



# COSMOS

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement N° 609043

## D7.7.2 Integration of Results (Year 2)

**WP7: Use cases Adaptation, Integration and Experimentation**

**Version:** 1.0

**Due Date:** 30/09/2015

**Delivery Date:** 30/09/2015

**Nature:** Report

**Dissemination Level:** Public

**Lead partner:** ICCS/NTUA

**Authors:** George Kousiouris, Achilleas Marinakis, Panagiotis Bourellos, Orfefs Voutyras (ICCS/NTUA), Juan Sancho (ATOS), Paula Ta-Shma (IBM), Francois Carrez, Adnan Akbar (UniS), Bogdan Târnaucă, Leonard Pitu (SIEMENS), Andres Recio (EMT)

**Internal reviewers:** Abie Cohen (HILD)

[www.iot-cosmos.eu](http://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
0.1	19/05/2015	George Kousiouris	ICCS/NTUA	Update of previous version ToC based on new version of Integration Plan
0.2	16/07/2015	George Kousiouris	ICCS/NTUA	Initial Definition of System and Integration Group testing, Definition of subchapter template
0.3	31/07/2015	George Kousiouris	ICCS/NTUA	Finaization of Integration Group testing and ToC assignments
0.4	8/09/2015	Panagiotis Bourellos	ICCS/NTUA	Adding contributions on Experience Sharing
0.5	8/09/2015	Juan Sancho	ATOS	Adding contributions on Situational Awareness process
0.6	08/09/2015	Juan Sancho	ATOS	Updated Madrid Data Feed section
0.61	11/09/2015	Panagiotis Mpourellos	NTUA	Contributions in Camden App Scenario, Data Feed and Proactive Exp sharing
0.62	15/09/2015	George Kousiouris	NTUA	Added contributions in 3.2.2
0.63	21/09/2015	Adnan Akbar	SURREY	Added input in Taipei Data feed and 3.2.4.1
0.64	21/09/2015	Juan Sancho	ATOS	Initial Madrid Traffic UC section
0.65	22/09/2015	George Kousiouris, Andres Martin	NTUA, EMT	Madrid Scenario Description
0.66	22/09/2015	Paula Ta-shma	IBM	Input on 3.2.1
0.67	22/09/2015	Juan Sancho	ATOS	Updated Madrid Traffic UC section
0.68	25/09/2015	Panagiotis Mpourellos, Abie Cohen	NTUA, HILD	Contributions in Camden App Scenario, Data Feed and Proactive Exp sharing and Camden DSO
0.69	25/09/2015	Leo Pitzu, Achilleas Marinakis	SIE, NTUA	Update of VE side integration chapter
0.7	27/09/2015	George Kousiouris	NTUA	Updated Exec Summary, Introduction, Conclusions
0.71	28/09/2015	Juan Sancho , Adnan Akbar	ATOS, SURREY	Fixings in Data Feed subsections
0.8	28/09/2015	George Kousiouris, Andres Martin	NTUA, EMT	Finalization of Madrid Scenario Description
0.81	28/09/2015	Bogdan Tarnauca	SIE	VE Registry incorporation
0.9	29/09/2015	Abie Cohen	HILD	Internal Review
1.0	30/09/2015	George Kousiouris	NTUA	Final Version Ready

## Table of Contents

Executive Summary .....	12
1. Introduction .....	14
1.1. Purpose of the Document .....	14
1.2. Integration Periods within the scope of this document .....	14
1.3. Document Structure .....	14
2. Integration Stage 1 (M10-M16) from D7.7.1 .....	16
2.1. Overview from Integration plan.....	16
2.2. Performed Tests .....	17
2.2.1. Subgoal: Data Management and Analytics Tests .....	17
2.2.2. Subgoal: Autonomous Behaviour of VEs.....	39
2.2.3. Component Tests .....	51
2.2.4. Occupancy detection model validation.....	51
2.3. Observed Deviations and Applied Mitigation Strategies .....	54
2.4. Future steps with regard to the advancements of this period .....	55
2.4.1. Security aspects.....	55
2.4.2. Data Analytics and Storage .....	55
2.4.3. Autonomous Behaviour of VEs.....	55
2.4.4. Modelling Aspects .....	55
3. Integration Stages 2 (M17-M22) & 3 (M23-M25) .....	56
3.1. Overview from Integration Plan.....	56
3.2. Defined Subsystems and Tests.....	57
3.2.1. VE Instances Descriptions, Registry interface and Data Schema Storage/Retrieval	57
3.2.2. Application Definition Framework through Node-RED Flows.....	67
3.2.3. VE side COSMOS Components integration/Installation.....	73
3.2.4. COSMOS components integration .....	76
3.2.5. Data Feeds integration .....	89
3.2.6. Madrid Scenario Application .....	99
3.2.7. Camden Scenario Application .....	107
3.2.8. Taipei Scenario Application .....	119
3.3. Evaluation of Plan Goals, observed deviations and applied mitigation strategies ...	123
3.4. Future steps with regard to the advancements of this period .....	125
4. Conclusions .....	126
References.....	127

Annex A: Unit/Component Tests .....	128
Data Mapping .....	128
Cloud Storage – Metadata Search.....	129
Cloud Storage – Storlets .....	129
Prediction .....	129
Semantic Description and Retrieval .....	129
Privelets.....	131
Planner .....	131
Experience Sharing.....	132
Hardware Security Board .....	133

## Table of Figures

Figure 1: COSMOS Platform Overall Deployment Diagram .....	16
Figure 2: Data Feed, Annotation and Storage Scenario .....	18
Figure 3: Data Feed, Annotation and Storage Subsystem .....	19
Figure 4: Subscribe/produce data adaptation .....	19
Figure 5: Bridge Configuration .....	20
Figure 6: RabbitMQ HTTP-based API - Topic "FromMosquitto" .....	20
Figure 7: JSON Message from the Camden UC .....	21
Figure 8: Configuration File .....	21
Figure 9: Sequence Diagram for Data Feed, Annotation and Storage Subsystem.....	22
Figure 10: Data Object Body .....	23
Figure 11: Data Object Metadata.....	23
Figure 12: Data Feed, Annotation and Storage Deployment Diagram .....	24
Figure 13: Metadata Search and Storlets Subsystem .....	24
Figure 14: Overview of EMT available data characteristics .....	25
Figure 15: Overview of Metadata and Storlets Scenario .....	26
Figure 16: Data Format in Object Storage.....	26
Figure 17: Metadata insertion in Object Storage.....	27
Figure 18: End user interface for the Metadata Search subsystem.....	28
Figure 19: Sequence Diagram for Storage and Analytics on Metadata Subsystem.....	28
Figure 20: Deploymnet Diagram for the Metadata Search.....	29
Figure 21: Security, Privacy and Storage .....	29
Figure 22: Security Demonstrator Show-Case .....	30
Figure 23: Hardware Security Board .....	31
Figure 24: Security, Privacy and Storage Deployment Diagram.....	31
Figure 25: Demonstrator Flow .....	32
Figure 26: Sequence Diagram for Storage and Security Subsystem .....	32
Figure 27: Client side image encryption.....	33
Figure 28: Platform Gateway for receiving and decrypting encrypted image .....	33
Figure 29: Facial Blurring Storlet Usage .....	33
Figure 30: Storlet Output (Blurred Image) .....	34
Figure 31: Storage and Modelling Subsystem.....	35
Figure 32: Overview of integration between Spark and Swift .....	36
Figure 33: Data Flow across the Subsystem.....	36

Figure 34: Input data format for Modelling case .....	37
Figure 35: Model Training using Spark MLlib, Storlets and Swift.....	37
Figure 36: Runtime Prediction using the trained model .....	37
Figure 37: Sequence Diagram for Modelling and Storage Analytics Subsystem.....	38
Figure 38: Modelling and Storage Analytics Deployment Diagram .....	39
Figure 39: Autonomous Behavior of VEs with minimum platform integration subsystem .....	41
Figure 40: Autonomous Behavior of VEs application GUI.....	41
Figure 41: Components of the main flat-VE.....	42
Figure 42: Case Base and Friend List examples.....	42
Figure 43: Visual representation of all scenario possibilities .....	43
Figure 44: Autonomous Behavior of VEs- Automated Event Detection and Incorporation of COSMOS Platform .....	44
Figure 45: Visual representation of the platform integration scenario .....	45
Figure 46: Sample DOLCE configuration file of freezing event detection.....	46
Figure 47: Example of a data stream as input to the CEP .....	46
Figure 48: Example of code for MB subscription for listening.....	46
Figure 49: Autonomous Behaviour of VEs Deployment Diagram .....	48
Figure 50: Autonomous Behaviour of VEs System Sequence Diagram.....	48
Figure 51: Enrich Case Base Content Sequence Diagram .....	49
Figure 52: React to Event Sequence Diagram .....	49
Figure 53: RequestSolution(Problem) Sequence Diagram.....	50
Figure 54: UpdateSocialMetricsLocal(Evaluation_of_Solution) Sequence Diagram .....	50
Figure 55: Performance of Classifiers.....	53
Figure 56: F-measure relation with training data .....	53
Figure 57: Time Complexity.....	54
Figure 58: Generic usage scenarios for VE registry .....	58
Figure 59: VE Registration Subsystem with identified IPs.....	59
Figure 60: Cosmos VE semantic annotation welcoming webpage .....	59
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).....	60
Figure 62: Mockup (top) and the actual implementation (bottom) for the Interface Endpoint Definition page.....	61
Figure 63: Sample JSON object for service endpoint request.....	62
Figure 64: JSON schema storage and association .....	63
Figure 65: Camden DSO fields .....	64

Figure 66: Reuse scenarios of Node-RED flows (Roles interactions Figure 17 from D7.6.2) .....	67
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).....	68
Figure 68: Share_flows_public flow notation example.....	69
Figure 69: Internal node configuration for the Share flows functionality .....	69
Figure 70: Sequence Diagram for sharing flows.....	70
Figure 71: Configuration of settings.js file in Node-RED for external Node.js libraries .....	72
Figure 72: Node-RED function node code for inclusion of external libraries .....	72
Figure 73: VE side COSMOS components.....	73
Figure 74: Madrid Traffic State Analysis Use Case .....	77
Figure 75: Traffic flow Subsystem Diagram.....	78
Figure 76: Steps for finding threshold values for CEP rules .....	79
Figure 77: Madrid Traffic Analysis Use Case - Deployment Diagram.....	81
Figure 78: Camden Flat VE overview.....	81
Figure 79: SAw FC and the CEP Engine.....	82
Figure 80: Autonomous Behaviour of VEs, basis for Proactive XP Sharing.....	84
Figure 81: Flow of extracted Complex Events into other VE sided FCs.....	84
Figure 82: Proactive Experience Sharing Sequence Diagram.....	87
Figure 83: Deployment Diagram for the Proactive Experience Sharing Subgroup .....	88
Figure 84: Data Feed, Annotation and Storage Subsystem .....	90
Figure 85: Example of Data Mapper request .....	91
Figure 86: Madrid data feed Activity Diagram .....	92
Figure 87: Madrid data feed Flow .....	92
Figure 88: Madrid Data Feed Deployment Diagram .....	93
Figure 89: Data Bridging Activity Diagram .....	94
Figure 90: Data Bridging Flow and Connectivity .....	95
Figure 91: Deployment Diagram for Camden Data Feed .....	96
Figure 92: Taipei data feed Activity Diagram .....	97
Figure 93: Deployment diagram for III data feed.....	98
Figure 94: Madrid Scenario Integration .....	100
Figure 95: Centralized archetype subsystem as used in the Madrid Scenario Application .....	102
Figure 96: Generic Data Output Flow towards the RBox system.....	102
Figure 97: Data format for ROUTESMAD.usertrack responses .....	103
Figure 98: Push notification to RB server datagram format .....	103
Figure 99: Sequence diagram for Madrid Scenario Application .....	105

Figure 100: VE2VE application archetype based on defined system cases of D2.3.2.....	109
Figure 101: Autonomous Behavior of VEs with minimum platform integration subsystem ....	110
Figure 102: Autonomous Behavior of VEs- Automated Event Detection and Incorporation of COSMOS Platform .....	110
Figure 103: Heating Scheduling App GUI .....	112
Figure 104: Steps for the Heating Schedule Management Application (General View) .....	112
Figure 105: Steps for the Heating Schedule Management Application (CBR and XP Sharing parts) .....	113
Figure 106: Camden Scenario Deployment Diagram .....	116
Figure 107: Low volume simulations.....	117
Figure 108: Medium volume simulations.....	118
Figure 109: High volume simulations.....	118
Figure 110: Real-time anomaly detection.....	120
Figure 111: Sequence diagram for Taipei Scenario.....	121
Figure 112: Deployment diagram for III Scenario .....	122



## List of Tables

Table 1: Data Feed, Annotation and Storage Test Case .....	22
Table 2: Planner with minimum integration to the COSMOS Platform Test Case .....	43
Table 3: Planner full integration with COSMOS Platform .....	47
Table 4: Flow sharing test case .....	70
Table 5: Flow Fetching test case .....	71
Table 6: Software dependencies of VE side components .....	74
Table 7: Prerequisite packages for FreeLan and Node-RED .....	74
Table 8: Test case for the CEP and ML cooperation .....	80
Table 9: Test case for Proactive Experience Sharing .....	87
Table 10: Test Case for the Madrid Data Feed incorporation .....	92
Table 11: Test Case for the Camden Data Feed .....	95
Table 12: Test Case for the Taipei Data Feed .....	98
Table 13: Set of code alarms for the Madrid events exchange .....	104
Table 14: Camden Scenario Application Complex Event Handling .....	114
Table 15: Heating Schedule Test Case .....	115
Table 16: Test case table for the Taipei application scenario .....	122
Table 17: Evaluation of goals over achieved results in M25 .....	124
Table 18: Results for Data Mapping Unit Test #1 .....	128
Table 19: Results for Data Mapping Unit Test #2 .....	128
Table 22: Results for Prediction Unit Test #1 .....	129
Table 23: Results for Semantic Description and Retrieval Unit Test #1 .....	129
Table 24: Results for Semantic Description and Retrieval Unit Test #2 .....	130
Table 25: Results for Privelets Unit Test #1 .....	131
Table 26: Results for Planner Unit Test #1 .....	131
Table 27: Results for Experience Sharing Unit Test #1 .....	132
Table 28: Results for Hardware Security Board Unit Test #1 .....	133
Table 29: Results for Hardware Security Board Unit Test #2 .....	133
Table 30: Results for Hardware Security Board Unit Test #3 .....	134
Table 31: Results for Hardware Security Board Unit Test #4 .....	134

## 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
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 2 of the COSMOS project. We continued the process from Y1, after installing the functional components of COSMOS in the testbed, following the details and requirements, as these are expressed in D7.6.1 “Integration Plan (Initial)” [1], with regard to component distribution, VM creation and configuration, and updated in D7.6.2 “Integration Plan(Updated)”, with relation to further integration between the COSMOS components and especially between the latter and the UCs. It is necessary to stress that we have retained the material from D7.7.1 and the previous integration periods (included in Chapter 2), 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 3 and the related integration periods that fall under the scope of this document.

Following, and by receiving input from the Integration Plan for periods PM17-22 and PM23-25, 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:

- VE Registration process, aiming to highlight VE capabilities and define sources of information in terms of interfaces, schemas and semantic notation, initially targeting the Camden UC
- The Proactive Experience Sharing mode, a new mode of operation aiming to alert participants of an upcoming event or situation, integrating components like the Planner, Experience Sharing, Situational Awareness and Trust and Reputation and being applied in the Camden UC
- Data flow from different types of sources, adaptively annotated, grouped and stored on a Cloud based object storage, extending from Y1 to include new sources (e.g. Taipei) or formats (Madrid City and EMT data feeds, enhanced Camden data feed), available through necessary Node-RED flows and bridges
- An enhanced Situational Awareness process, achieving the cooperation between the Cloud Storage, Machine Learning and CEP components and applied to the Madrid UC for traffic rules boundaries identification
- VE side component integration process, in terms of packaging, deployment and cooperation, which is a generic process, intended to be used in Y3 in the Camden UC
- Generic schema retrieval capabilities, with which an independent Application Developer may be notified about the format of the published information and thus exploit it to build ad-hoc applications
- Integration actions and necessary points that are needed for the application scenarios that have been defined for Y2 for the Camden UC (Heating Schedule), Taipei UC (Anomaly Detection) and Madrid UC (traffic boundaries identification), including bridging and adaptation points with the COSMOS components

- Application Definition framework that aims to exploit the offered functionalities and abstract them so that they can be grouped, reused and shared, thus enabling arbitrary or extendable application development

This new set of features comes to complement or abstract the already achieved ones at Y1, namely metadata search and visualization, increased security and privacy, decentralized and autonomous management and knowledge acquisition of VEs, enriched with social interactions and increased modelling capabilities, incorporating data preprocessing and integrating Apache Spark with object storage (Openstack Swift) for data analysis and modelling using historical data, leading to a feature-rich and flexible solution that the COSMOS environment envisions to be.

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 in the future in new additions that will be necessary for the iteration of this document in the final year 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. Where applicable, further testing is defined in terms of non functional aspects such as prediction model accuracy (from Y1 revised version) or solution scalability (e.g. Y2 experience sharing scalability test).

As a conclusion, the goals identified for this second 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. In relation to the integration plan, we have achieved a very satisfactory percentage of maturity levels, with 12 out of 14 subgoals achieved (while the other 2 are in progress), and 4 of them going beyond the anticipated level of maturity. Initial versions of the applications have been implemented on all available UCs at the COSMOS side, the logic and complexity of which will be enhanced in Y3, while data feeds have been incorporated for all cases, including significant differentiations from Y1, in terms of data richness, used protocols and sources of information. Significant work, that is expected to continue, 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, enabling different roles to be included in the process by abstracting functionalities and implementation details. Minor corrections have also been identified for feedback to the integration plan and based on our experiences during this period in terms of (3) subgoal extensions over integration periods as well as subgoal substitution in one case.

## 1. Introduction

### 1.1. Purpose of the Document

D7.7.2 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 D7.6.2 [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

The integration periods (as defined in D7.6.2) corresponding to this document are:

- Integration period 2 (PM17-22) and
- Integration period 3 (PM23-25).

Both of which are included in **Chapter 3**, 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 Chapter 2 from D7.7.1 (previous version of this document and relating to Integration period 1 from PM10 to 16), 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
- Chapter 3 contains the **new** additions for **Y2** and with relation to the goals of these periods (PM17-22 and PM23-25) 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 3.1) with relation to the achieved results is portrayed (in Section 3.3).
- In Chapter 4, 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.

