flows: in this case the use of pre-existing flows and templates will be pursued in order to enhance and ease application development
- Application archetypes potential extension: in this case, by investigating the remaining application scenarios, we will examine whether the defined archetypes can be extended in order to cover more typologies of applications through the combination of defined subsystems, or the possibility to include new extensions or abstractions to the

platform functionalities. This has been achieved by introducing a new Archetype, the "Events on events" one.

- Continuation of the concretization of the application scenarios with relation to the COSMOS components, including specific instantiations of components such as the Planner, CEP rules and prediction modelling.
- Introduction and implementation of a new concept, the Events Marketplace, in which developers may retrieve information about available events but also offer their events for consumption to other interested entities. This is the instantiation of the Events on events archetype, along with the exploitation of reusable flows for abstracted application creation

Furthermore we have also added a new Goal, relating to acquiring End User feedback, that may aid in enhancing COSMOS outcomes either in terms of functionalities offered or exposed interfaces. For each of the cases, the end user feedback has been transformed into changes in the technical or integration outcomes in order to accommodate the specific identified needs.

The work was organized in relevant technical groups, each of them having one or more (in case they were related) subgoals mentioned above. The outcomes of these groups are reflected in each one of the following subchapters. Furthermore, one group for each UC application was created, that was fed by the outcomes of these groups in order to adapt them to the application demo scenario.

4.2. Defined Subsystems and Tests

4.2.1. Incorporation of P&C Management and Consent Levels and integration with retrieval from Camden historical data

Privacy and consent management is one of the key requirements for IoT. Thus in the case of data collected and stored by COSMOS, relevant processes should be in place across the various stages of data acquisition, transfer and storage. When the data are at rest, an extra layer of consent management and access should be enforced, which is the purpose of this section. The design of such a system should enable a trade-off between the goals of fairness, motivation and symmetry in participation, privacy and system performance. In the COSMOS system, this protection on the data at rest is implemented via the Privacy and Consent system that resides on top of the Cloud storage and regulates access of external components (e.g. the Planner) to the raw data acquired from the IoT devices.

4.2.1.1 Testing Scenario

The Consent Management Architecture is described in the following figure:

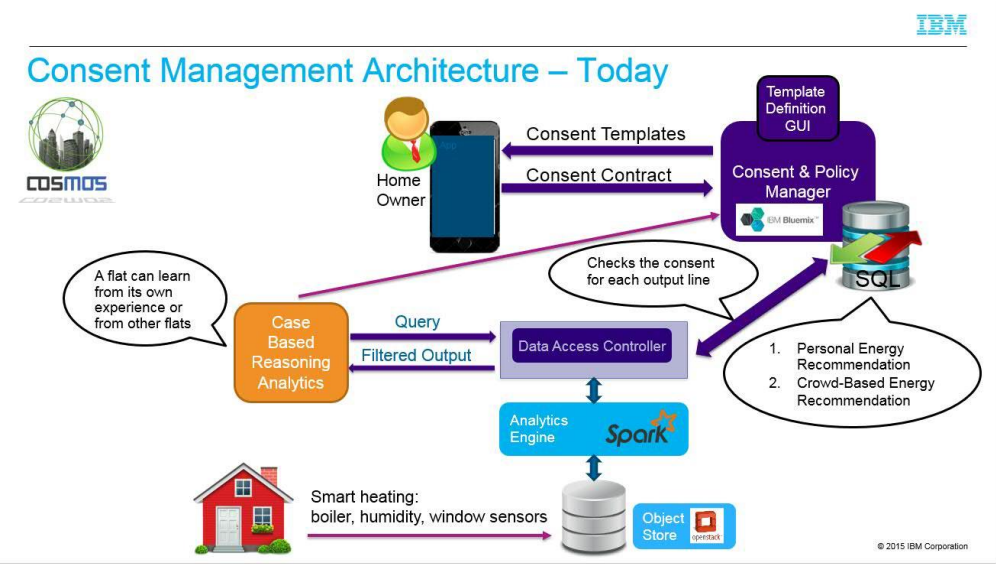


Figure 113: Overall architecture of Consent Management integration prototype

This figure describes the overall architecture of the prototype, as also described in D4.2.3 and repeated here for completeness of information but also since some minor corrections have been made since then. We have used IBM's Consent Manager Service on Bluemix (http://consentmanagement.eu-gb.mybluemix.net/indexCosmos.html) to prepare the consent templates for COSMOS, and are developing a special purpose Data Access Controller (DAC) for COSMOS. The DAC is responsible for controlling access to the data and enforcing the data privacy policies, based on the decisions made by the Consent Manager. We are implementing it for SQL-based data access, as a separate layer within the accessing application, based on Apache Spark and OpenStack Swift object store. IBM's Consent Manager Service on Bluemix manages everything related to the consent governing the use of data in the enterprise and responsible for the collection, storage, and maintenance of user consent. An initial version of the Consent Manager Service includes:

- Support for purpose based user consent

- Consent per data item corresponding to the purpose and service
- Purpose-based access control
- REST APIs for registering organizations and users, creating consent templates for services, and creating specific consent contracts
- Basic UI for consent template definition
- Basic logging and support for reporting solution

The major user of this mechanism in the COSMOS ecosystem is the Planner component residing at the VE level that needs to access raw data in order to create cases for the CBR technique used.

4.2.1.2 Subsystem from D7.6.3 and D2.3.3 with integration points

In Y3, the update in the architecture in D2.3.3 and the specialization of the System Case 9.3.3.3 to include P&C management has led to an updated subsystem image that includes two integration points (Figure 114). IP1 refers to the interface between the end user and the P&C mechanism, in order to define which parts of the data are available for usage by e.g. applications. The end user defines at the data field level whether the specific data field can be made accessible by 3rd parties (e.g. app developers). IP2 refers to the actions of an application developer using the Planner capabilities in order to drive a social autonomic app. For the Planner to have a populated knowledge base, it needs access to the historical data in order to create the relevant cases. Therefore, the app developer needs to configure the appropriate queries performed towards the Cloud storage for getting this data, based on the needed data fields. If for some reason the end user does not wish to reveal data fields needed by the Planner, then the specific user (e.g. house) will not be eligible for the specific application.

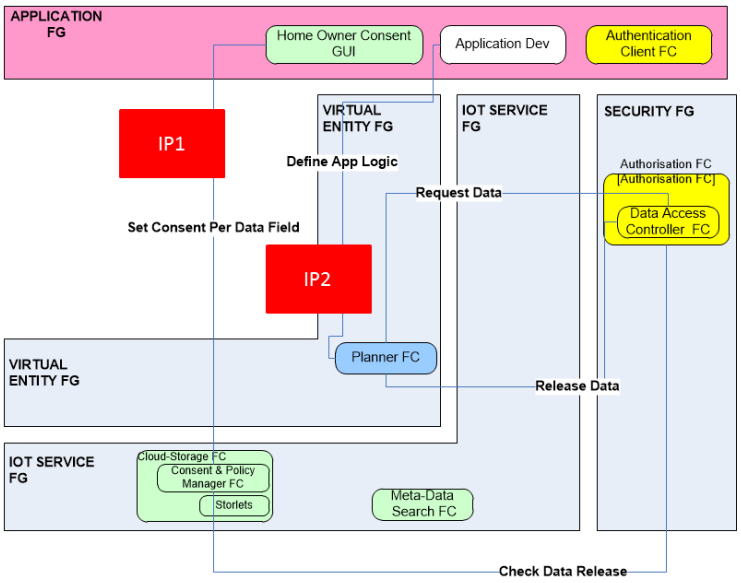


Figure 114: Combinatorial Subsystem for Privacy&Consent management in Case creation from historical data

The necessary modifications for integration purposes relate also to the internal structure of Experience Sharing, as portrayed in Section 9.3.3.7 of D2.3.3 [11], that needs to consult P&C before using local data, but given that this does not contain any IP as defined in the integration plan it is not included in the formal subsystem specification. However it is indeed a testing

scenario that is included in the integration process. Therefore overall, tests need to be performed for:

- Incorporation of declared consent (IP1)
- Modification of Planner application instance for inclusion of the respective queries that relate to the needed data fields as dictated by the application scope (IP2)
- Execution of the respective queries
- Modification of Experience Sharing to override local search when sharing is disabled

With relation to IP1, We have defined two consent templates on the CM service on Bluemix:

1. Personal Energy Recommendation (service_id = 40): the resident's personal energy data will be used to provide him recommendations
2. Crowd-based Energy Recommendation (service_id = 39): the resident's personal energy data will be shared with others to provide him crowd-based recommendations

The resident can either choose (1) or (2) or opt-out from both. Each of these templates contains the data fields included in the Camden data flow following items (both templates use the same items), as an example see the figure below.

1) apartment id (hid) – mandatory
2) last name – mandatory, anonymized
3) first name – mandatory, anonymized
4) e-mail – mandatory, anonymized
5) phone – optional, anonymized
6) address – optional, anonymized
7) estate – mandatory, anonymized
8) timestamp – mandatory
9) heatmeter – mandatory
10) sensors (temperature, humidity, window, door) – optional

In the Consent Manager Service on Bluemix (see: http://consentmanagement.eu-gb.mybluemix.net/indexCosmos.html) for each of the above items one can choose the following parameters (see the figures below):

- Mandatory (true/false)
- Anonymization method (none or exclude)
- Data types (e.g. text)
- Sensitivity (PII, medium, low)

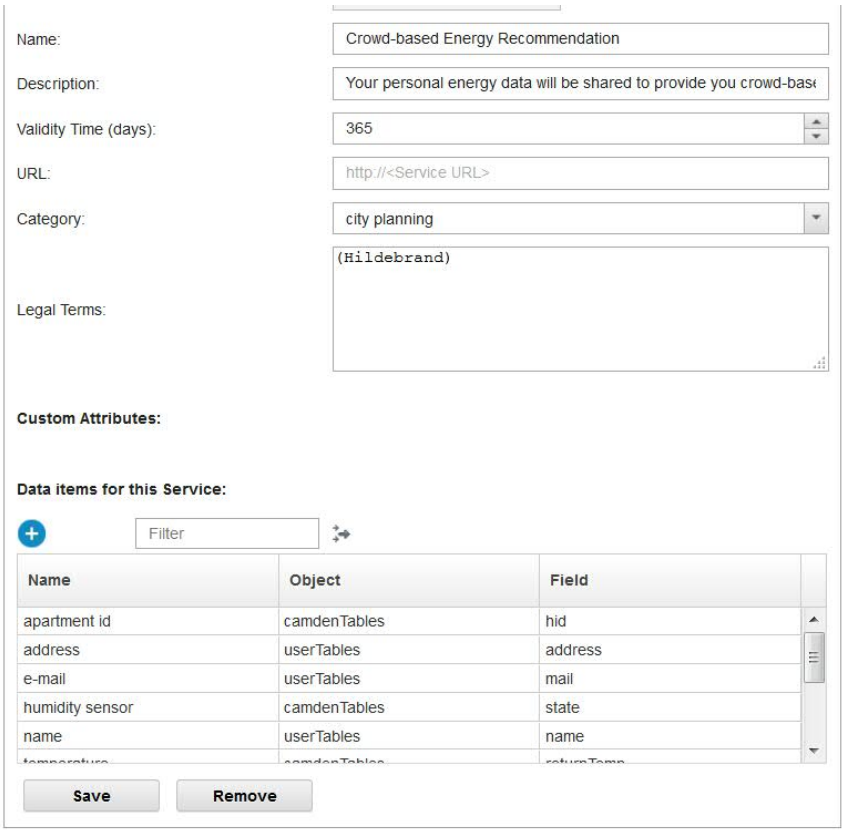


Figure 115: Consent template on Bluemix – Crowd-based Energy Recommendation

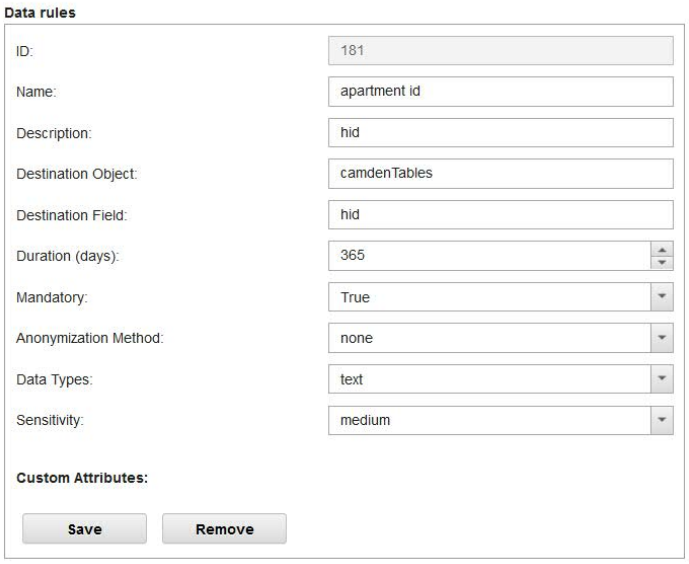


Figure 116: Consent template on Bluemix – defining a data item

For the remaining integration points, these are incorporated in the Planner code via the information presented in the following sections.

4.2.1.3 Message formats and configuration

4.2.1.3.1 Direct Access to the Consent Manager Service

The VE may contact the Consent Manager Service on Bluemix directly using the following API (for further details and examples see: http://consentmanagement.eu-gb.mybluemix.net/APIs/).

For example, in order to access for a single user and a single data element:

```
GET http://consentmanagement.eu-gb.mybluemix.net/cm/1.0/user_data/access_aproval
```

Where:

- service_id is 39 or 40;
- username is hid;
- dest_object is camdenTables or userTable;
- dest_field is one of the possible fields: in camenTables it is hid, ts, heatmeter, sensors, in userTables it is firstName, lastName, phone, email, address;

For example,

```
curl -X GET --header "Accept: application/json" --header "org: XXXXXX" --header "password: YYYYYYY" "http://consentmanagement.eu-gb.mybluemix.net/cm/1.0/user_data/access_approval?service_id=39&username=cZsogmiuZ9Fs&dest_object=userTable&dest_field=email&token=46764hdghghghgsdgfgshdghf874366"
```

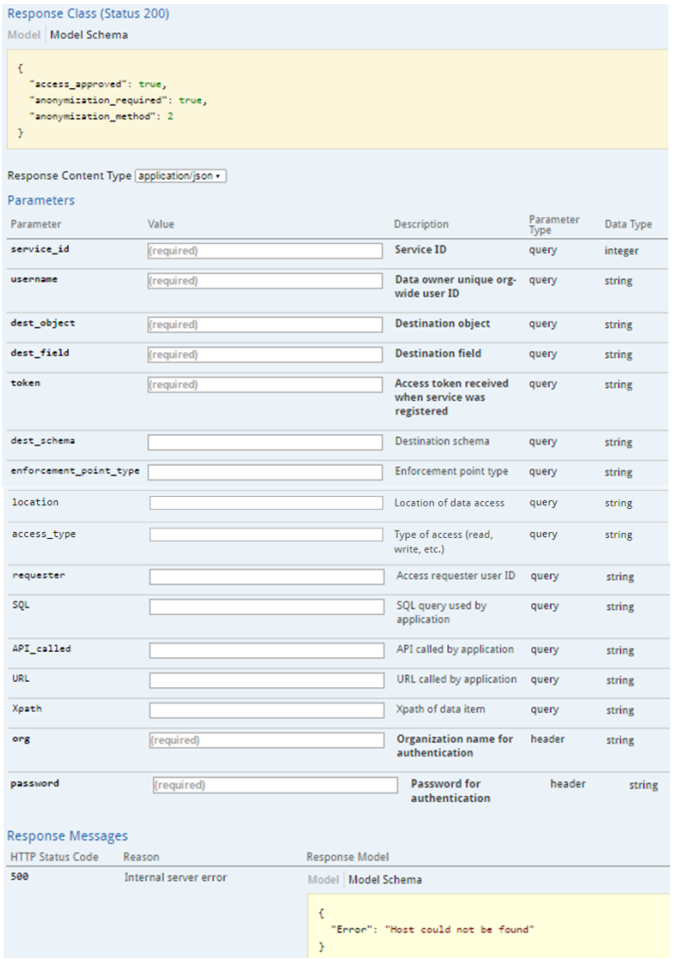


Figure 117: Consent Management

4.2.1.3.2 Connection to the Data Access Controller

The Data Access Controller contains the following components:

1. Consent Socket Client: obtains the SQL query and the token for the purpose from the CBR application, and outputs the result after processing it.
2. A filtering logic, including parsing the SQL request and response, and replacing values of items in the response that are not consented. If a user has completely opted out of a service, his record is removed from the result set. Otherwise, specific fields are "nullified" according to the field type (e.g., empty string, 0 for integers, null for objects, etc.). Other anonymization techniques will not be supported in the prototype.
3. Communication with the consent manager service, that compares the token for the purpose with the user's consent, and obtains the decision about if and how the data may be used for the stated purpose.
4. Consent Storage Connector: sends the SQL query to Spark SQL, and obtains the corresponding data records stored in OpenStack Swift.

The Data Access Controller (DAC) obtains the SQL query and purpose using the following RESTful API:

```
GET http://127.0.0.1:8080/sql?query="YOUR QUERY"&service_id
="YOUR SERVICE ID"&apartment_id="YOUR FLAT ID"&date_id="YOUR DATE"
```

For example:

```
curl -X GET --header "Accept: application/json" --header "org: COSMOS" --
header "password: XXXXXX"
"http://127.0.0.1:8080/sql?query=SELECT_HEATING_DATA&service_id=
39&apartment_id='cZsogmiuZ9Fs'&date_id='2016-05-27'"
```

The output is in JSON format (see examples below).

The possible SQL queries are:

1) SELECT_HEATING_DATA for selecting the data of one apartment for the heating schedule application.

```
SELECT camdenTables.hid AS hid, camdenTables.ts AS ts, camdenTables.heatmeter
AS heatmeter, camdenTables.sensors AS sensors, userTable.email AS email FROM
camdenTables, userTable WHERE camdenTables.hid=userTable.hid AND
userTable.hid='caAd0vwg1dzo' AND camdenTables.dt='2016-05-27'
```

- Output examples:

```
{"hid":"caAd0vwg1dzo","ts":1464307532,"heatmeter":{"instant":87,"flowRate":785
,"flowTemp":67.4,"returnTemp":67.2,"ts":1464307532,"cumulative":26861},"email"
:"dfdfdfd@hotmail.com"},
{"hid":"caAd0vwg1dzo","ts":1464307544,"sensors":[{"type":"Temperature","state"
:"20.906","ts":1464307544,"sid":6}],"email":"dfdfdfdfd@hotmail.com"}
```

2) SELECT_DAMP_DATA for selecting the data of one apartment for the damp identification application.

```
SELECT camdenTables.hid AS hid, camdenTables.ts AS ts, camdenTables.sensors AS
sensors, userTable.email AS email FROM camdenTables, userTable WHERE
camdenTables.hid=userTable.hid AND userTable.hid='cnsClXWefcFg' AND
camdenTables.dt='2016-05-27'
```

- Output examples:


```
{"hid":"cnsClXWefcFg","ts":1464307468,"sensors":[{"type":"Door
Sensor","state":"0","ts":1464307467,"sid":101}],"email":"dsfffgf@cnn.com"},
{"hid":"cnsClXWefcFg","ts":1464307468,"sensors":[{"type":"Temperature","state"
:"23.969","ts":1464307467,"sid":1}],"email":"gffgfgfg@cnn.com"}
```


Querying on more than one apartment can be done by sending several queries on each of the apartments.

There are two possible service id's representing the two possible purposes:

1) 39 for "Crowd-based Energy Recommendation"
2) 40 for "Personal Energy Recommendation"

In order to submit the query to the Data Access Controller one has to do the following:

1. Run Spark:

```
wp6@wp6:~/ConsentManager/ConsentManager$ ~/spark-1.5.0/bin/spark-submit --
class ConsentSQLApp ./ConsentManager-assembly-1.0.jar
```

2. Run the node-js application:

```
wp6@wp6:~/ConsentManager/ConsentManager/ConsentFilter$ node app.js
```

3. Send the query (using REST API) and obtain the answer (in JSON format), as above.

There are two use-case scenarios for the retrieval of the historical data:

1. **Heating schedule:** requires hid, email, timestamp, heatmeter and sensors (temperature, window) and is performed by the Case Based Reasoning analysis
2. **Damp identification:** requires hid, email, timestamp and sensors (temperature, humidity, window) and is performed by the data analytics analysis

A user can choose one of the following options:

- Opt-out the service completely and then all his data will be nullified, for example:

Query:

```
http://127.0.0.1:8080/sql?query=SELECT_HEATING_DATA&service_id=40&apartment_id
="cnsClXWefcFg"&date_id='2016-05-27'
```

Output:

```
{"hid":"","ts":0,"sensors":null,"email":""}
```

- Agree to share all his data, for example for the personal energy recommendation:

Query:

```
http://127.0.0.1:8080/sql?query=SELECT_HEATING_DATA&service_id=40&apartment_id
="caAd0vwg1dzo"&date_id='2016-05-27'
```

Output:


```
{"hid":"caAd0vwg1dzo","ts":1464310723,"sensors":[{"type":"Temperature","state"
:"20.875","ts":1464310723,"sid":6}],"email":"hdsfhdgghgsf@hotmail.com"},
{"hid":"caAd0vwg1dzo","ts":1464310747,"sensors":[{"type":"Temperature","state"
:"20.344","ts":1464310747,"sid":5}],"email":" hdsfhdgghgsf @hotmail.com"}
```

- Agree to share some of his data, for example for the crowd-based recommendation, only heatmeter data but not sensors data.

Query:

```
http://127.0.0.1:8080/sql?query=SELECT_HEATING_DATA&service_id=39&apartment_id
="caAd0vwg1dzo"&date_id='2016-05-27'
```

Output:

```
{"hid":"caAd0vwg1dzo","ts":1464310661,"heatmeter":{"instant":14,"flowRate":785
,"flowTemp":67.8,"returnTemp":67.7,"ts":1464310661,"cumulative":26861},"email"
:" hdsfhdgghgsf @hotmail.com"},
{"hid":"caAd0vwg1dzo","ts":1464310698, "sensors":null,"email":" hdsfhdgghgsf
@hotmail.com"}
```

4.2.1.4 Sequence Diagrams

The sequence diagrams for this case appear in Figure 118 and Figure 119.

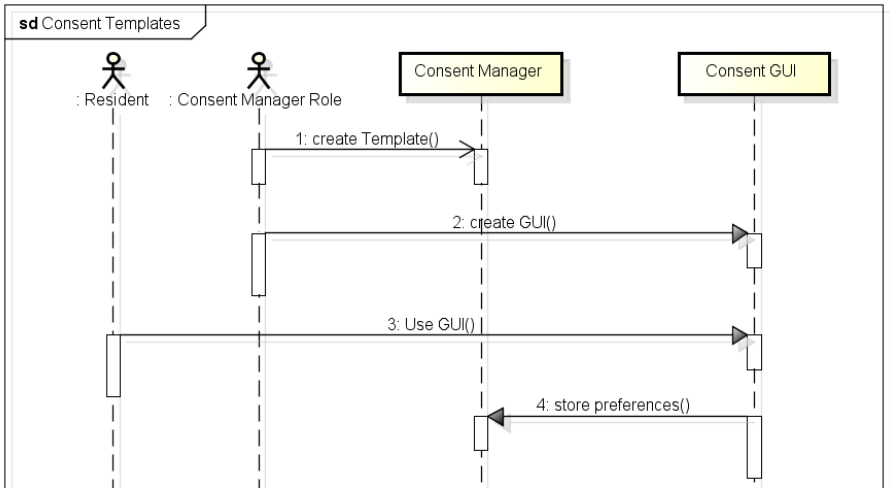


Figure 118: Sequence Diagram for Consent Template creation

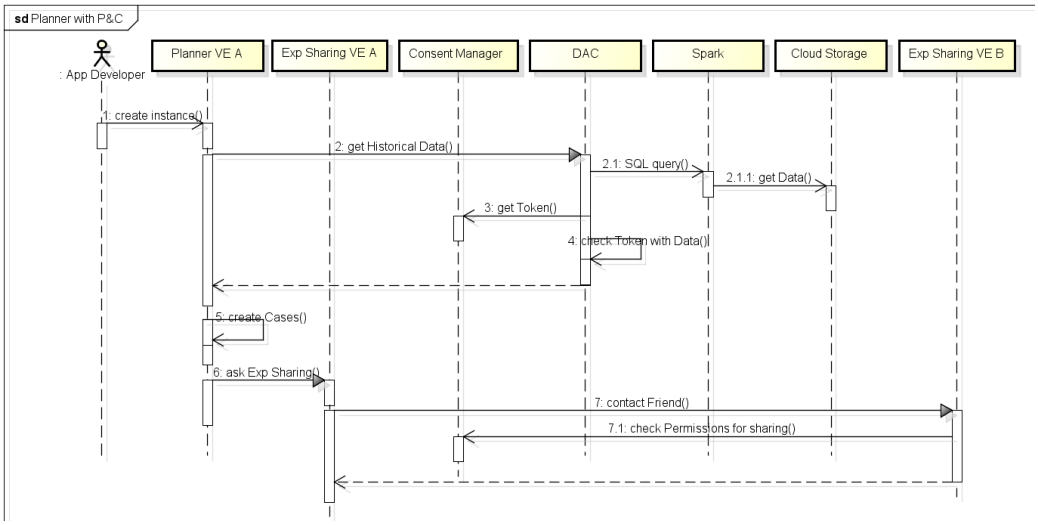


Figure 119: Sequence Diagram for Planner and P&C cooperation

4.2.1.5 Subsystem Test case table

Two test cases have been identified in this subsystem, appearing in the following tables.

Table 18: Test Case for Historical Data retrieval

Test Case Number Version	**Planner&CM_01**
Test Case Title	Planner Gets Historical Data
Module tested	Planner, Consent Manager, DAC
Requirements addressed	UR13
Initial conditions	The users have provided their consent preferences Spark and node-js application are up and running Node-RED instance and relevant flow
Expected results	Planner gets all the historical data if the user has provided its consent, otherwise the response contains “nullified” data fields which are not consented
Owner/Role	Application developer
Steps	Copy provided Flow in Node-RED instance Configure based on which is the specific apartment we are interested in (or all apartments) and which is the service (39 or 40) Trigger flow execution
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Table 19: Test Case for Experience Sharing disablement based on preferences

Test Case Number Version	**ExperienceSharing&CM_01**
Test Case Title	Override local case search when sharing is disabled
Module tested	Experience Sharing, Consent Manager
Requirements addressed	UR13
Initial conditions	The user has provided its consent preferences Consent Manager is up and running Node-RED instance and relevant flow
Expected results	When the user has not provided its consent for case sharing, the VE acts as a broker between its friends and the VE that requested a solution and therefore is able to provide an assist
Owner/Role	Application developer
Steps	Copy provided Flow in Node-RED instance Configure based on which is the specific apartment (VE) Trigger flow execution
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

4.2.1.6 Deployment Diagram

The deployment diagram for this case appears in the following figure.

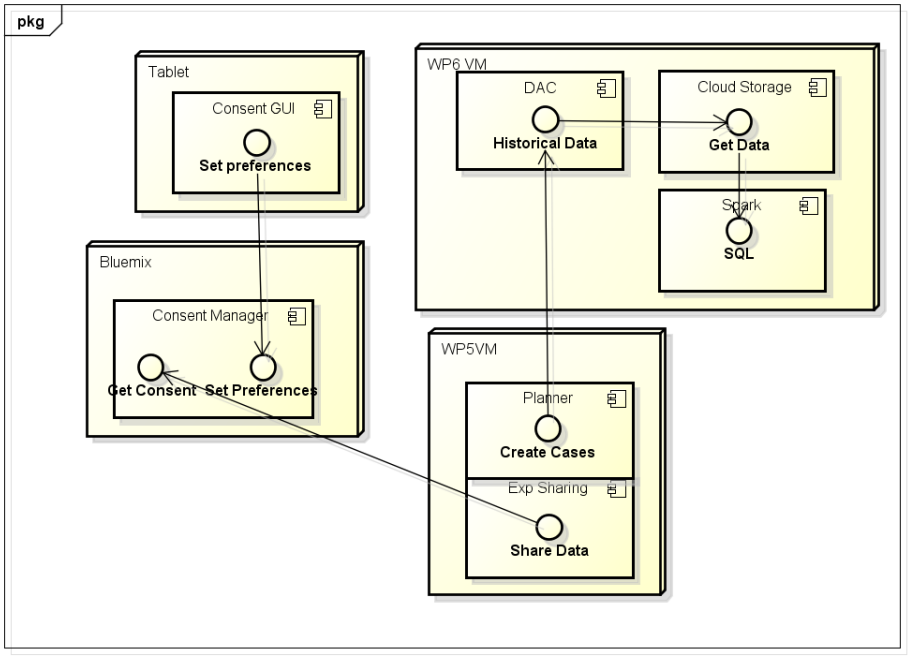


Figure 120: Deployment Diagram for the Planner and P&C Cooperation

4.2.2. Component installation to VE instances

With relation to the component installation to VEs, this has been already drafted in Section 3.2.3 along with the specifications for the software dependencies for the VE side image. The main difference with relation to Y2 refers to the inclusion also of the respective Node-RED flows in the image that refer to Y3 functionalities that appear in the following table.

Table 20: Y3 Node-RED flows incorporated in the VE side image

Functionality	Functional Component	GitHub link
Apply Privelets configuration	Privelets	https://github.com/COSMOSFP7/COSMOS-VE-Side/tree/master/Privelets
Push model of Privelets Service	Privelets	https://github.com/COSMOSFP7/COSMOS-VE-Side/tree/master/Privelets
Pull model of Privelets Service	Privelets	https://github.com/COSMOSFP7/COSMOS-VE-Side/tree/master/Privelets
Data feed processing and Event forwarding	µCEP, Planner	https://github.com/COSMOSFP7/COSMOS-VE-Side/tree/master/Planner
Retrieve and apply actuation plan, handling failsafes and user preferences	Planner	https://github.com/COSMOSFP7/COSMOS-VE-Side/tree/master/Planner

The integration point for the VE components is the Hardware Security Board. The device, equipped with a modern CPU, hosts the VE components as well as all security related hardware sub-systems.

In order to facilitate the integration and/or migration of these components, a brief description of the setup procedure is provided.

The hardware security components are accessible through memory mapped registry accesses. In order to use the components the operating system needs to be aware of the address space the modules are using. This is performed through the device tree which allows the kernel to map the address space correctly. To boot up the system 3 major components are needed:

- Bootloader – is a typical bootloader such as uBoot.
- Kernel – is a standard Linux kernel ported to ARM architecture and compiled for the Xilinx Zynq target (Xilinx itself provides the source code to do so).
- Userland – consists of a typical ARM release of Ubuntu core OS.

To boot up the system an SD card needs to be prepared following the guidelines provided by Xilinx (these step is similar to booting up a Raspberry Pi device).

Once the system has booted, following apps need to be installed (using the standard package manager) in addition to the dependencies presented for the Raspberry Pi device in Section 3.2.3:

- python;
- python-serial;
- nginx;
- shellinabox;
- build-essentials;
- libmosquitto;
- libmosquitto-dev;

For compatibility reasons a Linux operating system newer than 2014 is needed. The security related components, Privelets and uCEP/planner are copied and can be run directly – no manual configurations are needed. For the hardware related components please see deliverable D3.1.3, section 6 – Security Components [14].

4.2.3. VE Instances Descriptions and Registry population

4.2.3.1 Testing Scenario/Data Description

In the previous integration periods, details about the VE Registry GUI and process were displayed. In this version, the concrete APIs of the Registry are provided, along with the detailed schemas per case, that may enable the use of the component by the VE developers even in an automated programmable manner, useful for insertion of large scale VE descriptions or in a generally automated fashion like the one displayed in Section 4.2.4.

4.2.3.2 Subsystem from D7.6.3 with integration points

This is the same subsystem as described in Section 3.2.1.2, hence it is not repeated. Two main functionalities are foreseen in a nutshell, the insertion of VEs and their characteristics and the retrieval of VEs.

4.2.3.3 Message formats and configuration

The JSON schemas anticipated by the VE registry appear in the following list

- Virtual Entity

```
{"virtualEntityName":"",
"relatedPhysicalEntity":"",
"virtualEntityURI":"",
"location":{Location object},
"veProperties":[VEProperty object list]}
```

- VEProperty

```
{"virtualEntityURI":"",
"vePropertyURI":"",
"vePropertyName":"",
"vePropertyType":"ReadableWritableProperty or
WritableProperty or ReadableProperty",
"endPointURI":""}
```

- Location object

```
{"type":"fixed or mobile",
"locationURI":"",
"logicalLocationURI":"",
"latitude":"",
"longitude":""}
```

- Group Virtual Entity

```
{"name":"",
"uri":"",
"virtualEntities":[list of virtual entity URI's]}
```

- Message Bus Topic

```
{"topicRelatedEntity":"",
"topicURI":"",
"topicName":"",
"topicOwner":"",
"topicSubject":""}
```

- IoTService

```
{"serviceName":"",
"serviceURI":"",
"endpoints":[List of Endpoints],
"resourceLocation":Location object,
"relatedResource":""}
```

- VEService

```
{"serviceName":"",
"serviceURI":"",
"endpoints":[List of Endpoints(REST or MQ)],
"relatedResource":""}
```

- Endpoint
  - RESTEndpoint


```
{"endpointType":"COSMOS REST Service",
"endpointName":"",
"endpointURI":"",
"relatedResource":"",
"inputParameterStyle":"JSON",
"inputSchema":"",
"inputParameterURI":"",
"inputParameterName":"",
"inputDataType":"",
"inputSemanticType":"",
"returnValueStyle":"JSON",
"returnSchema":"",
"returnParameterURI":"",
"returnParameterName":"",
"returnDataType":"",
"returnSemanticType":""}
```


  - Message Bus Endpoint


```
{"endpointType":"RabbitMQ DB",
"endpointName":"",
"endpointURI":"",
"relatedResource":"",
"brokerURL":"",
"port":"",
"nodeName":"",
"exchangeName":"",
"routingKey":"",
"semanticType":"",
"serviceURI":"",
"rabbitMQConnectionConfigurationURI":"",
"topicURI":""}
```


4.2.3.4 Operations per entity

Following object format presentation in the previous chapter, the operations per entity appear in the following table.

Table 21: Operations per entity for the VE registry API

Entity	Operation	Details
VE	Insert	URI: /insert/virtual-entity Method: POST Input: Virtual Entity object in JSON format
VE Property	Insert	URI:/insert/ve-property Method: POST Input: VEProperty object in JSON format
GVE	Insert	URI:/insert/group-ve Method: POST Input: Group Virtual Entity object in JSON format
MessageBus Topic	Insert	URI:/insert/topic Method:POST Input: Message bus topic object in JSON format
Service	Insert IoT Service:	URI:/insert/iot-service Method:POST Input: IoT Service object in JSON format
Service	Insert VEService	URI:/insert/ve-service Method:POST

Input: VE Service object in JSON format

The search functionality is based on 2 main steps. The first one is getting search criteria for desired entity (one of these keywords: VirtualEntity, PhysicalEntity, VEProperty, IoTService, VEService, RESTEndPoint, MQEndPoint, MessageBusTopic). This is dictated by the need to reduce the code on the frontend side and to automate the process (if search criteria are changed, the change will be visible on frontend too). The second step is the actual search, which uses the criteria fetched earlier to get actual information about the desired entity.

Let's say we want to search for a virtual entity. First of all we need to fetch the criteria available for virtual entities. A GET request to http://cosmos.veregistry.eu/getcriterias?entity=VirtualEntity will return available criteria in JSON format (the url is just for the sake of demonstration, the relative path is important, marked with red). The JSON response looks like this:

```
{
"Physical Entity (URI)":["Equal","Starts With","Contains","After","Previous",
                         "Not Equal"],
"URI":["Equal","Starts With","Contains","After","Previous","Not Equal"],
"Name":["Equal","Starts With","Contains","After","Previous","Not Equal"],
"Location": null
}
```

Where each key represents an search criteria, and the values represent the operators available for that criteria ("Location" has a null value because it does not have any operators).

After we get our search criteria, we can build the JSON for the actual search. The JSON should look like this:

```
{
"type":"VirtualEntity",
"criterias":[
{"attribute":"Physical Entity (URI)","operator":"Contains","value":"brasov"},
{"attribute":"URI","operator":"Starts With","value":"http://brasov.ro"},
{"attribute":"Name","operator":"Contains","value":"council square"},
{"attribute":"Location","operator":0,
     "value":"45.439578220334305 24.996825456619263 132.56228245772854"}
],
"limit":"10",
"orderType":"asc",
"orderCriteria":"URI"
}
```

Where:

- "type" represents the type of the entity for searching,
- "criterias" represents a list with search criteria where each criterion must have an attribute, an operator and a value(attributes and operators must be from the search criteria JSON fetched earlier). If one wants to filter the search by location they could add to "criterias" another element with attribute "Location", operator 0 or null(it does not have any operators) and in the value field the coordinates of the point(latitude and longitude) and the distance around it in kilometers, with spaces between them(example: 44.5 22.3 256.45, where the first is latitude, second is longitude and third is distance in kilometers))
- "limit" is used to limit the number of entities returned by search process(0 or null to get all entities).
- "orderType" represents the type of ordering for result.It can be "asc" for ascendant or "desc" descendant.

- "orderCriteria" defines for each attribute of the entity the ordering which should be done(it's value is one of the search criteria fetched earlier).

A concrete URL example for search (the JSON is url encoded):

```
http://cosmos.veregistry.eu/search?json={%22type%22:%22VirtualEntity%22,%22cri
terias%22:[{%22attribute%22:%22Physical%20Entity%20(URI)%22,%22operator%22:%22
Contains%22,%22value%22:%22brasov%22},{%22attribute%22:%22URI%22,%22operator%2
2:%22Starts%20With%22,%22value%22:%22http://brasov.ro%22},{%22attribute%22:%22
Name%22,%22operator%22:%22Contains%22,%22value%22:%22council%20square%22},{%22
attribute%22:%22Location%22,%22operator%22:0,%22value%22:%2245.439578220334305
%2024.996825456619263%20132.56228245772854%22}],%22limit%22:%2210%22,%22orderT
ype%22:%22asc%22,%22orderCriteria%22:%22URI%22}
```

4.2.3.5 Subsystem Test case table

The subsystem test case table for the registry appears in the following table.

Table 22: Test Case for the VE Registry insertion and search functionality

Test Case Number Version	**VE Test_01**
Test Case Title	VE Registry Insertion and Search
Module tested	VE Registry
Requirements addressed	6.14, UR1-3, UR11
Initial conditions	Registry is online Relevant JSON fields have been populated
Expected results	VE description exists in the Registry
Owner/Role	VE developer
Steps	Create the relevant VE (or other entity) description Insert it via the GUI or via the automated API call Check search based on type (e.g. flat) Check search based on location coordinates and range
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

4.2.4. Populated registry integration and link to friend recommendation and fuzzification

Once the VEs have been included and described in the registry, they are available to be retrieved by other VEs who are searching for them based on similar attributes (this search has already been included in previous sections). However given that in many cases attributes need to be fuzzified based e.g. on user requests or on privacy considerations, an integration process needs to be performed including the fuzzification capabilities of the privelets mechanism. Once the VE has been described in the registry, the VE developer may apply the privelets mechanism, as this has been abstracted through the relevant Node-RED flow, retrieve the normal VE description and alter the attributes based on the fuzzification logic. The updated VE

description is then stored again in the registry and can be used in friend recommendation, with the advantages of the increased privacy of VE properties.

In general, friend recommendation can be based on a two- step process. Initially based on a query to the registry to get the similar VEs e.g. in terms of location, and based on the fuzzified values. Then from these we may query directly all of them at the Planning level in order to filter based on specific app level parameters or behavior (for example from the validation results of the Planner the desired Temp value of the resident is one key feature for getting usable cases).

4.2.4.1 Testing Scenario/Data Description

The specific testing scenario includes the existence in the registry of a set of flat VEs with location properties. However the residents do not wish for their actual location to be exposed, for a variety of purposes, including privacy considerations and risk of data leakage that could aid e.g. criminals from understanding the occupancy pattern of each flat. The residents however wish also to participate in the social aspects of COSMOS. The tradeoff achieved for this case is that the flat location should be fuzzified to a certain extent, so that their house is not exposed via its location property, but also the location property should not be too fuzzified thus contaminating the recommendation process (e.g. if we need a similar flat in London, the location should not be too fuzzified to show the flat in another city). The application developer on the other hand needs VEs within the specified radius (in red). The functionality based on the Registry GUI is portrayed in the following figure:

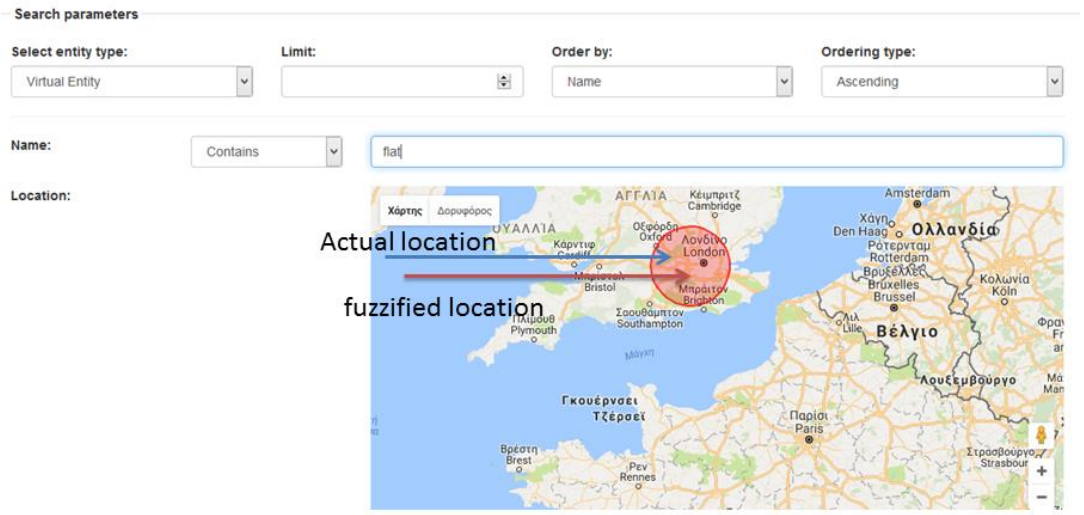


Figure 121: Search functionality based on radius to be tested with fuzzified values

4.2.4.2 Subsystem with integration points

The relevant subsystem is included in the following figure. One IP2 type integration point is between the App Developer and the registry to retrieve VEs and two IP4 type integration points refer to the actual description of the VE and the configuration of the privelets mechanism.

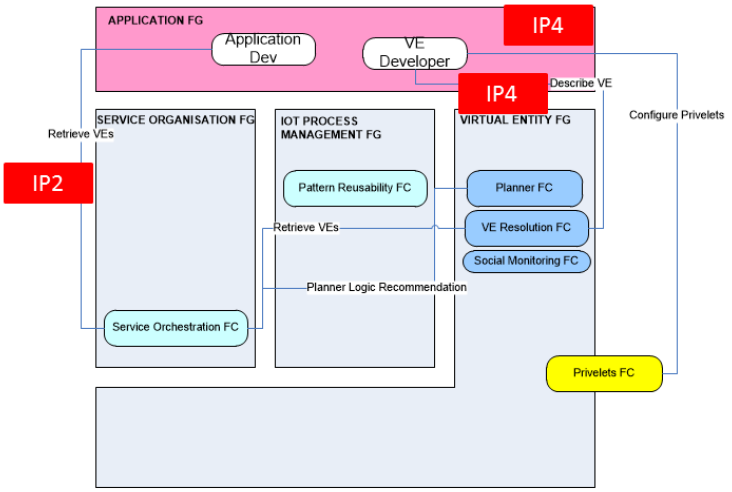


Figure 122: Privelets and VE registry integration subsystem

These integration points are portrayed in the following Node-RED flow and especially for the case of the VE developer (the App developer IP2 is simply a subcase of the included VE retrieval needed in the VE developer IP4 points). Privelet configuration process is included in detail in D3.2.3. Initially the VE developer may have registered the VE with the normal values in the respective fields. Upon privelets activation and configuration, they need to retrieve this description again, pass it through the privelets push model and then update the registry with the new data item. If the VE developer needs to directly insert the fuzzified description, then they may directly link the "Register the VE" subflow to the privelets push model node.

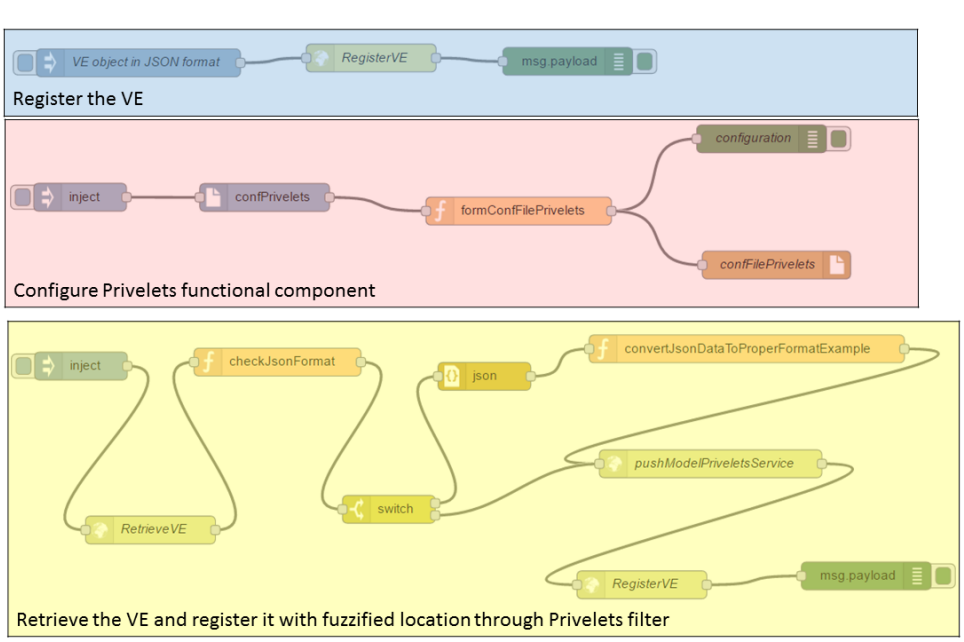


Figure 123: Node-RED flow for Privelets and Registry Integration

4.2.4.3 Message formats and configuration

Message formats for the registry component have already been described in Section 4.2.3 as parts of the official registry API.

4.2.4.4 Sequence Diagram

The sequence diagram for this case appears in the following figure.

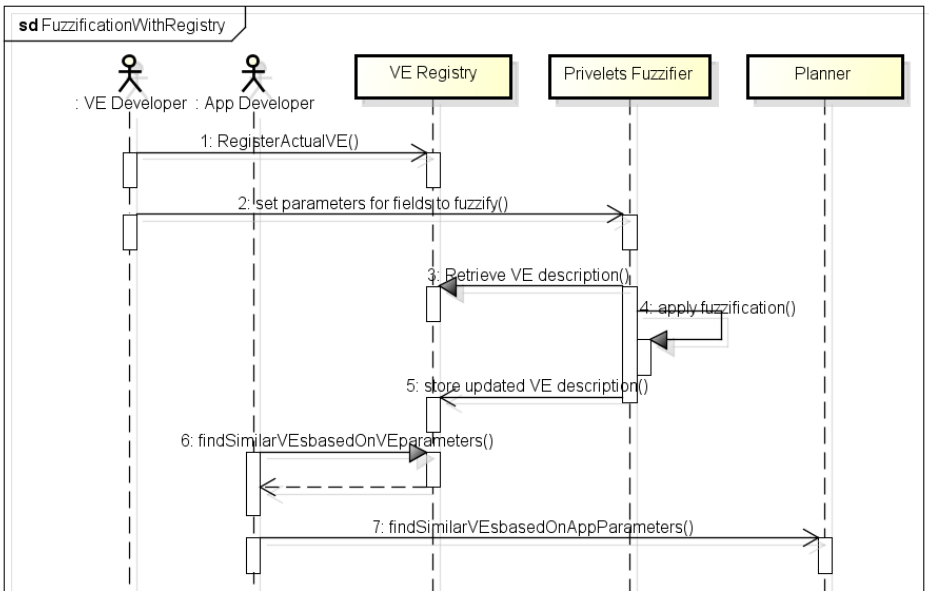


Figure 124: Sequence diagram for integration of VE registry and fuzzification/friend recommendation process

4.2.4.5 Subsystem Test case table

Table 23: Test Case for the Fuzzified property insertion

Test Case Number Version	**Fuz_Re 01**
Test Case Title	Fuzzify VE description properties
Module tested	Privelets, Registry, Node-RED Flow
Requirements addressed	UR13
Initial conditions	VE description existing in the registry
Expected results	A specific property of the VE is being fuzzified, with updated values based on VE developer preferences
Owner/Role	VE developer
Steps	Copy provided Flow in Node-RED instance Configure based on provided README file in flow the popup menu of the privelets, to determine which data field should be fuzzified and how much Check new description in Registry to have updated values Check search capabilities
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

4.2.4.6 Deployment Diagram

No deployment diagram is provided since this test case refers to established components of the platform and VE, included in previous sections of this document.

4.2.5. Data Feeds Health Checks

4.2.5.1 Testing Scenario/Data Description

In the previous integration periods, details about the Data Mapper were displayed. During Y3 we realized the importance of monitoring and automatically repairing failures in the component. These failures can occur due to lack of stability and maturity of the Secor project the mapper is based on. This section is relevant for the scenarios described in Sections 4.2.11, 4.2.12, 4.2.13. Since Secor has no internal mechanism that supports health functionalities we have created a system that takes advantage of the COSMOS MQTT Message Bus and the capacity to set rules within the µCEP.

Health detection – when Secor uploads an object to object storage it writes the upload status to its log, resulting in a small change in the log file size. Without needing to perform any modification at all in the machine where Secor is running, the µCEP, and more precisely, the Event Collect module, continually checks the properties of Secor logs. If for a predefined time (e.g. several hours), there was no change in log size, we can assume the data mapper is malfunctioning and the µCEP can generate an event.

Recovery – we can recover Secor as a response to a malfunction event, without any data lose. No data lose is possible as part of Secor offset update mechanism which stores in a zookeeper server the last offset of messages that were successfully uploaded from Kafka to object store. A new instance of Secor will continue uploading messages from the last offset stored in the zookeeper server. Worth mentioning that no additional script is needed at Secor machine in order to reboot the service, as the µCEP, and more precisely, the Complex Event Publisher submodule can actuate remotely on the machine.

4.2.5.2 Subsystem from D7.6.3

This relates to the classic data feed ingestion (already presented in the previous periods in Section 3.2.5.1.2), including the health check status.

4.2.5.3 Message formats and configuration

The only configuration that needs to be done in the machine where Secor is running is to provide the means to access the machine using SSH key pairs. This is easily done adding the public keys of the µCEP Collector and Publisher submodules into Secor machine's ~/.ssh/authorized_keys file.

The main configuration of the Collector and Publisher submodules relies on the actions to be performed remotely at the machine, which are provided as Python scripts:

```
def check_secor():
    exec_script = shlex.split("/usr/bin/ssh health@wp42.iot-cosmos.eu \
     'echo \"1 WatchSecor int size `stat -c%s ~/secor/secorBin/logs/secor.log\
     int ts `stat -c%Z ~/secor/secorBin/logs/secor.log`\" | mosquitto_pub -h \
     lab.iot-cosmos.eu -p 1883 -t \"cosmos/monitor/secor\" -s'")
    subprocess.Popen(exec_script)

def stop_secor():
    exec_script = shlex.split("/usr/bin/ssh health@wp42.iot-cosmos.eu \
        'pkill -f \"com.pinterest.secor.main.ConsumerMain\"'")
    subprocess.Popen(exec_script)

def start_secor():
    exec_script = shlex.split("/usr/bin/ssh health@wp42.iot-cosmos.eu \
        '/usr/bin/java -ea -Dsecor_group=secor_backup \
        -Dlog4j.configuration~/secor/secorBin/log4j.prod.properties \
```

```
        -Dconfig=~/secor/secorBin/secor.prod.partition.properties \
        -cp ~/secor/secorBin/secor-0.1-SNAPSHOT.jar:~/secor/secorBin/lib/* \
        com.pinterest.secor.main.ConsumerMain > \
        ~/secor/secorBin/logs/secor.log 2>&1 &'")
    subprocess.Popen(exec_script)
```

Basically, by subscribing to certain MQTT topics the submodules are able to trigger the above actions, being the parsed events that the collector sends to the engine similar to:

```
1 WatchSecor int size 306778 int ts 1472639203
```

And the complex events:

```
2 ResetSecor int cuenta 0 float average 306778.000000 string alarm "reset"
```

4.2.5.4 Sequence Diagram

The following sequence diagram describes the automated process to restart Secor in case of failure. Upon termination, the Publisher submodule sends a message to the Message Bus.

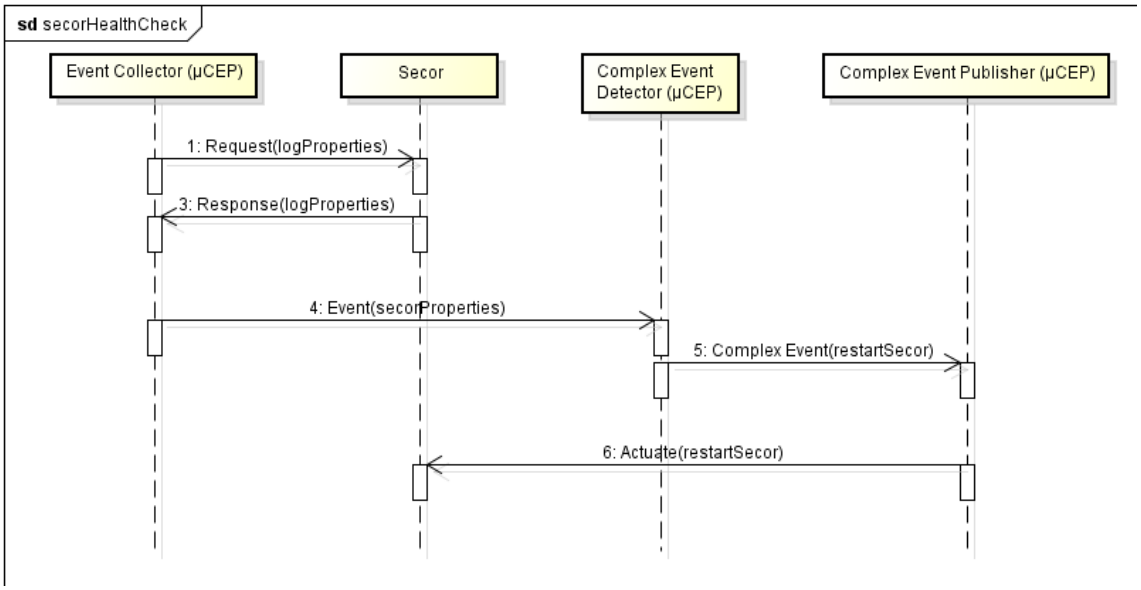


Figure 125: Secor health check Sequence Diagram

4.2.5.5 Subsystem Test case table

Test Case Number Version	**Secor_01**
Test Case Title	Secor health checks and fault identification
Module tested	Secor
Requirements addressed	6.4
Initial conditions	Secor is running MB is running µCEP is running
Expected results	Upon log file not changing, alarm raised and secor restarted
Owner/Role	COSMOS developer
Steps	Configure µCEP Collector submodule to read Secor log Configure µCEP Publisher submodule to actuate on Secor

	Configure µCEP rules Manually halt Secor to emulate failure
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

4.2.5.6 Deployment Diagram

No deployment diagram is provided since this test case refers to established components of the platform and VE, included in previous sections of this document.

4.2.6. Fab/Lab integration

4.2.6.1 Scenario Description

In the context of the IERC activities, AC4 is investigating aspects of bridging IoT with ART. To investigate such potential opportunities, we have contacted a fabrication laboratory (http://www.decodefablab.com/) in order to engage in a potential collaboration, in which COSMOS will act as the data provider, while the fab/lab will translate this into a form of creative arts design and prototype. The final target is to define a workshop structure that can then be added in the regular program of the Fab/lab. Workshops in the context of a fab/lab include the participation of external people (usually with a fee) that take part in the design and fabrication process, with the final goal of participating in a wider visibility event for which the workshop is targeted (e.g. produce an artistic item that will participate in an exhibition) and gaining experiences from the process.

The requirements posed by the Fab Lab were the following:

- Any concept combination would have to involve strictly localized data (e.g. from Athens, where the lab is located, or at most Greece) so that the public interest would be enhanced.
- Any concept would have to involve a fabrication stage (COSMOS would not be directly involved in this but the acquired data should enable it)
- The resulting product would be more of an artistic product than a purely functional one

One interesting candidate in this case was the data feed input to be weather data from Greece.

4.2.6.2 Meteo.gr data feed

Meteo.gr provides online real time data from weather stations across Greece (not directly on the sea such as buoys) with aspects such as Temperature, wind, rainfall, humidity etc. (http://meteo.gr/meteoplus/observationsFull.cfm)

It also includes predictions for the future (4 next hours) in terms of processed information including wave size (http://meteo.gr/meteoplus/wave_maps.cfm), but this does not seem to be real time. Prognosis can be based on further future steps.

So in this case COSMOS will need to access and retrieve the data for the overall area. What will be necessary for sure is the gathering of all the map tiles (individual tiles are given per REST call) and the exposure as a service of the results in the form of a URL to be passed to

Grasshoper for modelling purposes, in a csv or XML format. Thus the overall flow is represented in the following figure. Potential combinations can also be performed with the cooperation of social network data, e.g. for checking climate vs sentiment mappings.