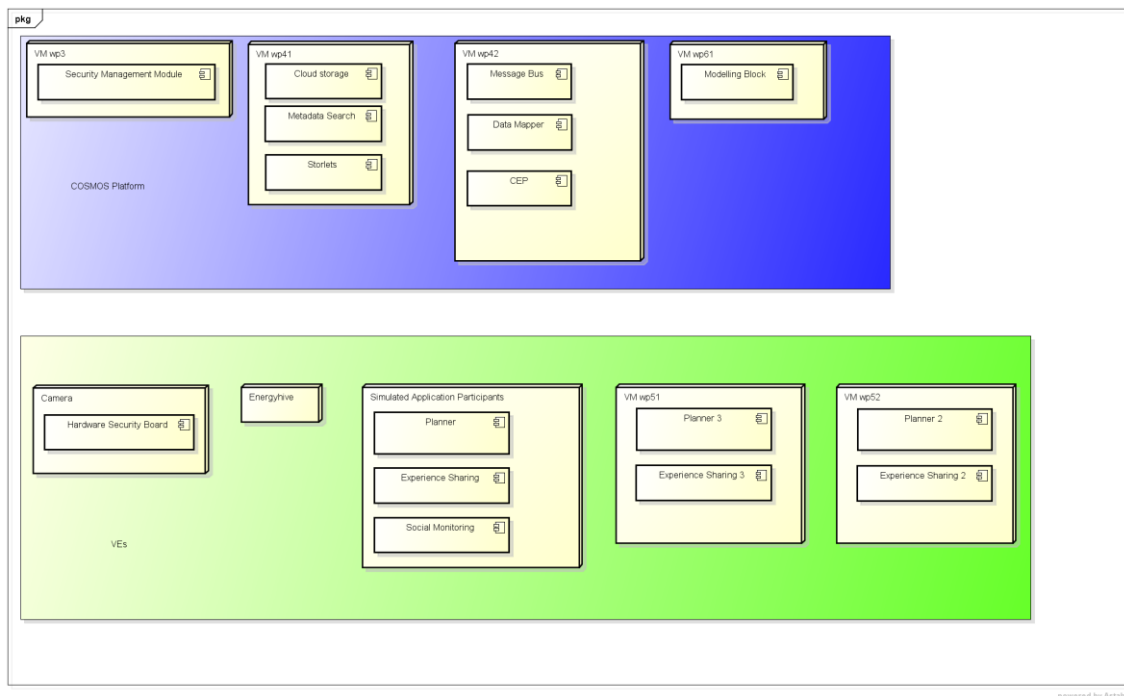


Figure 1: COSMOS Platform Overall Deployment Diagram



## 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].

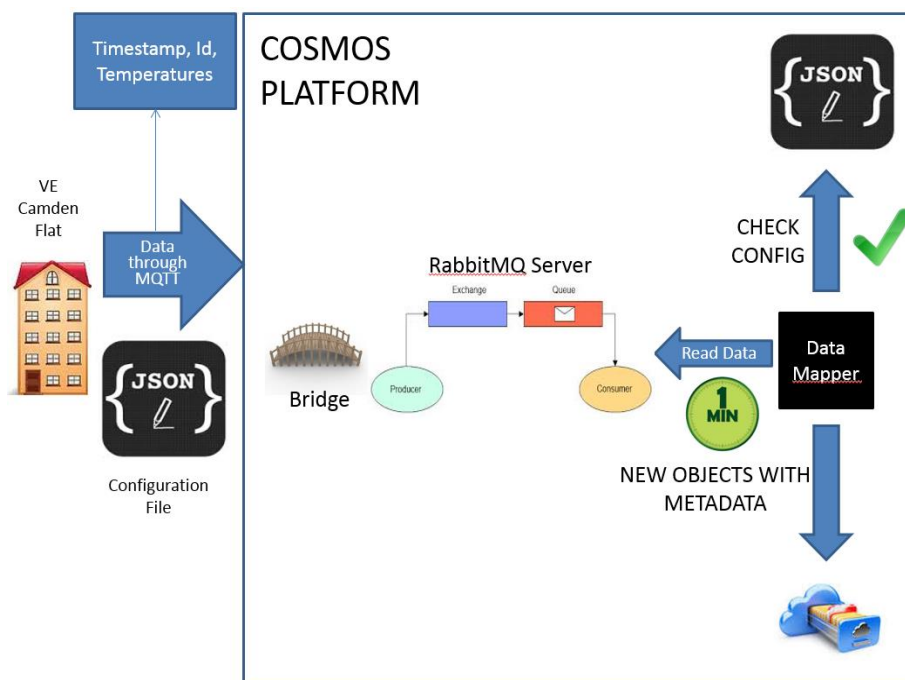


Figure 2: Data Feed, Annotation and Storage Scenario

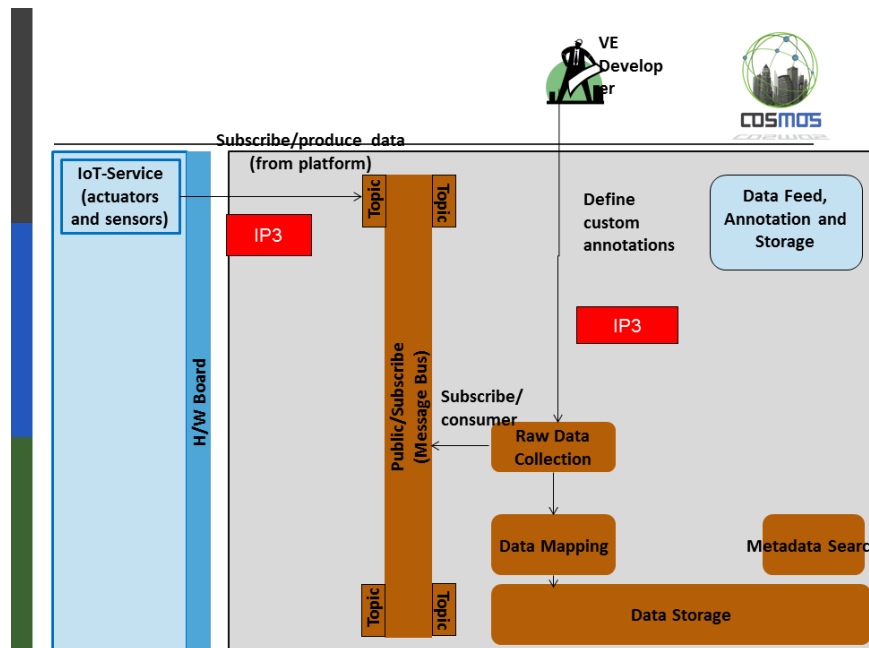


Figure 3: Data Feed, Annotation and Storage Subsystem

### 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.

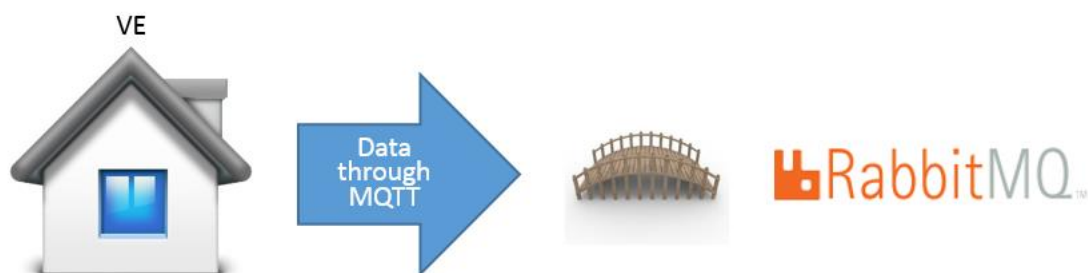


Figure 4: Subscribe/produce data adaptation

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

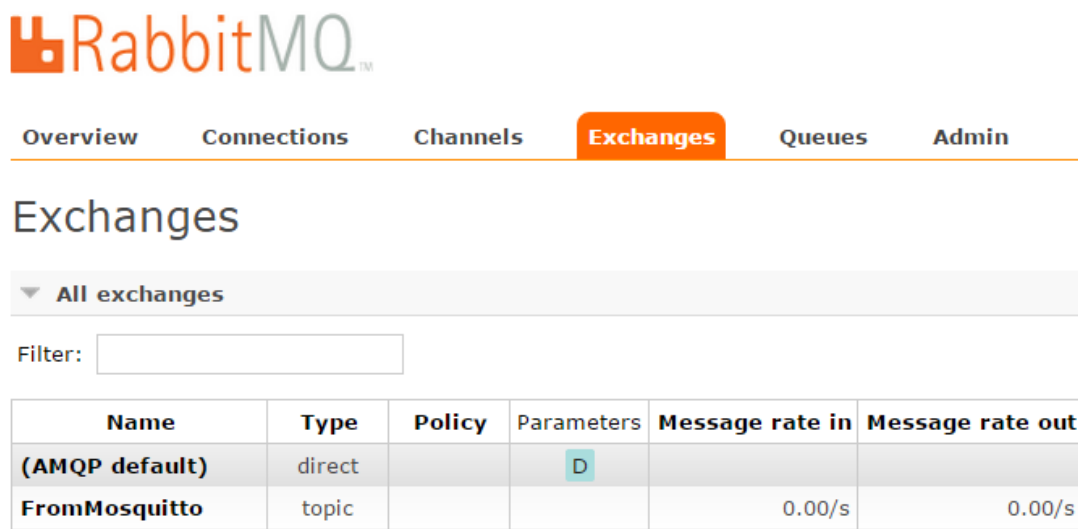


Figure 6: RabbitMQ HTTP-based API - Topic "FromMosquitto"

### 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.

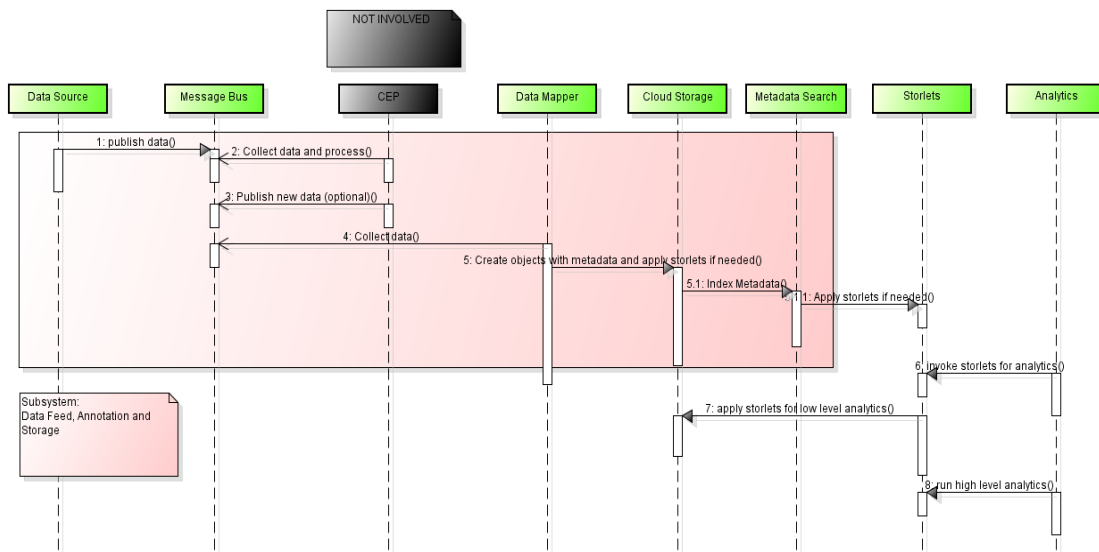


Figure 9: Sequence Diagram for Data Feed, Annotation and Storage Subsystem

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

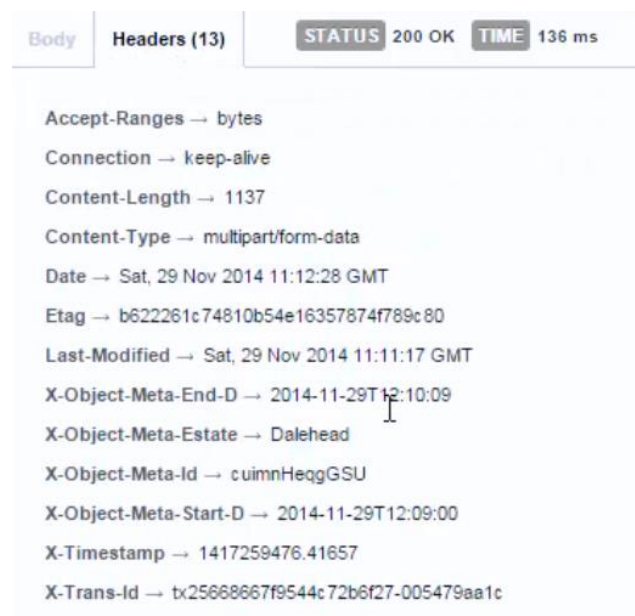
The following two figures depict the expected results that are written in the table above. Specifically, Figure 10 presents an aggregated JSON message that has been stored in the COSMOS Cloud Storage, whereas Figure 11 shows its metadata, fully aligned with the configuration file defined by the VE developer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



```
1 [{"estate": "Dalehead", "hid": "cuimnHeqgGSU", "ts": 1417259340, "instant": 3205, "returnTemp": 51.9, "flowTemp": 72.8, "flowRate": 133, "cumulative": 28724}, {"estate": "Dalehead", "hid": "cuimnHeqgGSU", "ts": 1417259350, "instant": 3206, "returnTemp": 51.9, "flowTemp": 73.0, "flowRate": 133, "cumulative": 28724}, {"estate": "Dalehead", "hid": "cuimnHeqgGSU", "ts": 1417259359, "instant": 3228, "returnTemp": 51.9, "flowTemp": 73.0, "flowRate": 133, "cumulative": 28724}, {"estate": "Dalehead", "hid": "cuimnHeqgGSU", "ts": 1417259370, "instant": 3209, "returnTemp": 52.0, "flowTemp": 72.9, "flowRate": 133, "cumulative": 28724}, {"estate": "Dalehead", "hid": "cuimnHeqgGSU", "ts": 1417259379, "instant": 3185, "returnTemp": 51.9, "flowTemp": 73.0, "flowRate": 132, "cumulative": 28724}, {"estate": "Dalehead", "hid": "cuimnHeqgGSU", "ts": 1417259389, "instant": 3259, "returnTemp": 52.0, "flowTemp": 73.1, "flowRate": 135, "cumulative": 28724}, {"estate": "Dalehead", "hid": "cuimnHeqgGSU", "ts": 1417259399, "instant": 3246, "returnTemp": 51.9, "flowTemp": 73.1, "flowRate": 134, "cumulative": 28724}, {"estate": "Dalehead", "hid": "cuimnHeqgGSU", "ts": 1417259409, "instant": 3202, "returnTemp": 51.9, "flowTemp": 73.1, "flowRate": 132, "cumulative": 28724}]]
```

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).



```
Accept-Ranges → bytes
Connection → keep-alive
Content-Length → 1137
Content-Type → multipart/form-data
Date → Sat, 29 Nov 2014 11:12:28 GMT
Etag → b622261c74810b54e16357874f789c80
Last-Modified → Sat, 29 Nov 2014 11:11:17 GMT
X-Object-Meta-End-D → 2014-11-29T12:10:09
X-Object-Meta-Estate → Dalehead
X-Object-Meta-Id → cuimnHeqgGSU
X-Object-Meta-Start-D → 2014-11-29T12:09:00
X-Timestamp → 1417259476.41657
X-Trans-Id → tx25668667f9544c72b6f27-005479aa1c
```

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.

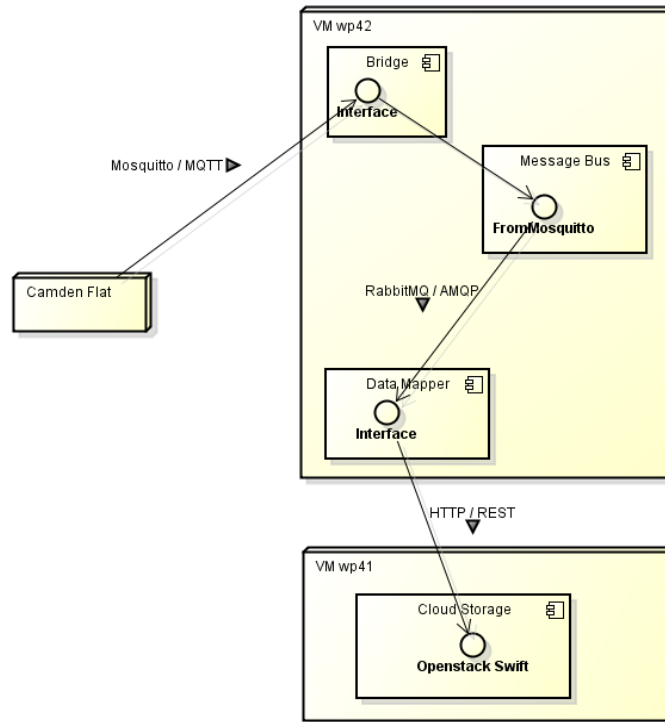


Figure 12: Data Feed, Annotation and Storage Deployment Diagram

### 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.

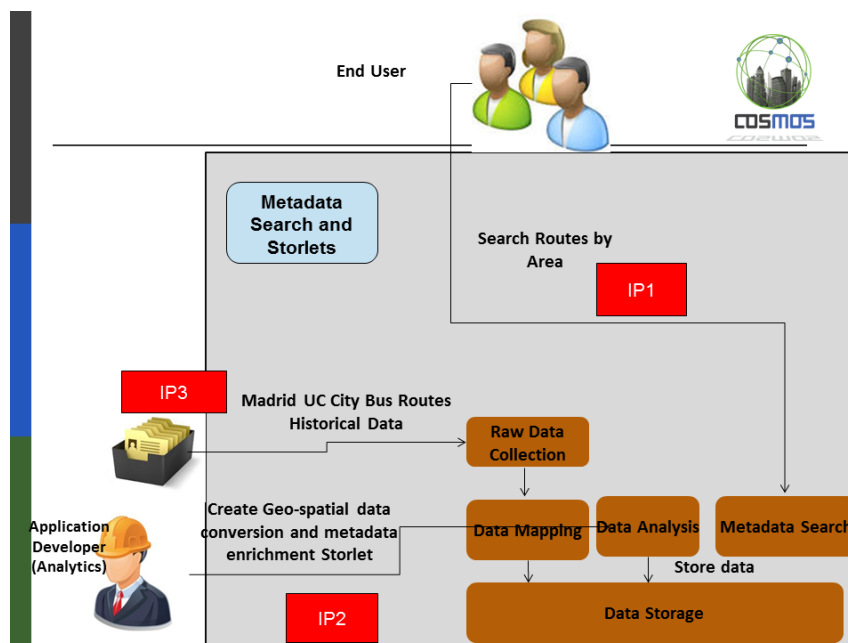


Figure 13: Metadata Search and Storlets Subsystem



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 communicate 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].

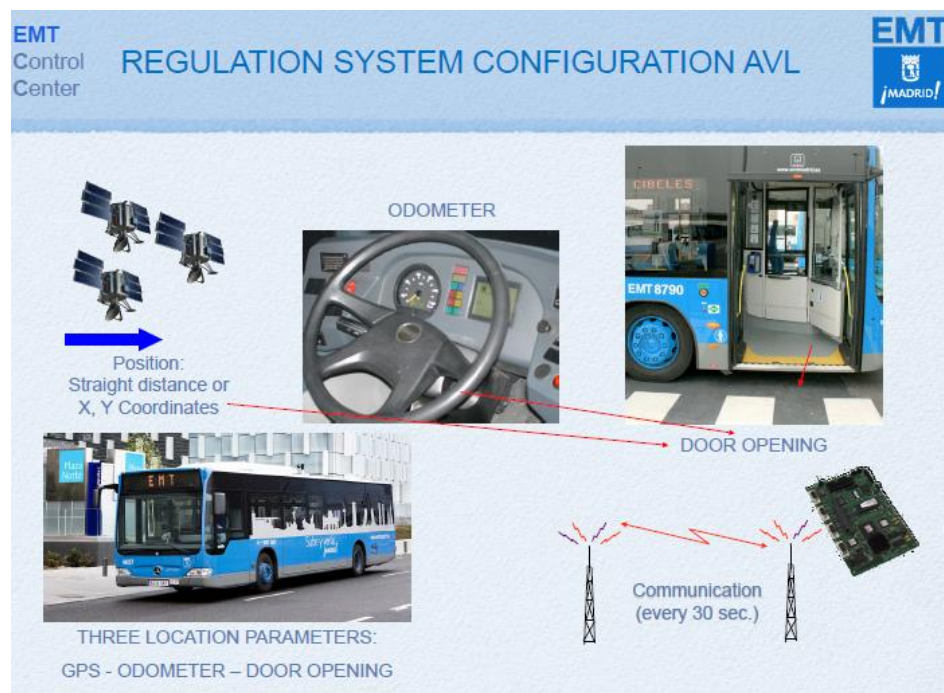


Figure 14: Overview of EMT available data characteristics

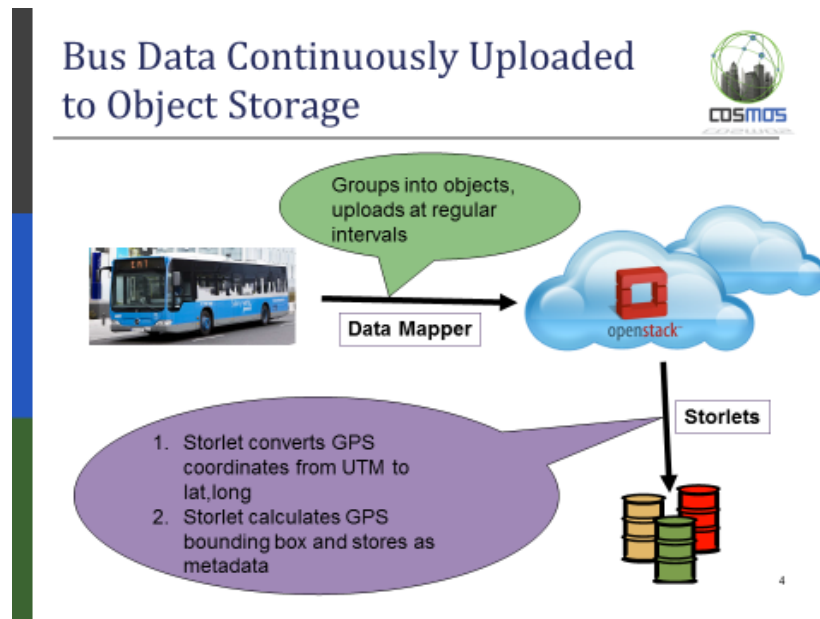


Figure 15: Overview of Metadata and Storlets Scenario

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.

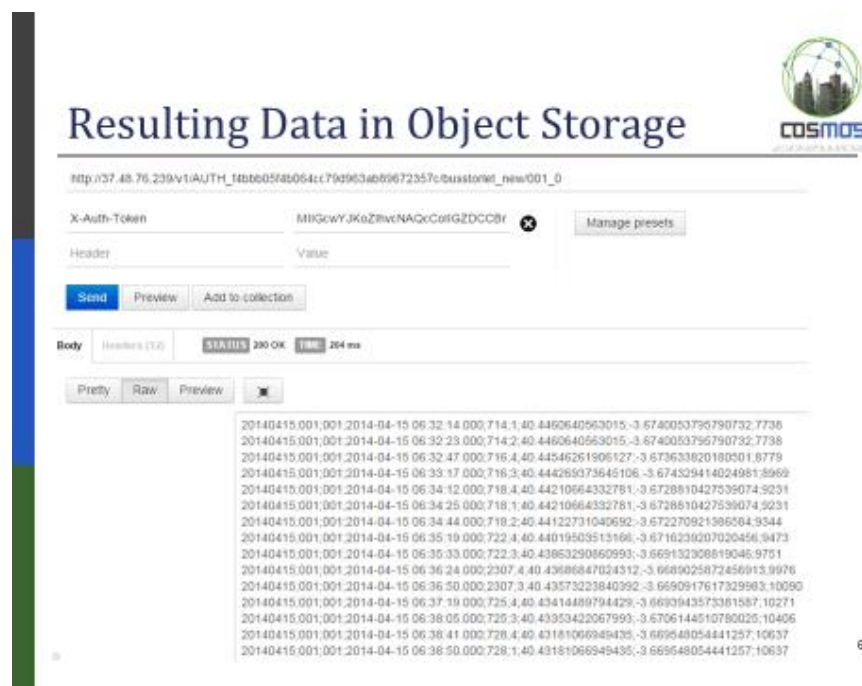


Figure 16: Data Format in Object Storage

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.