Figure 126: Data flow sequence for the Fab/Lab scenario

4.2.6.2.1 Data feed description (add template etc.)

Data from meteo are given in a map form from:

http://www.meteo.gr/meteoplus/wave_maps.cfm

For getting the data, a suitable GET call may be performed in order to extract from each map tile the individual points. A typical GET url is the following:

http://www.meteo.gr/meteoplus/Sailing/templates/waves_sp1JSON.cfm?SailMapID=4&FTime=5

with parameters the sailmapID and the Ftime. Tile ids have been identified and are portrayed in Figure 127. Data are given for static timestamps per 6 hour slots in the day (2:00, 8:00, 14:00 etc.). Ftime is the forward time, with the value of 3 indicating the next slot prognosis. Values higher than 3 indicate further time slots prognosis. We are mainly interested in the 1-step ahead, thus a sampling GET needs to be obtained every 6 hours.

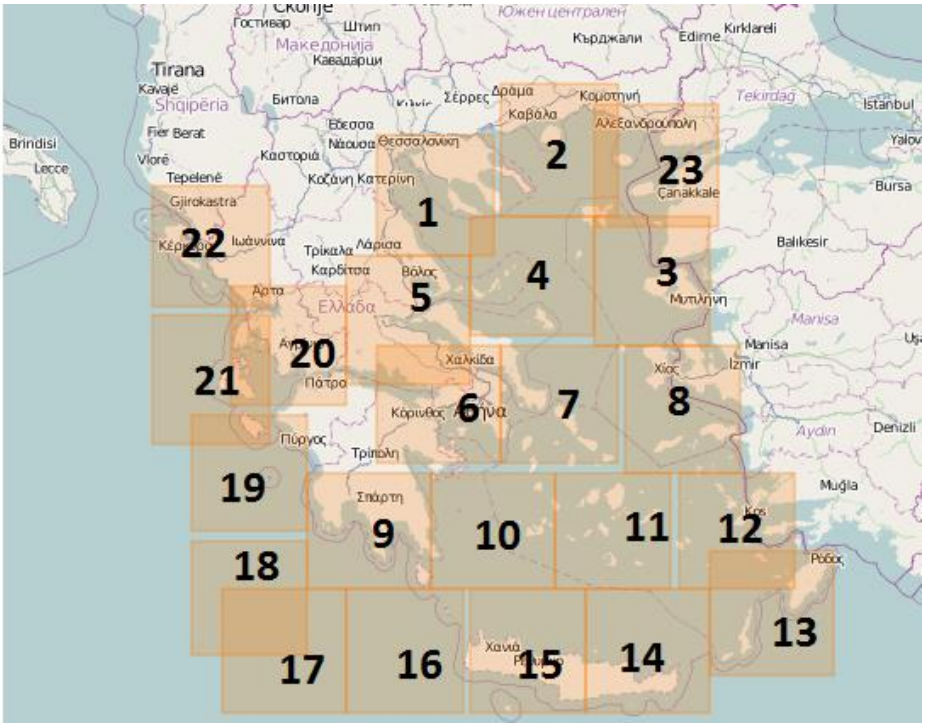


Figure 127: Map tile IDs for the Meteo.gr case

4.2.6.2.2 Subsystem from D7.6.2 with integration points

The subsystem used is the same as in Section 3.2.5.1.2, with the only difference that it does not come from an IoT Service or device, but it is adapted to the COSMOS platform through the use of the Service Orchestration component (implemented via Node-RED in our case). Furthermore it is not redirected to the MB and Cloud storage due to the CSV requirements posed by the Fab-Lab.

4.2.6.2.3 Message formats and configuration

Returned data are of the form of arrays of csv vectors such as the following:

```
[4,3784,61,1.10,1.12,61,3,0,24.20,38.60]
```

And the mapping to concepts is the following:

```
[Ftime, tile_point, degrees of wave, size of waves1, size of waves2, degrees, always 0, lon, lat]
```

The two different size of waves are very similar values, so for the purposes of the fab lab either of them can be used (probably indicates the significant size of the waves in the beginning and end of the interval or a margin of error in the estimated values). Tile id is not included in the data, but map point is an id of the point e.g. 3784. As an id potentially the lon,lat pair can also be used. Mapping between these potential ids and the coordinates in a fabricated structure should be performed.

The sequence of necessary actions includes:

- Triggering of the GET call every 6 hours to get the next prognosis
- Traversing across the various map tiles (and based on the area of interest) and getting all relevant tile id data, in which we need to add also the timestamp of the triggering time
- Filtering the values of interest, adapting to JSON format and clearing of duplicate values. Duplicate values appear due to the fact that a point may appear in the overlaps between areas (e.g. this can be seen from Figure 127 for tile IDs 1 & 4 among others)
- Adaptation to the format needed by Grasshopper and forwarding them in real time
- Storing the data either locally or in a database for querying of historical data.

4.2.6.2.4 Node-RED flows

No sequence diagram is provided in this case since it is a simple interaction between the Node-RED flow and the Meteo website. The Node-RED flow for the Fab-Lab data collection appears in Figure 128. This is triggered once and then automatically polls every 6 hours in order to obtain the new data, performing all calls for the needed tiles. The obtained data are formatted and stored either locally in a csv file (preferred solution by FabLab) or may be redirected towards a more organized data management solution (e.g. MySQL db).

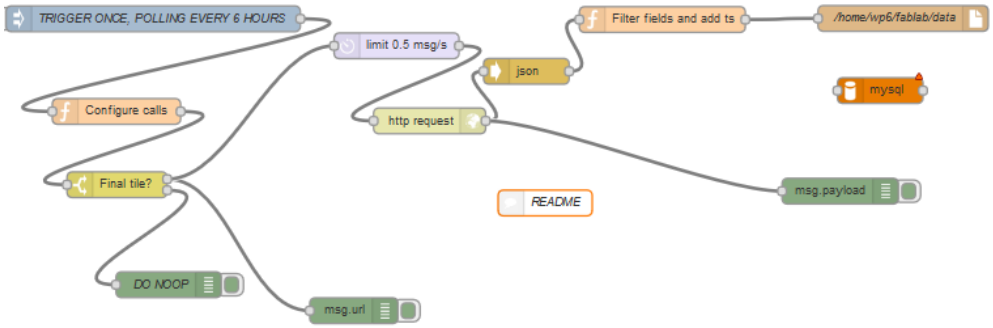


Figure 128: Node-RED flow for the Fab-Lab data collection

4.2.6.2.5 Subsystem Test case table

Table 24: Test case table for Fab Lab data acquisition

Test Case Number Version	FAB_01
Test Case Title	Fab Lab Weather Data Acquisition and Formatting
Module tested	Node-RED Flow
Requirements addressed	N/A for COSMOS related requirements. The requirements posed by the Fab Lab for data collection were included in the previous sections (4.2.6.1).
Initial conditions	Node-RED instance and relevant flow Time.js package installed and included in settings.js file (process mentioned in Section 3.2.2.6)
Expected results	Meteo.gr data stored locally in a csv format (compatible to Grasshopper)
Owner/Role	Application developer
Steps	Copy provided Flow in Node-RED instance Configure based on provided README file in flow Trigger flow to start polling every 6 hours
Passed	Yes
Bug ID	None
Problems	Duplicate values may exist due to overlapping areas between tiles Meteo.gr server seems to be overstrained by concurrent calls, hence we need to add an artificial delay between each tile call
Required changes	-

4.2.6.2.6 Specific tests needed

During the execution of the queries, it was observed that when trying to perform all GET calls simultaneously for all needed tile ids towards Meteo.gr, a varying number of them (typically 20%) failed, returning null data objects. For this reason, we have inserted a rate limiter in the Node-RED flow that holds requests in order not to exceed the set rate (currently set at 0.5 queries per second). This alleviates burst stress to the server and at the same time does not affect the data gathering process which in any case is much slower (data are sampled every 6 hours).

4.2.7. Incorporation of end user feedback in technical outcomes in a nutshell

4.2.7.1 Madrid Mobility Feedback Conclusions

In the case of the Madrid Scenario, end user feedback has been received with relation to the following items:

1) List of events that need to be identified in the context of the route monitoring for people with special needs by their caregivers
2) Front-end interfaces for people with special needs mobile app section and in the route monitoring and the caregiver portal
3) Madrid city council feedback on traffic management aspects with relation to which parts of the city network are most important
4) Madrid city traffic controllers with relation to the feedback and how this should be propagated back to the COSMOS platform

More details on the way this feedback was performed and overall conclusions are included from the respective UC side is in the respective use case documentation (Deliverables D7.5.3. and D7.3.3). In this section we highlight which aspects from the core COSMOS system were affected by this feedback.

4.2.7.1.1 Effect on Technical Implementations

From the core COSMOS system side, the conclusions presented have been incorporated in the following manner:

- Regarding point 3), this was interpreted in the MPs included in the scenario described in Section 4.2.11 and the specific road segments investigated
- Regarding point 4), this was interpreted and implemented as a feedback mechanism described in detail in Section 4.2.11.
- Regarding point 1) one of the most highlighted events in the caregiver survey (78.5%) was the incorporation of information about demonstrations and public events which in our case is identified through the usage of social data, identified in Section 4.2.11.

For the remaining points, these were handled within the UC systems as detailed in D7.5.3 [15] and D7.3.3 [16].

4.2.7.2 Camden Feedback Conclusions

In the case of the Camden Scenario, end user (residents) feedback has been received with relation to the following items:

1) The kind of **services** that the residents are expecting from a smart heating system: learning heating preferences, remote control and automatic adjustment according to external weather conditions.
2) The preference of the residents regarding the **desired temperature set-up**. Only 13% of them would prefer to set the temperature the way they do now (by using the programmer and temperature controls), whereas the rest of them would prefer the smart system to set the desired temperature (automatically) or to set the temperature on their own through an App.
3) The ease of use of a UI by the residents for setting the desired temperature. A simple UI was presented to the residents and it was well-perceived.

4) **Notifications & suggestions** that are of interest to the residents. Only 8% of the respondents answered that such a feature would be unnecessary. Such notifications & suggestions chosen or proposed by the residents are listed in the next subsection.
5) The **means** by which the residents would prefer to receive the above suggestions/notifications. The great majority of the respondents would prefer a mobile device.
6) The willingness of the residents to **share their data** with other users and/or third parties. About 40% of the respondents proved to be unwilling to share their data with other entities. However, it should be noted that they were more willing to share their data with other users rather than with third parties, showing that a community sharing app would be more preferable than centralised services from certain companies.
7) Since the smart system could automatically turn the residents' heating on/off based on **home occupancy**, the residents where asked about the preferred means of home occupancy detection by using another app (e.g. Google calendar), extra sensors, GPS or manual input. It was proven that most of the respondents did not like the idea of automatic home occupancy detection.
8) The interest of the respondents in providing their maximum daily or monthly **heating budget** as input to the system so that it could then predict if this is achievable with the current settings or send proposals to change them. More than 50% agreed with that option.
9) The end-users willingness to rate the performance of a smart heating system (for instance using a 5star rating interface) on a daily basis and for a few days following its installation. This would help the system to 'train' itself and learn from their specific requirements.
10) Any **special circumstances/events** (user preferences) that the residents recognized during which the heating schedule/settings should be changed. A list of such events is presented in the next subsection.

Identifying the Camden council and Hildebrand as end users consuming COSMOS services, one more topic can be added to the end user feedback:

11) **Failsafes** identified from the UCs and included as special configuration foreseen in the architecture of the COSMOS Smart Heating Application.

More details on the way this feedback was given and overall conclusions are included n the respective use case documentation (Section 7.4 at D7.4.3).

4.2.7.2.1 Effect on Technical Implementations

In this section we highlight which aspects from the core COSMOS system were affected by the feedback. The conclusions presented have been incorporated in the following manner:

- Regarding point 1), this feature is covered by the Planner and the CBR technique it uses at it is presented in the technical deliverables of WP5.
- Regarding points 2), 3) and 5), a new front-end was designed enabling the completion of a Smart Phone Application. Its features are being presented in subsection 4.2.9.
- Regarding point 4), the desired notifications proved to be the following:
  - "Temperature is at X range"
  - "Risk of damp/condensation"
  - "Turn on/off the heating"
  - "Close window" (to save energy/cost)

- "Risk of frost damage"
- "System is malfunctioning"
- Weather forecast
- Estimated costs of consumption to that point

At least part of these suggestions and notifications will be available through our application. More details can be found in subsection 4.2.9.

- Regarding point 6), the fact that the residents proved to be more willing to share their data with other users rather than with third parties is important for identifying the usefulness of technical components that focus on the sharing and storage of users' data (like the Experience Sharing component, Consent Management and the COSMOS Trust & Reputation model).
- Following point 7), it was decided to not include any kind of automatic home occupancy detection at the corresponding scenario.
- Regarding point 8), the COSMOS Smart Heating Application could be linked with **EnergyHive**. This featured is presented in subsection 4.2.9.
- Regarding point 9), a simple rating interface could be added to the front-end in order to receive the evaluation of the programmed heating schedules from the residents.
- Regarding point 10), special circumstances that have been identified by the residents are:
  - Period of illness and health issues
  - Change in occupancy because of holidays, business trips, staycation, etc.
  - Short notice/unexpected change of plans e.g. coming back from work early
  - Additional occupancy e.g. because of visitors
  - Individual needs e.g. elder people having different needs from younger ones
  - Extreme weather conditions and unseasonal weather
  - Case of repairs or system failure
  - Doing physical work at home vs sitting down for hours
  - Previous activities before entering house e.g. joking, swimming
  - Cooking (needing to turn off heating or opening window just in the kitchen)

  Through our application, at least part of these special circumstances will be available for selection. More details can be found in subsection 4.2.9.
- Finally, regarding point 11), certain technical failsafes (e.g. temperature/humidity sensors malfunctioning, low connectivity) provided Camden and Hildebrand can be monitored via specific Node-RED flows. When specific problems are being detected, notifications could be sent to the respective officers (introducing a wide alerting system for officers with relation to tenants). This feature is not present at the application front-end, this not visible from the residents.

To sum up, the aforementioned end-users points have significantly affected the technical implementation and provided functionality.

4.2.7.3 Sound Analyzer Feedback Conclusions

As mentioned in D7.6.3, in Y3 we have also investigated an extra Use Case scenario, not part of the official COSMOS cases, with relation to the usage of the Planner as a backend mechanism for identifying home environment sounds and visualizing them in order to be understood by people with hard of hearing issues. This was triggered by the assignment of a relevant MSc thesis at NTUA and was also supported as a proposed (and finally implemented) topic of the NTUA Hackathon.

The main point of end user involvement referred to a group of people with the specific health issue, which investigated the issue with relation to how such an app could be built (in terms of end user interfaces, functionality of these interfaces, ways of visualization etc.). The group consisted of 9 participants with an average age of 23, all being pre and post graduate students. They all agreed that the scope of such an app would be really useful in their day to day activities and living environment.

4.2.7.3.1 Effect on Technical Implementations

The main effect of the end user involvement referred to the determination of the sound list that needed to be taken under consideration based on common problems faced in the day to day activities of a person with hearing deficiency. This was determined along with potential visualization ways and is included in the following table (Table 25). The majority of these sounds have been incorporated in the respective scenario, based also on the available samples that could be obtained online for training of the CBR technique.

Table 25: End user with hearing impairment feedback on sound list

Sounds - Events	Led Color	Blinking Frequency (times/sec)	Further description
Door bell	White	1	
Knock on door	White	1	
House alarm	Orange	Very fast with strong light	
Fire alarm	Red	Very fast with strong light	
Baby crying	Green	Moderately fast	
Phone calls or SMS	Purple	1	
Wake up alarm	Light Blue	No light but mobile-smart watch vibration	Since deaf people will be sleeping, it would be preferable to wake up by really strong vibration from the smart watch they wear or the mobile phone.
Flowing Water	Blue	1	There are some cases that deaf people are washing their hands or taking a shower and then they are distracted by something else and forget the water that is still running.
Air Conditioning	Grey	1	There are some cases that deaf people leave the house without turning off the AC because they don’t hear it and forget about it.
Fridge Door open	Yellow	1	When you leave the door open, some fridges make a warning noise and deaf people can’t hear it.
Boiling water in the kitchen	Yellow	1	This one has the same color with the fridge because both events are located into the Kitchen. So, you just have to go over there and check which one of those two is happening.

4.2.8. Packaging and abstraction process

4.2.8.1 Assets creation

The updated and revised table of COSMOS artefacts (originally in D7.6.3 [7]) appears in the following table.

Item/Group name	Functionality	Way of exposure
Madrid Traffic data adaptation Node-RED flow	Retrieve, parse and store open data from sensors	Node-RED flow, Node-RED repository, COSMOS GitHub, other websites (open-platforms.eu), IBM Bluemix Madrid Traffic demo
Machine Learning scripts	Clustering, Prediction, Statistical Analysis	COSMOS website, GitHub repository, IBM Bluemix Madrid Traffic demo
Twitter Data ingestion flow	Following adaptation in Node-RED this flow is responsible for the ingestion in the Cloud storage of Tweets	Node-RED repository, COSMOS GitHub, other websites (open-platforms.eu)
EMT Reactive Box interfaces clients	Create registration and receive information from EMT Reactive Box, Push information as notifications to the app	Available through COSMOS GitHub account as Node-RED flows
µCEP	Complex Event Processing Engine	COSMOS GitHub, other websites (COSMOS website, open-platforms.eu)
Clients to consuming/producing data or Events	Register to MB and consume events	Through the Events MarketPlace (www.eventflows.com) with the use of clients (Java and Node-RED versions) available through COSMOS GitHub
Smart events flow	Rules boundaries via historical data and ML	source code, scripts and documentation which will allow running the Madrid Traffic demo on a user's Bluemix account using Bluemix services
Planner code	Dynamic CBR definition	As a Java library/code with easy configuration points, available through the COSMOS GitHub and website
Raspberry Pi image of VE side components	Overall VE side functionality	Download from website/GitHub
Trust model	As a simulator file for TRMSim-WSN	Download from website/GitHub
VE side management flows	Includes fuzzification and privacy definition interface, link to local MB, inclusion of fail-safes etc.	COSMOS GitHub

Sound Analyzer /Detector app	Instantiation of the Planner application for the Sound analysis scenario	Available through GitHub account of the participating Hackathon team
Smart Mobility App	SP front end	In Google Play in collaboration with the Inlife project
Fab/Lab data collection		Forwarded to interested Fab Lab
Fab Lab data collection flow		Available through COSMOS GitHub, website, Node-RED repository

4.2.8.2 Generalization and abstraction of flows

One of the aspects pursued in this period was also the improvement of the generalization and configuration capabilities of the various functionalities. This is necessary in order to enhance the usability of the tools provided by COSMOS. Indicatively, the privelets configuration menu through Node-RED is presented in the following figure, that simplifies the process for the developer.

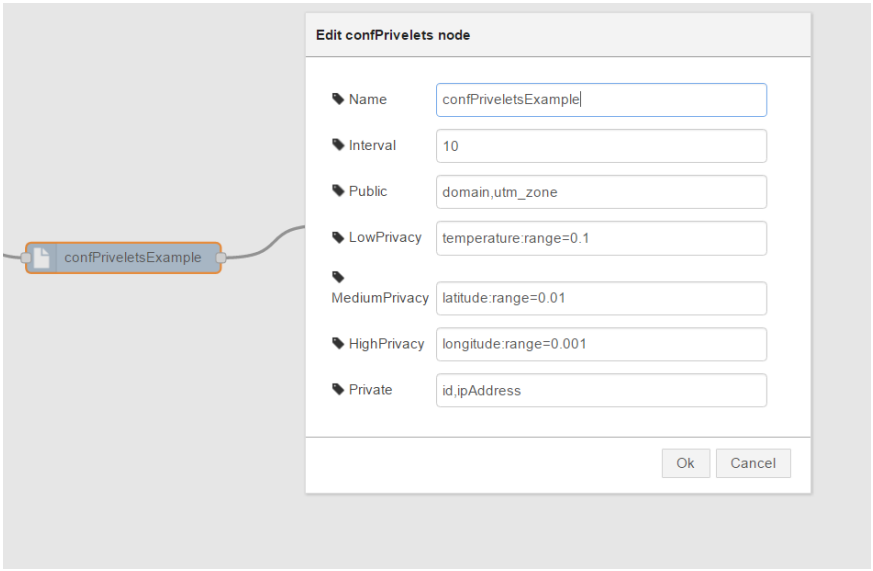


Figure 129: Example of privelets configuration abstraction

Another typical case refers to the Planner enhancement in order to operate as a service, exposing interfaces through which the component may receive problem structures and respond with solutions through a POST operation. This way calls may be more abstracted and flexible, as also indicated during the NTUA hackathon in which this form of communication was specifically appreciated.

```
{
    "payload": {
        "message_type": 1 or 2 (implemented 1),
        "problem_attributes": "hasOne#hasTwo#hasThree",
        "problem_values": "wow#such_blah#much_talk",
        "solution_attributes": "hasFour#hasFive#hasSix",
        "forSharing":"false",
        "weights":"0.5#0.4#0.1",
        "threshold":"1.0"
    }
}
```

Also the application of the subflows principle for hiding internal details of flows in Node-RED was used. Most of these additions were especially inspired through the NTUA Hackathon initial interaction with developers and their implementation during the hackathon duration was regarded as highly beneficial by the participating developers.

4.2.8.3 Online Availability

As also documented in more detail in D8.3.3 regarding dissemination activities, multiple paths have been applied for making the COSMOS developments available to the general public. Highlights of these relating to the technical parts have been included in this document.

Initially, we have created a Software and Manuals section on the COSMOS website for including relevant items (example Figure 130). This provides a short description and all relevant links to external repositories such as GitHub, Docker, etc.

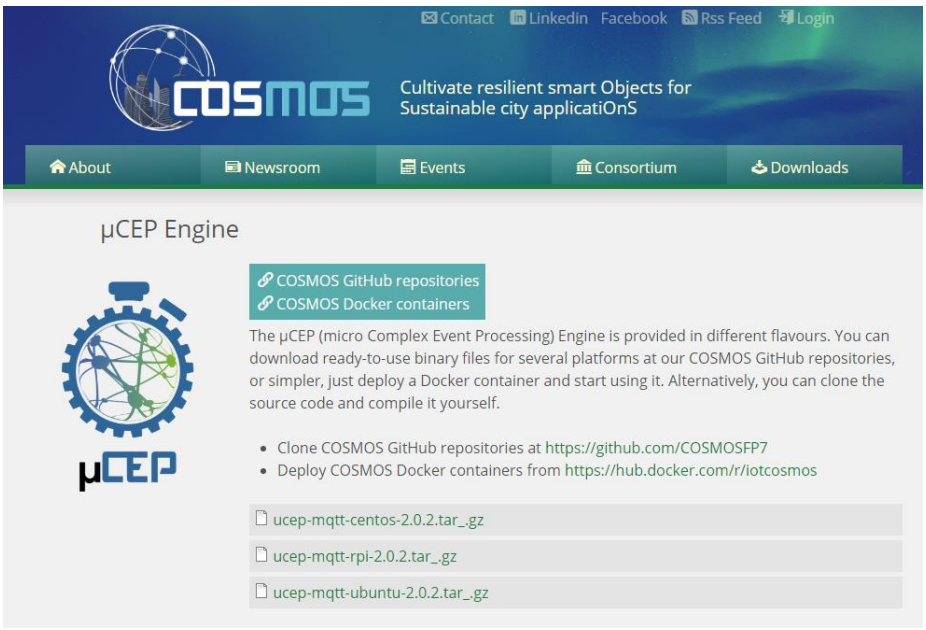


Figure 130: COSMOS website software and manuals example

The core of the release includes the COSMOS GitHub account, in which separate repositories for VE side, Platform side and Marketplace side have been created, including the relevant items per category.

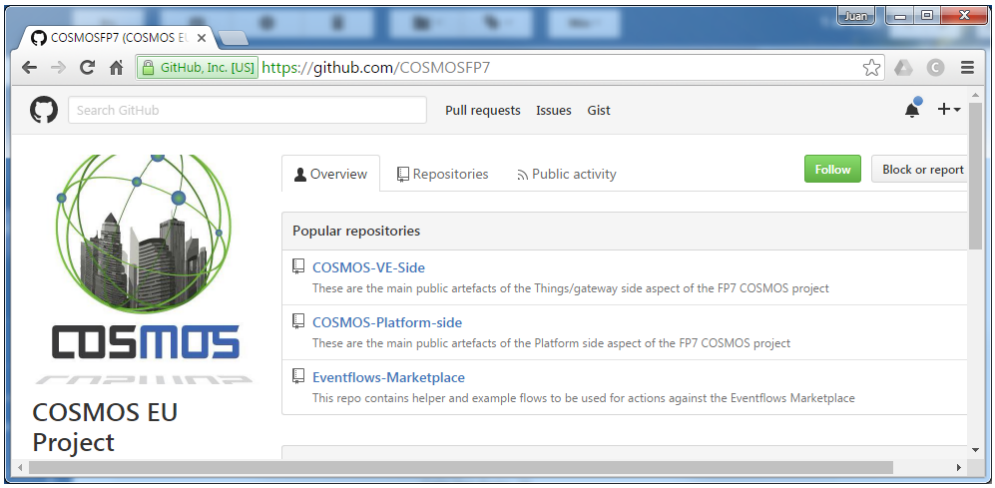


Figure 131: COSMOS GitHub repositories

Furthermore, details about specific outcomes have also been uploaded in open-platforms.eu for higher reachability.

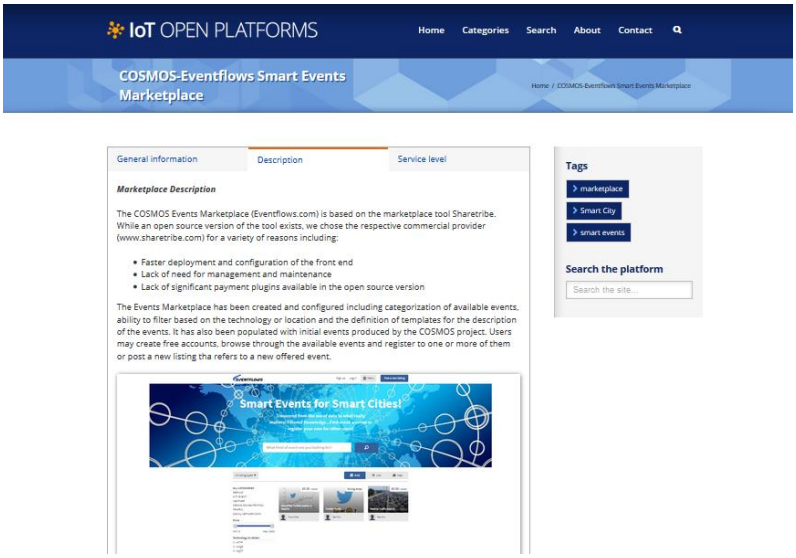


Figure 132: COSMOS artefacts on Open Platforms

Finally, specific repositories per case have been used where applicable, e.g. the Node-RED official flows repository (Figure 133) and the DockerHub repositories (Figure 134), where the µCEP has been provided to facilitate application developers the process to deploy a running instance of the engine as well as its management using PM2.

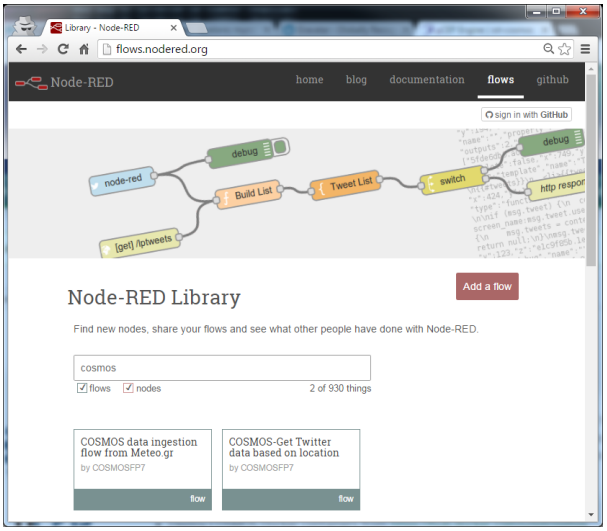


Figure 133: IoT COSMOS Node-RED flow repository

Figure 134: IoT COSMOS DockerHub repository

4.2.9. Camden Scenario Application

4.2.9.1 Scenario Description Update

One of the new features of Y3 is the incorporation of a set of **overrides** in the normal planning activities identified in D7.6.3 in order to address either **user-centric preferences**, as these were identified through the relevant surveys in the Camden scenario, or specific technical challenges that arise from a runtime operation of the system under realistic conditions and may relate to (un)expected circumstances such as a sensor malfunction (**failsafes**). In these cases, normal Planner operation in the context of a specific application such as the smart heating schedule needs to be integrated with the failsafe logic in order to reach a final decision regarding the actuation to be performed (typically through a relevant IoT service). This integration happens at the Service Orchestration layer in which various inputs may be simultaneously considered, such as the proposed actuation from the Planner reasoning, the specific user preferences set through an App GUI or runtime analysis achieved through monitoring and identification of failures. Thus, the final decision is based on all these features and the application design and rationale, as this is set forth in the Service Orchestration Logic. Typical examples of such cases may include:

- Sensor values going beyond a reasonable or feasible range. This may be a clear indication that the sensor has a malfunction, in which case the Planner may not be able to reason accurately on what needs to be done. Examples of these may include e.g. temperature sensor beyond a reasonable or even feasible range. Therefore, its decisions need to not be automatically applied.
- Failsafes in the normal Planner operation. If for some reason the schedule determined by the Planner results in the home temperature going below a risky threshold for the life of the resident, this plan may be overridden in order to ensure that no such case will be realized.
- User needs indicating that e.g. the user has gone on vacation, in which case the previous point should not be considered, resulting in multi-level overrides based on the specific app logic. Another example in this case would be the normal process including the deactivation of the heating in case of an opened window (to save energy and reduce costs). However, if the resident has specified exception from this rule (e.g. due to a temporary or permanent health issue that dictates the concurrent operation of heating with ventilation), this preference should be applied to override the normal cost efficiency operation.

Based on the input provided from the residents, a GUI has been created to give them the opportunity interact with their heating system, enabling the completion of the Smart Heating Application Scenario. This GUI is further presented in subsection 4.2.9.2.1.

Furthermore, another case of scenario inclusion is the incorporation of user consent in the operation of the app and based on the provided input. A technical viewpoint of these activities has already been provided in Section 4.2.1, in this section the instantiation of various specific components for the Camden scenario is presented.

4.2.9.2 Subsystem from D7.6.3 with integration points

The primary subsystem used is the one mentioned in Section 3.2.7.2 which relates to the Social autonomic apps archetype. However, in Y3 it also includes the performed extension regarding the handling of exceptions, user preferences and failsafes as indicated in D2.3.3's Section 9.3.6.3 [11], which appears in the following figure (Figure 135) together with the respective integration points. The process involves two primary integration points:

- IP1 which relates to the operation of the Smart heating application through a user interface (e.g. mobile app front end), that is responsible for receiving user preferences and adapting the Planner operation accordingly.
- IP2 which relates to the Service Orchestration FC that needs to be adapted in order to implement the overrides at the app logic layer.

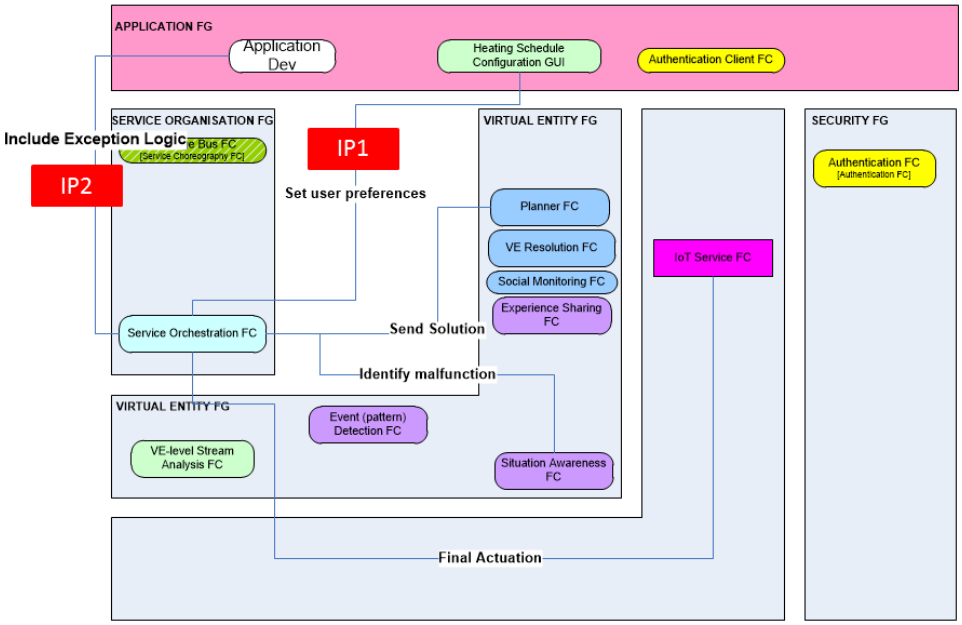


Figure 135: Camden Scenario Failsafes incorporation in Planner operation

The specific implementation logic for IP2 appears in the following figure. Inside the service orchestration layer (Node-RED), a new section is added on real time detection of the failsafe logic. This is responsible for listening to the emitted data and checking them against the implemented malfunction logic according to the scenario description. If and when such a case appears, a relevant context variable is set to true and an email may be sent to the e.g. maintenance team of the houses in order to notify them about this abnormal behavior.

Furthermore, in the case of the Planner, each time it receives a request it may calculate the normal plan, but it is up to the Exception logic to determine if this plan will be forwarded to the actuation phase, based on the checking of the User preferences and Failsafes variables. If these are false the plan proceeds accordingly. If they are true, respective corrective actions are considered thus bypassing the plan actuation. This Exception logic covers points 4 and 11 presented in subsection 4.2.7.2.

Another interesting case refers to the third section in the flow (Data SLA monitoring), which is closely related to our work in the context of the ISO 19086-2 draft standard series on Cloud SLA metrics (more information on this may be obtained in COSMOS Deliverable D8.2.3 [12]). As indicated in that document, and in an effort to demonstrate the process of monitoring IoT SLAs that focus e.g. on data delivery guarantees, a suitable logic may be added that monitors the flow and detects when the data delivery falls short of the guarantee made by the data source provider. Upon detection, in this case a relevant notification may also be sent to e.g. the customer or the technical maintenance team.

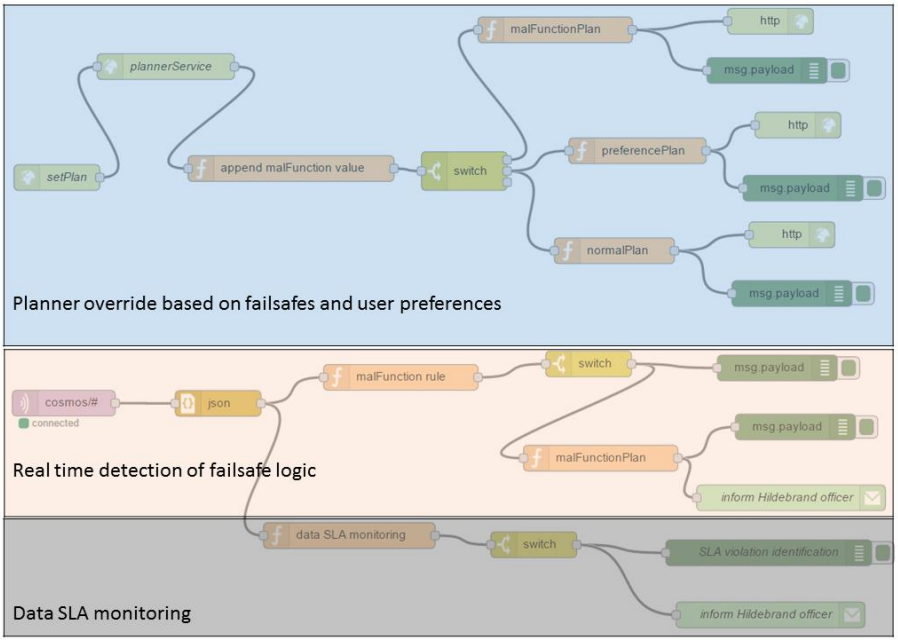


Figure 136: Node-RED flow incorporating Planner bypasses based on Exceptions

4.2.9.2.1 Home Owner Smart Heating Application GUI

After taking under consideration the feedback provided by the Camden residents we implemented the following GUI for the Smart Heating Application:

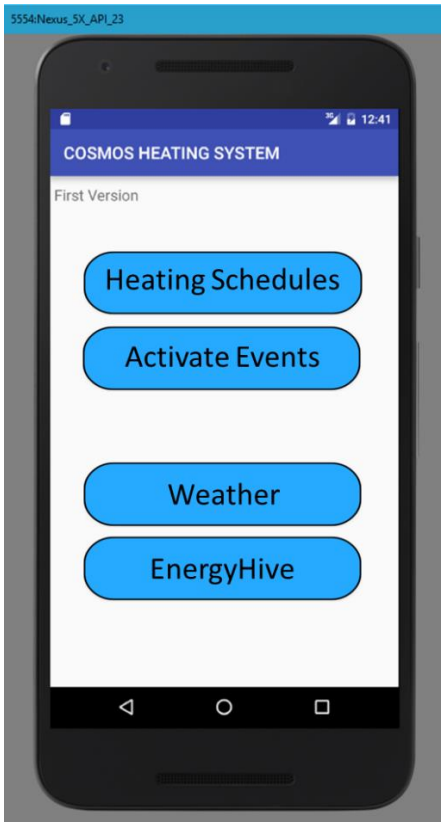


Figure 137: Main menu of the Smart Heating Application GUI.

When the end user presses the “Heating Schedules” button, the user is forwarded to a new page where a list of the current programmed heating schedules is presented in the format presented in the first picture of the following figure. Under this list, a button for the creation of a new heating schedule is available. After pressing this button, the options presented in the second picture of the following figure appear.

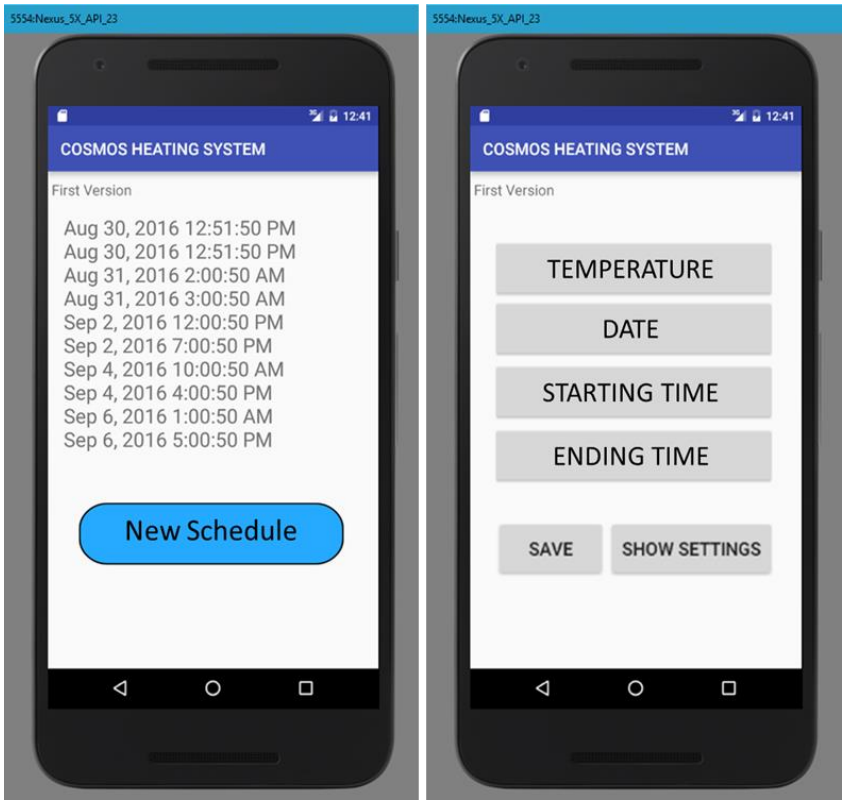


Figure 138: List of heating schedules and new schedule set-up.

In order to create a new schedule, first of all, the end user has to state which the desired temperature is by pressing the “TEMPERATURE” button. It should be noted that the definition of the desired temperature may be optional, since the application may have a default desired temperature extracted from the Case Base on which the Planner reasons to create new schedules. The screen that appears when the end user inserts the desired temperature is shown in Figure 139. The specific design was selected having in mind the creation of a simple to understand and use GUI, even for people that are not well accustomed with technology.

By pressing the “DATE”, “STARTING TIME” and “ENDING TIME” buttons, the end user can state at which day the new schedule should be set and which should be its duration. In Figure 140 it is shown that the design that was selected was the calendar and analog clock view, since these features are widespread and their ease of use has been proven in similar scenarios (e.g. when setting alarm clocks).

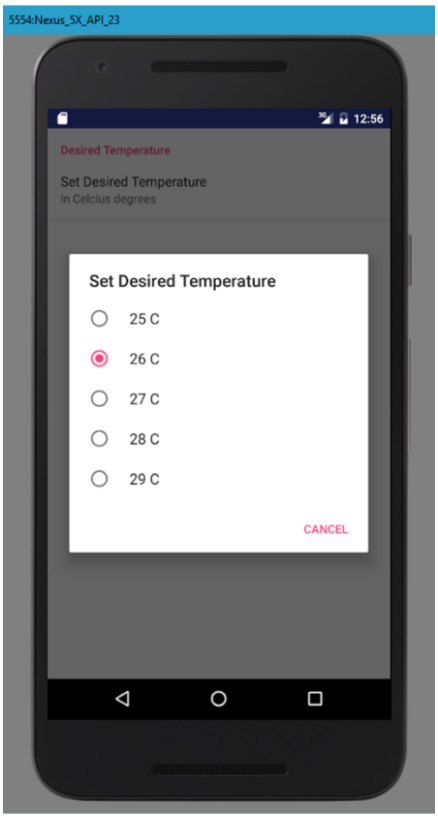


Figure 139: Setting the desired temperature for a new heating schedule.

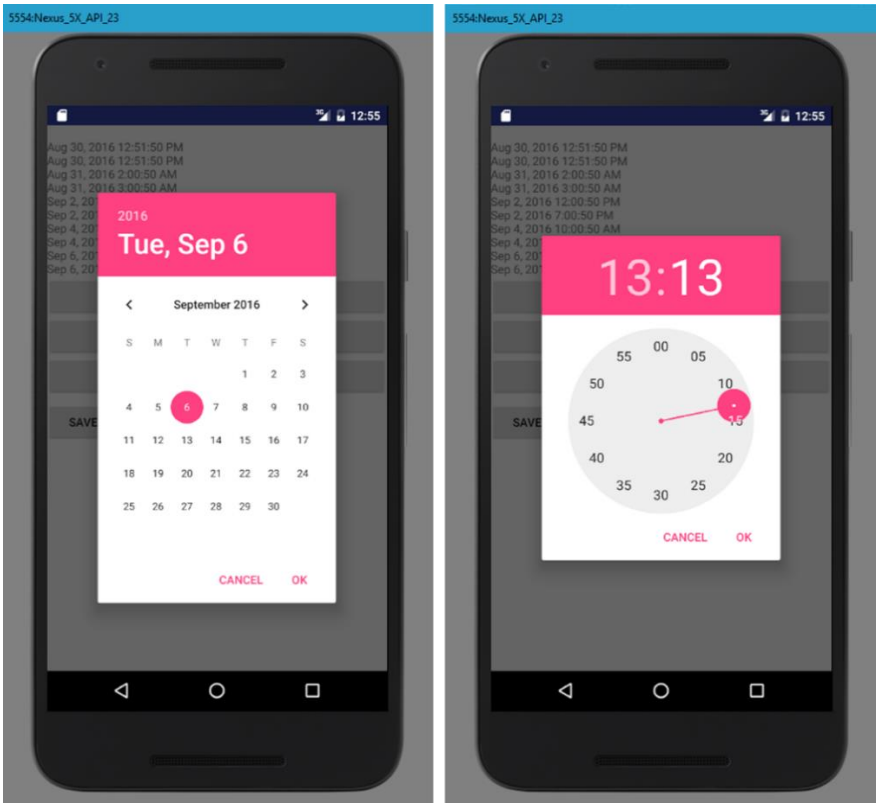


Figure 140: Setting the date and starting-ending time of a new heating schedule.

Coming back to the main menu of the GUI of the application, when the “Activate Events” button is pressed, the end user comes across a check list of various preferences/events that the application logic should take under consideration, in case the heating schedule originally created by the Planner has to be overridden. The list of such preferences was presented in subsection 4.2.7.2.

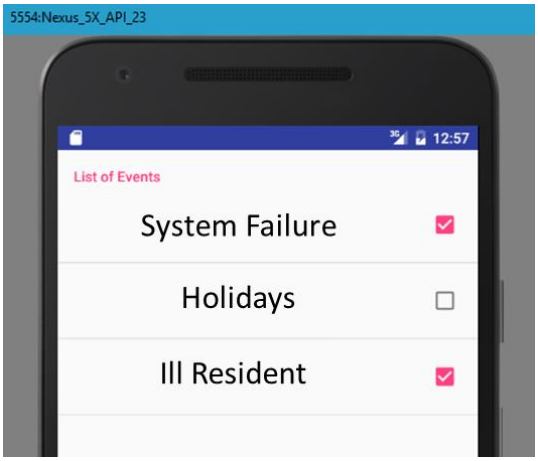


Figure 141: Events declaration based on the residents circumstances.

Finally, taking under consideration points 4 and 8 presented in subsection 4.2.7.2, two more buttons are present at the main menu of the application. The “Weather” button forwards the end user to a weather website where information regarding weather forecasts is being provided. The “EnergyHive” button forwards the user to the Camden EnergyHive website which displays the energy usage of the resident.

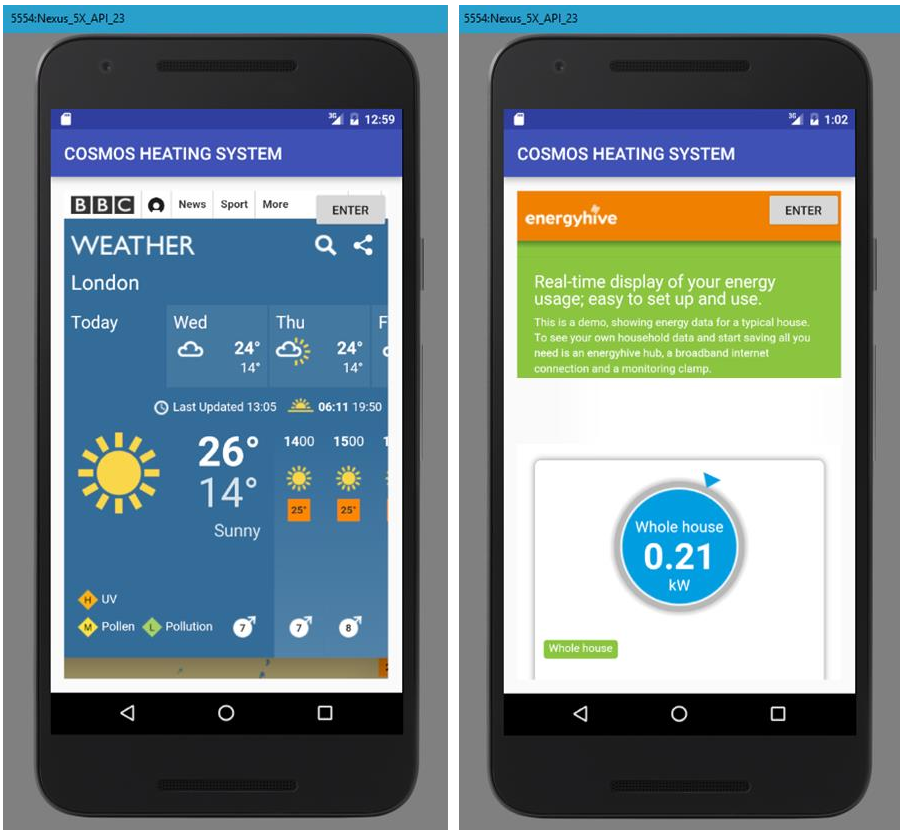


Figure 142: Link to weather website and EnergyHive.

4.2.9.2.2 Home Owner Consent GUI Instantiation

Presents the consent template and purpose to the end-user and obtains the consent contract, e.g. a consent screen that will be provided on the tablet to the end-users by the housing security officer (see https://cosmos.energyhive.com/ and the figure below).

Estate: Oxenholme
Flat: 1

Name	Crowd-based Energy Recommendation	
Description	Your personal energy data will be shared to provide you crowd-based recommendations	Allow / Deny
Data Name	window sensor	Allow / Deny
Data Name	relative humidity sensor	Allow / Deny
Data Name	door sensor	Allow / Deny
Data Name	heatmeter	Allow / Deny
Data Name	sensors	Allow / Deny
Data Name	estate	Allow / Deny
Data Name	resident phone number	Allow / Deny
Data Name	resident e-mail	Allow / Deny
Data Name	resident first name	Allow / Deny
Data Name	resident last name	Allow / Deny

Figure 143: Home Owner Consent GUI excerpt.

The Home Owner Consent GUI Connects to the Consent Manager Service on Bluemix using the following RESTful API (for further details and examples see: http://consentmanagement.eu-gb.mybluemix.net/APIs/)

1) Get consent templates:

```
GET http://consentmanagement.eu-gb.mybluemix.net/cm/1.0/service
```

2) Register user to consent manager:

```
POST http://consentmanagement.eu-gb.mybluemix.net/cm/1.0/user
```

3) Save user consent preferences:

```
POST http://consentmanagement.eu-gb.mybluemix.net/cm/1.0/user_service
```

4) Save user consent preferences for each data item in the consent service:

```
POST http://consentmanagement.eu-gb.mybluemix.net/cm/1.0/user_data
```

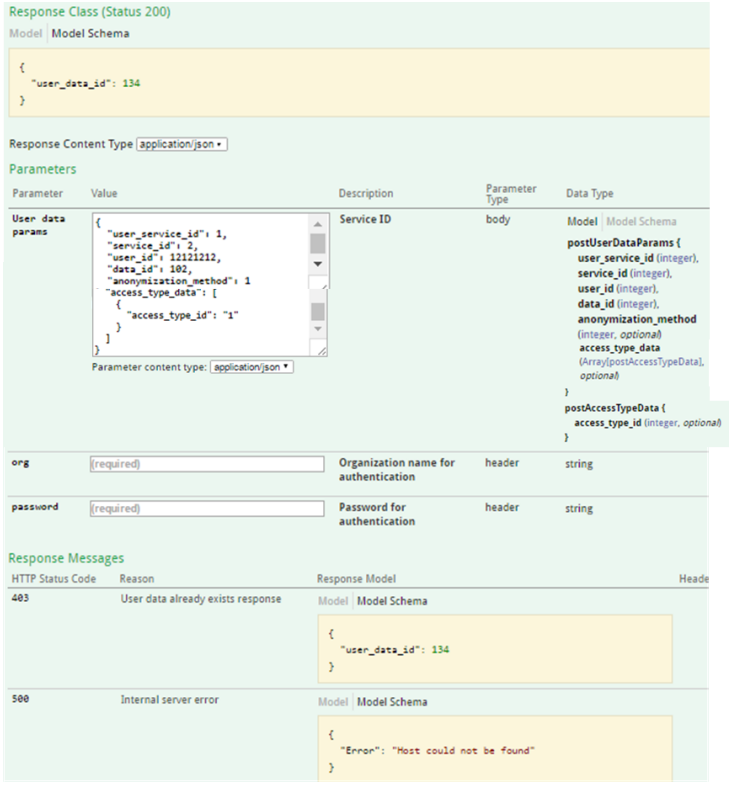


Figure 144: Save User Consent Preferences for each data item in the consent service.

More information and details on this can be found in D4.2.3 [17].

4.2.9.3 Message formats and configuration

4.2.9.3.1 Home Owner Consent Case

For the home owner consent GUI and data format, data is collected from various IoT devices in the COSMOS smart-heating use-case scenario, such as window activity and humidity sensors, boiler/mixed fuel output sensor, heating valve actuations, stored in a container in Object Store and is given in the following schema (camdenTables):

```
|-- estate: string
|-- hid: string
|-- ts: long
|-- heatmeter: struct
|    |-- instant: integer
|    |-- flowRate: integer
|    |-- flowTemp: float
|    |-- returnTemp: float
|    |-- ts: long
|    |-- cumulative: integer
|-- sensors: array (e.g. temperature, humidity, door, window)
|    |-- element: struct
|    |    |-- type: string
|    |    |-- state: string
|    |    |-- ts: long
|    |    |-- sid: long
|-- dt: string
```

Here are examples for a few lines of data (in JSON format):


```
{"estate":"Oxenholme","hid":"c7aB4rxpoJDI","ts":1464315525,"heatmeter":{"instant":89,"flowRate":7,"flowTemp":65.8,"returnTemp":54.6,"ts":1464315525,"cumulative":53605},"dt":"2016-05-27"}

{"estate":"Oxenholme","hid":"cD60ZifP5U2U","ts":1464315525,"heatmeter":{"instant":71,"flowRate":7,"flowTemp":66.2,"returnTemp":57.6,"ts":1464315525,"cumulative":43205},"dt":"2016-05-27"}

{"estate":"Oxenholme","hid":"cYCNMjzyx5mI","ts":1464315525,"heatmeter":{"instant":97,"flowRate":5,"flowTemp":63.5,"returnTem.rep":46.5,"ts":1464315525,"cumulative":43469},"dt":"2016-05-27"}

{"estate":"Oxenholme","hid":"cxI5PuusVeds","ts":1464315525,"heatmeter":{"instant":103,"flowRate":9,"flowTemp":65.5,"returnTemp":55.5,"ts":1464315525,"cumulative":16630},"dt":"2016-05-27"}
```


We have fabricated user data (userTable) according to the following schema, and stored it in Object Store:

|-- hid: string
|-- firstName: string
|-- lastName: string
|-- phone: string
|-- email: string
|-- address: string

Here are examples for a few lines of data (in JSON format):


```
{"hid":"c0CboM9tMBPk","firstName":"Sally","lastName":"Shaw","phone":"555-222-1212","email":"sallyshaw@gmail.com","address":"27 Craven Park London NW10 UK"}

{"hid":"c0spwvCppWAM","firstName":"Shannon","lastName":"Fields","phone":"5551214213","email":"shannonfields@gmail.com","address":"32 Lawn Rd London NW3 UK"}

{"hid":"c15DusfPAGPY","firstName":"Fannie","lastName":"Bass","phone":"1-479-257-26","email":"fannie.bas@gmail.com","address":"620-624 A4006 London NW9 9HF UK"}

{"hid":"c1ifCp7EnkEA","firstName":"Vilma","lastName":"Reed","phone":"080.736.6536","email":"vilma.ree@gmail.com","address":"1 Oak Hill Park London NW3 7LB UK"}
```


4.2.9.3.2 Home Owner Smart Heating Application

The input provided through the GUI by the residents is being forwarded to the back-end (Planner and corresponding Node-RED flows) in JSON format. An example of a new schedule forwarded to the Planner is given:


```
{"Temperature":"26","StartingTime":"2016-08-30T13:25:43","EndingTime":"2016-08-30T18:50:54", "EventA":"true", "EventB":"false", "EventC":"true"}
```


The main message formats and configuration regarding the Planner have already been presented in subsection 3.2.7.3.

4.2.9.4 Sequence & Deployment Diagram

Given in subsections 3.2.7.4 and 3.2.7.6 respectively.

4.2.9.5 Subsystem Test case table

Table 26: Failsafes Test Case

Test Case Number Version	Failsafe_01
Test Case Title	Override Planner based on failsafes and user preferences
Module tested	Planner
Requirements addressed	UNI.253, Required failsafes
Initial conditions	Planner is up and running Node-RED instance and relevant flow
Expected results	When a failsafe is detected or the user has a special preference, then the normal plan is not executed, but instead respective corrective actions are considered thus bypassing the normal plan actuation.
Owner/Role	Application developer
Steps	Copy provided Flow in Node-RED instance Trigger flow execution Fill in the Planner app front-end
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

4.2.9.6 Heating Schedule Efficiency example

In the case of the Smart Heating schedule, what is important is also that the schedule created for the user to reduce energy costs while maintaining a normal operation (e.g. desired temperature levels) based on user preferences. In the case of the planning data from Camden, this desired temperature was acquired from the dataset by measuring the average temperature of each flat in the investigated period. The reason for not taking directly the temperature from when the users deactivate their heating is the feedback from Camden officers that typically residents leave the heating on well beyond their comfort levels and simply drop the temperature by opening e.g. the window. On the other hand, minimum acceptable temperature was extracted from the temperature of the flat when the heating was turned on. This was also the start temperature of the examined schedules.