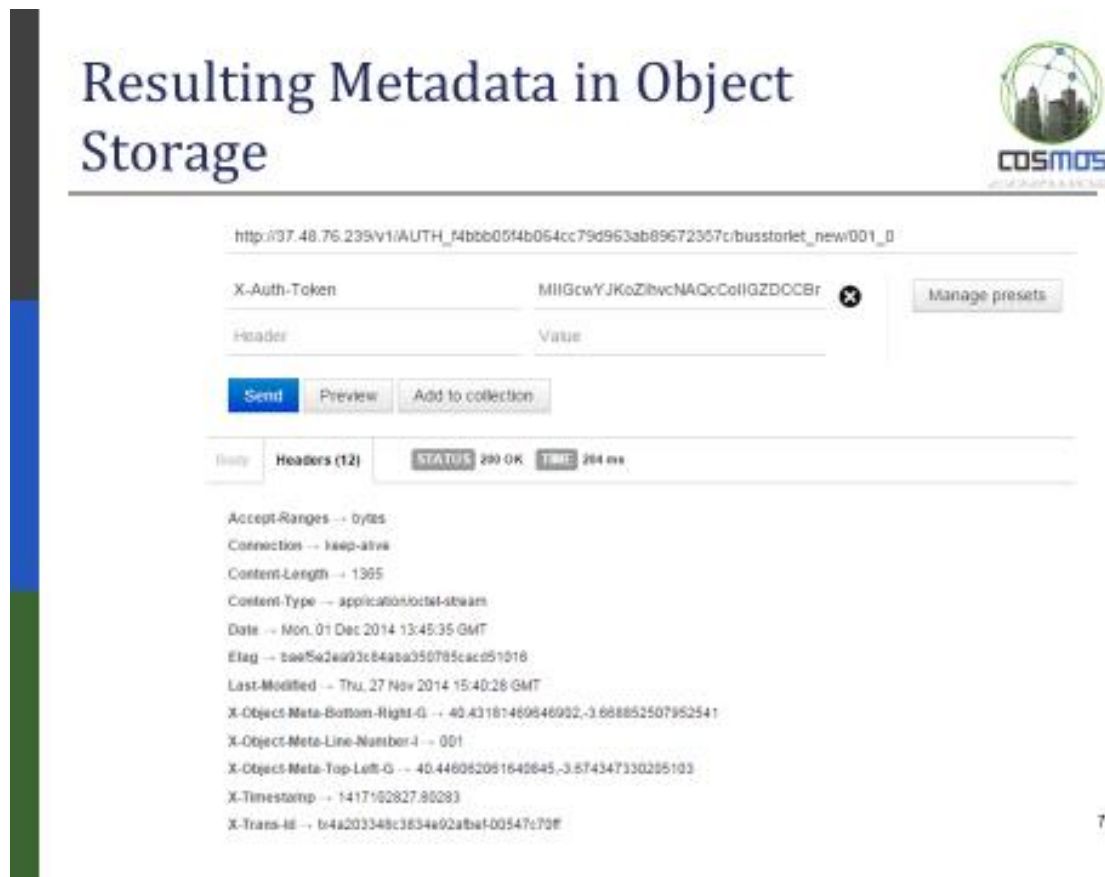


Figure 17: Metadata insertion in Object Storage

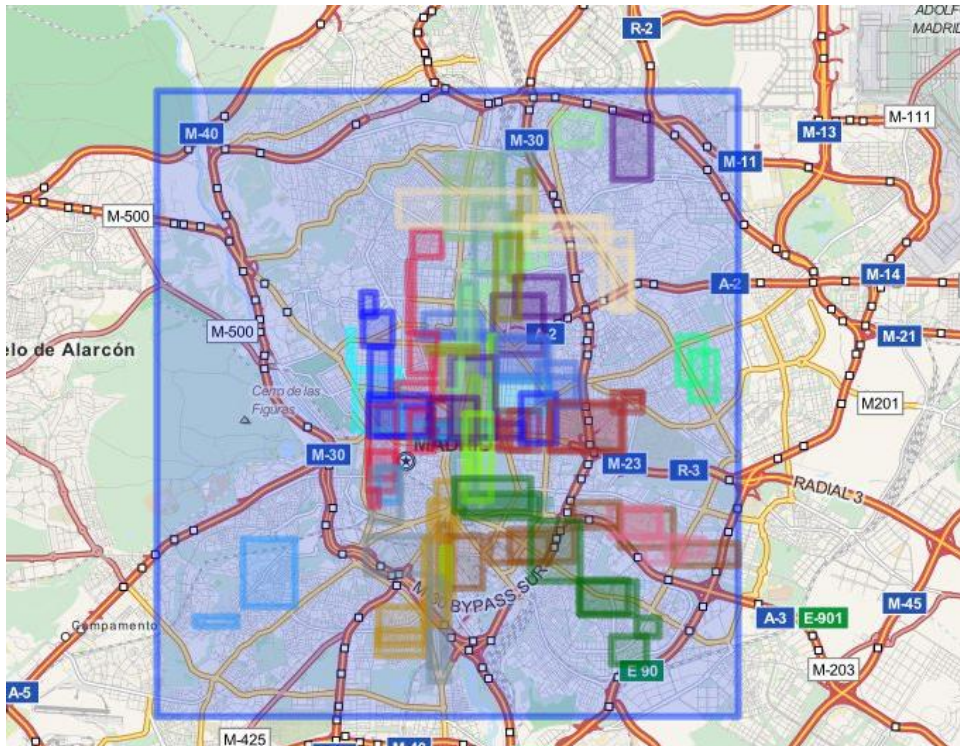


Figure 18: End user interface for the Metadata Search subsystem

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.

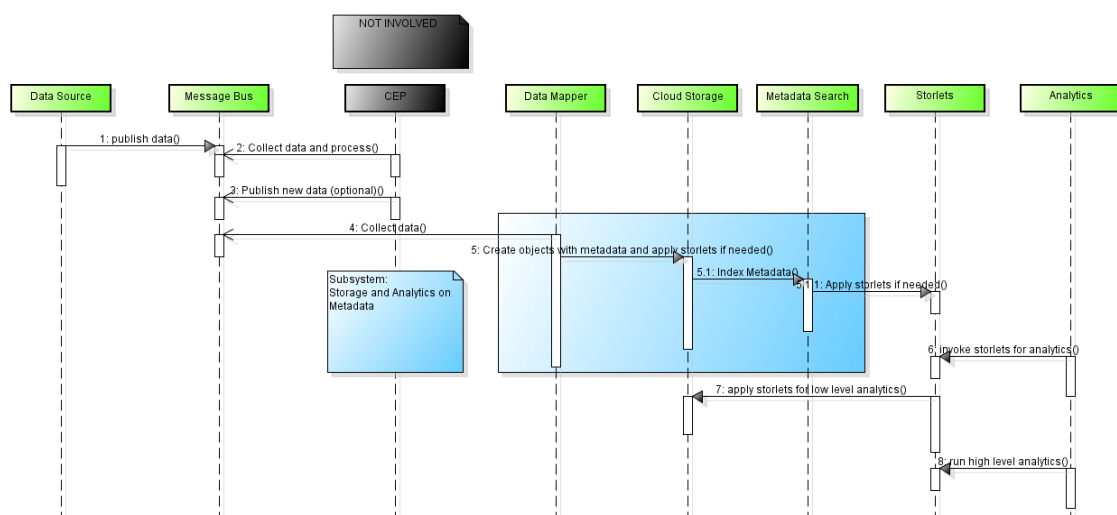


Figure 19: Sequence Diagram for Storage and Analytics on Metadata Subsystem



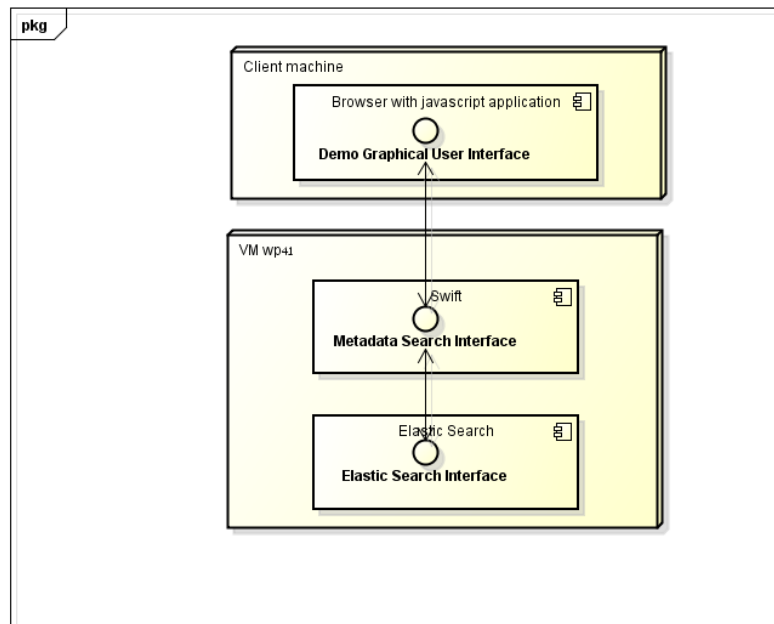


Figure 20: Deployment Diagram for the Metadata Search

### 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.

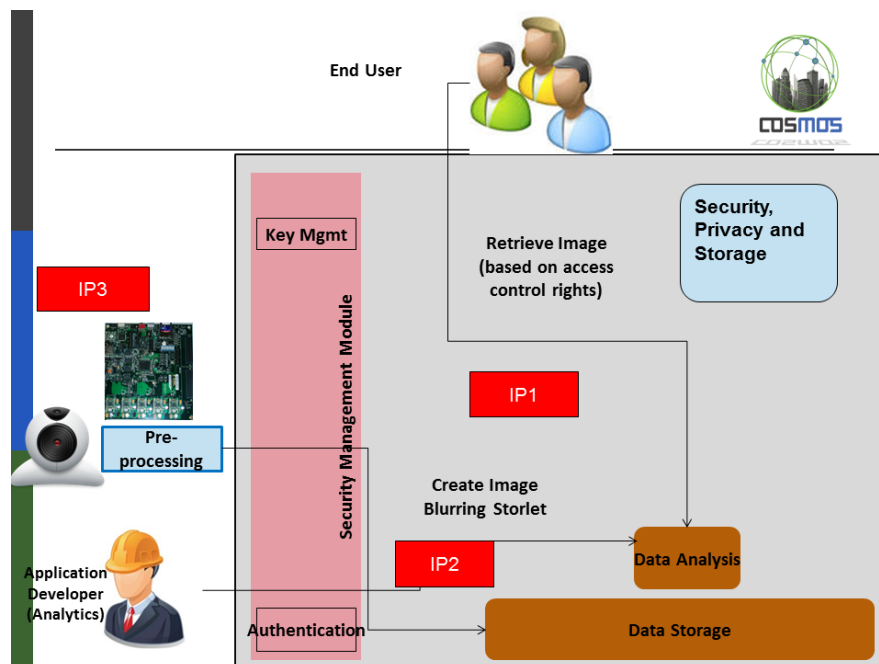


Figure 21: Security, Privacy and Storage

### 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.

### Demo: Passenger Photos Securely Uploaded to Cloud

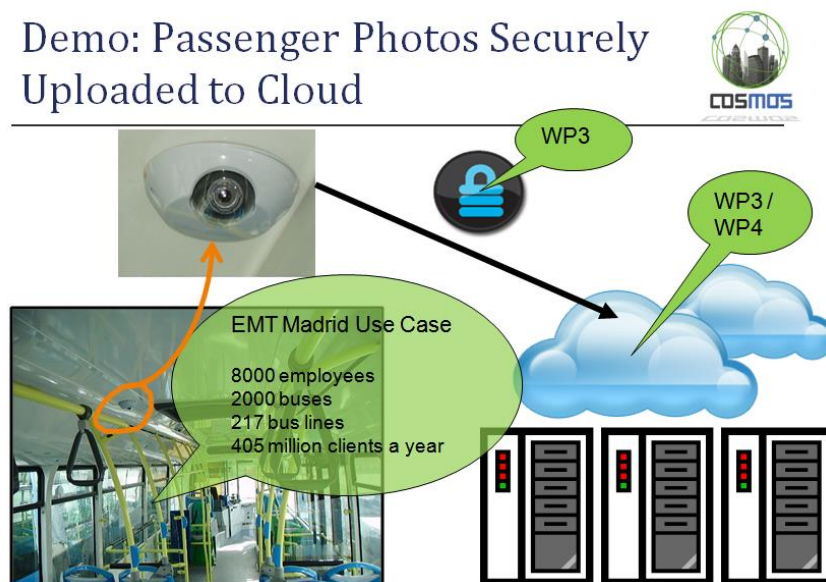


Figure 22: Security Demonstrator Show-Case

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.

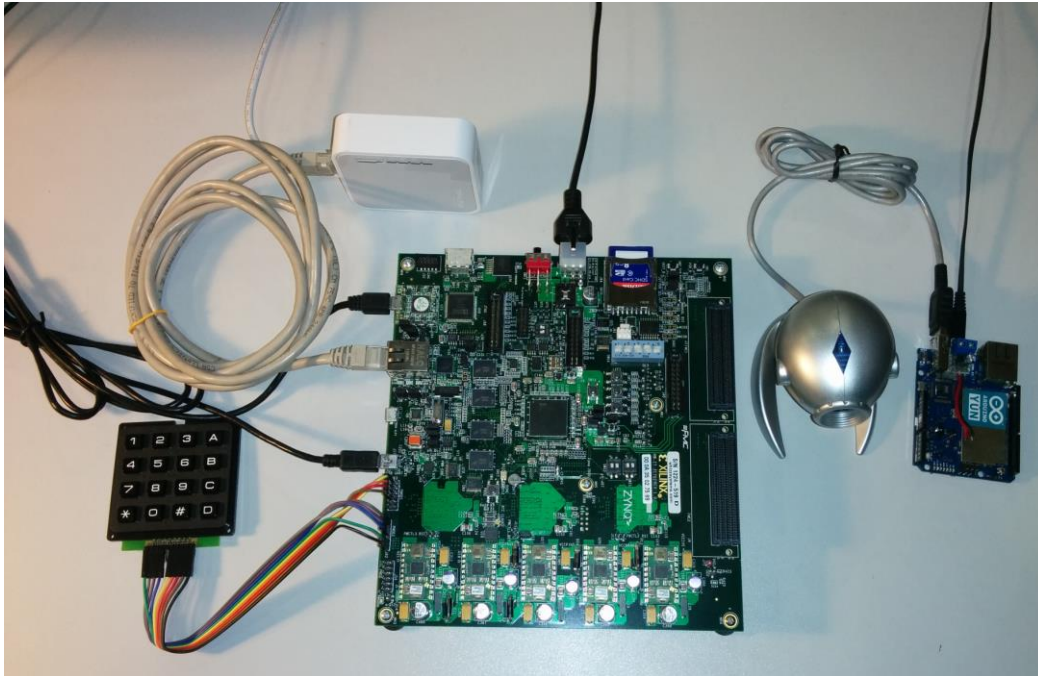


Figure 23: Hardware Security Board

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.

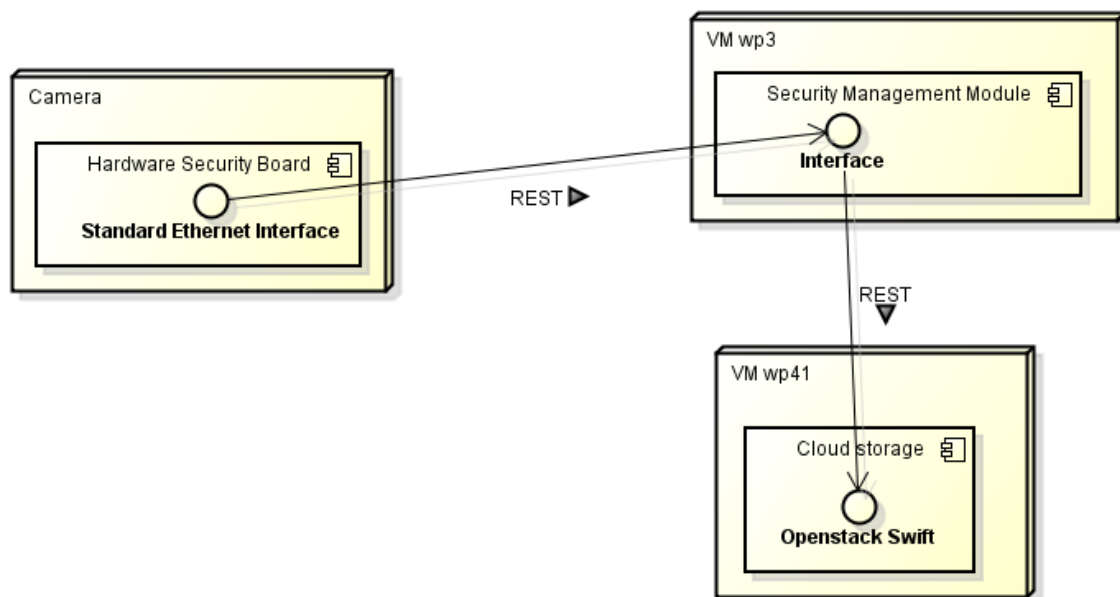


Figure 24: Security, Privacy and Storage Deployment Diagram

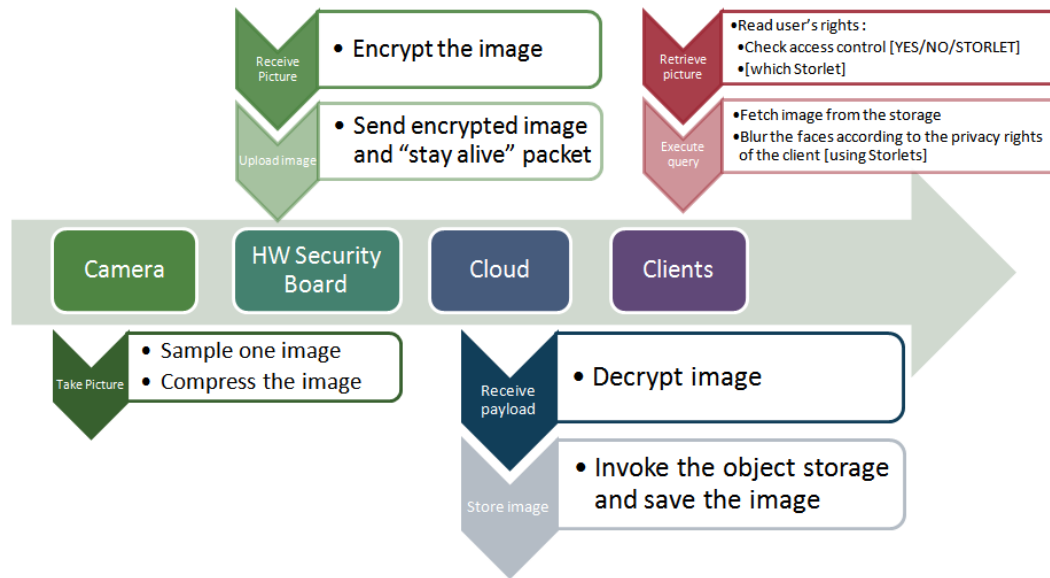


Figure 25: Demonstrator Flow

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

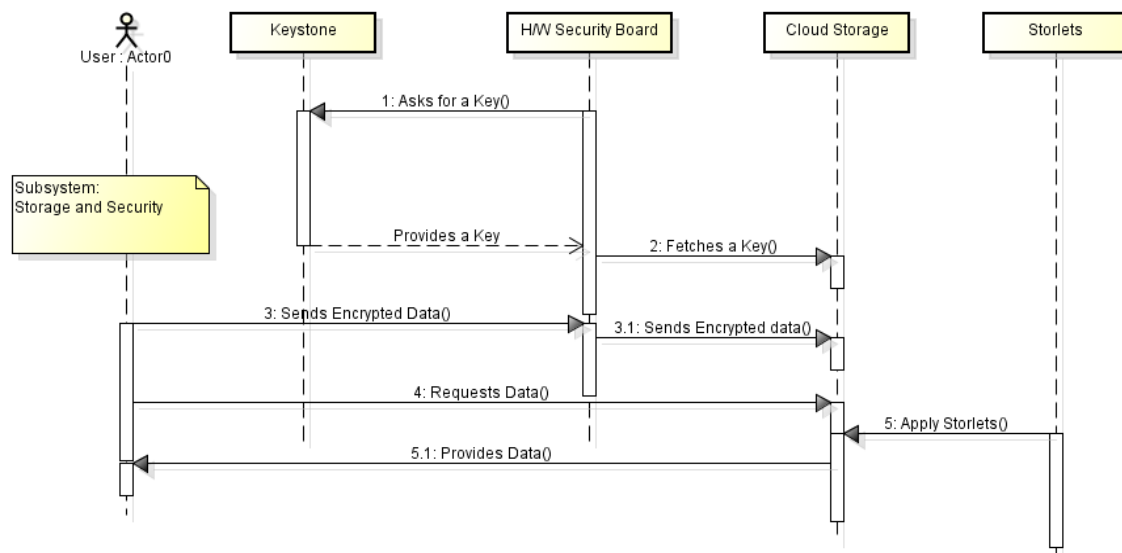


Figure 26: Sequence Diagram for Storage and Security Subsystem



```

Your group is currently "mkgroup". This indicates that neither
your gid nor your pgsid (primary group associated with your SID)
is in /etc/group.

The /etc/group (and possibly /etc/passwd) files should be rebuilt.
See the man pages for mkpasswd and mkgroup then, for example, run

mkpasswd -l [-d] > /etc/passwd
mkgroup -l [-d] > /etc/group

Note that the -d switch is necessary for domain users.

rolv01v9@MD15052C ~
$ python client.py
connecting to server...
taking picture...
saving picture...
encrypting using key: 2b7e151628aed2a6abf7158809cf4f3c
upload the encrypted picture...