In the validation data we used real data from 3 housing units of Camden, for a two-week period lasting from 13/5/2016 to 2/6/2016. The specific set was based on data availability and had the requirement to include active heating. 16 schedules were derived from the data (as schedule we indicate any time interval in which the heating was constantly on and if this interval is larger than one hour). As a starting point we consider the temperature at each flat at the respective interval start. The planner then has the goal to maintain the average desired temperature for the duration of the interval. The evaluation was performed in the end, measuring how much time the Planner dictated that the heating should be on and which is a % fragment of the real case (as mentioned previously, in the real case the heating was on for the entire interval). For each flat we also calculate the rhythm of heat loss, as derived from the data, which is used to calculate when the planner should be activated again (we consider as a boundary the limit of 3 degrees less than the average desired temperature). An indicative schedule for 1.13 hours appears in the following table:

- **1.13h (4068s)**:
  - Start schedule-
  - Status: Heating ON (True)
  - Rythm is: -2.8552227288790836E-4
  - Time Left: 2206.0 (CHECKPOINT to deactivate)
  - Status: Heating OFF (false)
  - Time Left: -8301.061216824068 (Time for Temp to drop to 3 degrees below desired so that it needs to reopen- reopening happens at Td-3 degrees) Negative time implies that the schedule will have finished by the time the heating needs to be turned on again
  - Status: heating On
  - Total Scheduled On Time: 1862/4068 (% of interval heating was ON, operation without planner is 100% ON based on the real data)

The average percentage gained in the various schedules was approximately around 60%, especially for small duration schedules, while reducing for the larger cases to around 35%.

Another conclusion from this experimentation phase is that in any given house of the experiment, using the knowledge set of a different user can lead to better efficiency than a native knowledge base, with an upper barrier of the knowledge set's improvement percentage. It stands to reason that the more diverse sets, have better chances of achieving the desired result described here.

In this specific example, the efficiency of housing units 2 and 3 are improved by using the knowledge set of housing unit 1, along with their own preferences in temperature. Though in this specific case, housing unit 1 still has a larger native schedule efficiency.

This leads us to notice that in any given, privacy and consent management component application, with restrictive preferences for sharing and receiving knowledge, the user may still benefit from the Planner CBR approach, given time to diversify his knowledge base. Of course it is also evident, that the process may be greatly accelerated, by simply partaking in the concept of Experience Sharing.

4.2.10. Taipei Scenario Application

In year 2, we used the data from III office in Taipei, which was set up for COSMOS usage and demonstrated the anomaly detection scenario. In year 3, III developed APIs to provide access to data from more than just one flat. We ingested this data in COSMOS storage and extended the scenario from year 2 to work with several appliances installed in various flats. An example of current threshold values for two appliances from one flat is shown below in Figure 145. The scope is to extend Y2's work (included in Section 3.2.8) on abnormal detection status for the flat cases in Y3.

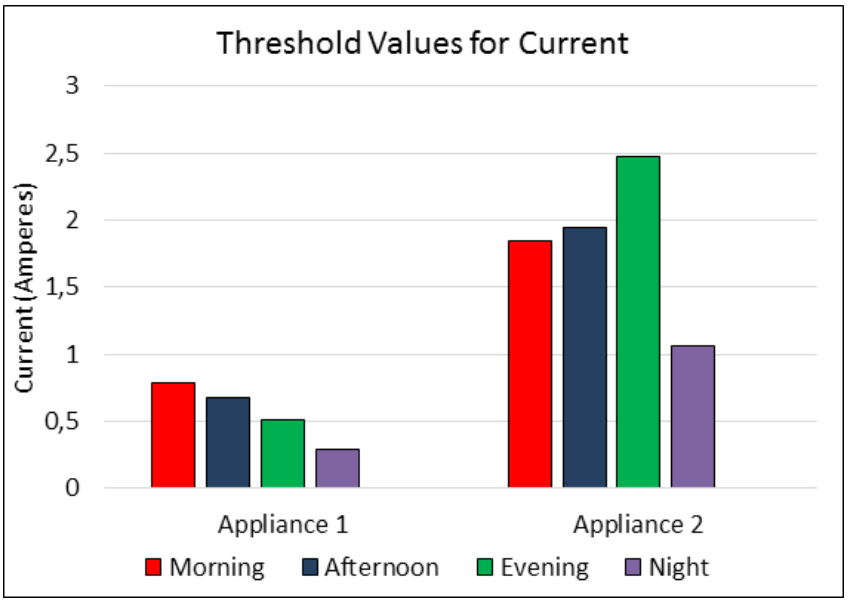


Figure 145: Current threshold values of two appliances

4.2.11. Madrid Scenario Integration

The Madrid Scenario Application was previously introduced in section 3.2.6, so in this new update we are going to focus on the evolution and growth that has been performed in order to cope with the feedback received from different actors such as end users and experienced developers, and also the comments expressed by the members of the CE. During Y3, we have aligned and finished the integration of the different technical components, and also provided the means to visualize the outputs of the traffic analysis in a friendly GUI (Figure 146). Thanks to this, users of the RB WebPanel are now able to visualize in an easier manner the different analytics provided by COSMOS. Additionally, a dedicated WebGUI is made available to a selected group of Traffic Operators in order to capture the accuracy of said analytics by means of feedback widgets (Figure 147).

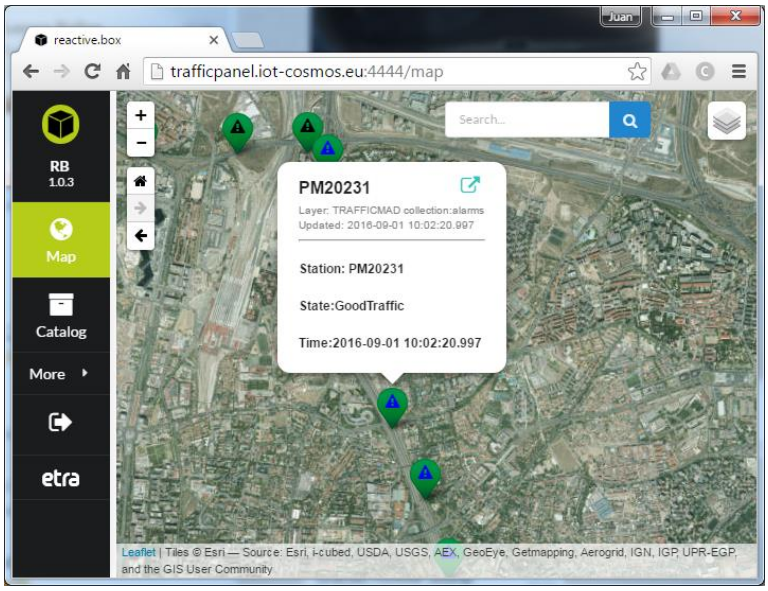


Figure 146: ReactiveBox Traffic Panel

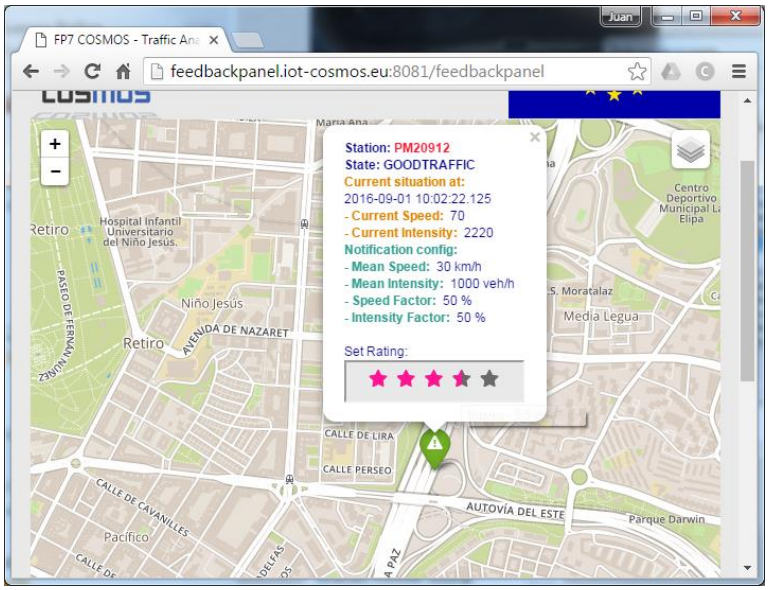


Figure 147: Traffic Feedback WebPanel

The completion of the scenario from previous releases till the work done during the last year of the project has been done in several manners. First, by adding additional analytics to the incoming Data Feed of traffic data we provide more Data Output layers that are ingested into the ReactiveBox system. Furthermore, COSMOS functionalities with regard to this scenario provide a way to receive interactions from selected end users, whose inputs are used particularly to fine-tune the percentage for raising alarms –as requested by Madrid Traffic Operators–. Besides that, in an effort to cope with the whole value chain, we complete the scenario in Y3 with a mechanism to collect the feedback provided by Traffic Operators, which refer to the preciseness of the calculations based on their gained experience.

4.2.11.1 Scenario Description Update

During Y3, extensive discussions were performed with the Madrid City council in order to get their view on Y2 functionalities with relation to traffic identification and potential requirements that could be incorporated. Based on this feedback, the main concern of the Council is with relation to specific prediction models for the ring road M-30 of Madrid (Figure 148), which is considered key for the overall traffic state of the city.

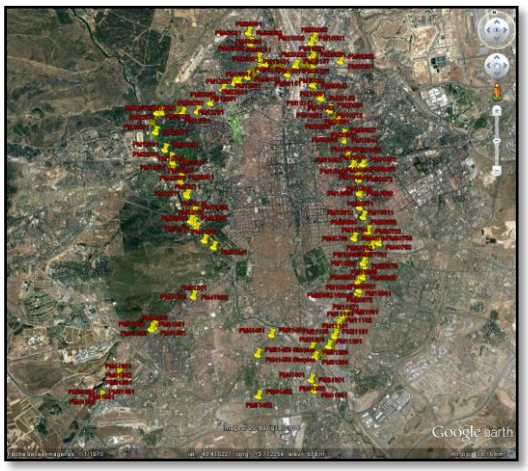


Figure 148: Traffic radar sensors deployed in the M-30 ring

Furthermore, the incorporation of a new set of external services/data, such as weather conditions or social network data from Twitter was considered. The usage of external data services is very useful from an application creation point of view, in order to enhance application scope but also to extend in combinatorial reasoning and/or data analytics aspects. Except for the internal data used in the context of COSMOS from the available partner testbeds (that have been already described in Section 3.2.5), external services have been used this year for the purposes of the scenarios. In this section we indicate the details behind the incorporation of these sources of data and their adaptation to be used in the context of COSMOS.

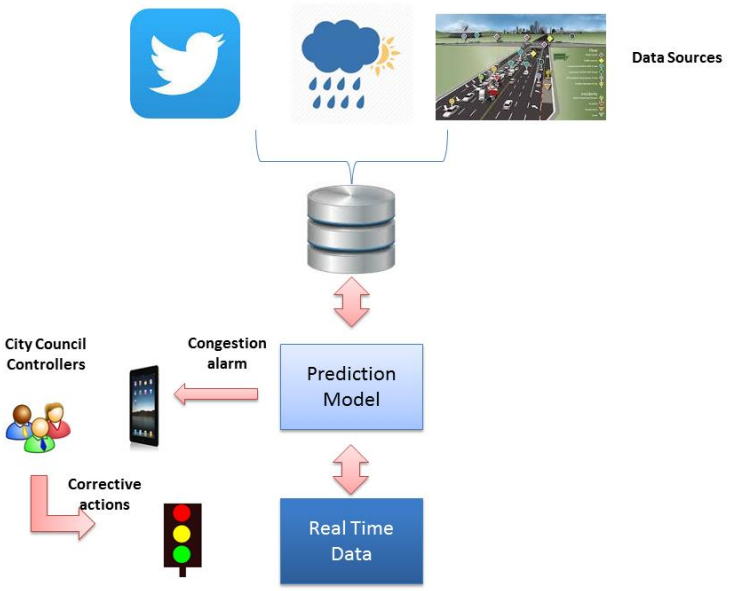


Figure 149: Mobility scenario - High Level view

In order to describe the new additions performed during this year, let's briefly summarize the different parts that compose the scenario, and focus in the ones that have suffered improvements. To put some context, let's reuse the Information Lifecycle Management Diagram and map over it the different aspects of our solution. In first place, if we pay attention to the two flows of Figure 150, what we can see is that information from the real world is processed in real-time by the µCEP (orange flow), but also stored in an efficient manner (blue flow). In the case of the Madrid Scenario, historical data that are being stored refer to Traffic, Weather and Twitter messages.

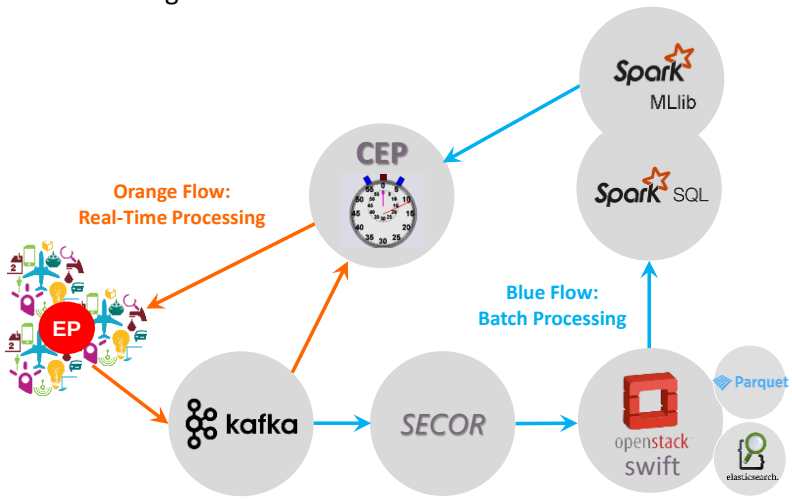


Figure 150: Information Lifecycle Management Diagrams

In year 2, we proposed and implemented a lambda architecture for large scale IoT applications and demonstrated it on traffic data of Madrid. It provides an efficient way to store large IoT historical data and carry complex analytic tasks using Apache Spark and at the same time provides the functionality to process real-time data streams accurately and detect complex events such as traffic state in real-time. In year 3, we made the same architecture as our baseline and ingested other data sources into our system in order to provide a truly global context view for IoT applications. In this regard, weather and social media data was ingested into COSMOS, as explained later on.

4.2.11.1.1 Twitter Data feed description

As mentioned in D7.6.3 [7], the main steps of integrating an external flow can be summarized as follows in the case of Twitter:

- Obtaining of the data in a source adaptive way (e.g. registration to Twitter to get the feeds for a specific subset of the tweets that are from Madrid bounding box), based on the needs of the query and the use case for the data
- Wrapping them to be used from within Node-RED (consumption, transformation and forwarding).
- Definition of an AVRO schema necessary for describing how the data fields information will be stored in the Cloud storage and which fields will be maintained, which also affects the Node-RED manipulation
- Processing of the tweets in order to extract only the useful information per case (e.g. based on the AVRO schema), while adapting the format and JSON structure of the output to the MB
- Potential post-processing in order to add extra information such as sentiment analysis or otherwise e.g. through Apache Spark to generate rules for runtime identification of a respective event.

Following discussions with the partners expected to use the data, the fields that appear in Table 27 were decided. Timestamp is necessary for the aggregation of values in intervals and identification of the normal tweet numbers in these intervals. The text of the tweet is not necessary in this stage but it was considered interesting to maintain it, especially since sentiment analysis was expected to be applied on it. Geographical coordinates were also necessary to be maintained in order to be used as grouping factors for the tweets while the user id (coming from Twitter) was maintained in order to ensure that potential peaks in activity could not originate from a single user (or for general post-processing). Finally sentiment analysis was added in order to aid in potential extensions of the post processing scenarios, such as the identification of happy and sad areas, that could be by themselves a separate event category (potentially useful for e.g. marketing promotions).

Table 27: Twitter fields description

Name	Type	Required
Ts (timestamp from Twitter)	Long	Yes
Text (Actual tweet content)	String	Yes
Longitude (as obtained from Twitter geolocation)	Double	Yes
Latitude (as obtained from Twitter geolocation)	Double	Yes
Twitter id of user (necessary for potential post-processing)	String	Yes
Sentiment Analysis score	Double	No

The main use of Twitter data is to act as another source of information that may aid in the prediction of the Madrid Traffic state (as described in Section 4.2.11). For this reason, filtering of the required messages from Twitter needs to be performed. This is done upon registration, in which we define the geographical bounding box from where we need the relevant tweets to be forwarded. This is taken under consideration by the respective Twitter API and only the tweets that have enabled geolocation and fall within this box are forwarded. It is necessary to stress that not all of the tweets are forwarded, but only a percentage of them, according to the Twitter API documentation.

The testing scenario includes the registration to Twitter to receive the necessary data, the relevant data manipulation dictated by the previous schema (e.g. removal of unnecessary fields), the enrichment of the tweet with sentiment analysis (using both the built-in Node-RED node but also a service offered by the FP7 LeanBigData project) and the forwarding of the messages towards the COSMOS Message Bus. From there it is picked up by the Cloud Storage component, annotated and ingested into the Openstack Swift objects.

4.2.11.2 Subsystem from D7.6.3 and D2.3.3 with integration points

The subsystem used is the same as in Section 3.2.5.1.2, with the only difference that it does not come from an IoT Service or device, but it is adapted to the COSMOS platform through the use of the Service Orchestration component (implemented via Node-RED in our case). In Y3 and based on the new architectural system case 9.3.4.8 of D2.3.3, an updated subsystem has been identified that appears in Figure 151. There are two main integration points between the App developer and the COSMOS system (IP2), and refer to the following cases:

- Definition and agreement on the AVRO schema needed for describing the data feed
- Adaptation of the data gathering process from the external source to the process used in COSMOS and the alignment with the aforementioned template.

A further testing case refers to the actual ingestion of the data in the COSMOS Cloud storage as a final mean of verification that the data have been gathered.

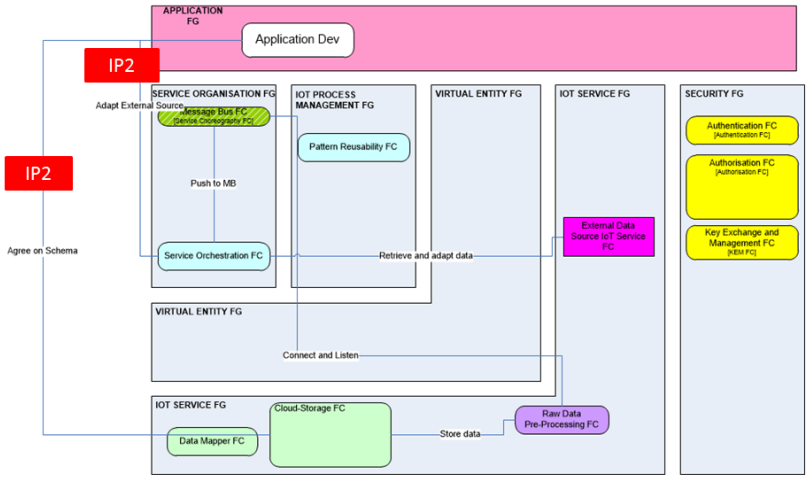


Figure 151: External Data ingestion Subsystem with integration points

The specific process is instantiated in the following sections for two cases of external data, the Madrid City Weather and Twitter information.

4.2.11.3 Message formats and configuration

4.2.11.3.1 Reactive Box messages

The complex events generated by the µCEP are embedded in a VEProt datagram –the format expected by the ReactiveBox– and sent to the EMT system. An example of one these datagrams is showed below for the TRAFFICMAD.alarms collection.


```
{
    "_id": "PM20952",
    "subsystem": "PUTDATA",
    "function": "REPLACE",
    "layer": {
        "owner": "EMT.SERVICIOS.TRAFFICMAD",
        "type": "public",
        "name": "TRAFFICMAD.alarms"
    },
    "levelAlarm": "I",
    "instant": "2016-08-18 16:40:49.867",
    "rating": "-1",
    "geometry": {
        "type": "Point",
        "coordinates": [-3.666663, 40.401878]
    },
    "textStatus": "GoodTraffic",
    "trafficState": {
        "codeStation": "PM20952",
        "speed": "75",
        "intensity": "2400"
    },
    "system": "LAYERS",
    "serviceLevel": "0",
    "conditionAlarm": {
        "factorSpeed": "0.500000",
        "factorIntensity": "0.500000",
        "meanSpeed": "30",
        "meanIntensity": "1000"
    },
    "codeAlarm": "40",
    "shape": {
        "type": "marker",
        "options": {
            "shape": "circle",
            "markerColor": "green",
            "prefix": "fa",
            "icon": "fa-exclamation-triangle"
        }
    },
    "state": {
        "color": "white",
        "instant": "2016-08-18 16:40:49.867",
        "description": "<b><p>Station:
PM20952</p><p>State:GoodTraffic</p><p>Time:2016-08-18
16:40:49.867</p></b>",
       "value": "1",
       "format": "text"
    },
    "alertReceived": "0"
}
```


The content of this datagram is very interesting as it highlights the benefits of the ReactiveBox system when it comes to the representation of the visual marker in the RB Map. Actually, the attribute *"shape"* allows us to define dynamically the type of icon and color of the marker, being possible to display different information with different color codes, as good/bad traffic.

Additionally, as soon as a Traffic Operator provides feedback on a marker, it gets updated so that an additional attribute is added to the previous datagram in the TRAFFICMAD.rating collection (same as above but including this new attribute).

```
"feedback": {
   "rating": 4,
   "ts": "1470399756",
   "tf": "1:22:35",
   "opid": "OperatorId"
}
```

4.2.11.3.2 Weather feed

Regarding the weather information, both historical and live weather feeds have been extracted using the Wunderground.com API service. There are some limitations such as maximum calls per min and day, but they fall within our operational scopes. Other parties have also been tested so to be used as backup services, such as OpenWeatherMap.org API service.

An example of the query to get historical data is provided hereafter:

```
https://www.wunderground.com/history/airport/LEMD/2015/01/02/DailyHistory.html
?reqdb.zip=&reqdb.magic=&reqdb.wmo=&format=1
```

And the corresponding output:

```
TimeCET,TemperatureC,DewPointC,Humidity,SeaPresureLevelhPa,VisibilityKm,WindDirection
,WindSpeedKm/h,BurstSpeedKm/h,RainFallmm,Events,Conditions,WindDirDegrees,DateUTC
12:00 AM,-2,-4,78,1041,18,Norte,5.6,,,,,350,2015-01-01 23:00:00
12:30 AM,-2.0,-5.0,80,1037,-9999.0,NO,5.6,-,N/A,,Despejado,320,2015-01-01 23:30:00
1:00 AM,-2,-4,75,1042,18,Norte,7.4,,,,,350,2015-01-02 00:00:00
1:30 AM,-2.0,-5.0,80,1037,-9999.0,NNO,5.6,-,N/A,,Despejado,330,2015-01-02 00:30:00
[…]
10:00 PM,2,-2,69,1042,30,Norte,7.4,,,,,350,2015-01-02 21:00:00
10:30 PM,2.0,-2.0,75,1039,-9999.0,Norte,9.3,-,N/A,,Despejado,350,2015-01-02 21:30:00
11:00 PM,1,-2,72,1042,20,Norte,5.6,,,,Despejado,350,2015-01-02 22:00:00
11:30 PM,0.0,-2.0,87,1039,-9999.0,Norte,5.6,-,N/A,,Despejado,350,2015-01-02 22:30:00
```

In order to extract weather information from 2008 till 2016, a specific scrapper script has been built in order to request all the required information, which was then transformed conveniently following an Apache AVRO schema. After that, the information was sent to the Cloud Storage.

```
{"namespace": "cosmos",
  "type": "record",
  "name": "weatherData",
  "fields": [
    {"name": "TimeCET", "type": ["null", "string"]},
    {"name": "TemperatureC", "type": ["null", "string"]},
    {"name": "DewPointC", "type": ["null", "string"]},
    {"name": "Humidity", "type": ["null", "string"]},
    {"name": "SeaLevelPressurehPa", "type": ["null", "string"]},
    {"name": "VisibilityKm", "type": ["null", "string"]},
    {"name": "WindDirection", "type": ["null", "string"]},
    {"name": "WindSpeedKmh", "type": ["null", "string"]},
    {"name": "GustSpeedKmh", "type": ["null", "string"]},
    {"name": "Precipitationmm", "type": ["null", "string"]},
    {"name": "Events", "type": ["null", "string"]},
    {"name": "Conditions", "type": ["null", "string"]},
    {"name": "WindDirDegrees", "type": ["null", "string"]},
    {"name": "DateUTC", "type": ["null", "string"]},
    {"name": "ts", "type": ["null", "long" ]},
    {"name": "dt",  "type": ["null", "string"]},
    {"name": "tf", "type": ["null", "string"]}
  ]}
```

With respect to the live weather feed, periodical calls are made using the following Node-RED flow, which receives a json payload with the data and parses it to fit with the aforementioned AVRO schema.

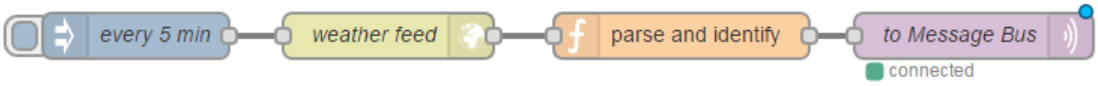


Figure 152: Live weather feed Node-RED flow

4.2.11.3.3 Twitter feed

The AVRO schema that was created in accordance with the fields defined in Section 4.2.11.1.1 appears below.

```
{"namespace": "cosmos",
  "type": "record",
  "name": "TwitterData",
  "fields": [
    {"name": "ts", "type": "long"},
    {"name": "text", "type": "string"},
    {"name": "lon", "type": "double"},
    {"name": "lat", "type": "double"},
    {"name": "twitter_id", "type": "string"},
    {"name": "sentiment", "type": ["null", "double"]},
  ]
}
```

One of the key points of configuration that needs to be addressed is the server time from which the request to register to Twitter data is performed. Due to potential desynchronization, the server time needs to be updated periodically, e.g. through the command ***`ntpdate -u 2.pool.ntp.org`***. Otherwise the timestamp of the request may not match the timestamp of Twitter and this leads to authentication error due to the Oath protocol used. Developers also need to follow instructions on getting OATH credentials from Twitter in [10].

4.2.11.4 Combining Events

As described earlier, we incorporated external data feeds in year 3 and combined it with existing traffic data provided by use case partner. In this regard, Twitter and weather data were ingested in COSMOS storage in order to exploit fully enriched IoT data analysis. Following individual data streams were analyzed to detect complex events.

- **Traffic Data.** Our approach consists in gathering information from the traffic sensors deployed in the city, and use the knowledge derived from their historical and real-time analysis. In year 2, we proposed a lambda architecture for large-scale IoT applications and demonstrated it with the help of traffic data in order to detect traffic states in real-time as shown in Figure 153 where Analytics block covers our lambda architecture.

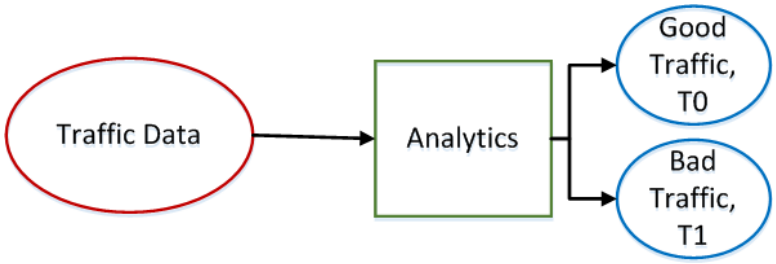


Figure 153: Traffic data analytics

- **Weather Data.** The purpose of using weather information is twofold: on one hand, this additional data source has been modeled and used to derive a better understanding of traffic conditions in the city; on the other hand, it takes advantage of the µCEP capacity to correlate different data sources, thus being an extensible example for other potential use cases, as well as enabling the creation of more complex rules that demonstrate the capacity of the engine to derive meaningful context from different input events. Figure 154 below shows three different categories of weather after doing analytics in COSMOS.

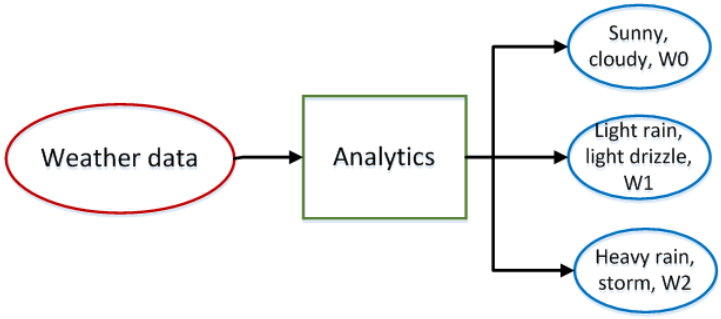


Figure 154: Weather data analytics

- **Twitter messages.** Social media is one of the fastest ways to extract real time information about the events at city level. In order to provide a true global view for Madrid scenario, we also exploited Twitter data feed coming from Madrid City and applied the same data analytics architecture as described in Figure 150. In our approach, we aggregate the total number of tweets coming from a specific region in Madrid for every 30 minutes. It is a common observation that in case of an event, people like to tweet about. For example, if there is a concert or football match in a nearby location, the total number of tweets from that region increases. We exploit this fact and find the mean number of tweets for a region from historical data and if we detect a spike in aggregated number of tweets, we generate an event.Figure 155 shows the summary of twitter data analytics where analytics block shows COSMOS analytics part and E0, E1 and E2 represent the events based on the level of crowd sourcing based on tweets count.