```

Figure 27: Client side image encryption

```

wp3admin@wp3: ~
Using username "wp3admin".
wp3admin@37.48.76.238's password:
Access denied
wp3admin@37.48.76.238's password:
Welcome to Ubuntu 13.10 (GNU/Linux 3.11.0-26-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

System information as of Mon Dec  1 01:40:56 CET 2014

System load:  0.04               Processes:    73
Usage of /:   0.7% of 193.54GB    Users logged in:  0
Memory usage: 3%                 IP address for eth0: 37.48.76.238
Swap usage:   0%

Graph this data and manage this system at:
  https://landscape.canonical.com/

Last login: Mon Dec  1 01:40:57 2014 from 78.129.47.172
wp3admin@wp3:~$ python server.py
('deceiving data from:', ('78.129.47.172', 53402))
('decrypting data using key:', '2b7e151628aed2a6abf7158809cf4f3c')

```

Figure 28: Platform Gateway for receiving and decrypting encrypted image

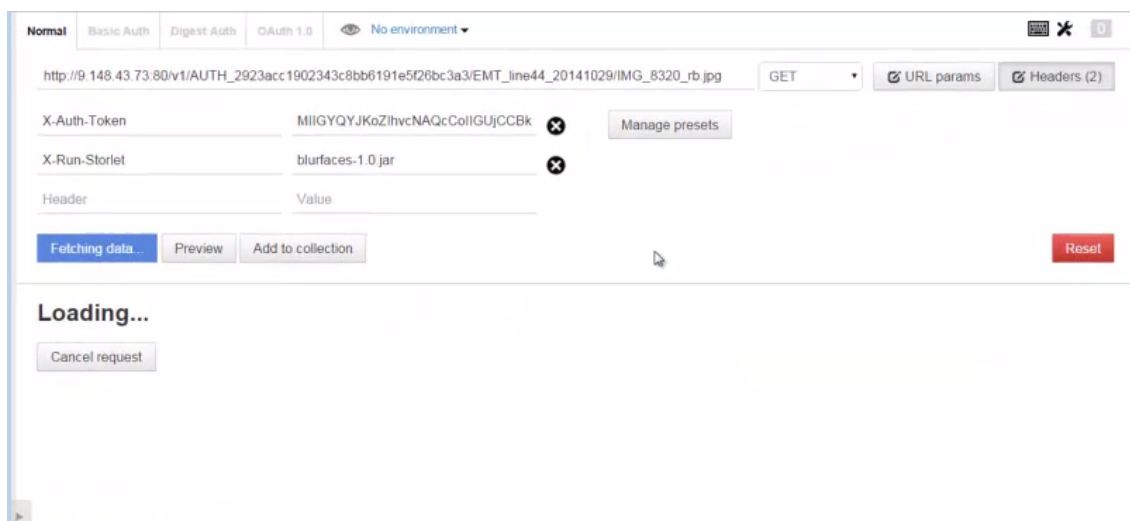


Figure 29: Facial Blurring Storlet Usage



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

### Integration of New Models

The diagram illustrates the Smart Platform Data Analytics Architecture, showing the interaction between two application developers and a central smart platform.

**Application Developer (Modeling):** Represented by a person wearing a hard hat and a suit. This developer interacts with the **Create Occupancy Detection Model** and the **Modeling** component within the Smart Platform.

**Application Developer (Analytics):** Represented by a person wearing a hard hat and a suit. This developer interacts with the **Smart Platform Historical Data**, **Raw Data Collection**, **Data Mapping**, and **Data Analysis** components within the Smart Platform.

**Smart Platform:** The central processing unit, represented by a large grey box. It contains several components and data flows:

- Modeling and Storage Analytics:** A blue box at the top left of the Smart Platform.
- Create Occupancy Detection Model:** A text label above the **Modeling** component, connected to the **Application Developer (Modeling)**.
- IP2:** A red box located between the **Create Occupancy Detection Model** and the **Modeling** component.
- Modeling:** An orange box at the top right of the Smart Platform, connected to the **Application Developer (Modeling)** and the **Use Preprocessing Storlet** component.
- Smart Platform Historical Data:** A text label above the **Raw Data Collection** component, connected to the **Application Developer (Analytics)**.
- Raw Data Collection:** An orange box in the middle of the Smart Platform, connected to the **Smart Platform Historical Data** and the **Data Mapping** component.
- Data Mapping:** An orange box in the middle of the Smart Platform, connected to the **Raw Data Collection** and the **Data Analysis** component.
- Data Analysis:** An orange box in the middle of the Smart Platform, connected to the **Data Mapping** and the **Get data** component.
- Use Preprocessing Storlet:** A text label above the **Data Analysis** component, connected to the **Modeling** component.
- Get data:** A text label above the **Data Storage** component, connected to the **Data Analysis** component.
- Data Storage:** A large orange box at the bottom of the Smart Platform, connected to the **Get data** component.
- Create Preprocessing Storlet:** A text label above the **IP2** box, connected to 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].

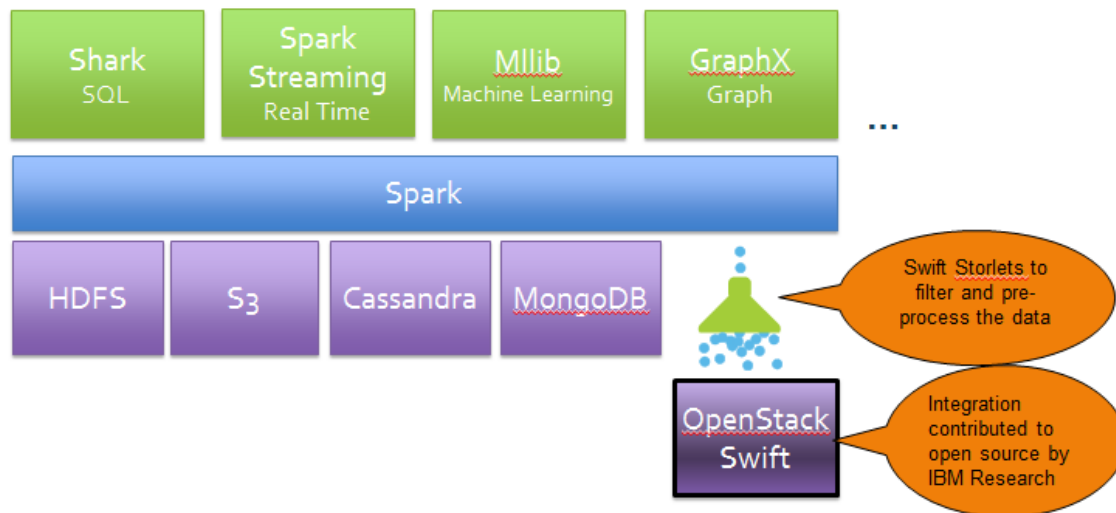


Figure 32: Overview of integration between Spark and Swift

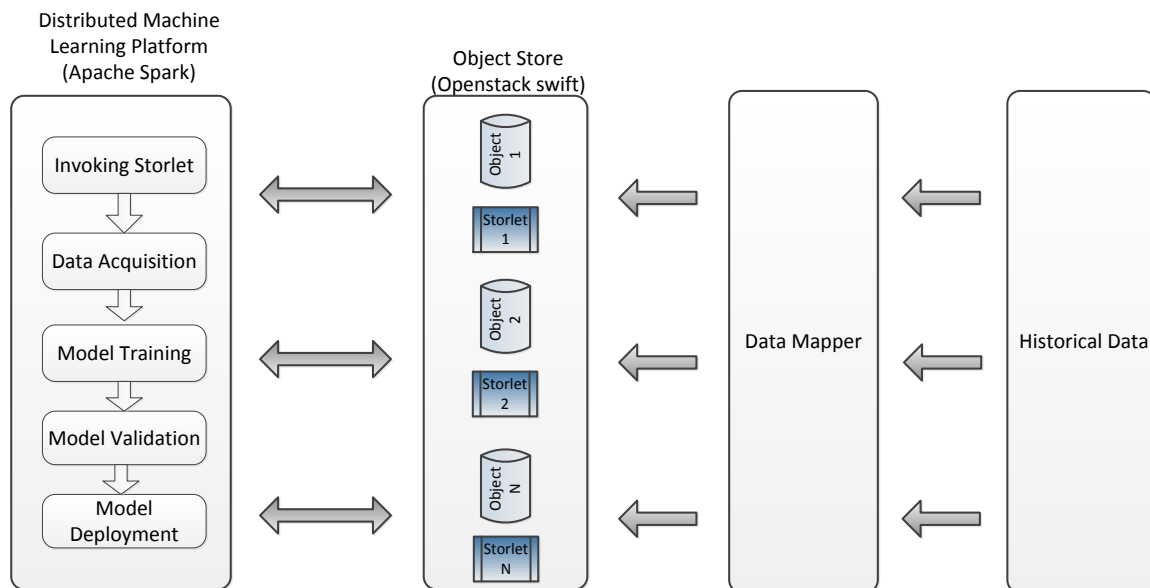


Figure 33: Data Flow across the Subsystem

The format of the data used appears in Figure 34 and contains information on the building 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.

```

1 node-id,time_stamp,PIR,mic,temp,light,watts,frequency,RMS_voltage,RMS_current,reactive_power,phase_angle
2 108,"2014-06-18 00:00:15",2,6,1230,19,1.24,50.0,238.9,0.1,147507.04,273.0
3 108,"2014-06-18 00:00:25",0,15,1229,3,1.24,50.1,239.1,0.1,147507.04,273.0
4 108,"2014-06-18 00:00:35",0,13,1224,14,1.34,50.1,239.1,0.1,147506.94,273.0
5 108,"2014-06-18 00:00:45",3,2,1223,11,1.34,50.0,239.1,0.1,147507.04,273.0
6 108,"2014-06-18 00:00:55",0,0,1229,10,1.24,50.0,238.7,0.1,147506.94,273.0
7 108,"2014-06-18 00:01:05",1,13,1220,14,1.13,50.0,238.5,0.1,147506.94,273.0
8 108,"2014-06-18 00:01:15",2,6,1225,10,1.03,50.0,238.5,0.1,147506.94,273.0
9 108,"2014-06-18 00:01:25",0,0,1220,19,1.13,50.0,238.5,0.1,147506.83,273.0
10 108,"2014-06-18 00:01:35",1,5,1228,8,1.24,50.0,238.4,0.1,147506.83,273.0
11 108,"2014-06-18 00:01:45",2,2,1221,13,1.34,50.0,238.6,0.1,147506.94,273.0
12 108,"2014-06-18 00:01:55",1,2,1218,8,1.24,50.0,238.7,0.1,147506.94,273.0
13 108,"2014-06-18 00:02:05",1,2,1229,10,1.24,50.0,238.6,0.1,147506.83,273.0
14 108,"2014-06-18 00:02:15",1,5,1223,9,1.13,50.0,238.7,0.1,147506.94,273.0
15 108,"2014-06-18 00:02:25",2,18,1222,13,1.03,50.0,238.8,0.1,147506.94,273.0
16 108,"2014-06-18 00:02:35",1,16,1235,15,1.13,50.0,238.4,0.1,147506.83,273.0
17 108,"2014-06-18 00:02:45",1,1,1216,11,1.24,50.0,238.4,0.1,147506.83,273.0
18 108,"2014-06-18 00:02:55",1,7,1223,11,1.34,50.0,238.6,0.1,147506.94,273.0
19 108,"2014-06-18 00:03:05",2,1,1224,11,1.24,50.0,238.7,0.1,147506.94,273.0
20 108,"2014-06-18 00:03:15",1,6,1226,13,1.24,50.0,238.7,0.1,147506.94,273.0
21 108,"2014-06-18 00:03:25",1,1,1217,11,1.03,50.0,238.7,0.1,147506.94,273.0
22 108,"2014-06-18 00:03:35",2,11,1222,0,1.13,50.0,238.7,0.1,147506.94,273.0
23 108,"2014-06-18 00:03:45",1,9,1231,11,1.13,50.0,238.7,0.1,147506.94,273.0
24 108,"2014-06-18 00:03:55",0,1,1227,11,1.24,50.0,238.9,0.1,147506.94,273.0
25 108,"2014-06-18 00:04:05",1,5,1214,10,1.24,50.0,238.8,0.1,147506.94,273.0
26 108,"2014-06-18 00:04:15",1,2,1221,14,1.34,50.0,238.9,0.1,147506.94,273.0
27 108,"2014-06-18 00:04:25",1,5,1227,15,1.24,50.0,238.9,0.1,147506.94,273.0
28 108,"2014-06-18 00:04:35",0,4,1218,9,1.13,50.0,238.9,0.1,147506.94,273.0
29 108,"2014-06-18 00:04:45",1,17,1229,12,1.13,50.0,238.9,0.1,147506.94,273.0
30 108,"2014-06-18 00:04:55",1,1,1222,12,1.13,50.0,238.9,0.1,147507.04,273.0
31 108,"2014-06-18 00:05:05",2,15,1228,13,1.24,50.0,238.9,0.1,147506.94,273.0
32 108,"2014-06-18 00:05:15",1,15,1227,14,1.24,50.0,239.0,0.1,147506.94,273.0
33 108,"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

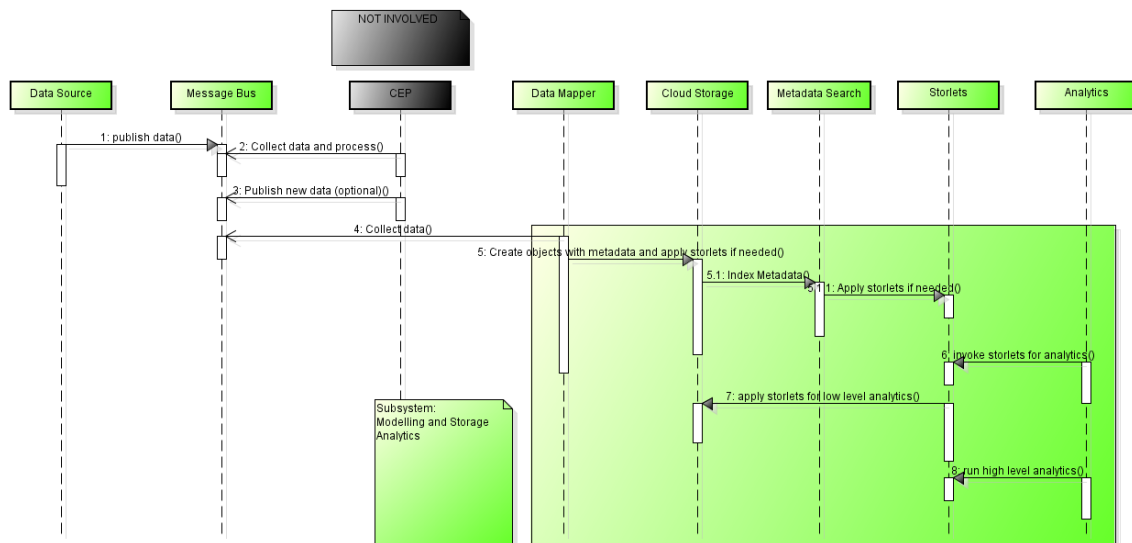


Figure 37: Sequence Diagram for Modelling and Storage Analytics Subsystem

### 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	<ol style="list-style-type: none"> <li>1) Data will be stored in the object storage in the form of objects</li> <li>2) Storlets will be provoked and it will perform aggregation as a pre-processing step</li> <li>3) Algorithms implemented in Apache Spark will access the pre-processed data from the object storage and machine learning models will be trained</li> <li>4) The trained model will be validated using validation data set</li> <li>5) The model will be deployed on real time data and it will detect the occupancy state of a user</li> </ol>
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

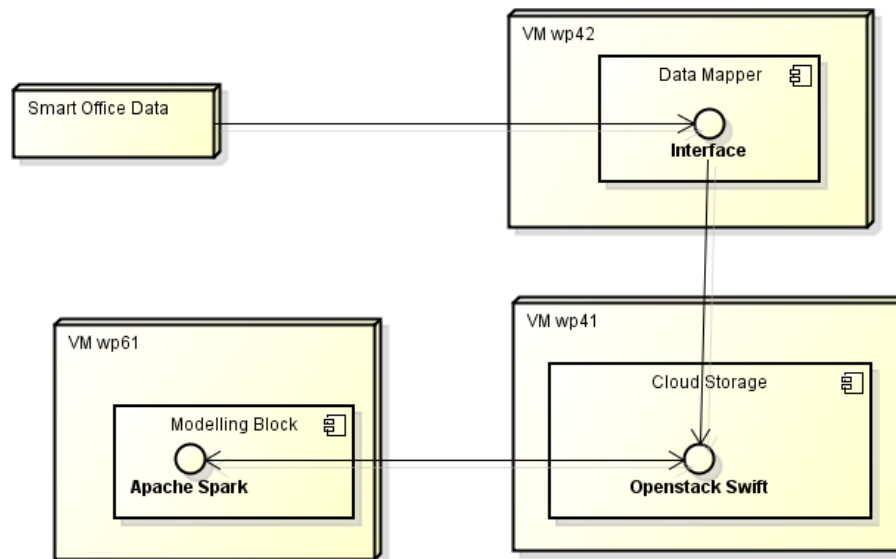


Figure 38: Modelling and Storage Analytics Deployment Diagram

### 2.2.2. Subgoal: Autonomous Behaviour of VEs

The subgoal includes the following components:

#### Message Bus

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

#### CEP

The CEP 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 CEP, 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 CEP and after receiving data for Case creation (simulating local observations), the event is triggered by user input, which corresponds 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.



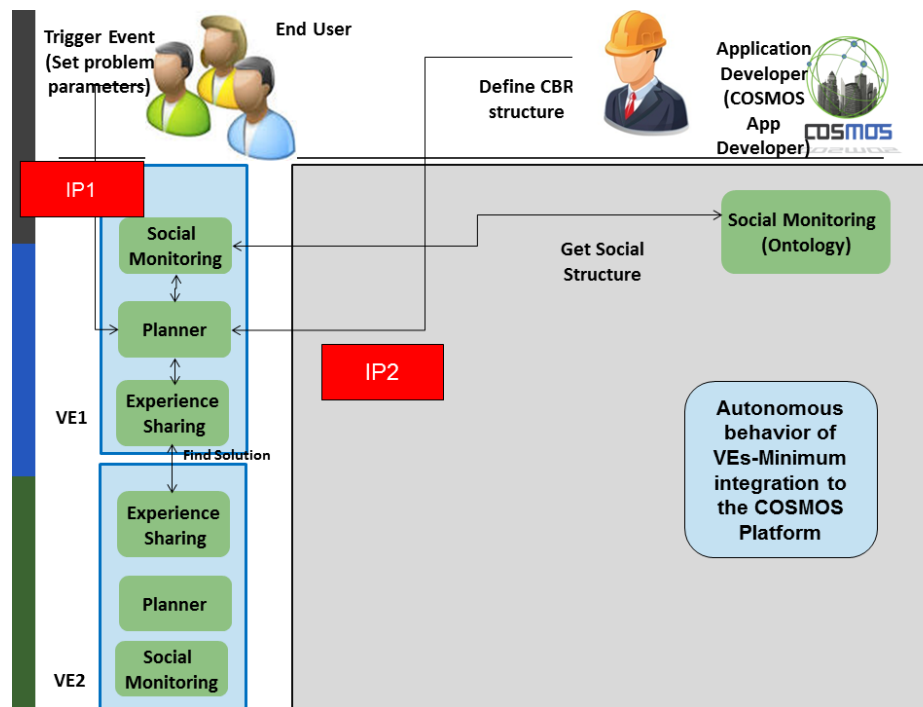


Figure 39: Autonomous Behavior of VEs with minimum platform integration subsystem

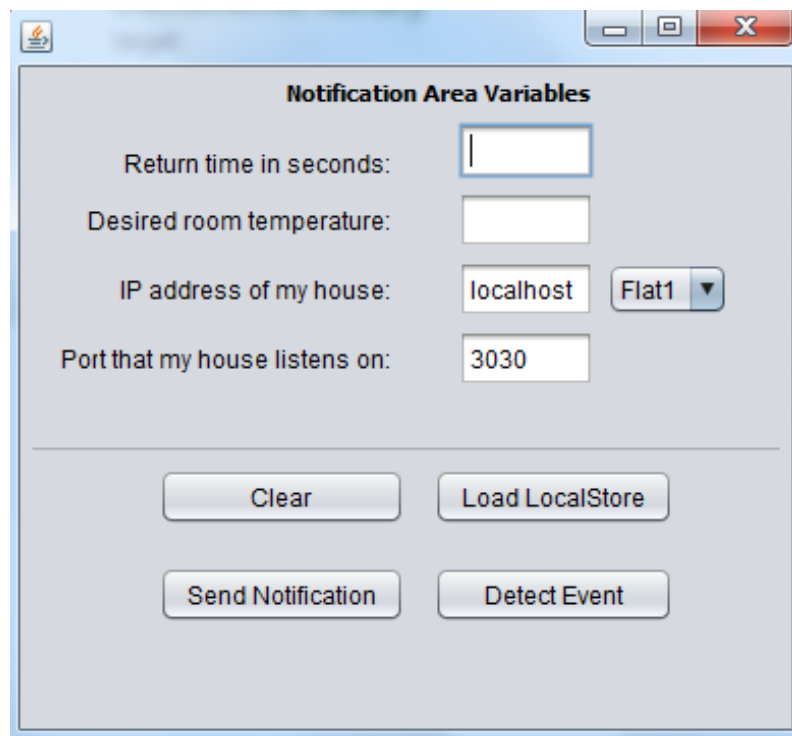


Figure 40: Autonomous Behavior of VEs application GUI

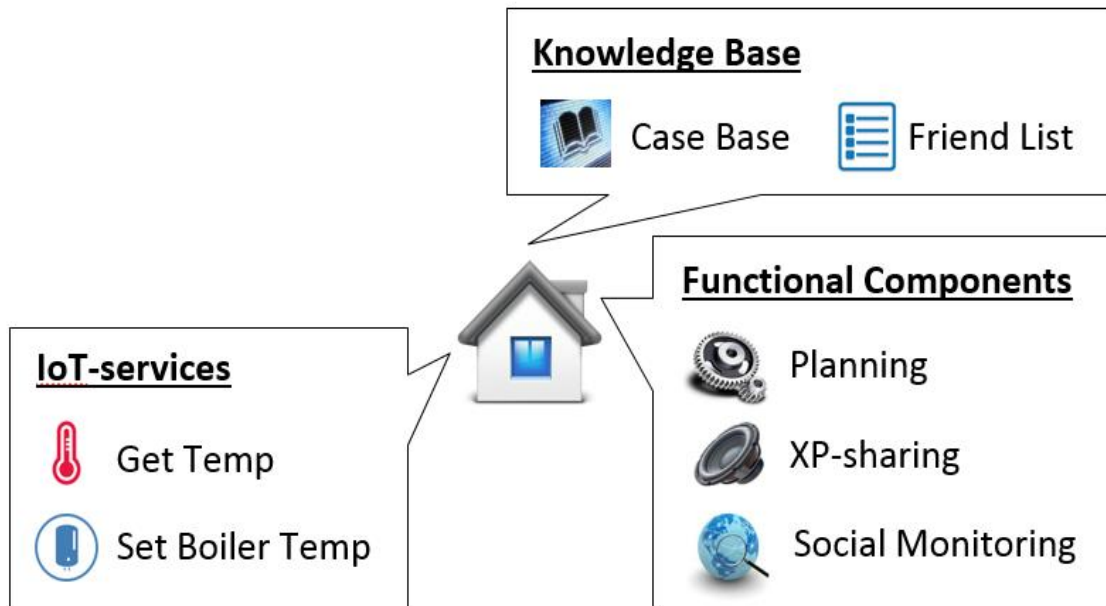


Figure 41: Components of the main flat-VE

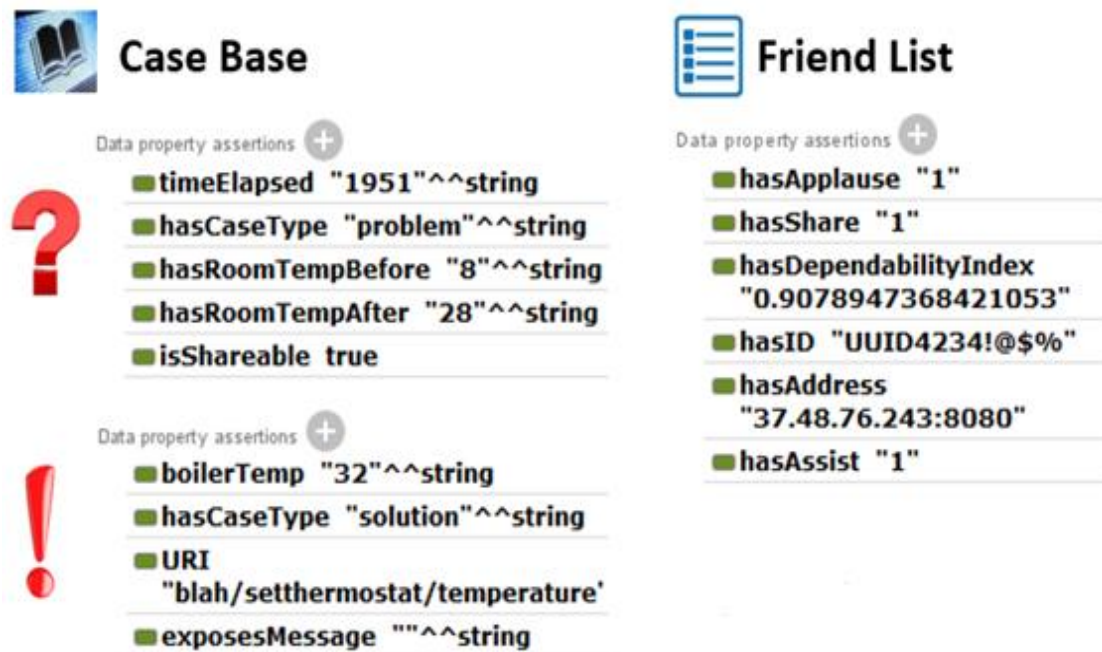


Figure 42: Case Base and Friend List examples

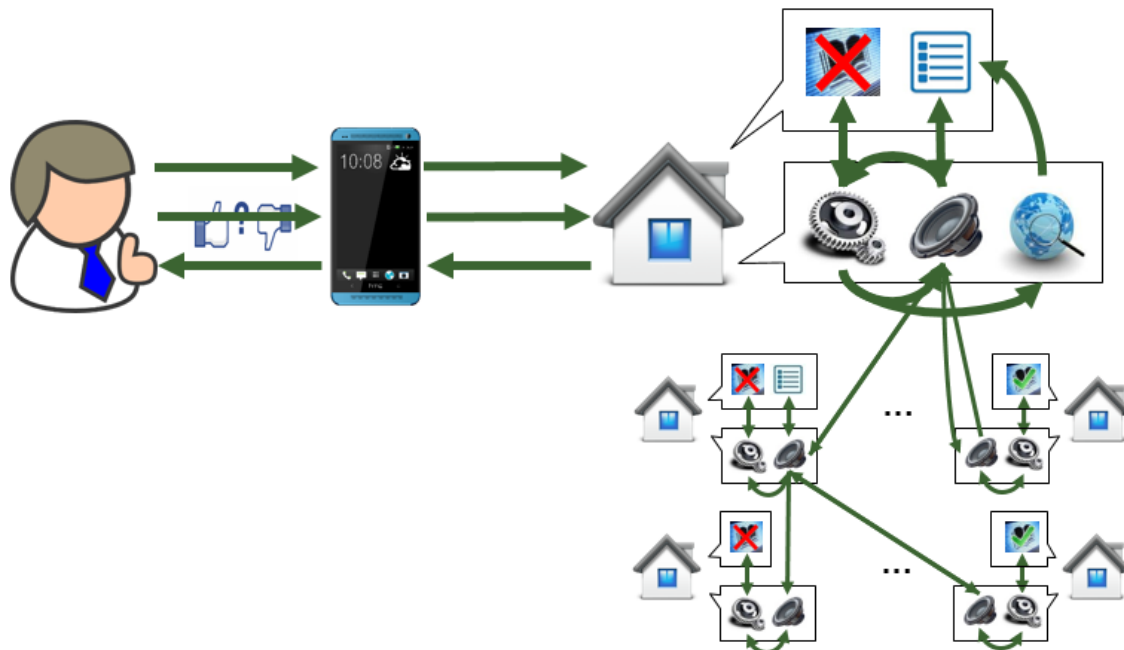


Figure 43: Visual representation of all scenario possibilities

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

<b>Test Case Number Version</b>	<b>AB_1</b>
<b>Test Case Title</b>	Reaction to User Generated Event
<b>Modules tested</b>	Planner, Experience Sharing, Social Monitoring
<b>Requirements addressed</b>	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
<b>Initial conditions</b>	Have a CB and appropriate Friend lists inside the test bed Have the VE and application simulation jars inside the test bed
<b>Expected results</b>	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
<b>Owner/Role</b>	End User, Application Developer
<b>Steps</b>	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
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	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 CEP 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.

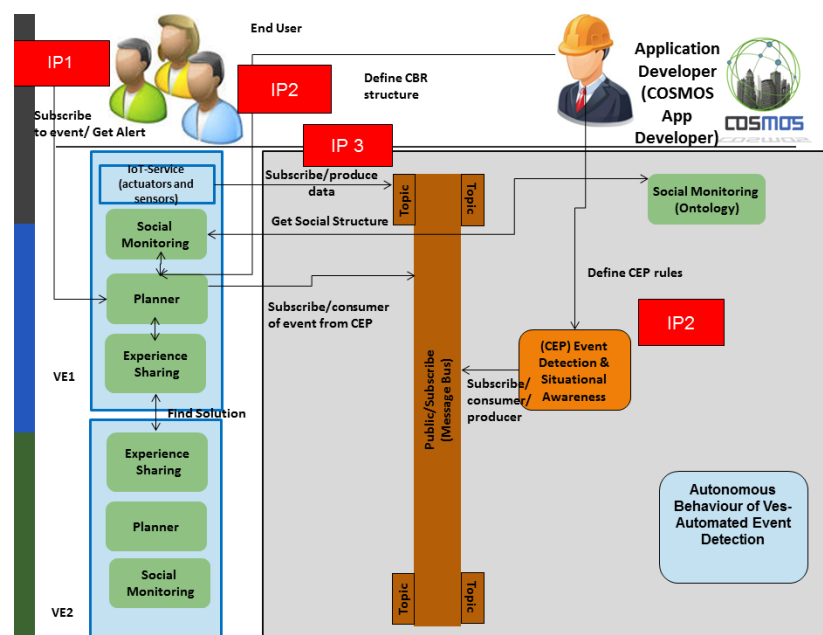


Figure 44: Autonomous Behavior of VEs- Automated Event Detection and Incorporation of COSMOS Platform

Many applications have to run continuously and without the intervention of the End-Users. Based on certain rules, the COSMOS CEP 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:

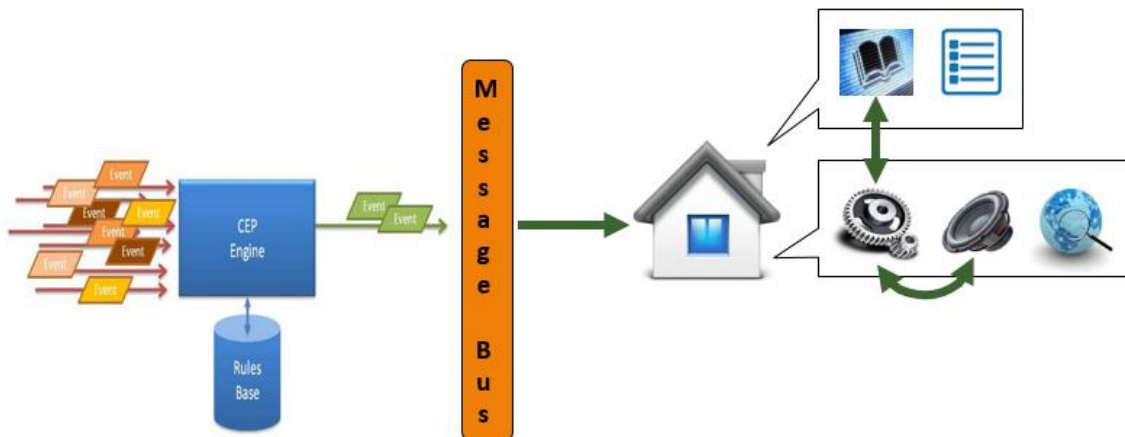


Figure 45: Visual representation of the platform integration scenario

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 CEP. In that case it is necessary to provide a way for the developer to inject detection rules, in the CEP. Right now, the only way to add new rules to CEP is by updating the rule's file which is coded in DOLCE.

Figure 46 shows an example of a DOLCE configuration file.