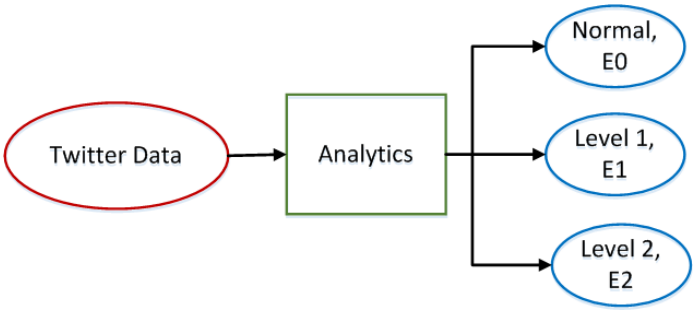


Figure 155: Twitter data analytics

Once we have all the detected events from individual data feeds, they are combined using µCEP running at the platform level. In this regard, we extended state of methods and explored a probabilistic event processing approach as explained in D6.1.3 [13]. In state of the art CEP systems, complex event detect is either true or false but real world is probabilistic in nature. A more efficient way is to detect complex event in terms of probabilities. For example, a rule for

combining all these events to detect congestion can be that if (traffic state is bad) AND (event is detected from Twitter) AND (heavy rain) → generate complex event Congestion. Now in conventional CEP systems if either of the condition becomes false, complex event will not be detected. But in real world, the probability of Congestion might be reduced but still there will be a non-zero probability of having a congestion. In our system, we model these probabilities using Bayesian Networks which is a probabilistic directed acyclic graphical model and represented by Figure 156 below.

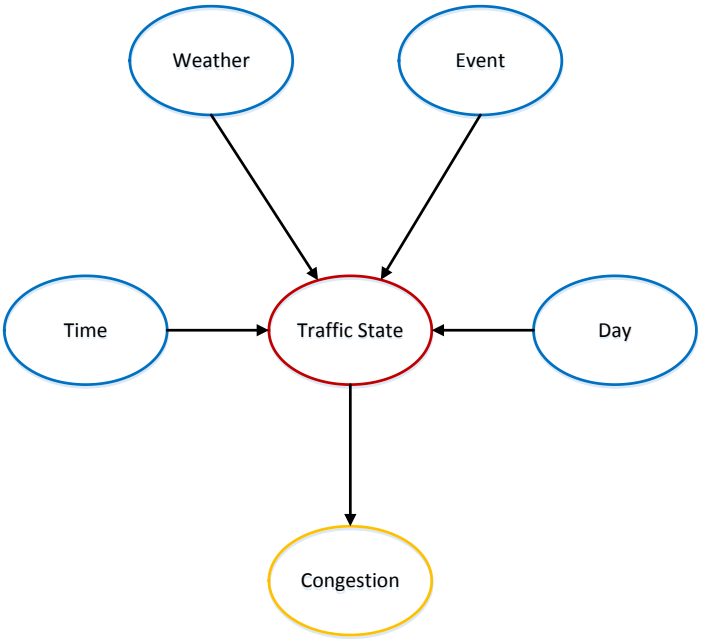


Figure 156: Bayesian network representation for Madrid probabilistic events

Here each node is a random variable which represents the possible events directed from individual data feed (output from Figure 153, Figure 154 Figure 155). In addition, we also take time and day context in our model as morning traffic on weekday will be different as compared to morning traffic on weekend. Traffic state (good or bad) is dependent on different factors such as time, day, weather and events as represented in the figure. Bad traffic can lead to congestion which is an example of anomaly or traffic can become bad. We applied Bayesian chain rule to infer how the impact of individual events propagate to effect probability of congestion as shown below.

```
P(Time, Weather, Event, Day, Traffic, Congestion) =
            P(Time) · P(Weather) · P(Day) ·
            P(Traffic|Time,Weather,Event,Day) ·
            P(Congestion|Traffic)
```

For example, if we want to find the probability of Congestion when the weather is rainy, Event detected from Twitter, Traffic state is bad on a weekday, it will be given by:

$$P(W^1,\ E^1,\ T^1,\ C^1,\ D^0) = P(W^1).P(E^1).\ P(D^0).P(T^1|W^1,E^1,\ D^0).P(C^1|T^1)$$

Where,

- W^1 = rainy weather,
- E^1= level 1 event detected from twitter,
- T^1 = bad traffic and
- D^0 = weekday.

All the probabilities were calculated empirically from historically data. In this regard, SQL queries proved to be helpful. A snippet from the code is shown below to give an idea how SQL queries can be used to find above-mentioned probabilities.

```
#P(T0|W0, E0, D0)
df_traffic_1 = sqlContext.sql("select * FROM total WHERE traffic_state = '0'
AND weather_level = '0' AND twitter_level = '0' AND day =        '0'").count()
a1 = df_traffic_1/count

#P(T0|W0, E0, D1)
df_traffic_2 = sqlContext.sql("select * FROM total WHERE traffic_state = '0'
AND weather_level = '0' AND twitter_level = '0' AND day =        '1'").count()
a2 = df_traffic_2/count

#P(T0|W0, E1, D0)
df_traffic_3 = sqlContext.sql("select * FROM total WHERE traffic_state = '0'
AND weather_level = '0' AND twitter_level = '1' AND day =        '0'").count()
a3 = df_traffic_3/count

#P(T0|W0, E1, D1)
df_traffic_4 = sqlContext.sql("select * FROM total WHERE traffic_state = '0'
AND weather_level = '0' AND twitter_level = '1' AND day =        '1'").count()
a4 = df_traffic_4/count
```

4.2.11.5 Sequence Diagrams

4.2.11.5.1 ReactiveBox sequence

Once a new traffic alarm is issued by the µCEP, the following Node-RED flow parses it and creates the corresponding traffic notification to the sent to ReactiveBox system.

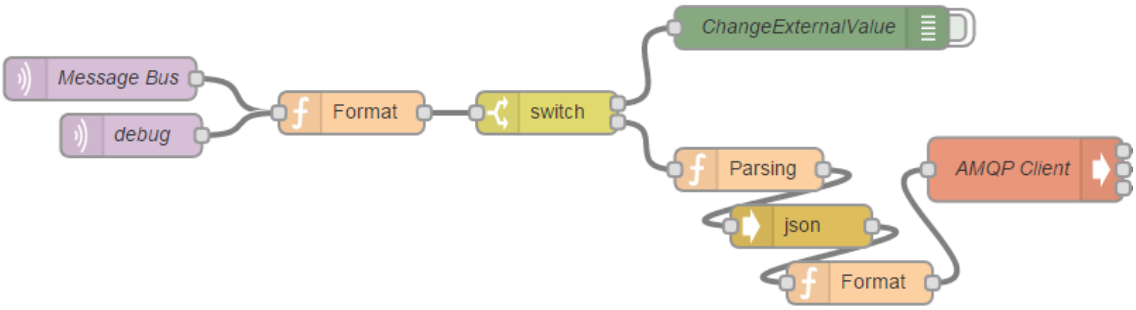


Figure 157: Ingestion of Complex Event to ReactiveBox

Additionally, a complimentary Node-RED flow has been implemented in order to subscribe to specific collections of the ReactiveBox system, being then able to obtain enriched information and store it in the CloudStorage for future reference. For instance, apart of the traffic notifications, the feedback provided by the Traffic Operators with relation to a specific notification is received and stored.

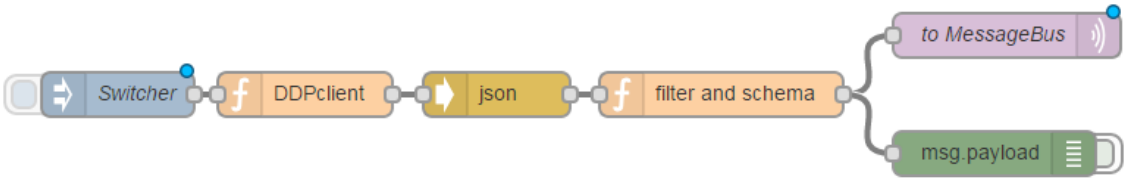


Figure 158: Subscribing to ReactiveBox notifications via DDP

One the particular functionalities that have been developed to manage incoming ReactiveBox messages relies in the fact that, by nature, those messages contain only the updated attributes and not the whole message –that is how DDP/Meteor.js work with MongoDB databases–. As a result, a cache mechanism is used.

```
var obj = msg.payload;
if (obj.msg == "changed" || obj.msg == "added"){
  var cachedObj = context[obj.id] || {};
  cachedObj = {
    "id": obj.id,
    "levelAlarm": gLP(obj.fields.levelAlarm, cachedObj.levelAlarm),
    "instant": gLP(obj.fields.instant, cachedObj.instant),
    "codeAlarm": gLP(obj.fields.codeAlarm, cachedObj.codeAlarm),
    "coordinateX": ((obj.fields.geometry && obj.fields.geometry.coordinates)?
gLP(obj.fields.geometry.coordinates[0], cachedObj.coordinateX): cachedObj.coordinateX) ,
    "coordinateY": ((obj.fields.geometry&& obj.fields.geometry.coordinates)?
gLP(obj.fields.geometry.coordinates[1], cachedObj.coordinateY): cachedObj.coordinateY) ,
    "textStatus": gLP(obj.fields.textStatus, cachedObj.textStatus),
    "speed": (obj.fields.trafficState)? gLP(obj.fields.trafficState.speed,
cachedObj.speed) : cachedObj.speed,
    "intensity": (obj.fields.trafficState)? gLP(obj.fields.trafficState.intensity,
cachedObj.intensity ) : cachedObj.intensity,
    "serviceLevel": gLP(obj.fields.serviceLevel, cachedObj.serviceLevel),
    "factorSpeed": (obj.fields.conditionAlarm)?
gLP(obj.fields.conditionAlarm.factorSpeed, cachedObj.factorSpeed ) :
cachedObj.factorSpeed,
    "factorIntensity": (obj.fields.conditionAlarm)?
gLP(obj.fields.conditionAlarm.factorIntensity, cachedObj.factorIntensity ) :
cachedObj.factorIntensity,
    "meanSpeed": (obj.fields.conditionAlarm)? gLP(obj.fields.conditionAlarm.meanSpeed,
cachedObj.meanSpeed ) : cachedObj.meanSpeed,
    "meanIntensity": (obj.fields.conditionAlarm)?
gLP(obj.fields.conditionAlarm.meanIntensity, cachedObj.meanIntensity ) :
cachedObj.meanIntensity,
    "alertReceived": gLP(obj.fields.alertReceived, cachedObj.alertReceived)
  };
  context[obj.id] = cachedObj;
  msg.payload = cachedObj ;
return msg;
}

# gLP stands for getLatestPropery
gLP = function(optionalField, cachedFiled) {
  return (optionalField)?optionalField : cachedFiled;
}
```

With relation to the feedback provided by the Traffic Operators, the following sequence diagram depicts how the Feedback WebPanel behaves. More info will be given in next section 4.2.11.8.

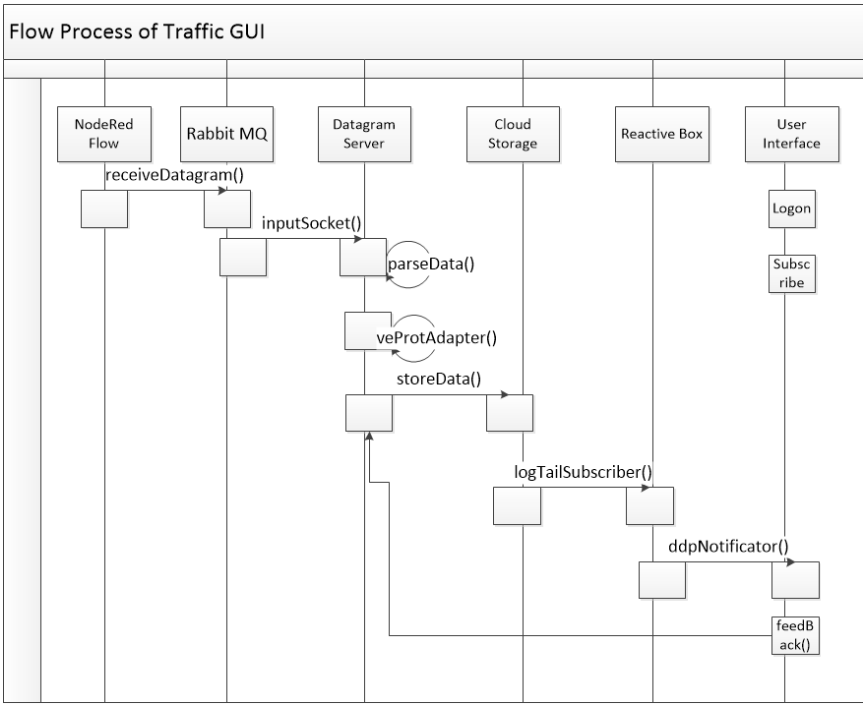


Figure 159: Feedback WebPanel - Sequence Diagram

4.2.11.5.2 Twitter sequence

The activity diagram for the Twitter case is similar to the case of section 3.2.5.1.4, hence it is not repeated. The sequence diagram for this case appears in Figure 161.

The Node-RED flow for the feed is portrayed in Figure 160. In this case the registration to the Twitter API is performed and the respective tweets are forwarded towards the COSMOS MB, but also towards a counting facility. The latter is responsible for maintaining a global bounding box data structure and counter structure, to measure in real time different tweet counts in a bounding box. This information will afterwards be compared to the statistical analysis of historical data. A potential peak in tweets may be indicative of a large crowd concentration event, which may be associated with the user GPS location and forwarded to the CG portal (in a similar manner to which in Y2 bad traffic was propagated to the CGs). This generic process has been already highlighted in Section 3.2.6 of this document and updated in D7.6.3 especially in that document's figure 39, which highlights the generic integration flow.

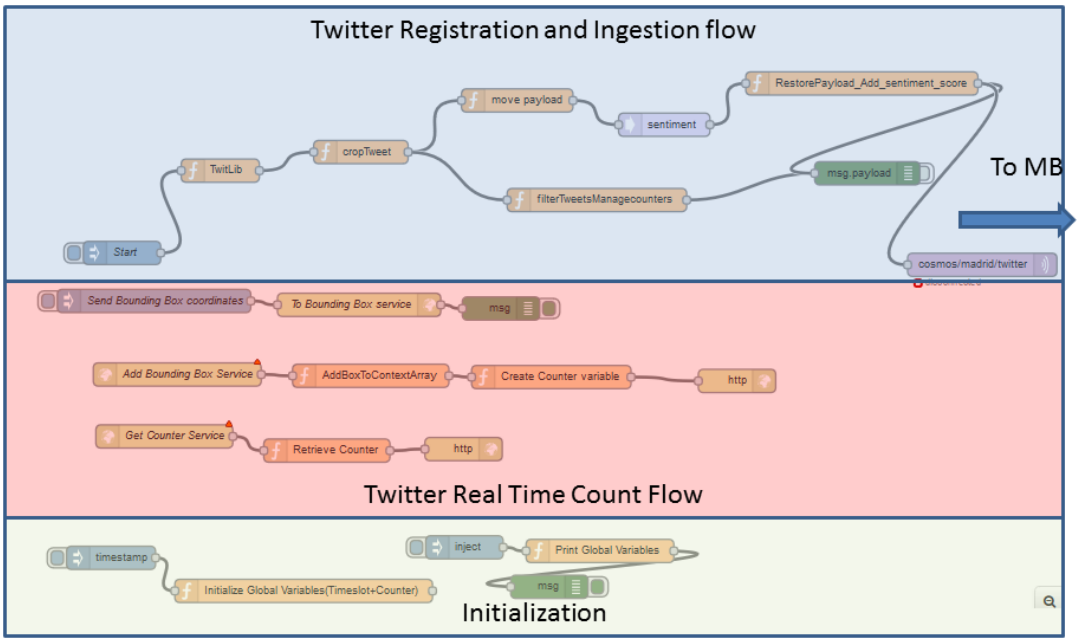


Figure 160: Node-RED flow for Twitter data ingestion and real time count

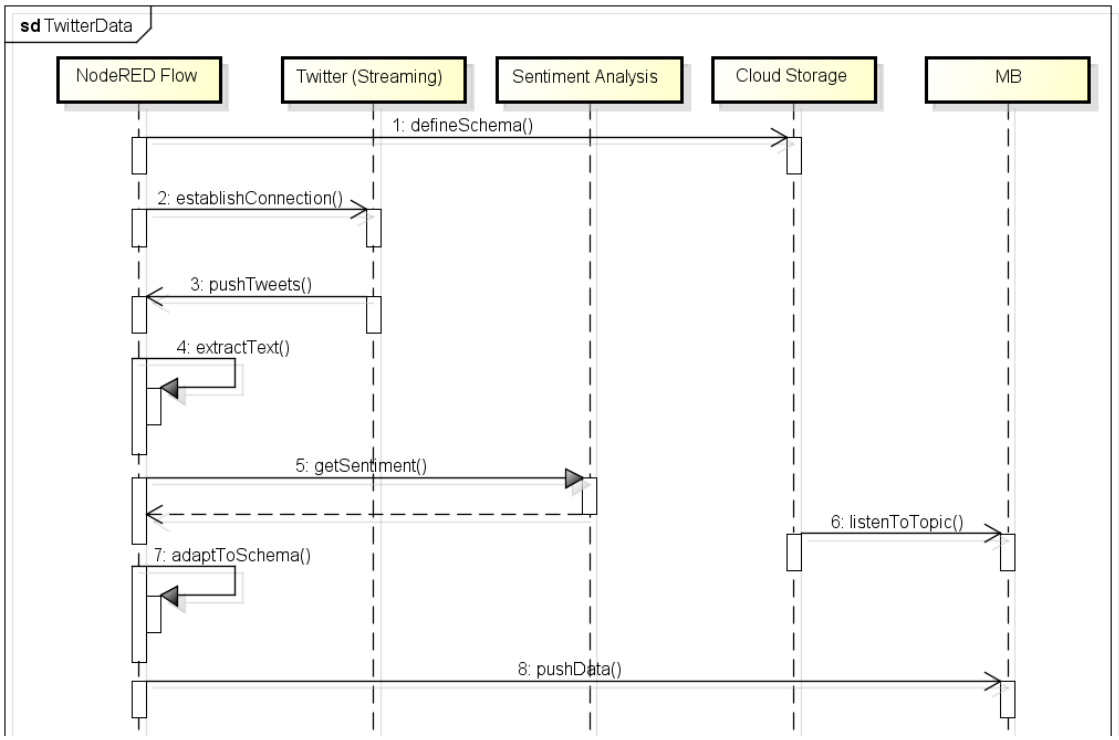


Figure 161: Sequence Diagram for the Twitter data feed.

4.2.11.6 Subsystem Test case table

Table 28: Twitter Get Data Test Case

Test Case Number Version	**Twi_01**
Test Case Title	Twitter Get Data
Module tested	Message Bus, Cloud Storage, Node-RED Flow
Requirements addressed	4.1, 4.2, 4.9, 6.1, 6.5
Initial conditions	A functional Message Bus AVRO Schema shared with Cloud Storage Node-RED instance and relevant flow Credentials on Twitter API
Expected results	Twitter data made available in real-time through Message Bus Twitter data stored for later analysis in Cloud Storage
Owner/Role	Application developer
Steps	Copy provided Flow in Node-RED instance Configure based on provided README file in flow Trigger flow to register to Twitter API based on geolocation bounding box coordinates Retrieve data asynchronously Parse data, include sentiment analysis Publish into Message Bus Swift receives data and stores it
Passed	Yes
Bug ID	None
Problems	Desynchronization of server time periodically causes authentication failures with OATH Built-in Sentiment analysis of Node-RED does not seem to provide highly accurate scores
Required changes	Use cronjob in server with command: *ntpdate -u 2.pool.ntp.org*

4.2.11.7 Deployment Diagram

The deployment diagram for the Twitter case is very similar to the one presented in Section 3.2.5.1.6, hence it is not repeated.

4.2.11.8 Specific tests that may be needed

Taking advantage that the Traffic Analysis Use Case is being tested by Madrid Traffic Operators, a dedicated Feedback WebPanel has been developed allowing COSMOS to get and record direct feedback from the users, for instance to fine-tune the underlying algorithms of the solution.

The feedback panel consists of a reactive Meteor page synchronized with the "TRAFFICMAD.alarms" collection, so that any inserted, modified or deleted alarm will be immediately seen in the panel. For each traffic measurement point, "rating star" widget is referenced, what is used to rate the accuracy and usefulness of the notification.

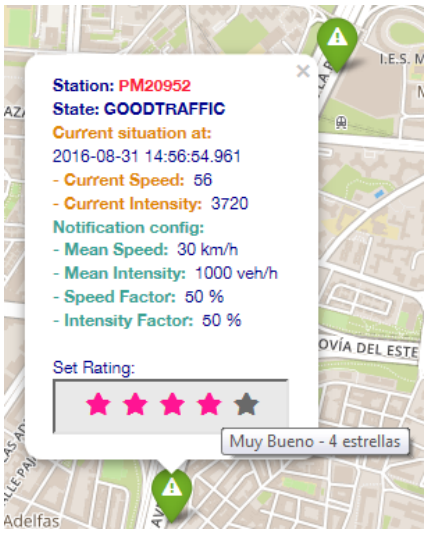


Figure 162: Feedback WebPanel - Rating widget

The synchronization is performed by Meteor (Node.js) platform, which allows a customer to subscribe to one or more MongoDB collections, previously published by the server.

```
Meteor.publish('TRAFFICMAD.alarms', function() {
  return traffic.find();
});

Meteor.subscribe('TRAFFICMAD.alarms');
```

When voting in the “Set Rating” area, an event is trigger to insert a new document in a separate collection called "TRAFFICMAD.rating", using exactly the same fields and values that the ones in the document "TRAFFICMAD.alarms" on which one is voting, but adding an additional "object" called "feedback" with the rating field (vote value), timestamps (ts, tf) and OPID (operator who has carried the vote).

(2) {_id : PM13041}	{ 17 fields }	Document
_id	PM13041	String
subsystem	PUTDATA	String
function	REPLACE	String
layer	{ 3 fields }	Object
levelAlarm	I	String
instant	2016-09-01 08:02:12.814	String
rating	3	String
geometry	{ 2 fields }	Object
textStatus	GoodTraffic	String
trafficState	{ 3 fields }	Object
system	LAYERS	String
serviceLevel	0	String
conditionAlarm	{ 4 fields }	Object
codeAlarm	40	String
shape	{ 2 fields }	Object
state	{ 5 fields }	Object
alertReceived	0	String

feedback	{ 4 fields }	Object
rating	3.5	Double
ts	1470400518	String
tf	12:35:18	String
opid	IvanLedesma	String

With relation to scalability, for the case of Madrid (thus a large city, prone to technology that is anticipated to produce a significant number of tweets) the provided tweets were easily handled within the Node-RED environment used in COSMOS, with no evidence of lost messages or server bottleneck.

4.2.12. Sound Detection Scenario Application

4.2.12.1 Scenario Description

One new scenario examined during Y3 is providing a service to enable end users with hearing impairment that enhances their awareness regarding their surroundings at home. This might be achieved by the transformation of surrounding sounds to a visualization or notification effect that can be understood by this user category. A different effect should be used per different sound, as well as sound identification, in order to enable the users to distinguish between the different types of events (e.g. a doorbell ringing, a fire alarm going off, etc.). To this end, COSMOS technologies may prove useful and enable such a process.

This scenario includes direct end user involvement in the definition of the requirements of this process, which has been described in Section 4.2.7.3. It was studied and implemented from application developers using the COSMOS components during the NTUA Hackathon, with satisfactory results.

Initially sounds that are captured through sensors attached to the Raspberry Pi 2 used in the context of the COSMOS VE case or sample files for testing purposes are being used. The input sounds are transformed to data points (first Integration Point in the figure), which may then be analyzed in terms of their dominant frequencies. This vectorization is then used to define a new CBR Problem that is then fed to the Planner component. CBR then attempts to categorize the incoming sound and match it against stored vectors of initialized sounds. This is the second Integration Point in the figure which includes the definition and population of the case structure for this scenario. In case of match, the actuation Solution of the Case structure is activated, which may imply the visualization of the respective sound via multiple means, such as wifi connected light bulbs, Bluetooth connected smart watches, the COSMOS Platform MB, Twitter tweets or other means (e.g. even through IFTTT recipes since many smart devices have such interfaces). This is the third Integration Point in the figure and can be implemented via available Node-RED nodes.

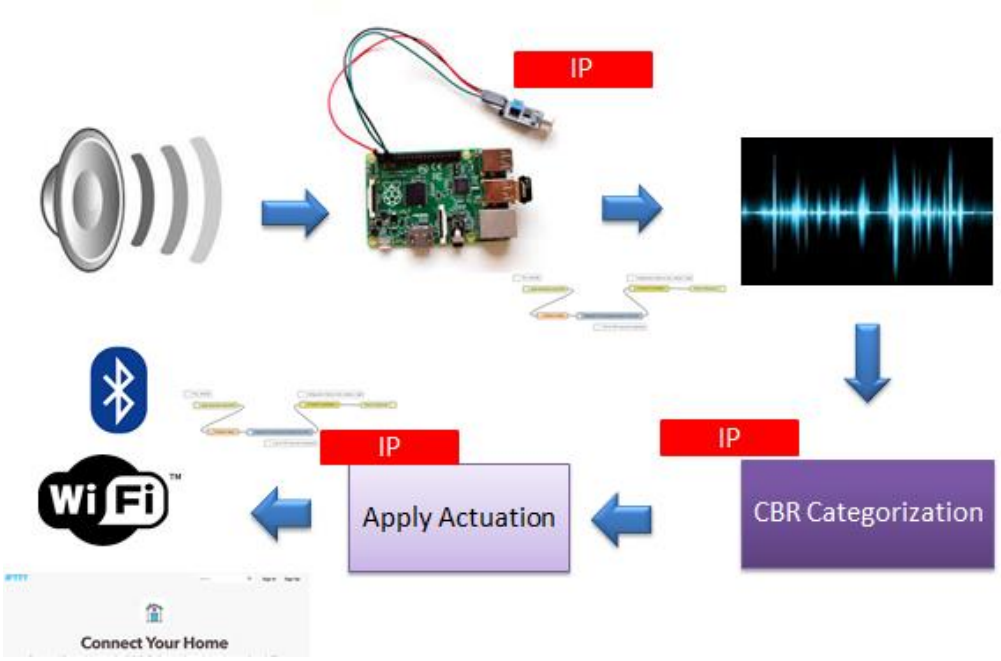


Figure 163: Technical sequence of steps for Smart Home Sound Detection Application.

4.2.12.2 Subsystem from D7.6.3 with integration points

For the sound analysis part (vectorization of the sounds) the Python programming language is being used. Originally, the script normalizes and isolates the sound on one channel and then performs Fast Fourier Transformation (FFT). Thus, having the frequency spectrum of the sound, we adjust it to real frequencies using the sampling frequency of the signal. Finally, we keep the five most dominant frequencies of the spectrum. Next, an initial Case Base consisting of the dominant-harmonic frequencies for each sound category of the scenario extracted from test sounds can be created using the Protégé tool and stored in RDF / XML file format as an ontology. Lastly, the Planner reasons on the knowledge base in order to give adequate results through CBR.

The synthesis of the above as a web service as well as the actuation part of the scenario (Twitter post) was implemented via Node-RED.

The subsystem that corresponds to the Planner, the Case Base and the Experience Sharing component is already described in Section 3.2.7.2, hence it is not repeated here.

4.2.12.3 Message formats and configuration

The application receives the path of .wav files and then forwards the files as input into a python script which returns the five dominant frequencies of the sound. Based on these, a .json file is created which represents the http post request to the Planner.

Thereafter, the Planner categorises the sound after comparing it with its initial Case Base by giving as a result to the defined Problem a .json file which is parsed in order to acquire the solution attributes (Sound Category: telephone audio, doorbell, fire-alarm, knocking-door and crying baby). Finally, using a branch (switch node) the corresponding message can be created and visualised (e.g. as a tweet/post to the end-user like “Attention! A fire-alarm is ringing.”).

4.2.12.4 Sequence Diagram

The Node-RED flow of the above procedure is being presented.

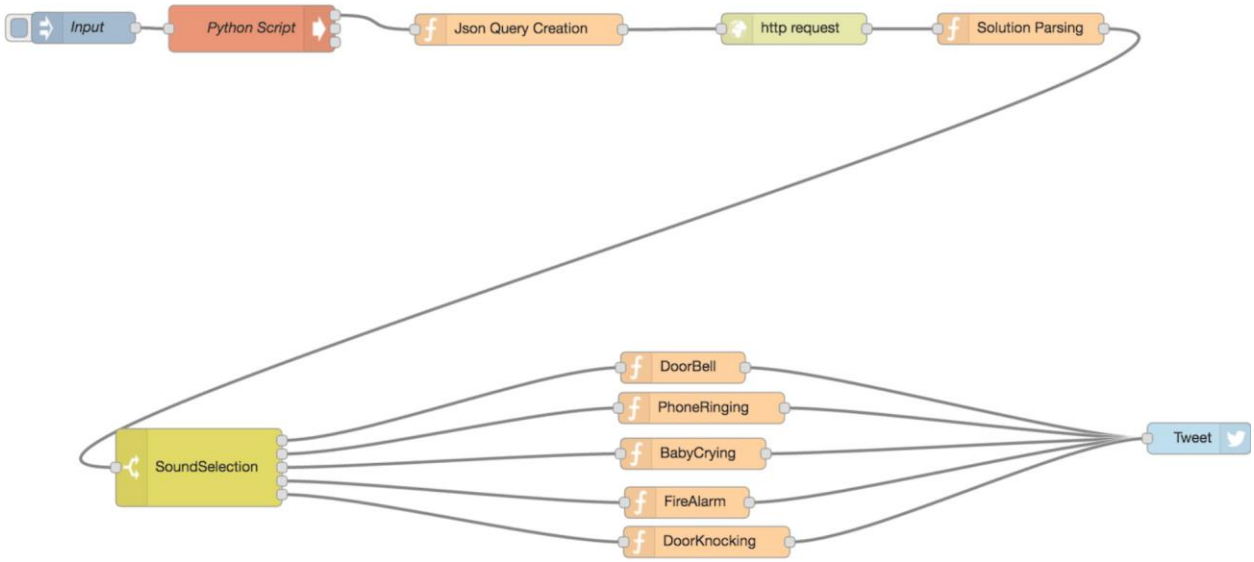


Figure 164: Node-RED flow for Sound Detection through tweets.

4.2.12.5 Subsystem Test case table

Table 29: Sound Detection Application Test Case

Test Case Number Version	**Sou_01**
Test Case Title	Sound Detection Application
Module tested	Planner, Node-RED Flow
Requirements addressed	Requested sounds by the end users are being recognized
Initial conditions	Available test sound files A functional Planner An instantiated Case Base Node-RED instance and relevant flow Credentials on Twitter API
Expected results	Notifying the end users through tweets or other means
Owner/Role	Application developer
Steps	Record or provide .wav files as input Python script returns dominant frequencies Create Problem statement as .json file Planner categorizes and identifies the sound The final tweet is posted
Passed	Yes
Bug ID	None
Problems	Medium response time: can be decreased Medium-Good sound recognition accuracy: can be increased
Required changes	See subsection 4.2.12.6

4.2.12.6 Specific tests that may be needed

In its current version, our application can differentiate audio signals between telephone audio, doorbell, fire-alarm, knocking-door and crying baby sounds and forward the appropriate twitter-post to the end user.

To test our application, two **sound sources** were/can be used:

- .wav files available online (e.g. http://karol.piczak.com/papers/Piczak2015-ESC-Dataset.pdf)
- .wav files created by recording our own sounds (real-world test)

The **application variables** of interest are:

- the response time of the application
- the memory size that the Case Base requires (depends on number of sounds and sound variables)
- the Quality of Information, i.e. percentage of cases where we had proper sound recognition. The corresponding sub-variables are:
  - True Positive (TP) - label is positive and prediction is also positive
  - True Negative (TN) - label is negative and prediction is also negative
  - False Positive (FP) - label is negative but prediction is positive
  - False Negative (FN) - label is positive but prediction is negative

- the required computing power, RAM, etc.

The above application variables can be greatly influenced by the several design/implementation decisions that can be made. The corresponding **design variables** are:

- the number of the dominant frequencies being extracted/stored per sound
- the number of different sounds stored
- the possible changes to the sound analysis script (sounds with complex spectrums such as baby-crying, phone-ringing need more advanced sound processing than others)
- the threshold used by the Planner for the sounds comparison

So far, the tests have proven that this scenario has reached a satisfactory level. Indicatively, using a Case Base of 70 sounds with 5 dominant frequencies for each one of them and setting the Planner threshold close to 90%, about **70%** of 60 input sounds (with both relevant and irrelevant sounds) have been matched against the 5 Sound Categories correctly. Through further experimentation, the results can be improved.

Furthermore, one key outcome of the validation process was the need to check for different similarity measures based on each case. The Bray-Curtis distance used in the Camden Scenario application for Smart heating is not suitable in this case, given that it creates a highly insensitive measure when high frequencies may appear in the compared sounds. Given that this measure sums in the denominator the values of the vector, having high frequencies in the top 5 frequency list (even with low contribution to the energy of the signal) may dominate the numerical value of the measure thus leading to wrong conclusions if the threshold value is not adjusted..

4.2.13. Events On Events Archetype with reusable flows and Implementation through a Marketplace Concept

4.2.13.1 Archetype Scope

As mentioned in D7.6.3 (Sections 6.2.4 and 6.2.10), one of the key aspects of COSMOS and the applied methodology is the ability to combine events in an abstracted manner (without having knowledge of how events are created and published), in order to gradually ascend in awareness levels. This may be achieved by combining events with each other, or combine events with different data inputs and a specific application logic that may lead to the generation of new events. Due to the technologies used in the project (such as the MB, Node-RED, etc.) these combinations may be achieved in a very efficient and agnostic manner, enabling external developers to build upon results and extend them according to their specific use case or vision.

Thus we envision demonstrating scenarios, in which multiple events and data sources may be combined in order to enhance extracted conclusions. So for example, the traffic events detection performed by the COSMOS Smart Events flow may be combined with similar flows coming from social network data in order to identify extremes in crowd concentration. In this case a combination of a good traffic state and a spike in Twitter activity is indicative that in that area a large pedestrian concentration is created.

However, in order to empower the sustainability of the COSMOS outcomes as well as the participation of external developers building upon the initial event pool, motivation must be provided in terms of how easy it is to plug in and exploit the publication of these events or tools as well as the monetization of this effort.

One way to enable this is through the creation of an Events Marketplace, in which participants may act as consumers and producers of events that are propagated through a common messaging structure and with a common agreed format (not in content but in type e.g. JSON). Such a structure may also provide discovery of the available events and the ability to receive them. Once received these events may be enriched, enhanced and then reshared as a higher level event through the marketplace.

The practical steps needed for such a process, that were initially highlighted in D7.6.3 include:

- Creation of the Marketplace front end, in which producers and consumers will be able to register their events, the format of the transmitted event (e.g. JSON schema) and potentially semantics of each field
- Creation of a common pub/sub structure for producers and consumers to log on and push/listen for events

COSMOS technologies may aid participants of such a scheme in the following manner:

- Assist developers in event generation through
  - COSMOS SE flow and specific instances (Twitter, Madrid etc.)
  - COSMOS Node-RED flows and examples

However anyone will be able to register for producing events with any kind of technology, for better generalization. The only requirement is that for each published event the schema is clear (does not need to be the same schema for all events) and that it is in JSON.

4.2.13.2 Subsystem from D7.6.3 with integration points

The Events on Events archetype presented in D7.6.3 Section 6.2.4 appears also in the following figure. Integration points refer to Application (Event) developers for a variety of actions such as

- Creating a new event description and linking it to the Marketplace description template
- Pushing event related information into a common messaging structure
- Browsing through the available events, selecting and registering for one
- Listening to event related information from the common messaging structure

Specific details per action are given in the following sections.

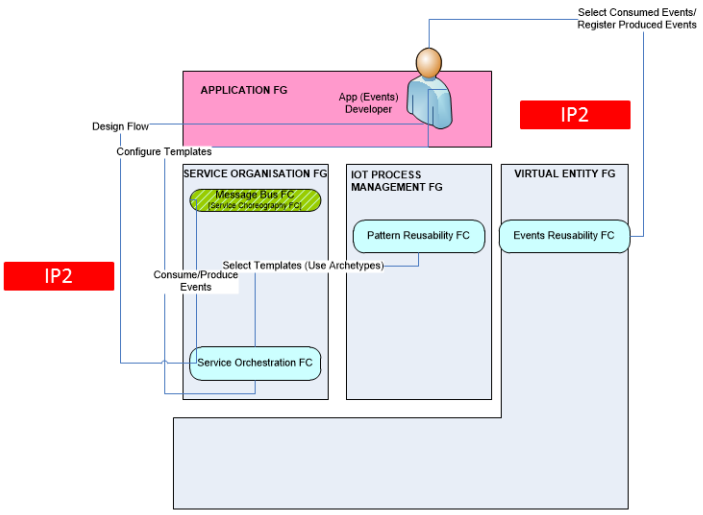


Figure 165: Events on Events Archetype subsystem

4.2.13.3 Marketplace Front-End

The COSMOS Events Marketplace (Eventflows.com) is based on the marketplace tool Sharetribe. While an open source version of the tool exists, we chose the respective commercial provider (www.sharetribe.com) for a variety of reasons including:

- Faster deployment and configuration of the front end
- Lack of need for management and maintenance
- Lack of significant payment plugins available in the open source version

The Events Marketplace (Figure 166) has been created and configured including categorization of available events, ability to filter based on the technology or location and the definition of templates for the description of the events. It has also been populated with initial events produced by the COSMOS project. Users may create free accounts, browse through the available events and register to one or more of them. They may also create their own event (details on this to follow).

Figure 166: Events Marketplace Front End Design

4.2.13.4 Creation of a new event by a producer

In order for a producer to create a new event, the following steps need to be undertaken:

- Create an account on Eventflows.com
- Check out flows from COSMOS that may assist you to get started
  - https://github.com/COSMOSFP7/COSMOS-Platform-side
  - More details on this step are included in Section 4.2.13.4.1
- Imagine a combination of data or data&processing that can help identify an interesting event and create the sequence that generates it. This is probably the most critical step in order for the event to be meaningful and interesting for external users. Especially events that may be used in the context of mobile apps should be considered, since these cases would have a wide exploitation field. For the sequence, any kind of technology may be utilized, however COSMOS tools are mainly based on Java and Node-RED. Especially for the latter it is strongly advised to utilize it since it is very flexible and able to apply data transformations easily and combinations of published information
- Register your event in the marketplace and insert the necessary descriptions
  - E.g. data schema of messages
  - More details on this follow in Section 4.2.13.4.2

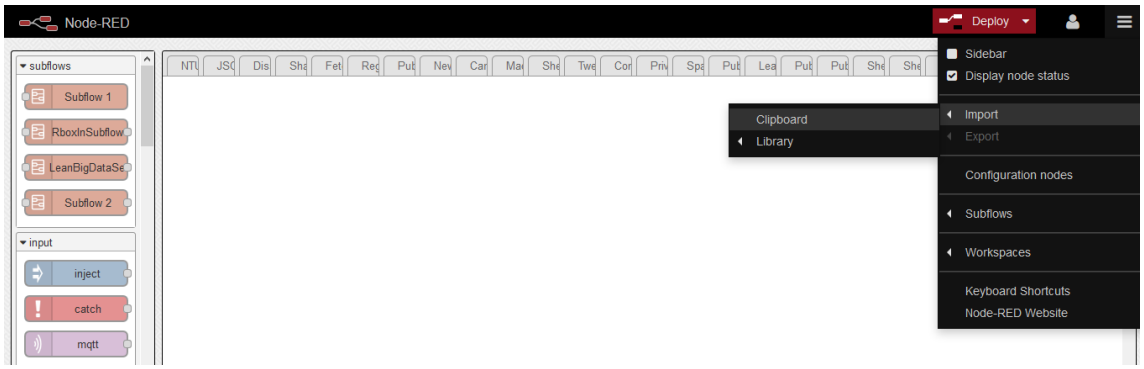
- Credentials for accessing the Messaging system (AMQP protocol) will be sent via email along with the endpoint to which the data should be pushed
- Use and configure provided Node-RED flows or Java clients for pushing data to the provided endpoint
    - https://github.com/COSMOSFP7/Eventflows-Marketplace/tree/master/NodeRED_Clients
    - More details on this follow in Section 4.2.13.4.3

4.2.13.4.1 Usage of existing flows as starting point

As mentioned previously, one of COSMOS aims is to abstract large parts of the process, through the reutilization of common flows. For this reason a number of flows have been made available through our GitHub repository, that can be easily copied and integrated directly in any new attempt for event creation. Details on how to do this are included in the following paragraphs, using as an example the Twitter registration flow provided.

The detailed steps are:

- Data flow for registering to Twitter and getting data, described in detail in Section 4.2.11 The flow is available here:
    - https://github.com/COSMOSFP7/COSMOS-Platform-side/tree/master/DataIngestion/TwitterDataIngestion
    - The flow registers to Twitter API and receives tweets based on geolocation.
        - Can be manipulated afterwards in multiple ways
            - Stored e.g. in a DB , object storage etc. and analyzed statistically to find peaks
            - Passed through sentiment analysis to find happy and sad spots
- In order to use it, copy the json file and paste it in a Node-RED tab (Figure 167)
    - Import-> From Clipboard
    - Paste json file
    - Flow will appear in Tab (can be connected to whatever output e.g. MQTT node etc.)
- Needs configuration. Details are included as comments in the flow
    - Credentials as a Twitter registered developer
    - Configuration of geographical coordinates if one is interested in another area other than Madrid



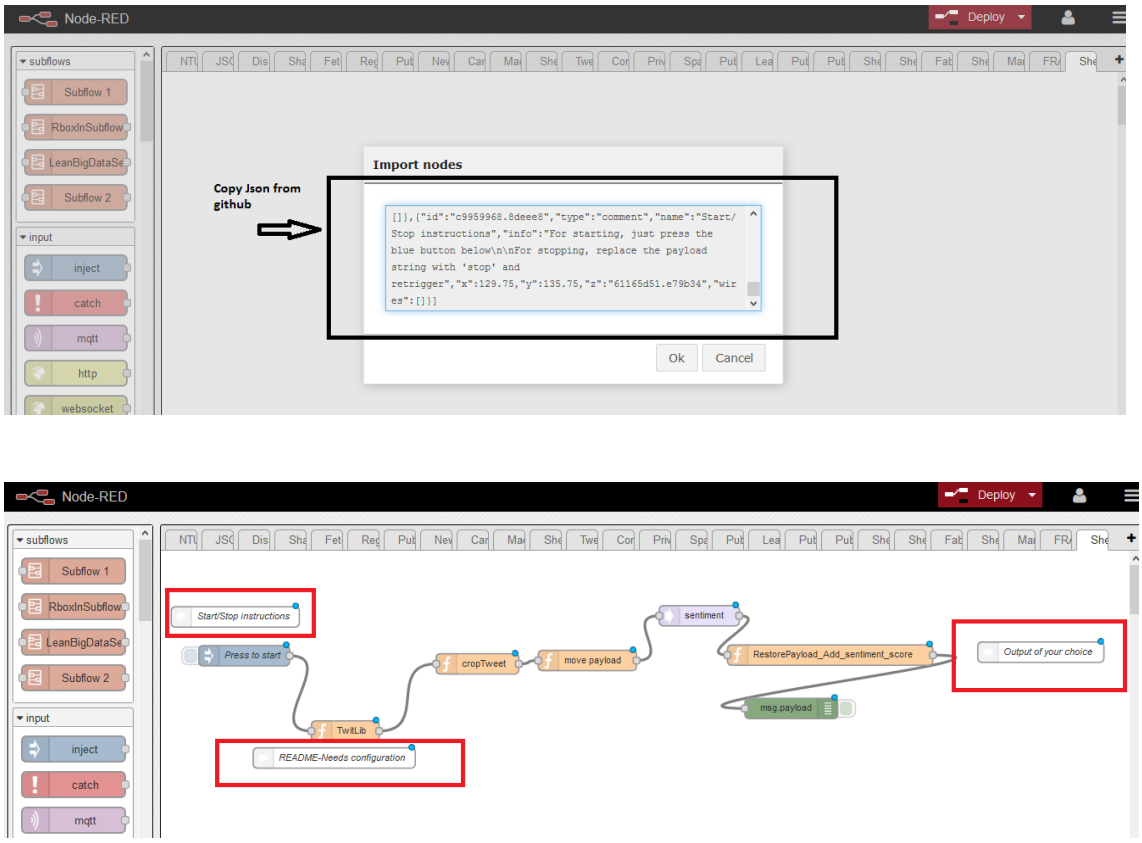


Figure 167: Process of importing an existing Node-RED flow and configuration details

4.2.13.4.2 Description of an event on the Marketplace

Following an account creation on the Marketplace and an event production, the producer needs to register this event on the Marketplace. In order to do that, they need to:

- Follow the process of posting a new listing (button available on Front end)
- Define the category in which the listing is included for semantic and filtering purposes (indicative categories may include City event, Weather event, Social networks event etc.)
- Define the type of the listing (free or for purchase)
- Populate the fields on the event template description that appears in Figure 168

Especially for the latter, the fields are the following:

- Description of the event. This ideally should include details on how it is produced, as well as key information with relation to e.g. boundaries of the event or of the provided fields.
- JSON schema. This should include the schema under which the event is published. There is no limitation with relation to the content of the schema, since this is event-related, only that it should be in JSON.
- JSON example. An example data item based on the previous schema could be useful for potential consumers.
- Endpoint. This is necessary if the event is published outside the messaging system of Eventflows, or it can contain also the respective endpoint of the latter under which the data are available.

- Estimated produced number of events per hour. This is an indicative field that should be populated if applicable, in order to inform potential consumers regarding the message rate they are expected to ingest. Remember that consumers also login through the messaging system and receive the data, but the post processing of this data on their end should also be present. Therefore this information will be useful in order for them to determine what kind of resources they need to accommodate for this purpose.
- QoS Features. This is an indicative field that may be populated by the producer in order to let consumers know key Quality of Service characteristics of the event. Examples of such cases may be the error of a prediction service, as tested against specific conditions/data set, the sensitivity of the sensors, the rate of guaranteed data acquisition etc. This relates to our work within the ISO committee for the draft 19086-2 standard on Cloud metrics and how these can also be applied in the IoT context. More information can be found on our collaborative work with the SLALOM project and how such metrics can be applied in the IoT domain (http://www.iot-cosmos.eu/sites/default/files/cosmos/files/content-files/articles/7_COSMOS_SLALOM_IoT_SLAs.pdf), candidate metrics that have been identified through an online survey (http://www.iot-cosmos.eu/sites/default/files/cosmos/files/content-files/articles/Report_from_survey.pdf) and the analysis of this work in COSMOS D8.2.3.

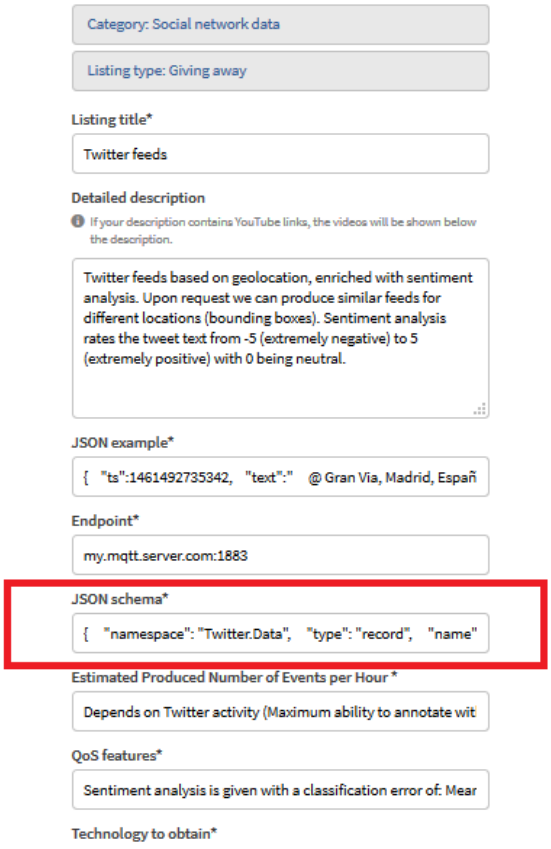


Figure 168: Event template description on Eventflows.com

4.2.13.4.3 Pushing to the Marketplace Messaging System

After producing the event and preparing its description for the Marketplace, now it is time for actually pushing it for publication in the Eventflows messaging system. As mentioned in the previous sections, the users of the Marketplace (both consumers and producers) are expected to use the common Pub/Sub system based on RabbitMQ in order to exchange information. However, in order for the system to have consistency in terms of namespaces and management, these users have limited rights for configuration. This creates the problem that widely used Node-RED nodes that could be used for accessing RabbitMQ (based on the AMQP protocol) like https://www.npmjs.com/package/node-red-contrib-amqp can not be used. The reason for this is that the specific implementation requires extensive rights in order to be able to operate, such as configuration rights for creation of exchanges and queues. However in the marketplace pub/sub system, these rights are reserved only for the administrator role and not for the producer or consumer role.

For this reason we created a set of customized Node-RED flows that may be able to override this issue. The main library used (from many tested) was amqplib (https://github.com/squaremo/amqp.node), which was included and configured appropriately in the subscriber and consumer implementation flows (and alternatively Java clients also) that have been published on the COSMOS GitHub (Figure 169). The flow consists of a main function incorporating the node.js function of amqplib, appropriately configured to be used with the Eventflows messaging system. However a set of configuration steps are also needed, as indicated in the README file included in the flow, which includes inserting e.g. credentials acquired, exchange name and url of the messaging system, as provided by the marketplace. Furthermore, a manual testing trigger is included as well as a json node which may be needed if the produced event is encoded as a JSON object. In that case it needs to be transformed to a string in order to be visible to the consumer. The link of the produced event to the publication flow is expected to be performed either in this JSON node or in the function node ("Push data to marketplace").

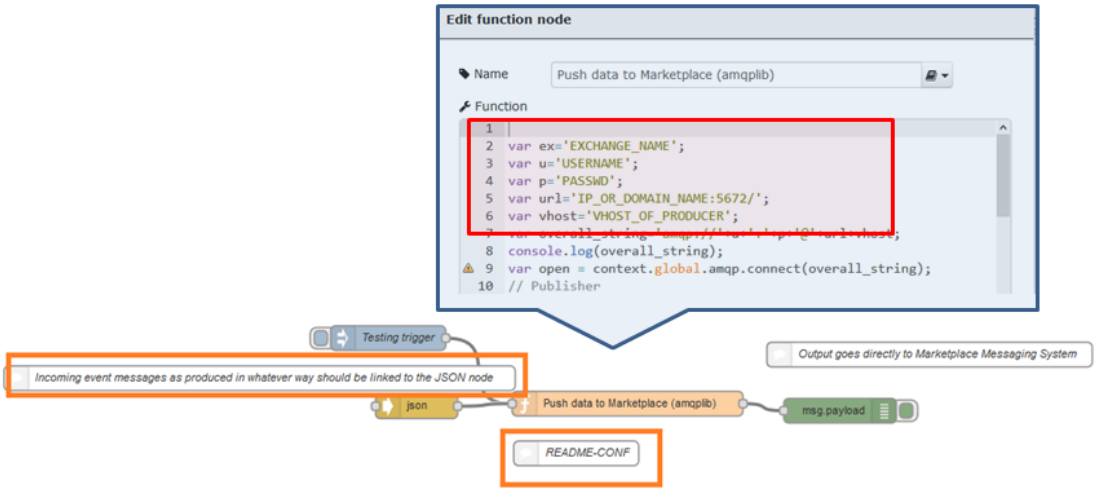


Figure 169: Producer events publication flow

4.2.13.5 Registration to an event by a consumer

Following the availability of a set of events, a consumer may be interested in acquiring the data for their own means and further processing/exploitation. In order to do that they must undergo through the following set of steps:

- Check and filter offered events from website via the offered parameters (Figure 170)
- Register for an event (and pay if it is not given away for free) (Figure 171)
- Credentials for accessing the Messaging system (AMQP protocol) will be sent via email along with the endpoint
- Use and configure provided Node-RED flows or Java clients for consuming data from the respective endpoint (https://github.com/COSMOSFP7/Eventflows-Marketplace/tree/master/NodeRED_Clients) (Figure 172)

Especially for the latter step, configuration is also needed (as was the case in the Producer flow) in terms of credentials for the specific user and endpoint. Furthermore, a manual trigger for starting and stopping data acquisition is available as well as detailed README files inside the flow.

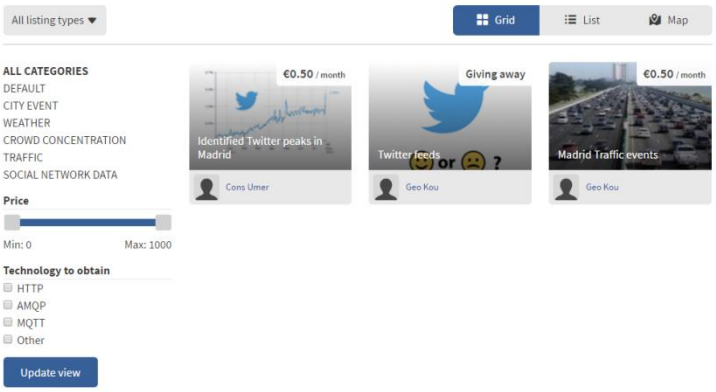


Figure 170: Events filtering on Marketplace

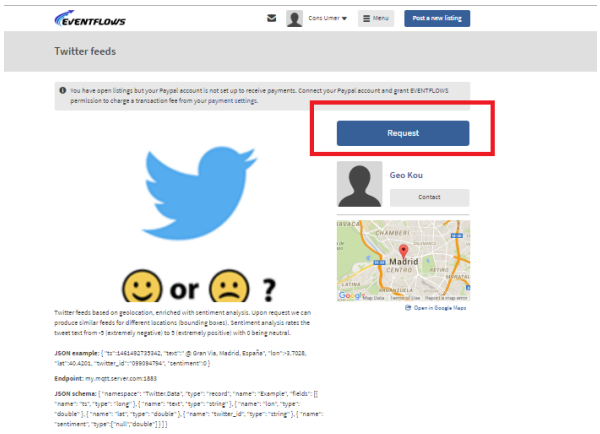


Figure 171: Event registration for consumers via the marketplace

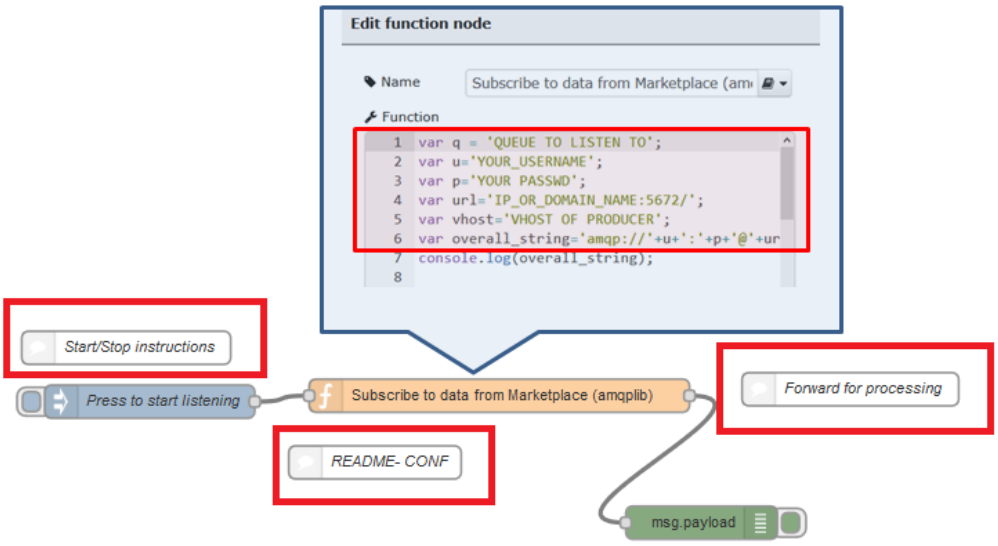


Figure 172: Consumer data acquisition flow

4.2.13.6 Management of the common Pub/Sub structure

The management of the messaging system is of course a task that should be centrally driven by the Marketplace entity in order to maintain order and consistency in the way resources are created and assigned across the various participants of the structure. The way these resources are managed is a combination of the used technology (in our case RabbitMQ and the AMQP protocol) and the foreseen business model. Therefore a mapping should be performed between the common use cases of the system, the access levels enforced for each role and the way these are implemented and set.

4.2.13.6.1 Access Levels definition

As indicated in the previous sections, there are two major roles in the Marketplace, the events producer and the events consumer. To guarantee management and controlled access to the common messaging structure, managed by the marketplace entity, none of these roles have configuration rights on the structure (that is the ability to create any kind of resource such as exchanges, queues, accounts, bindings etc.). Only the administrator of the structure has this kind of configuration capacity. Therefore, for the producers the only applicable action is to have write privileges on their respective exchanges and for the consumers to have read privileges on their respective queues. Bindings and resource creation is automatically handled by the system upon other actions such as user account creation (on the web site), event registration etc. These actions also trigger the creation of the respective accounts and resources on the messaging back-end.

4.2.13.6.2 Management Automation and administration

In order to aid in the management and administration of the marketplace backend, suitable Node-RED flows have been created. These undertake to implement the sequence of actions that are necessary following a producer's publication of a new event or a consumer's registration for consuming an event. These actions mainly refer to the organization of the backend common Pub/Sub system in order to create the necessary accounts, RabbitMQ structures (exchanges and queues), access and configuration rights and endpoints and appear in Figure 173.