```

10 external int TEMP_ALERT = 39;
11 external duration T_WINDOW = 10 seconds;
12
13 /*
14     System Test 1:
15     Event filtration by constant value (sensor source)
16 */
17
18 event temperature
19 {
20     use
21     {
22         int sensor_id,
23         int value
24     };
25
26     accept { sensor_id == 253 };
27 }
28
29
30 complex freezeThreshold
31 {
32     detect temperature
33     where value < 4;
34 }

```

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 CEP, the VM must be accessed directly and the CEP 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 1 temperature int sensor_id 253 int value 3

```

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

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 messages
channel.exchangeDeclare(exchangeName, "topic"); //there are a few exchange types
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**

<b>Test Case Number Version</b>	<b>AB_2</b>
<b>Test Case Title</b>	Reaction to CEP generated event
<b>Module tested</b>	CEP, Message Bus, Planner, Experience Sharing, Social Monitoring
<b>Requirements addressed</b>	Same as <b>AB_1</b> 5.20
<b>Initial conditions</b>	Have a CB and appropriate Friend lists inside the test bed Have the VE and application simulation jars inside the test bed A functional CEP in the test bed A functional Message Bus
<b>Expected results</b>	The answer to the CEP 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
<b>Owner/Role</b>	End User, Application Developer, VE developer
<b>Steps</b>	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 CEP Grade the Solution as helpful or unhelpful if needed
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	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.

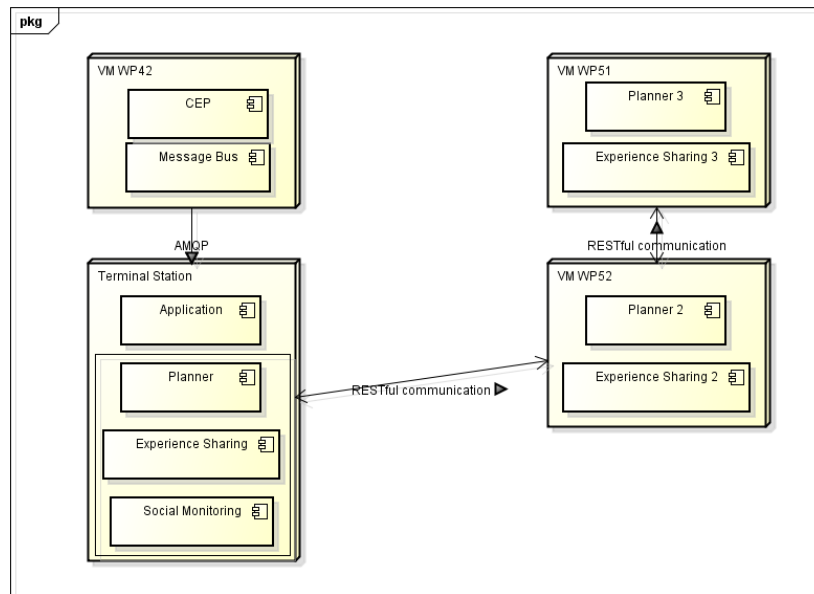


Figure 49: Autonomous Behaviour of VEs Deployment Diagram

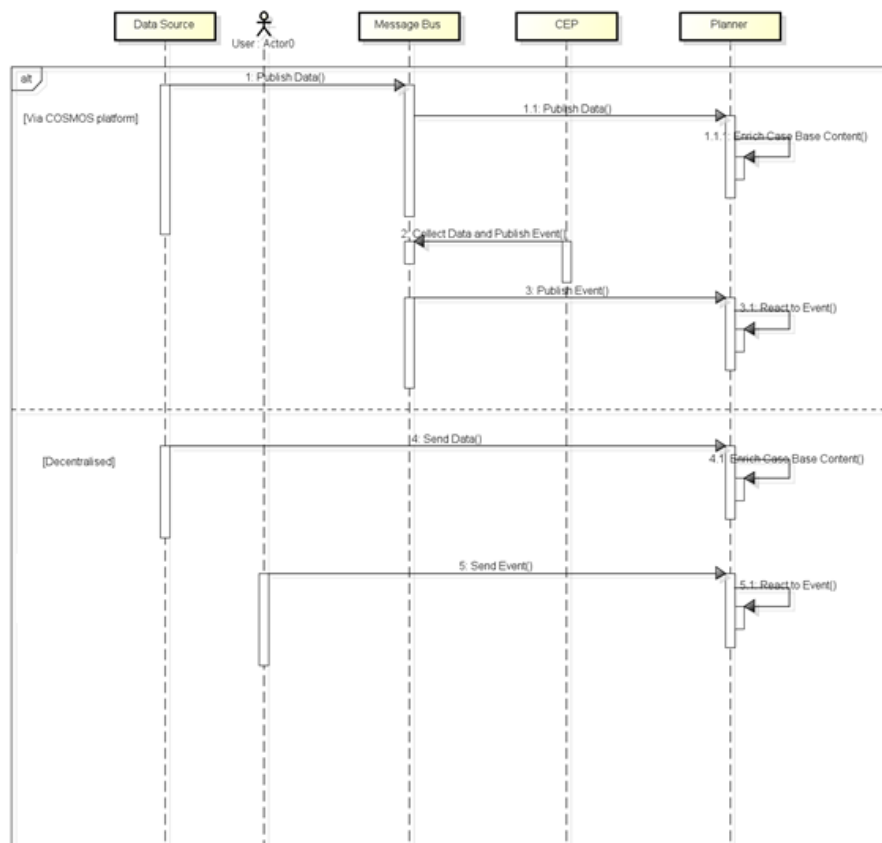


Figure 50: Autonomous Behaviour of VEs System Sequence Diagram



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

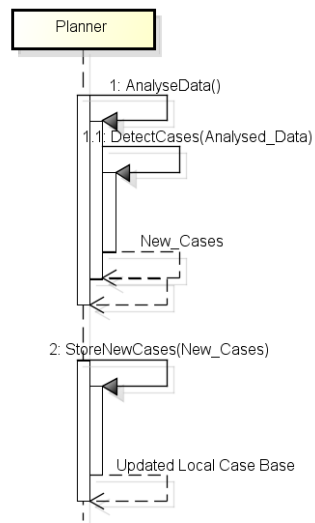


Figure 51: Enrich Case Base Content Sequence Diagram

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

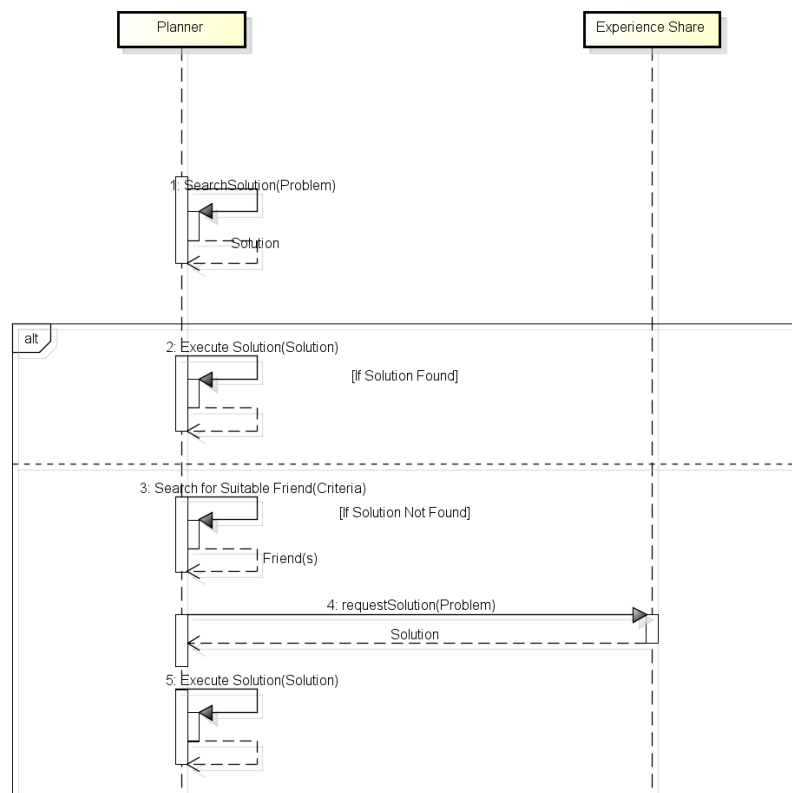


Figure 52: React to Event Sequence Diagram

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

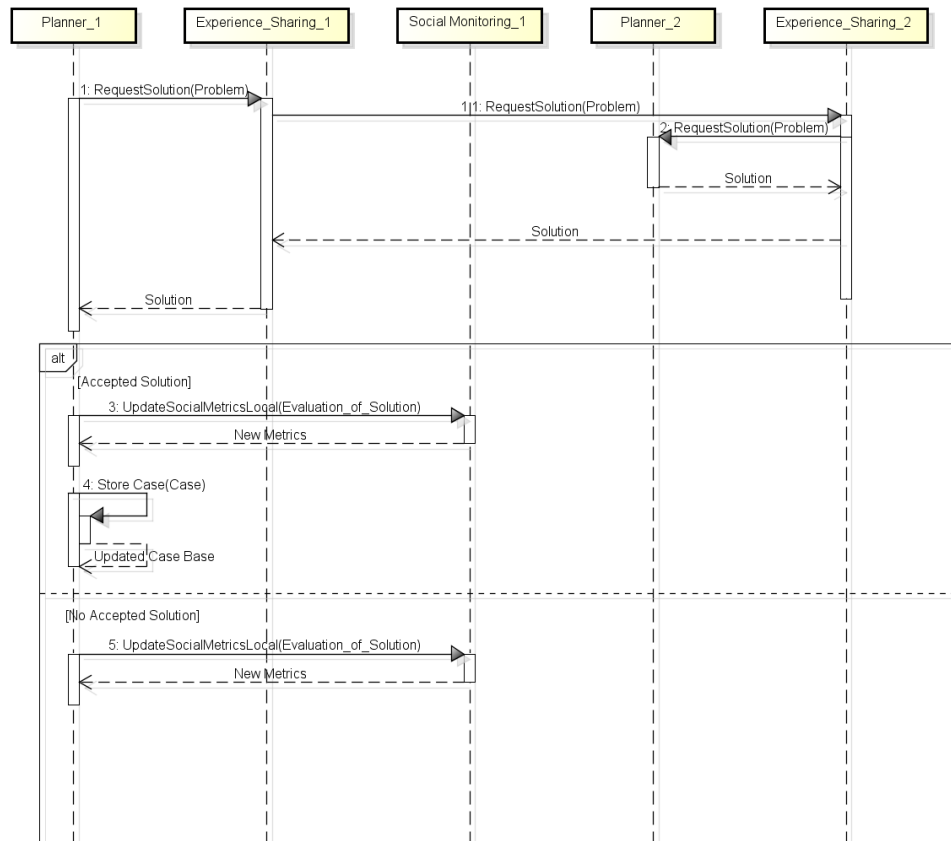


Figure 53: RequestSolution(Problem) Sequence Diagram

UpdateSocialMetricsLocal(Evaluation\_of\_Solution) in Figure 53 is detailed in Figure 54:

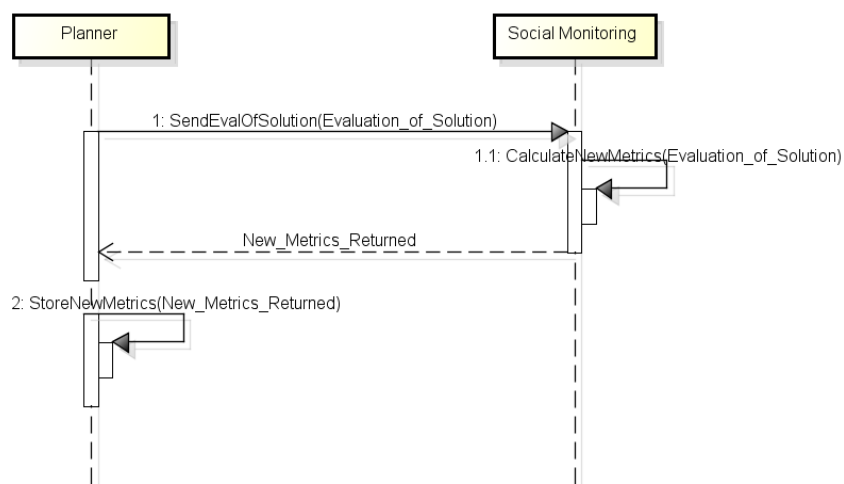


Figure 54: UpdateSocialMetricsLocal(Evaluation\_of\_Solution) Sequence Diagram

### 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.

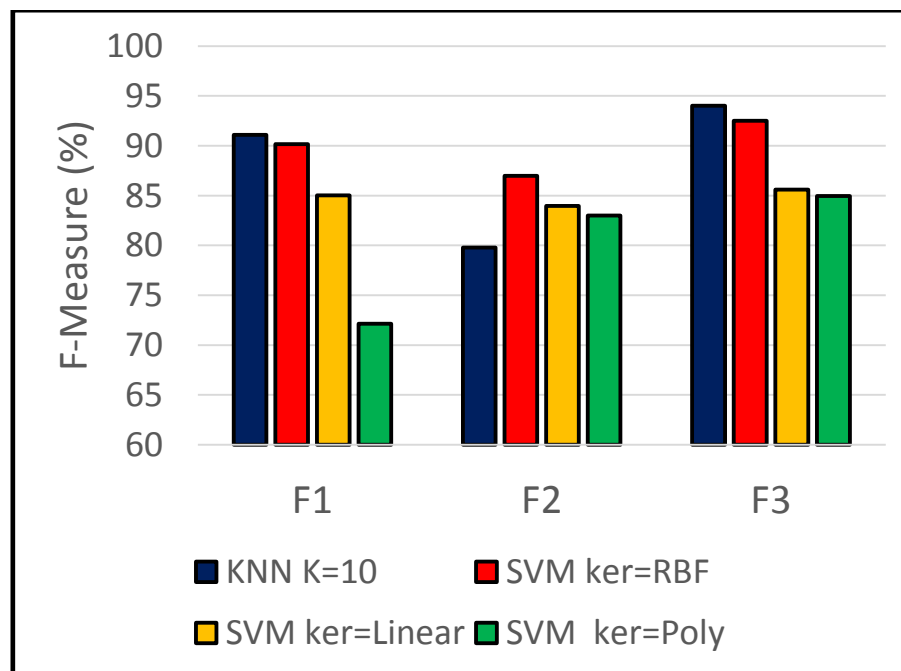


Figure 55: Performance of Classifiers

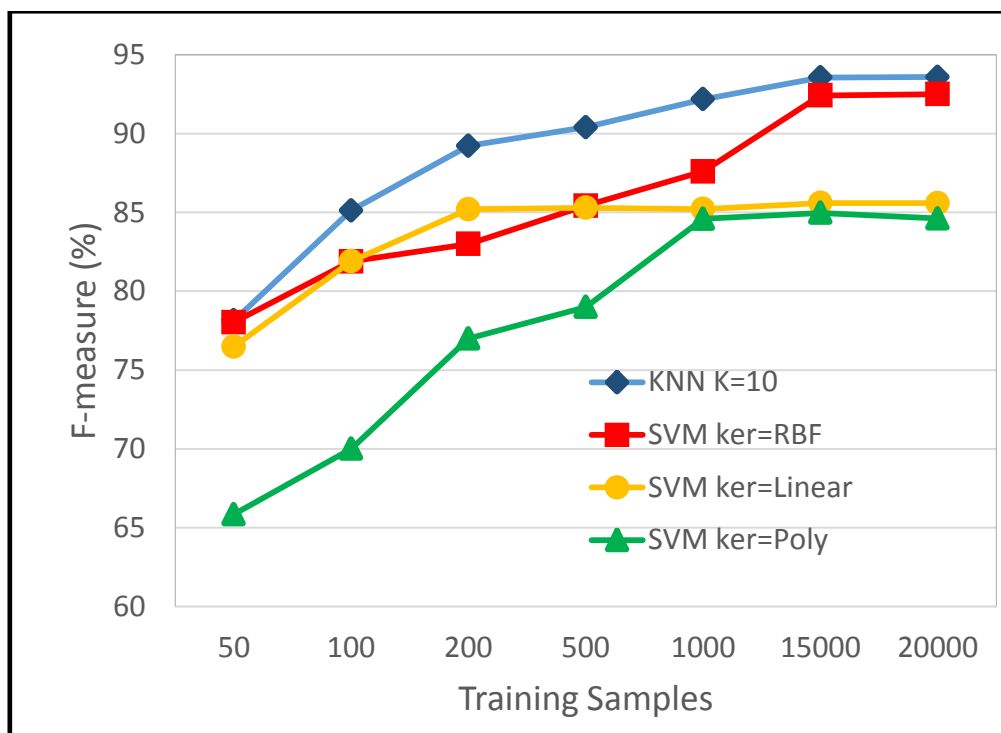


Figure 56: F-measure relation with training data

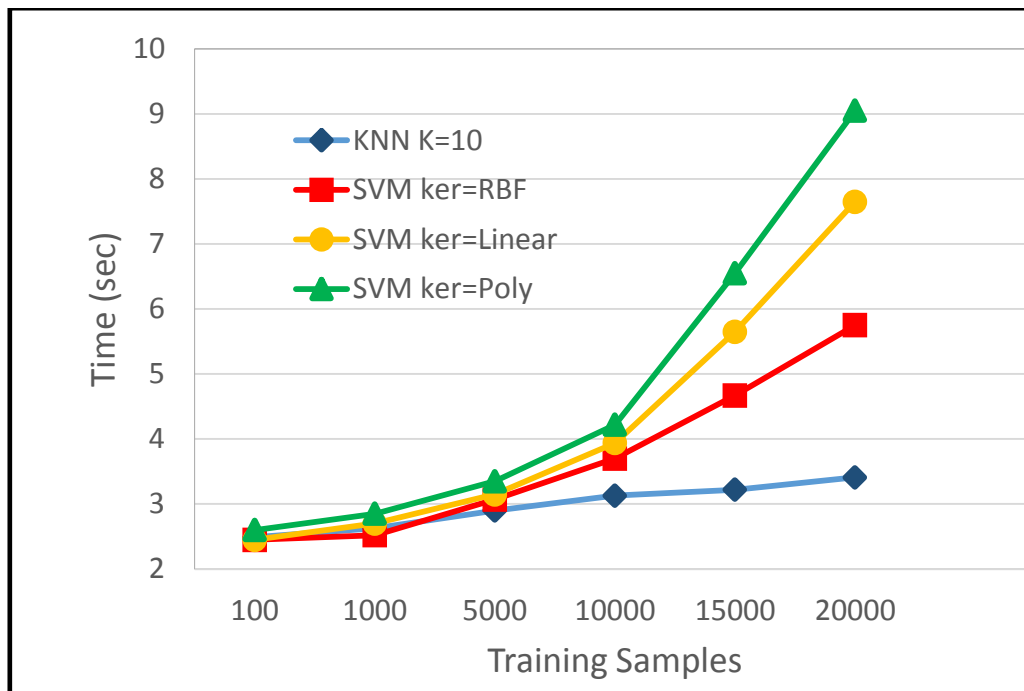


Figure 57: Time Complexity

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 identified 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)

#### 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 Nodered 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.

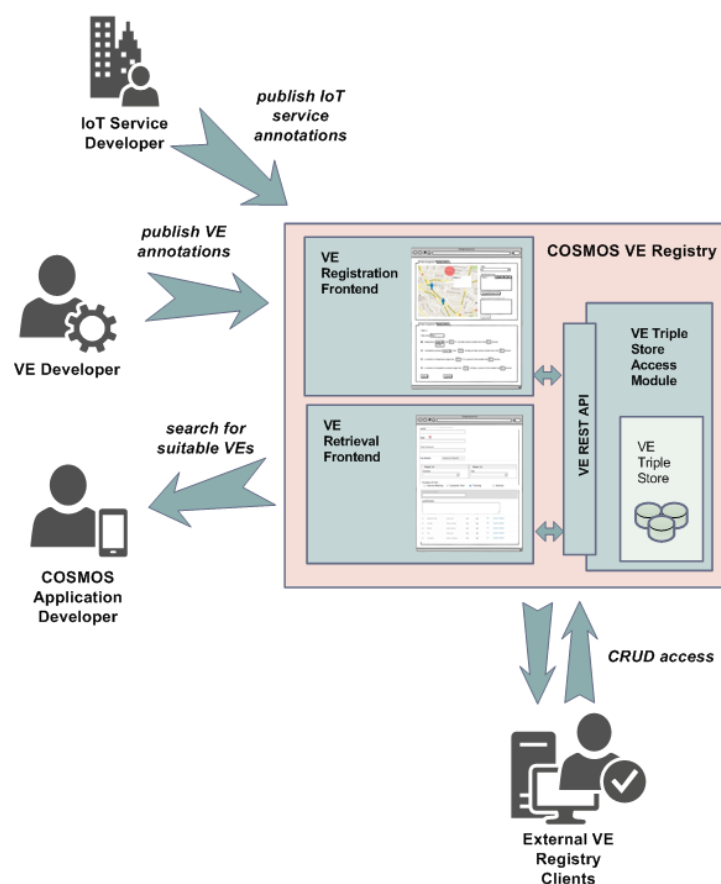


Figure 58: Generic usage scenarios for VE registry

### 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.

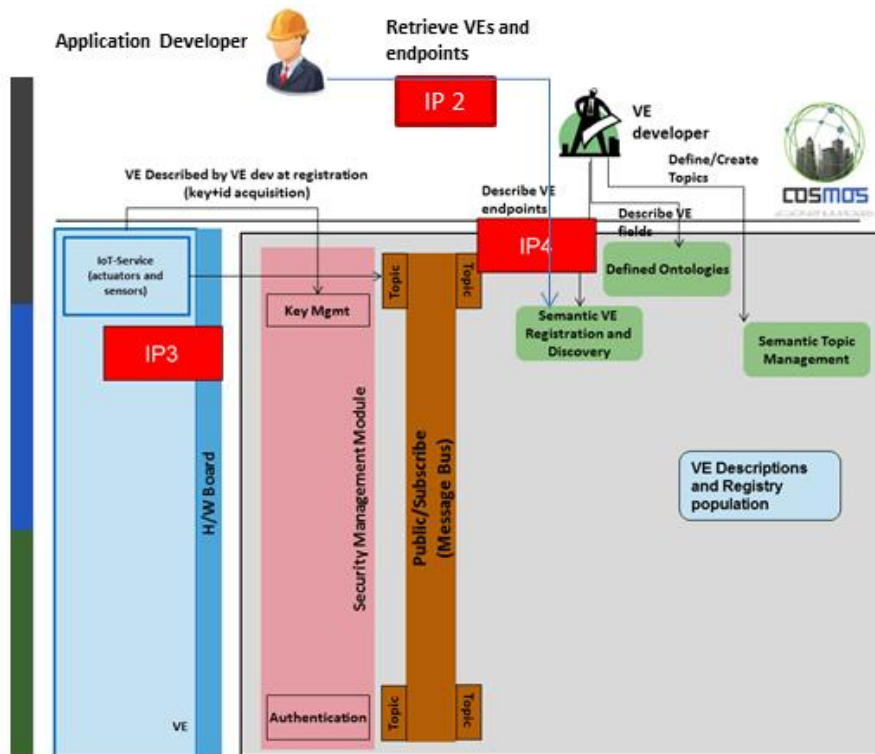


Figure 59: VE Registration Subsystem with identified IPs

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.

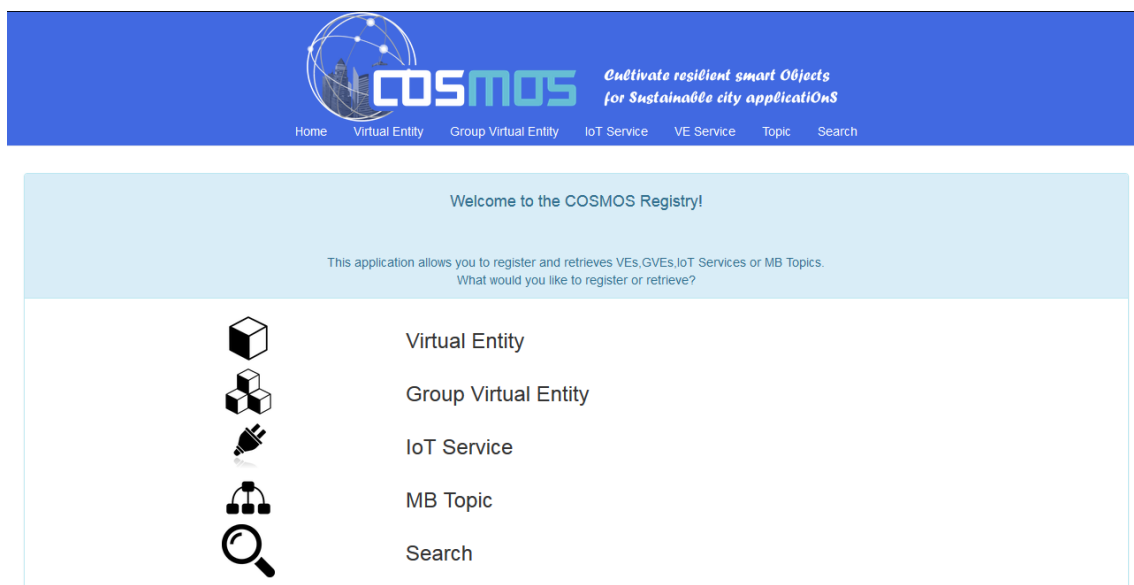
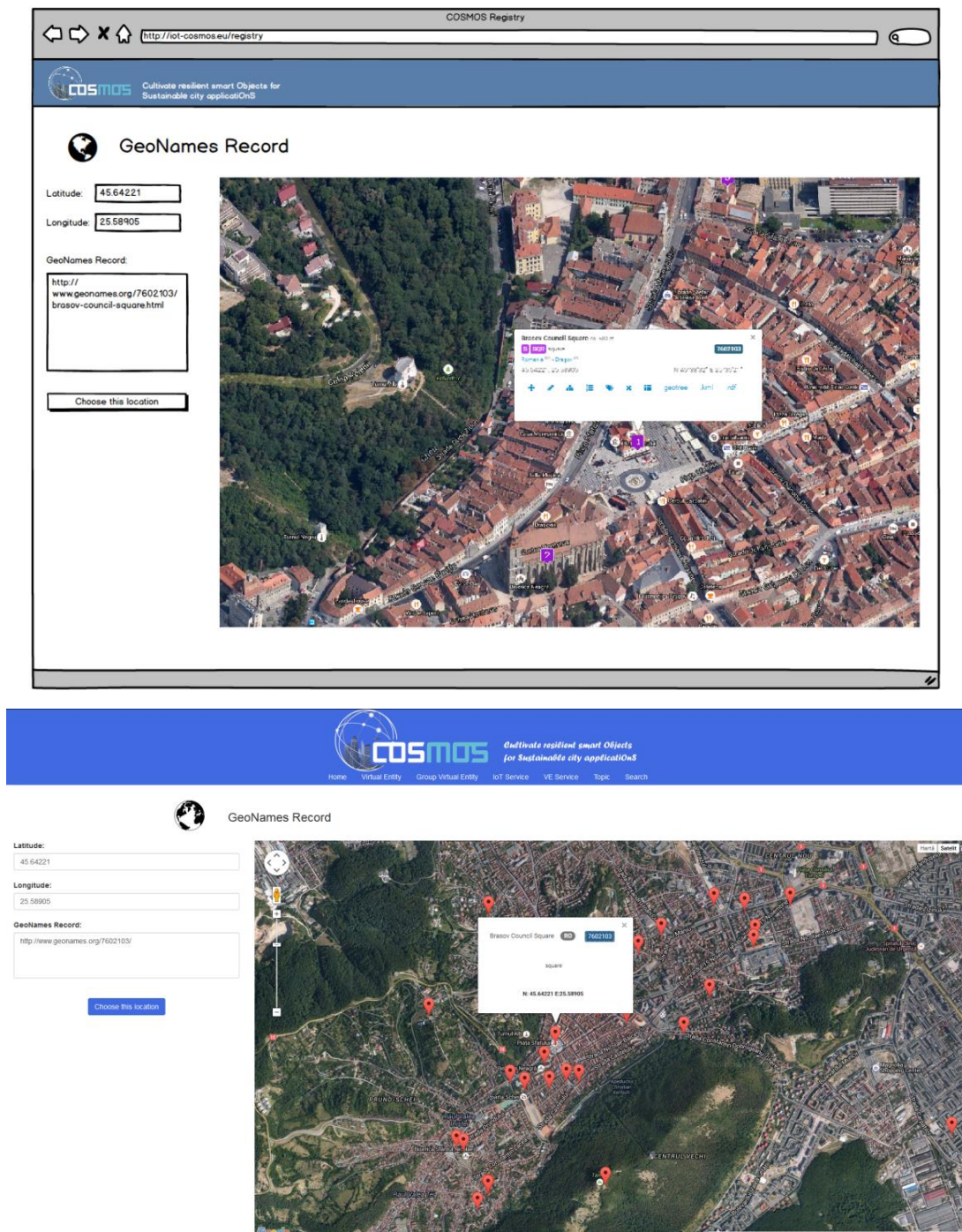


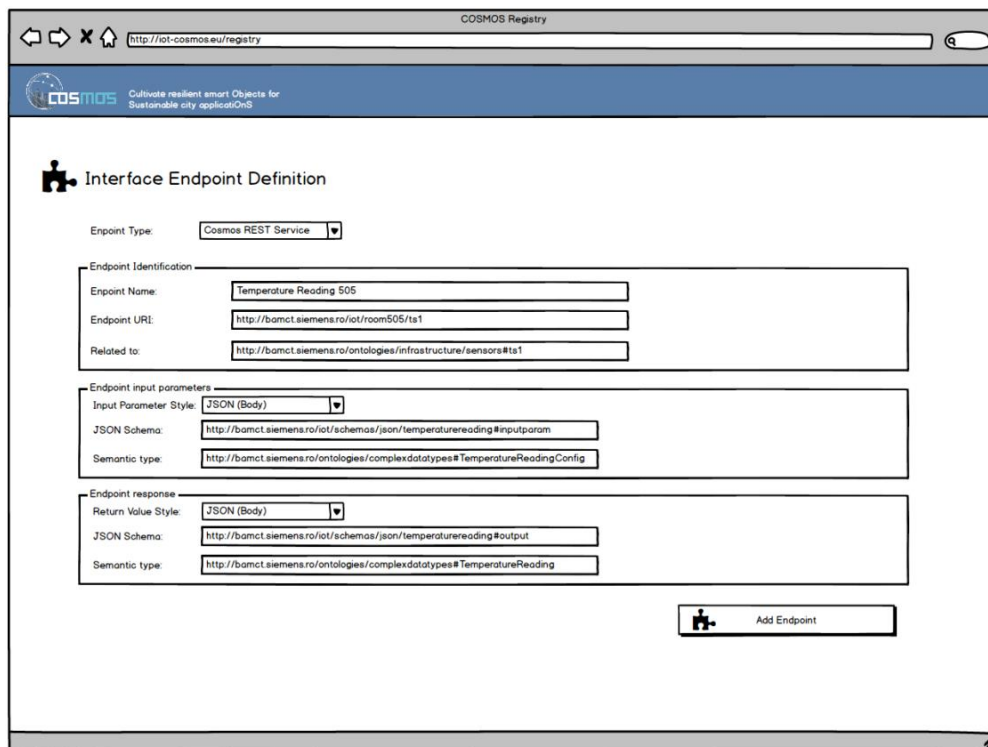
Figure 60: Cosmos VE semantic annotation welcoming webpage

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.

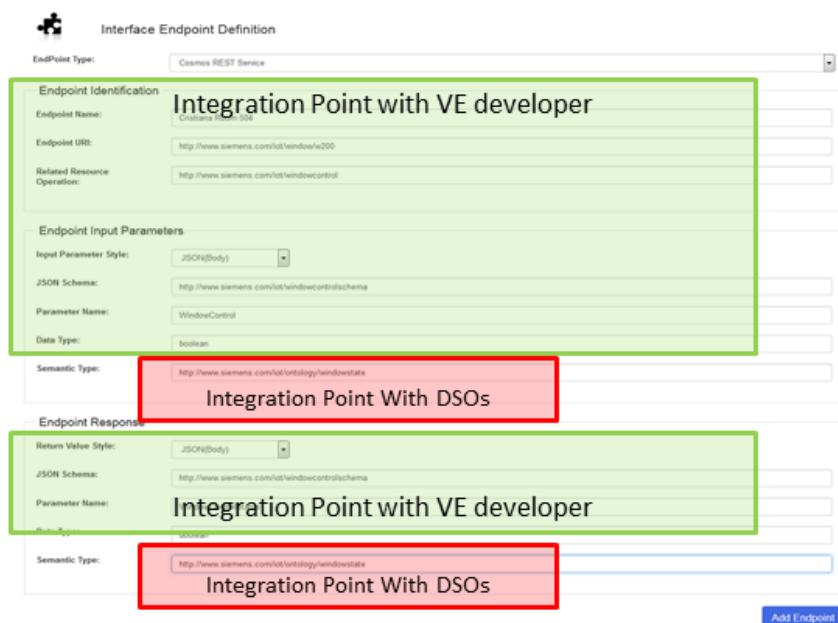


**Figure 61: Mockup (top) and the actual implementation (bottom) for one of the location browser pages (in this case the GeoNames Record based location indicator)**





The mockup shows a web browser window titled "COSMOS Registry" with the URL "http://iot-cosmos.eu/registry". The page header includes the COSMOS logo and the tagline "Cultivate resilient smart Objects for Sustainable city applicationOnS". The main section is titled "Interface Endpoint Definition" and contains a form for defining a new endpoint. The form is divided into several sections: "Endpoint Type" (set to "Cosmos REST Service"), "Endpoint Identification" (with fields for "Endpoint Name" (Temperature Reading 505), "Endpoint URI" (http://bamct.siemens.ro/iot/room505/ts1), and "Related to" (http://bamct.siemens.ro/ontologies/infrastructure/sensors#ts1)), "Endpoint input parameters" (with fields for "Input Parameter Style" (JSON (Body)), "JSON Schema" (http://bamct.siemens.ro/iot/schemas/json/temperaturreading#inputparam), and "Semantic type" (http://bamct.siemens.ro/ontologies/complexdatatypes#TemperatureReadingConfig)), and "Endpoint response" (with fields for "Return Value Style" (JSON (Body)), "JSON Schema" (http://bamct.siemens.ro/iot/schemas/json/temperaturreading#output), and "Semantic type" (http://bamct.siemens.ro/ontologies/complexdatatypes#TemperatureReading)). An "Add Endpoint" button is located at the bottom right of the form.