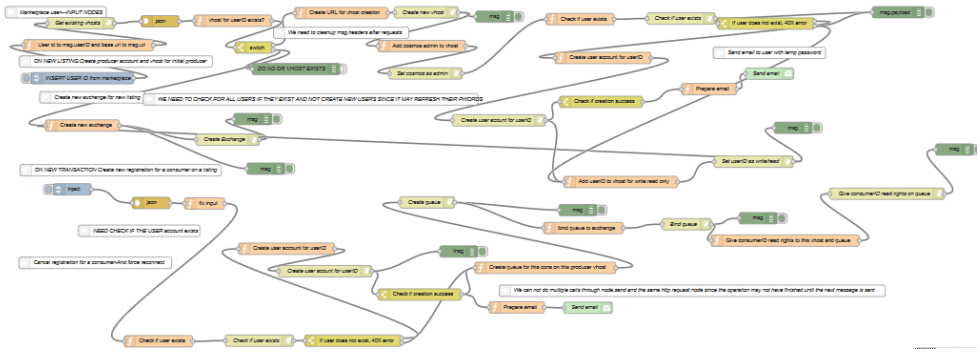


Figure 173: Automation Flow for Back-end common messaging system between producers and consumers

4.2.13.7 Message formats and configuration

The messaging formats are defined by the producer in the marketplace. The only limitation that we pose is that they are in JSON, since this is the main format supported in Node-RED. With relation to configuration aspects, the only case of needed information is the fields that are necessary to be populated in the template consumer and publisher flows, with the respective e.g. user credentials, as mentioned in detail in the previous sections.

4.2.13.8 Sequence Diagram

The sequence diagram containing the overall operations for the producer phase appears in Figure 174.

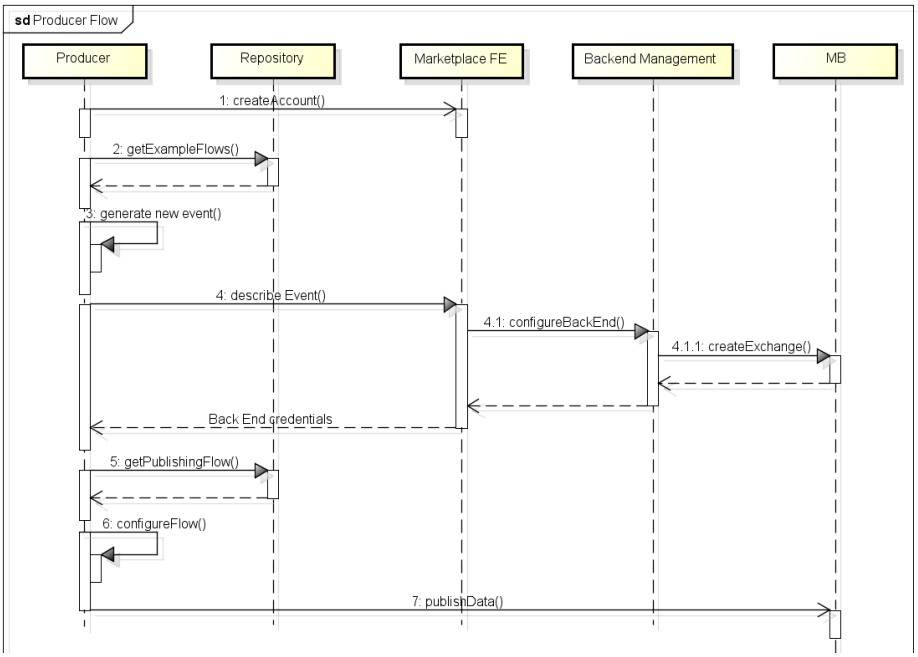


Figure 174: Sequence diagram for overall Producer phase

The sequence diagram containing the overall operations for the consumer phase appears in Figure 175.

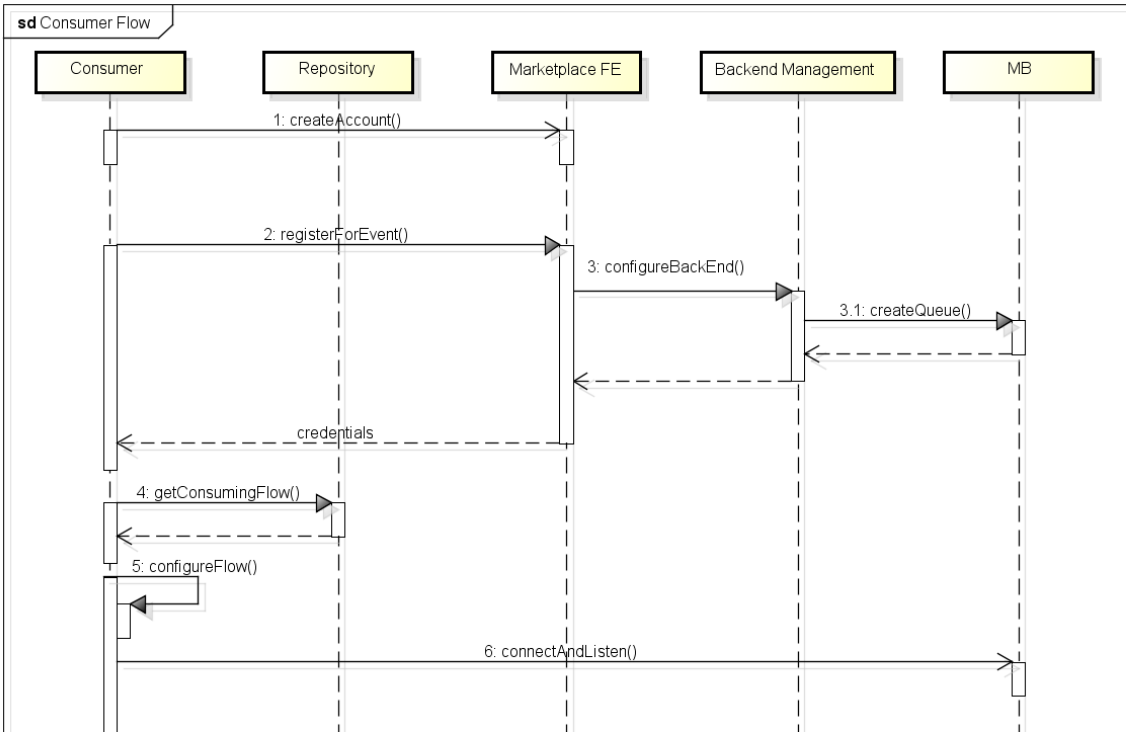


Figure 175: Sequence diagram for overall Consumer phase

4.2.13.9 Subsystem Test case table

The following test cases have been performed for the Marketplace functionalities.

Table 30: Producer Test case for the Marketplace

Test Case Number Version	MAR_01
Test Case Title	Marketplace producer flow
Module tested	Marketplace Front End, Marketplace BackEnd, Node-RED Flows (management and publication)
Requirements addressed	6.41, 6.43, 6.47, 7.1, UR4, UR7, UR10, UR14
Initial conditions	A functional Message Bus based on RabbitMQ Node-RED instance
Expected results	User account created for producer on Marketplace and on MB Ability of the producer to push messages on their resources on the MB
Owner/Role	Application/Events developer
Steps	Create account on Eventflows.com Retrieve flows from COSMOS GitHub and paste in local Node-RED instance Create event description in Eventflows.com Admin to trigger backend actions Get access credentials and configuration details on the backend MB Publish into Message Bus
Passed	Yes

Bug ID	Automatic password generation needs to avoid special characters since it is afterwards URL encoded in the path of the MB.
Problems	Due to the Marketplace solution not exposing the user database, at the moment the account creation on the backend following the account creation on the front end needs to be triggered manually
Required changes	At the moment the publishing flow creates a new connection for each new data item that needs to be pushed. This was followed in order to optimize for cases of infrequent events delivery, which is considered as the majority of the cases (for example in the case of Madrid traffic one new message is published per 5 minutes per MP). However if there are cases where frequent events are identified this could be more optimized if the connection remained open.

Table 31: Consumer Test case for the Marketplace

Test Case Number Version	**MAR_02**
Test Case Title	Marketplace Consumer flow
Module tested	Marketplace Front End, Marketplace BackEnd, Node-RED Flows (management and subscription)
Requirements addressed	6.41, 6.43, 6.47, 7.1, UR4, UR7, UR10, UR14
Initial conditions	A functional Message Bus based on RabbitMQ Node-RED instance
Expected results	User account created for consumer on Marketplace and on MB Ability of the consumer to retrieve messages for the registered events on the MB
Owner/Role	Application/Events developer
Steps	Create account on Eventflows.com Retrieve consumption flows from COSMOS GitHub and paste in local Node-RED instance Admin to trigger backend actions Get access credentials and configuration details for the backend MB Retrieve data fromMessage Bus
Passed	Yes
Bug ID	Automatic password generation needs to avoid special characters since it is afterwards URL encoded in the path of the MB.
Problems	Due to the Marketplace solution not exposing the user database, at the moment the account creation on the backend following the account creation on the front end needs to be triggered manually
Required changes	-

Table 32: Authorization Test Case for Marketplace illegal access

Test Case Number Version	MAR_03
Test Case Title	Marketplace authorization flow
Module tested	Marketplace BackEnd, Node-RED Flows (management, publication and consumption)
Requirements addressed	6.41, 6.43, 6.47, 7.1, UR4, UR7, UR10, UR14
Initial conditions	A functional Message Bus based on RabbitMQ Node-RED instance
Expected results	Producer should not be able to alter the configuration, create bindings or other resources on the MB Consumer should only be able to receive the information for which they have registered Any non registered user should be denied access
Owner/Role	Application/Events developer
Steps	Retrieve flows from COSMOS GitHub and paste in local Node-RED instance Get access credentials and configuration details on the backend MB Producer to attempt performing bindings, generating new exchange or queues Consumer to attempt performing bindings between their queue and another exchange (other than the ones for which they have registered)
Passed	Yes
Bug ID	-
Problems	Given the strict definition of privileges, many out of the box solutions such as the packaged Node-RED node for AMQP will not work (since they attempt to create e.g. queues upon initialization). Therefore it is strongly suggested to use only the flows provided by COSMOS.
Required changes	-

4.2.13.10 Deployment Diagram

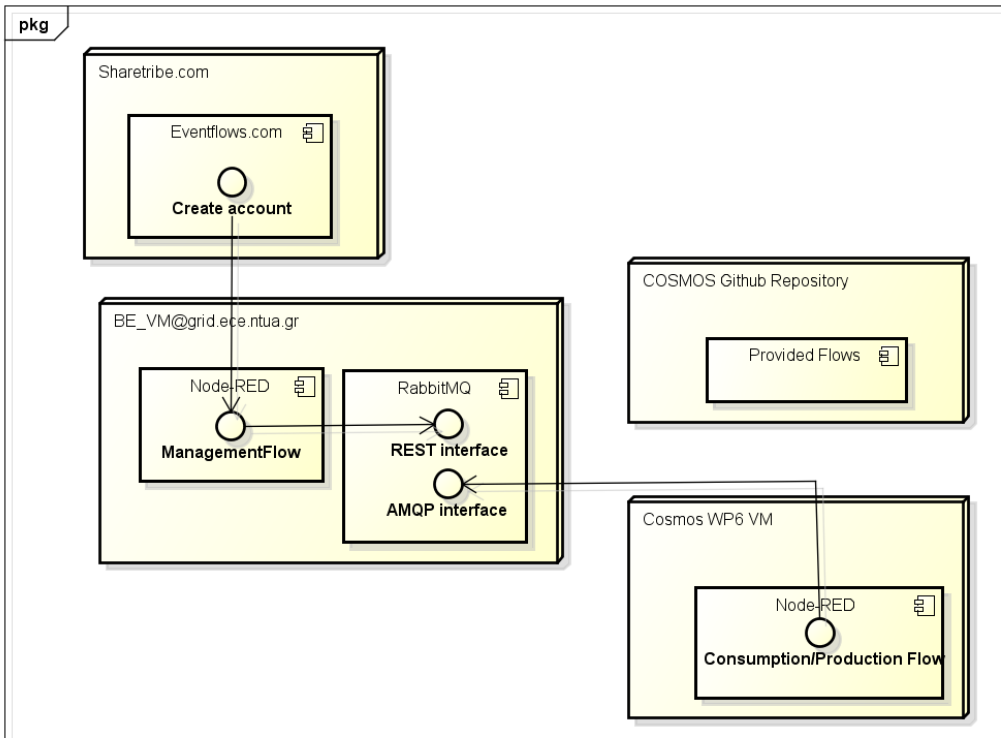


Figure 176: Deployment Diagram for Marketplace elements

4.3. Evaluation of Plan Goals, observed deviations and applied mitigation strategies

In terms of evaluation of the implementation plan, the results appear in the following table for the goals that are still active in the third year of the project.

What is critical to notice is that the project extended in two external use cases that were not included in the formal use case planning (Fab Lab cooperation and Sound Analyzer cases), demonstrating the ability of the tools to adapt to different circumstances and be versatile enough to handle diverse problems.

Furthermore, the incorporation of the Marketplace entity is considered a key point for exploiting COSMOS outcomes and enabling their sustainability, while easily integrating with the COSMOS outcomes in terms of event production and consumption. The Marketplace concept is also considered as a very realistic mean of achieving sustainability following the project end.

Finally, end user feedback has been incorporated in the majority of the scenarios, as indicated in Section 4.2.7 and detailed in the respective implementation tasks, enabling the latter to adapt to user requirements, in a variety of cases ranging from front end interfaces up to the core functionalities of each scenario.

Table 33: Planned Goals and Status for the respective periods (M26-36)

Goal	Subgoal	Status (M36)
VE Description and Linking to functionalities	VE descriptions	Available
	Integration to functionalities	Privelets fuzzification and VE recommendation
COSMOS Platform Integration	Data Management, Analytics and Autonomous VE behavior	P&C integration, Fuzzification integration, multi-source prediction framework
	Packaging process	Variety of packaged components through either source code, Node-RED flows, event offering or image-based logic
Application Definition and Creation	Scenarios Concretization	Available, with extended functionalities for Y3
	Application Definition Framework reusable flows	Extended with Y3 flows and interfaces, Available
	Extension of Archetypes Definition and Marketplace investigation	Available and extended with 1 Y3 archetype (including its instantiation through the Eventflows Marketplace)
VE side components integration	Endpoint Definition	Available through Registry
	Installation process	Raspberry Pi Image
Data Model	Data template/schema for new flows	Available for two new cases of data used
End User feedback	Interfaces	Included in GUIs for consent management, Caregiver and SP Front Ends and Smart Heating App
	Functionalities	Included in event list identification for caregiver alarms, in sound detection list for hearing impairment. In User preferences exceptions for heating schedule and failsages logic for Planner, and traffic event identification quality for traffic controllers
External exploration of Cases	Fab/Lab	For all the external cases investigated, the maturity level has been significant and beyond the original expectations in the Integration Plan (D7.6.3). For the Fab/Lab the data have been made available, the Eventflows Marketplace is operational and with automated back-end management and for the Sound Detection case a prototype has been implemented.
	Marketplace Investigation	
	Sound Detection scenario	

5. Conclusions

As a conclusion and following the implementation of the integration plan finalized in D7.6.3, we can conclude that all the major and minor integration goals have been achieved.

In the end of Y1 of the project we deployed an internal COSMOS platform for hosting the components coming from the technical WPs. These components have been grouped in specific subsystems that have the goal of providing added value functionalities. For these functionalities, a set of tests and scenarios have been drawn, integrated and executed successfully, paving the way for its migration to the actual application UCs, as these have been defined in the relevant documentation (D7.1.1).

Following, in Y2 we have centralized the integration process around the linking between the UCs (in terms of data feeds and/or functionalities) and the architectural subsystems. The latter have also been expanded with new or combined capabilities (e.g. proactive experience sharing reversing the information flow, combinatorial subsystem of Machine Learning and CEP for rules boundaries definition), that were directly applied in the context of the UCs. Initial versions of the applications have been implemented on all available UCs, including significant differentiations from Y1, in terms of data richness, used protocols and sources of information. Furthermore, related schemas for information exchange have been defined where appropriate, for inter-component communications. Significant work was also the incorporation of Node-RED as a multi-purpose tool in the context of the project (as a workflow, integration, development and testing environment), that has enabled faster, easier integration and the ability to perform arbitrary or reusable combinations, as envisioned by the COSMOS project and laid out in D7.6.2. This has also enabled the abstraction of the application archetypes, envisioned in D7.6.2, and their extendibility for Y3. Furthermore, it has helped clarify and hide integration points, thus allowing for added underlying complexity to be achieved.

During Y3, the main goal was to further abstract the integration and reusability potential of the COSMOS artefacts, which has been achieved via the extension of the number and type of available flows. Extended versions of the application scenarios have been provided, along with the investigation and successful incorporation of new UC types such as the Sound Analyzer UC and the Fab/Lab UC, that have succeeded in showing COSMOS applicability in even more application categories. New data feeds have been included in order to enhance scenarios but also to demonstrate the various concepts and ease of integration achieved through the strategy and its implementation. Furthermore, the technical presentation of the Eventflows marketplace was included, to act as an entity through which developers may monetize their efforts from exploiting open data sources and fine-graining the acquired knowledge through the event identification. Last but not least, end user feedback has been taken under consideration from a variety of viewpoints in terms of interfaces, functionalities, special circumstances, fail-safes of operation and ease of use, from a variety of roles ranging from application developers to Camden residents and Madrid citizens. This feedback has enabled the enhancement of the technical outcomes and has led to specific improvements that have been documented in the previous chapters of this deliverable.

References

[1] COSMOS Deliverable 7.6.1 “Integration Plan (Initial)”, ICCS/NTUA and other partners, June 2014

[2] Support Vector Machine (SVM): http://en.wikipedia.org/wiki/Support_vector_machine

[3] K-Nearest Neighbours (KNN): http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[4] COSMOS Deliverable 2.2.1 “State of the Art Analysis and Requirements Definition (Initial)”, UniS and other partners, February 2014

[5] COSMOS Deliverable 5.1.1 “Decentralized and Autonomous Things Management: Design and Open Specification (Initial)", ICCS/NTUA and other partners, April 2014

[6] COSMOS Deliverable 4.1.1 “Information and Data Lifecycle Management: Design and Open Specification (Initial)", IBM and other partners, April 2014

[7] COSMOS Deliverable 7.6.3 “Integration Plan (Final)”, ICCS/NTUA and other partners, April 2016

[8] http://Node-RED.org/docs/writing-functions.html#global-context

[9] https://github.com/madridopenlabmobility/MOBILITY-MADRID-virtual-entities/tree/master/CodeSamples/Python%202.7

[10] Twitter credentials process for developers: https://dev.twitter.com/oauth/overview/application-owner-access-tokens

[11] COSMOS Deliverable 2.3.3 “Conceptual Model and Reference Architecture (Final)”, UniS and other partners, August 2016

[12] COSMOS Deliverable 8.2.3 “Report on Roadmap and Standardisation activities (Y3)”, Siemens and other partners, August 2016

[13] COSMOS Deliverable 6.1.3 “Reliable and Smart Network of Things: Design and Open Specification (Final)”, UniS and other partners, April 2016

[14] COSMOS Deliverable 3.1.3 “End-to-End Security and Privacy: Design and Open Specification (Final)”, Siemens and other partners, April 2016

[15] COSMOS Deliverable 7.5.3 “Smart events and protocols for smart public transport (Y3 Functionality)”, EMT, June 2016

[16] COSMOS Deliverable 7.3.3 “Smart events and protocols for smart public transport (Year 3)”, EMT, August 2016

[17] COSMOS Deliverable 4.2.3 “Information and Data Lifecycle Management: Software prototype (Final)”, IBM and other partners, June 2016

Annex A: Unit/Component Tests

Data Mapping

Table 34: Results for Data Mapping Unit Test #1

Test Case Number Version	4_DM_1
Test Case Title	Storing live data, coming from the Message Bus, in the Cloud Storage.
Module tested	Data Mapping
Requirements addressed	4.1, 4.2
Initial conditions	RabbitMQ service is installed and running Hildebrand data stream is available
Expected results	Data are stored as objects with timestamps, metadata and a lower threshold for the size.
Owner	Achilleas Marinakis
Steps	Execute: /NTUA/DataMapping/target java -jar DataMapping-Y1.jar Hildebrand
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Table 35: Results for Data Mapping Unit Test #2

Test Case Number Version	4_DM_2
Test Case Title	Storing historical data, coming from a static file, in the Cloud Storage
Module tested	Data Mapping
Requirements addressed	4.1, 4.2
Initial conditions	The path to the static file is provided.
Expected results	Data are stored as objects with timestamps and id metadata and a lower threshold for the size.
Owner	Achilleas Marinakis
Steps	Execute: /NTUA/DataMapping/target java -jar DataMapping-Y1.jar Surrey
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Cloud Storage – Metadata Search

We have developed unit tests for the metadata search component. These tests verify requirements 4.1, 4.3 and 5.0.

Cloud Storage – Storlets

We have developed unit tests for the Storlets component. These tests verify requirements 4.1, 4.4, 4.5, 4.6, UNI.041.

Prediction

Table 36: Results for Prediction Unit Test #1

Test Case Number Version	**6_ML_1**
Test Case Title	6_ML_1
Module tested	Prediction
Requirements addressed	6.2, 6.3, 6.6, 6.22, 6.23
Initial conditions	Python installed with numpy, scikit, pandas and statsmodel. Time series historical data file.
Expected results	Machine Learning model which can predict the user Occupancy state.
Owner	University of Surrey
Steps	Install the required libraries. Store the historical data file. Run the code.
Passed	Yes
Bug ID	N/A
Problems	No
Required changes	No

Semantic Description and Retrieval

Table 37: Results for Semantic Description and Retrieval Unit Test #1

Test Case Number Version	**5_SDR_1**
Test Case Title	VE semantic description and retrieval
Module tested	Triple store access
Requirements addressed	5.0, 5.2, UNI.414, UNI.416, UNI.425, UNI.426
Initial conditions	First version of the semantic model available. Test triple store installed. Test query API available.
Expected results	Successful recording of the VE triples and matches between the retrieval results and the expected VE descriptions.
Owner	SIEMENS
Steps	1)Use a template form to fill in the description of a VE; 2)Call the API to store the description of the VE into the triple store;

	3) Repeat steps 1 and 2 for a number of VEs with different attributes; 4)Query the triple store in order to retrieve the description of different VEs based on various search criteria; 5)Compare the results with the expected outcome.
Passed	Yes
Bug ID	
Problems	
Required changes	

Table 38: Results for Semantic Description and Retrieval Unit Test #2

Test Case Number Version	**5_SDR_2**
Test Case Title	VE description validation
Module tested	Triple store access
Requirements addressed	5.0, 5.2, UNI.414, UNI.416, UNI.425, UNI.426
Initial conditions	First version of the semantic model available. Test triple store installed. Test query API available.
Expected results	Inconsistent VE descriptions should not be stored into the triple store.
Owner	SIEMENS
Steps	1)Use a template form to fill in the description of a VE with erroneous or inconsistent fields; 2)Call the API to store the description of the VE into the triple store; 3) Repeat steps 1 and 2 for a number of VEs with different attributes; 4)Whenever the VE description does not meet the consistency requirements, and error condition must be signalled and no commit to the triple store should be made; 5)Query the triple store in order to retrieve the description of different inconsistent VEs whose description and storage attempt was made; 6)No complete or partial VE description should be retrieved.
Passed	Yes
Bug ID	
Problems	
Required changes	

Privelets

Table 39: Results for Privelets Unit Test #1

Test Case Number Version	**3_PR_1**
Test Case Title	Privacy
Module tested	Privelets
Requirements addressed	UR13
Initial conditions	Two VEs have entered COSMOS VPN using FreeLan and have acquired virtual IP addresses Both VEs have received the necessary keys to communicate with each other safely Privelets configuration file is filled in properly Followers and Followees List are updated
Expected results	VEs exchange public data with anonymity
Owner	Achilleas Marinakis
Steps	Install Privelets package under the main root of a machine Inside the /Privelets directory execute the jar file: java –jar Privelets-Y2.jar In another machine send a GET or POST request to the first machine
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Planner

Table 40: Results for Planner Unit Test #1

Test Case Number Version	**5_PL_1**
Test Case Title	Notification Service and CBR
Module tested	Planner
Requirements addressed	UNI. 704, 706, 708, 715, 719 5.31 5.29, UNI.010 5.21 5.10 5.9, UNI.251
Initial conditions	Have a CB inside the test bed. Have the VE and app simulation jar inside the test bed.
Expected results	The answer to the notification of the User Generated Event in the GUI. Command Line or Terminal Log with similarities retrieved and queries used.
Owner	Panagiotis Bourelos, Orfefs Voutyras
Steps	Place the CB in the localstore folder of the home directory. Open command line or terminal.

	Navigate to the jar folder. Execute “java –jar DemoProjectMaven-1.0-SNAPSHOT.jar”. Keep the terminal or cmd open to view log info and locate the GUI. Enter a set of temperature and time that exist locally (time: 3535 and temp: 26). Press “Send Notification”.
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Experience Sharing

Table 41: Results for Experience Sharing Unit Test #1

Test Case Number Version	**6_ES_1**
Test Case Title	Experience Sharing and returned Solution
Module tested	Planner, Experience Sharing
Requirements addressed	Same as those in 5_PL_1 plus: 5.11, 5.12, 6.8, 6.18, 6.19, 6.34, 6.35
Initial conditions	Have a CB and Friend List inside the test bed. Have the VE and app simulation jar inside the test bed. Have one or both auxiliary XP sharing VEs active in the VMs, along with their CBs and Friend Lists.
Expected results	The answer to the notification of the User Generated Event in the GUI Command Line or Terminal Log with similarities retrieved and queries used The Logs of auxiliary VEs in the VMs with the Planner result and the Experience Sharing steps
Owner	Panagiotis Bourelos, Orfefs Voutyras
Steps	Place the CB and Friend List in the localstore folder of the home directory; Open command line or terminal; Navigate to the jar folder; Execute “java –jar DemoProjectMaven-1.0-SNAPSHOT.jar”; Keep the terminal or cmd open to view log info and locate the GUI; Boot the VEs in their respective VMs; Enter a set of temperature and time that exist in another VE (time: 1800 and temp: 26 for VE Flat 2/ time: 3535 and temp: 28 for VE Flat 3); Press “Send Notification”
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Hardware Security Board

Table 42: Results for Hardware Security Board Unit Test #1

Test Case Number Version	**3_HSB_1**
Test Case Title	Communication over standard interfaces
Module tested	Communication module
Requirements addressed	3.1
Initial conditions	Configure standard communication interfaces (I2C, SPI, USB, Ethernet)
Expected results	Data flow can take place over the specified interfaces. Sensors and the Ethernet network can be accessed.
Owner	Leonard Pitu
Steps	1. Load the hardware design (*.bit file) 2. Load the firmware 3. Boot the firmware 4. Obtain an IP address 5. Read the sensors 6. Make the data available over ETH.
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Table 43: Results for Hardware Security Board Unit Test #2

Test Case Number Version	**3_HSB_2**
Test Case Title	Secure the data flow (simulation)
Module tested	Cryptographic accelerator
Requirements addressed	3.3
Initial conditions	Configure the FPGA with the hardware design
Expected results	The cryptographic accelerator shall meet the FIPS-180 standard.
Owner	Leonard Pitu
Steps	Standard test vectors 1. Load the hardware module in the simulator (ModelSIM) 2. Load the test cases according to FIPS-180 3. Run the tests Random test vectors 1. Load the hardware module in the simulator (ModelSIM) 2. Load private test vectors generated with Crypto++ 3. Run the tests
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Table 44: Results for Hardware Security Board Unit Test #3

Test Case Number Version	**3_HSB_3**
Test Case Title	Secure the data flow (board)
Module tested	Cryptographic accelerator
Requirements addressed	3.3
Initial conditions	Configure the FPGA with the hardware design.
Expected results	The cryptographic accelerator shall meet the FIPS-180 standard.
Owner	Leonard Pitu
Steps	Standard test vectors 1. Load the hardware design (*.bit file) 2. Load the test cases according to FIPS-180 3. Run the tests Random test vectors 1. Load the hardware design (*.bit file) 2. Load private test vectors generated with Crypto++ 3. Run the tests
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Table 45: Results for Hardware Security Board Unit Test #4

Test Case Number Version	**3_HSB_4**
Test Case Title	HSB enrolment
Module tested	Cryptographic accelerator
Requirements addressed	3.7
Initial conditions	Configure the FPGA with the hardware design Load the software (Linux + drivers + test application)
Expected results	Use the ECDH to securely exchange the key
Owner	Leonard Pitu
Steps	1. Load the hardware design (*.bit file) 2. Load the software 3. Boot the system 4. Run the test cases and fetch the key from Keystone
Passed	Yes
Bug ID	None
Problems	None
Required changes	None