The actual implementation shows the same "Interface Endpoint Definition" form, but with additional annotations. The "Endpoint Identification" section is highlighted with a green box and labeled "Integration Point with VE developer". The "Endpoint Input Parameters" section is highlighted with a green box and labeled "Integration Point with VE developer". The "Endpoint Response" section is highlighted with a green box and labeled "Integration Point with VE developer". The "Semantic Type" field in the "Endpoint Input Parameters" section is highlighted with a red box and labeled "Integration Point With DSOs". The "Semantic Type" field in the "Endpoint Response" section is highlighted with a red box and labeled "Integration Point With DSOs". An "Add Endpoint" button is located at the bottom right of the form.

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.



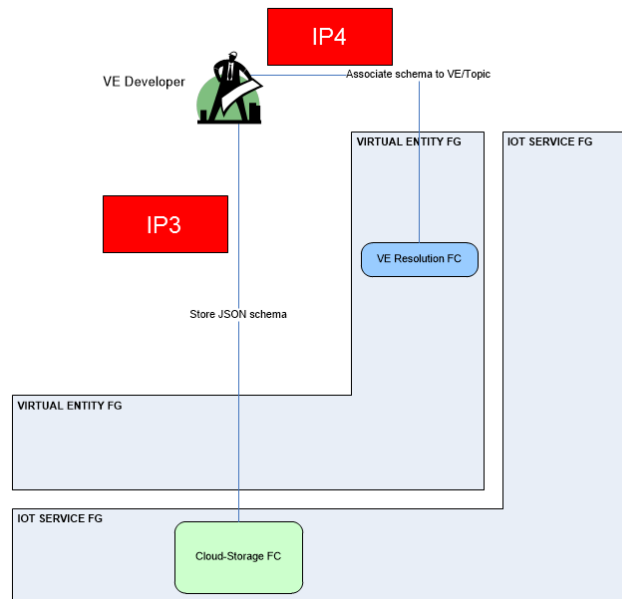


Figure 64: JSON schema storage and association

### 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).

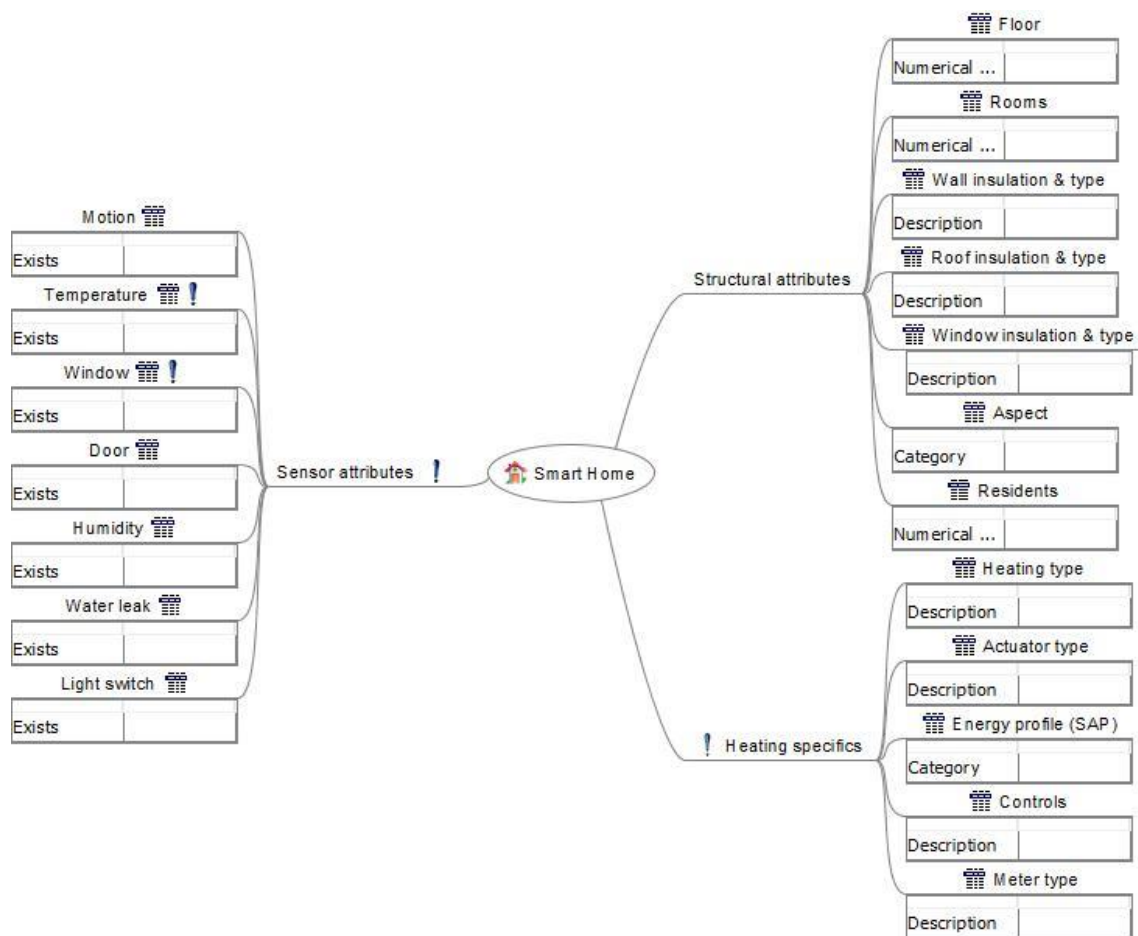


Figure 65: Camden DSO fields

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

[http://37.48.76.239/v1/AUTH\\_a32a367c5ded4a3c860ea4e81255f315/secorSchema](http://37.48.76.239/v1/AUTH_a32a367c5ded4a3c860ea4e81255f315/secorSchema)

[Send](#) [Preview](#) [Add to collection](#)

---

body Headers (9) STATUS 200 OK TIME 90 ms

[Pretty](#) [Raw](#) [Preview](#)   [JSON](#) [XML](#)

```
1 III.metaKeys
2 III.schema
3 TrafficFlowMadridPM.metaKeys
4 TrafficFlowMadridPM.schema
5 camden.metaKeys
6 camden.schema
```

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.

http://37.48.76.239/v1/AUTH\_a32a367c5ded4a3c860ea4e81255f315/secorSchema/TrafficFlowMadridPM.schema GET

Send Preview Add to collection

Body Headers (9) STATUS 200 OK TIME 330 ms

Pretty Raw Preview JSON XML

```

1 { "namespace": "cosmos",
2   "type": "record",
3   "name": "TrafficFlowMadridPM",
4   "fields": [
5     { "name": "codigo", "type": "string" },
6     { "name": "descripcion", "type": [ "null", "string" ] },
7     { "name": "accesoAsociado", "type": [ "null", "long" ] },
8     { "name": "intensidad", "type": "int" },
9     { "name": "ocupacion", "type": "int" },
10    { "name": "carga", "type": "int" },
11    { "name": "nivelServicio", "type": "int" },
12    { "name": "velocidad", "type": [ "null", "int" ] },
13    { "name": "intensidadSat", "type": [ "null", "int" ] },
14    { "name": "error", "type": "string" },
15    { "name": "subarea", "type": [ "null", "int" ] },
16    { "name": "ts", "type": "long" },
17    { "name": "tf", "type": "string" }
18  ]
19 }
```

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:

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

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

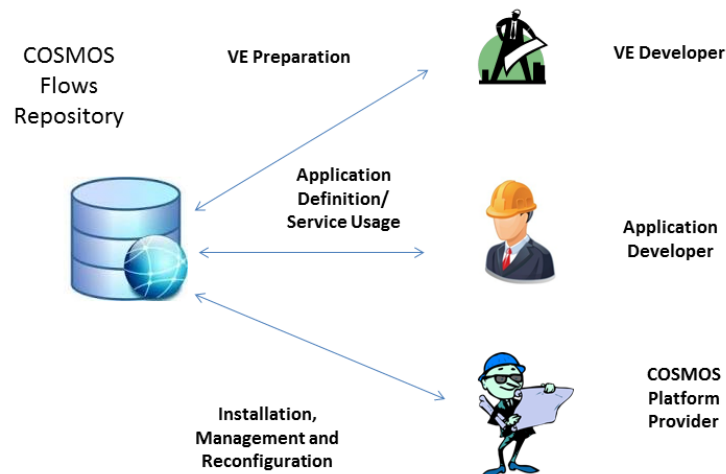
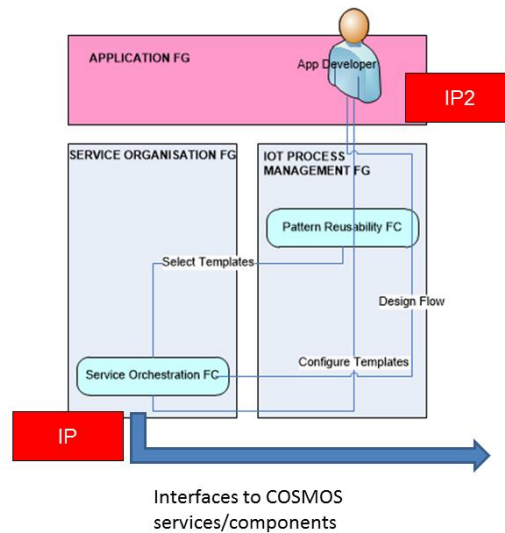


Figure 66: Reuse scenarios of Node-RED flows (Roles interactions Figure 17 from D7.6.2)

#### 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.

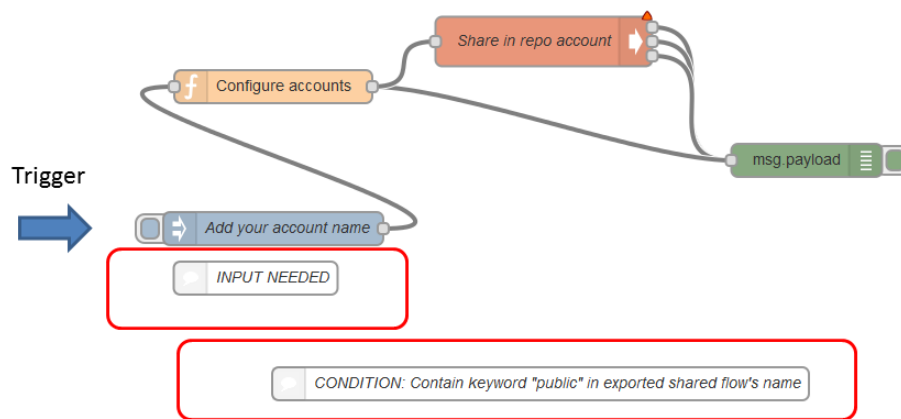
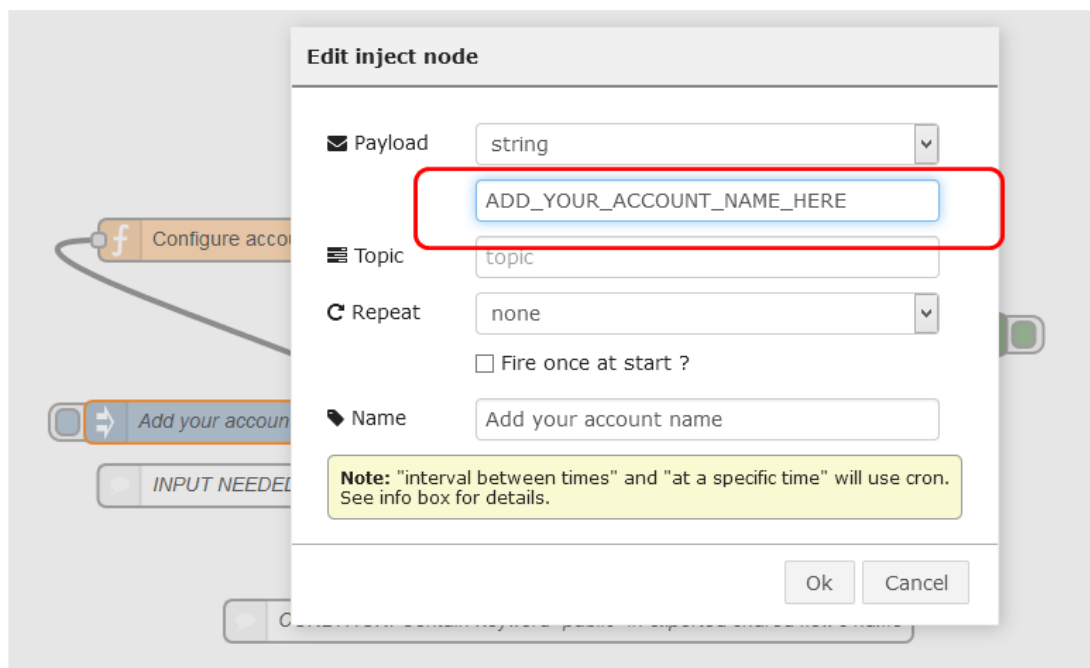


Figure 68: Share\_flows\_public flow notation example



**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.

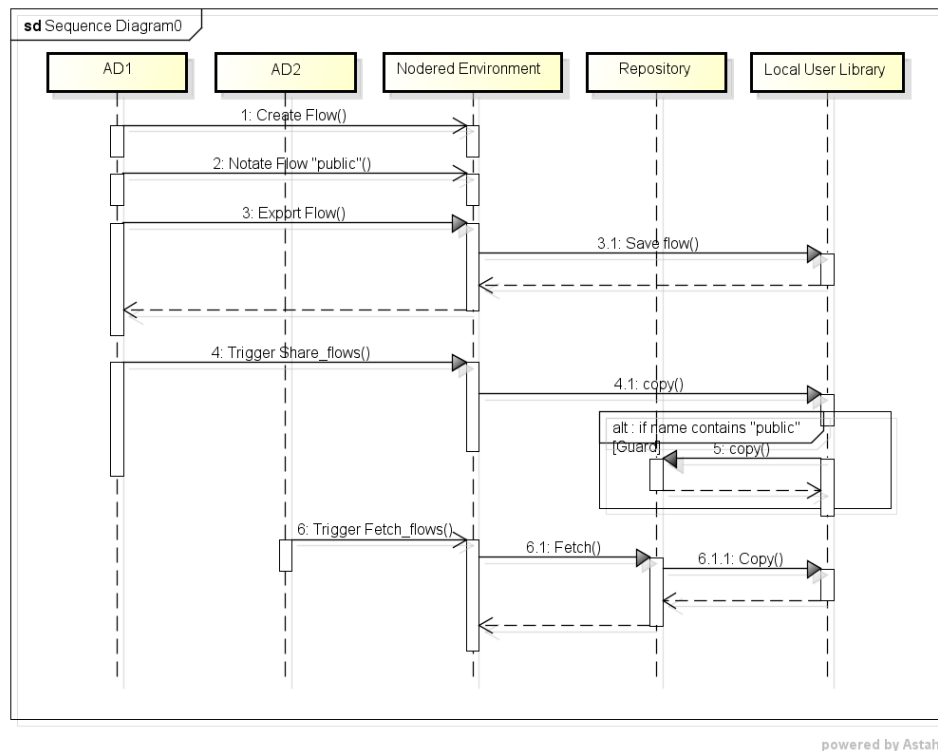


Figure 70: Sequence Diagram for sharing flows

### 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.
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	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

<b>Test Case Number Version</b>	<b>ADF_2</b>
<b>Test Case Title</b>	Flow Fetching from Repository
<b>Module tested</b>	Pattern Reusability FC (Repository flow), Service Orchestration FC
<b>Requirements addressed</b>	UNI.408, 5.31, UR1
<b>Initial conditions</b>	The developer has an account in Node-RED. There is a “repo” account available in the Node-RED environment
<b>Expected results</b>	All repo flows from the repo account are copied in the user’s account.
<b>Owner/Role</b>	Developer
<b>Steps</b>	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.
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	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***

- 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

- 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.

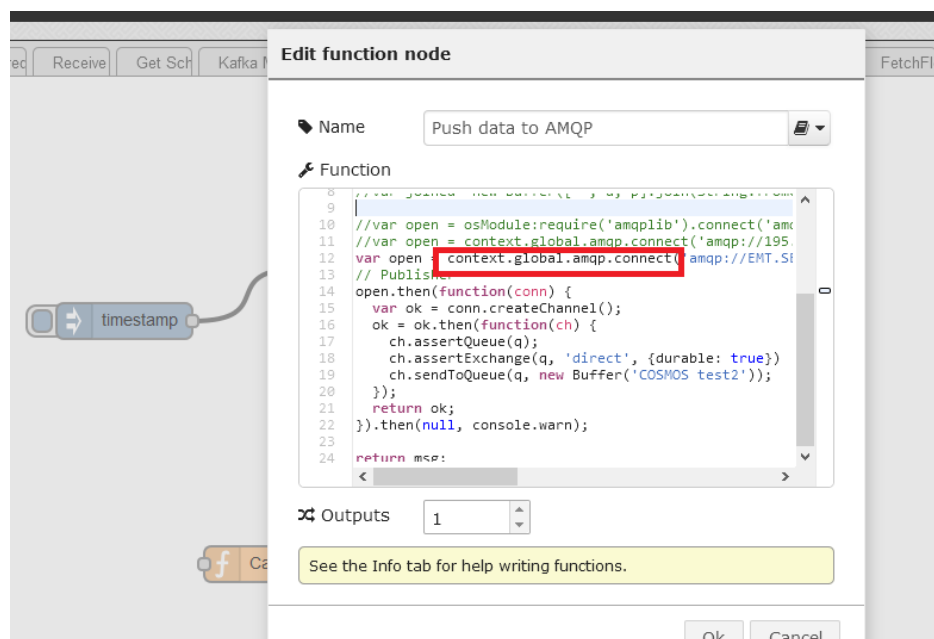


Figure 72: Node-RED function node code for inclusion of external libraries

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;
- $\mu$ CEP – the  $\mu$ CEP 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.

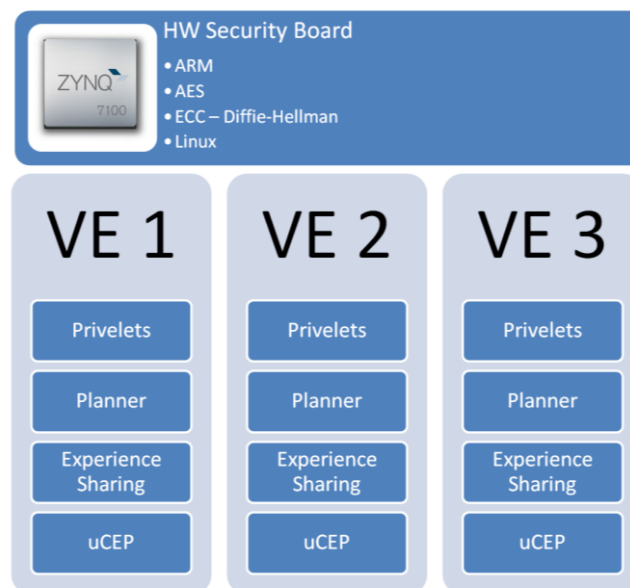


Figure 73: VE side COSMOS components

#### 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  $\mu$ CEP 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  $\mu$ CEP Engine.

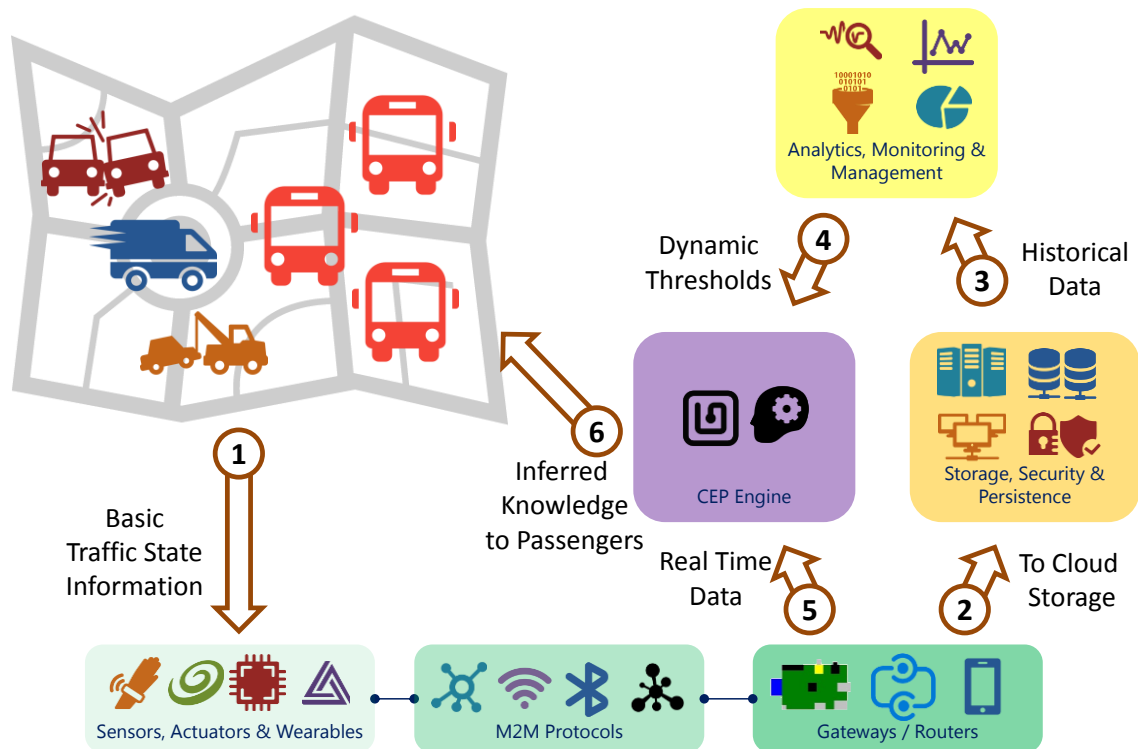


Figure 74: Madrid Traffic State Analysis Use Case

### 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.



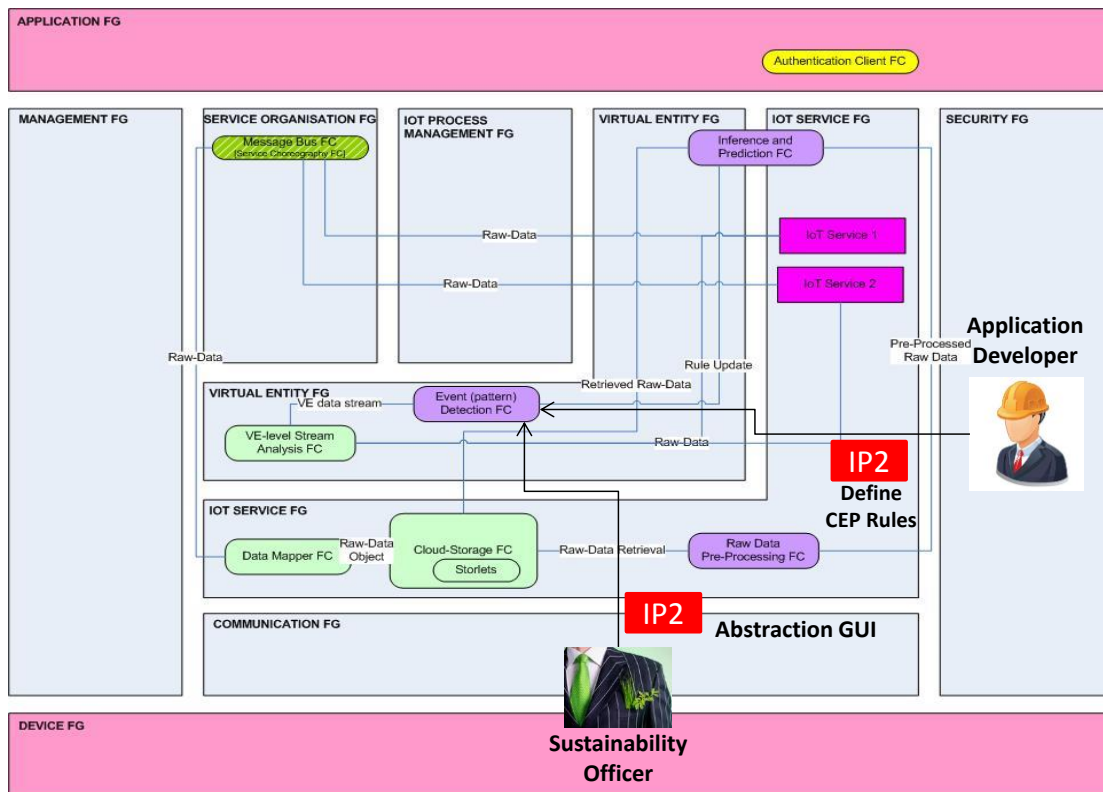


Figure 75: Traffic flow Subsystem Diagram

### 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  $\mu$ CEP 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.

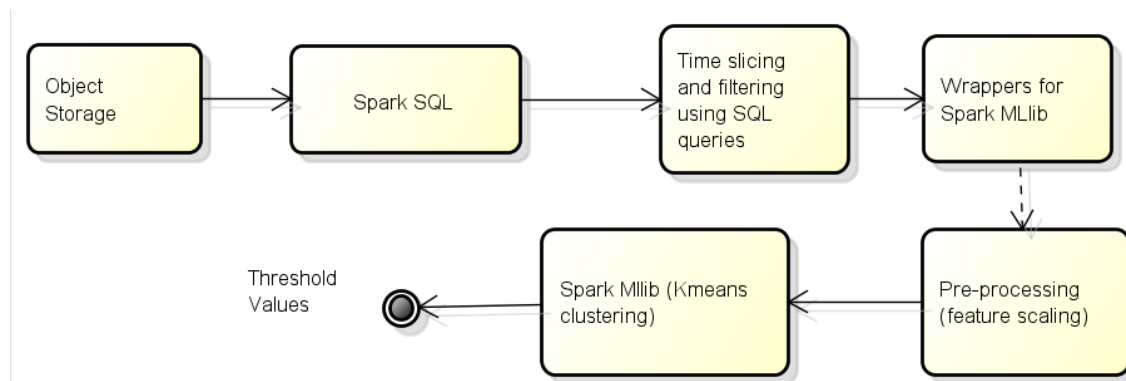


Figure 76: Steps for finding threshold values for CEP rules

### 3.2.4.1.5 Subsystem Test case table

The subsystem test case appears in Table 8.

**Table 8: Test case for the CEP and ML cooperation**

<b>Test Case Number Version</b>	<b>MAD_02</b>
<b>Test Case Title</b>	Traffic thresholds calculation
<b>Module tested</b>	Message Bus, Cloud Storage, Node-RED Flow, $\mu$ CEP, Machine Learning
<b>Requirements addressed</b>	5.20, 5.21, 6.43, 6.5, 6.6
<b>Initial conditions</b>	Real-time traffic data feed Historical traffic information stored Node-RED rules with dynamic thresholds
<b>Expected results</b>	New thresholds calculated by Machine Learning $\mu$ CEP traffic congestion detection rules are adjusted to actual conditions with new thresholds
<b>Owner/Role</b>	VE developer
<b>Steps</b>	Machine Learning (ML) retrieves data from Cloud Storage ML computes new thresholds $\mu$ CEP receives new thresholds and updates rules
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	Lack of persistence of new thresholds upon $\mu$ CEP Engine reboot
<b>Required changes</b>	ML component stores new thresholds in Cloud Storage $\mu$ CEP 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  $\mu$ CEP 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  $\mu$ CEP 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 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 CEP each time it starts, so a dedicated daemon may handle this message and send the latest calculated thresholds.

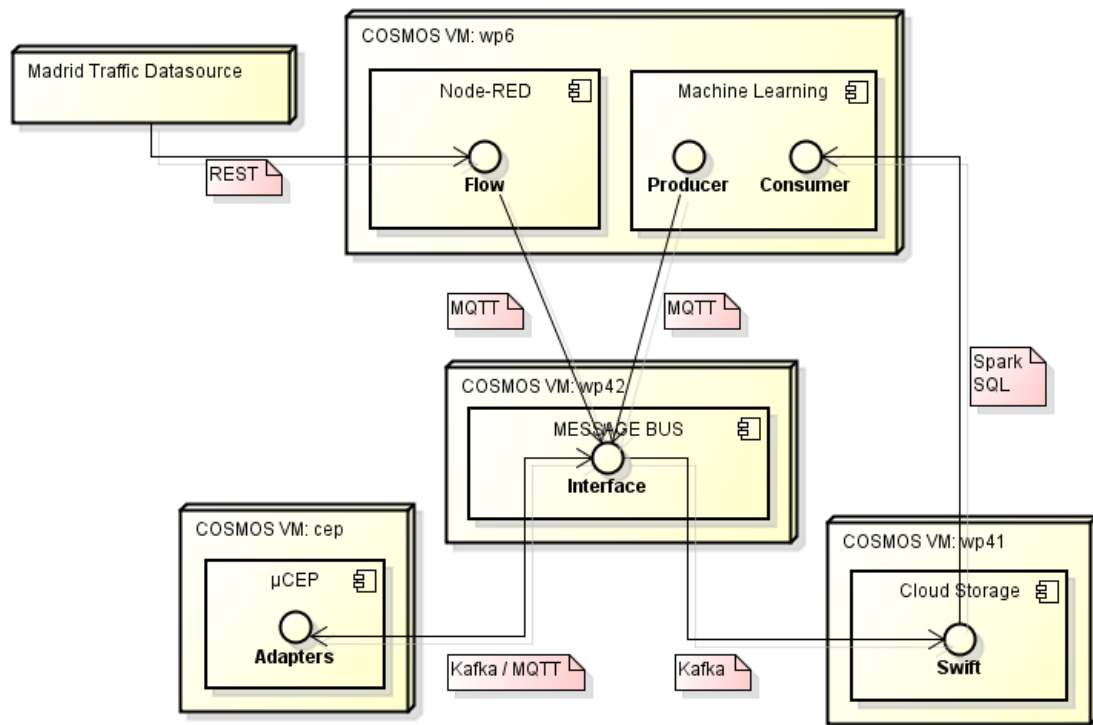


Figure 77: Madrid Traffic Analysis Use Case - Deployment Diagram

### 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.

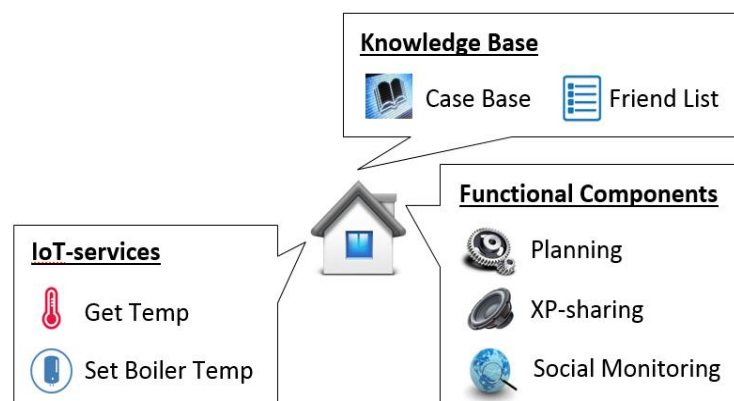


Figure 78: Camden Flat VE overview

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

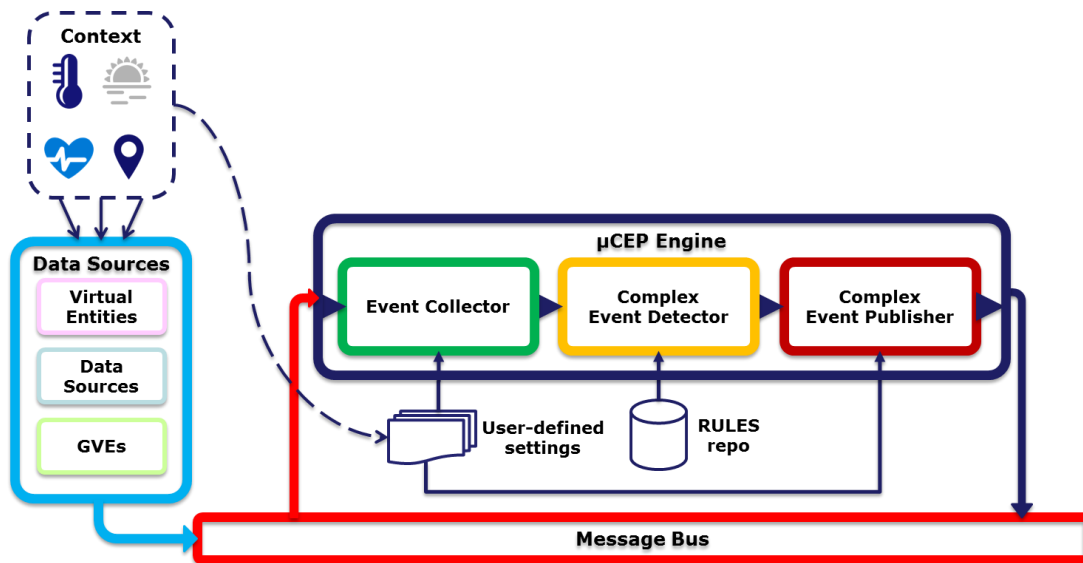


Figure 79: SAw FC and the CEP Engine

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 CEP engine instance running on the VE side. This analysis is what the CEP 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. Therefore 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 CEP 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.

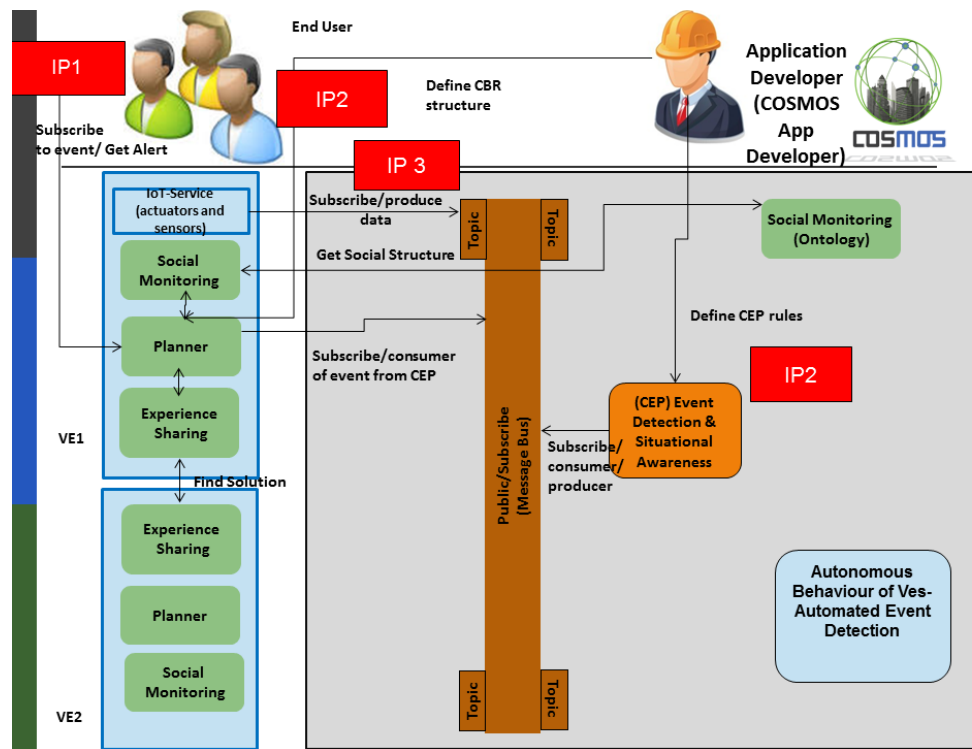


Figure 80: Autonomous Behaviour of VEs, basis for Proactive XP Sharing

In Figure 81, there is a clear demonstration of the primary flow of Complex Events between the CEP 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 CEP 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.

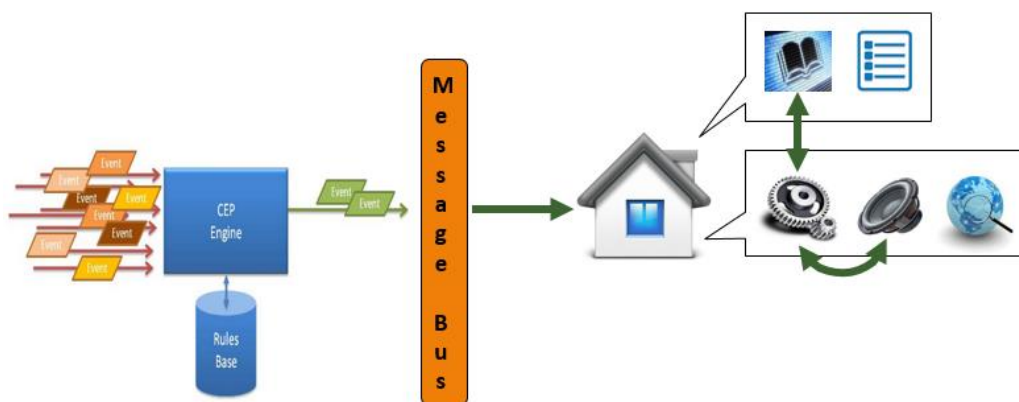


Figure 81: Flow of extracted Complex Events into other VE sided FCs

### 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": "aaaabbbbccccddd",
  "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 CEP 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 CEP engine will make use of the specific rules uploaded to it by the Application developer.

If and when the CEP 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

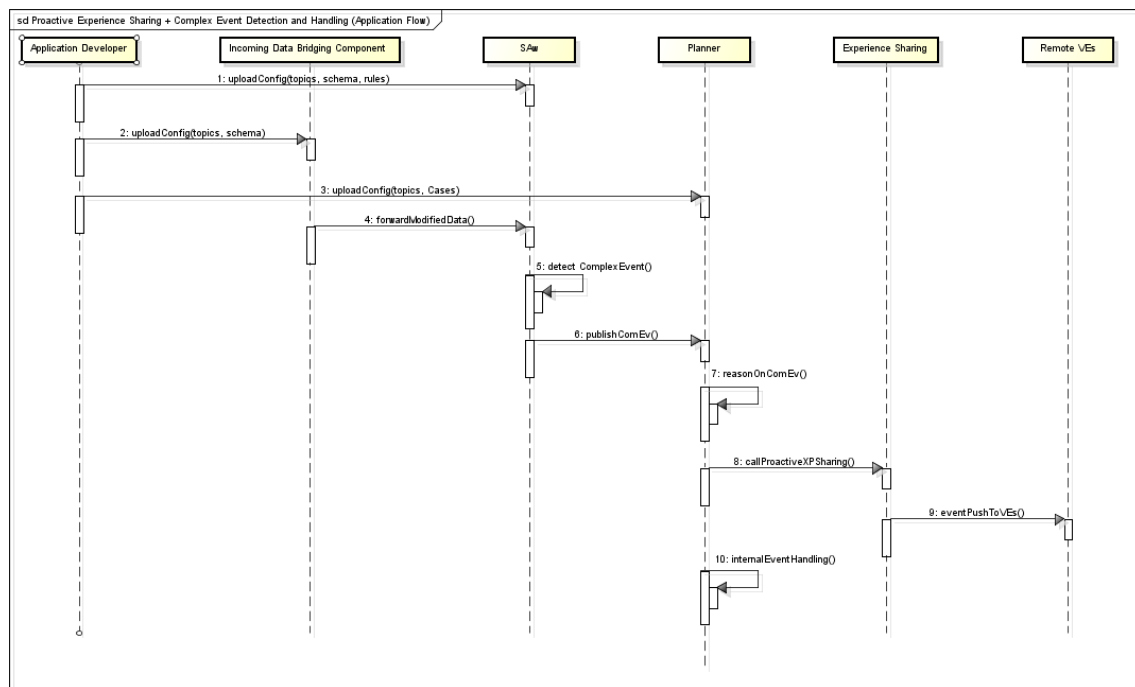


Figure 82: Proactive Experience Sharing Sequence Diagram

### 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

<b>Test Case Number</b>	<b>PRO_01</b>
<b>Version</b>	
<b>Test Case Title</b>	Proactive Experience Sharing
<b>Module tested</b>	Data Bridging, SAW FC's $\mu$ CeP engine, Planner FC, Experience Sharing FC, Social Monitoring FC
<b>Requirements addressed</b>	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]
<b>Initial conditions</b>	Uploaded Configurations to Planner, SAW FCs and the Bridging Component, Connectivity between all Components
<b>Expected results</b>	Identify and handle Complex Event through Sharing it to other VEs, or handling it internally, on the basis of CBR Solution retrieval
<b>Owner/Role</b>	Application Developer VE developer
<b>Steps</b>	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

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

### 3.2.4.2.6 Deployment Diagram

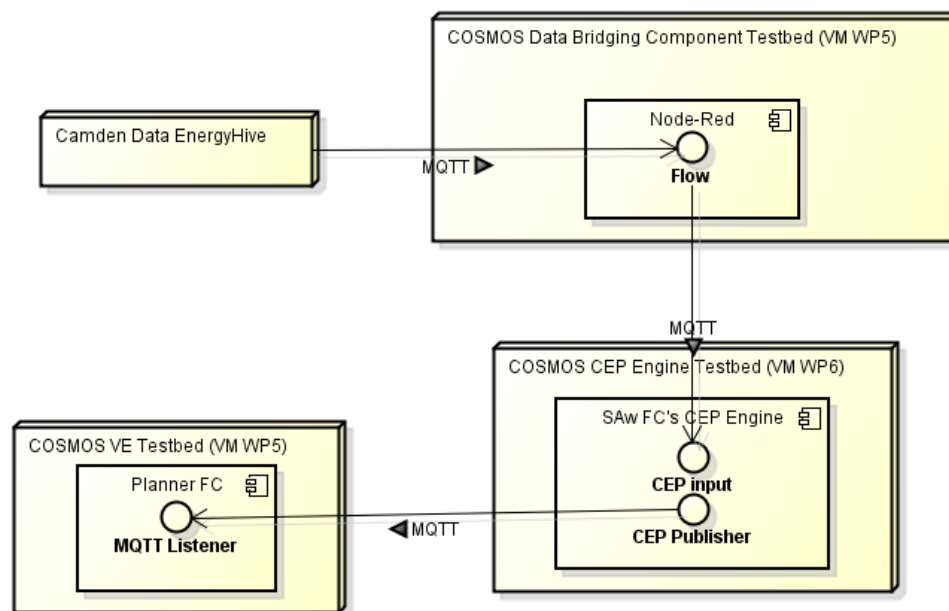


Figure 83: Deployment Diagram for the Proactive Experience Sharing Subgroup

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 CEP 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<sup>1</sup>. The sensorization of the traffic is done by 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
<b>ID</b>	Unique identifier of the Measurement Point
<b>Intensity</b>	Intensity in number of vehicles per hour (a.k.a. traffic flow)
<b>Occupancy</b>	Percentage of vehicle occupancy in the measurement point
<b>Load</b>	Calculated from intensity, occupancy and characteristics of the street, this parameters informs about the traffic load level
<b>Service Level</b>	Not implemented
<b>Speed</b>	Average speed of the vehicles detected in the last “integration period” (usually 5 minutes)
<b>Error</b>	-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>
```

<sup>1</sup> Traffic intensity in Real-Time – Madrid Opendata Portal: <http://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnnextoid=02f2c23866b93410VgnVCM1000000b205a0aRCRD&vgnnextchannel=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.

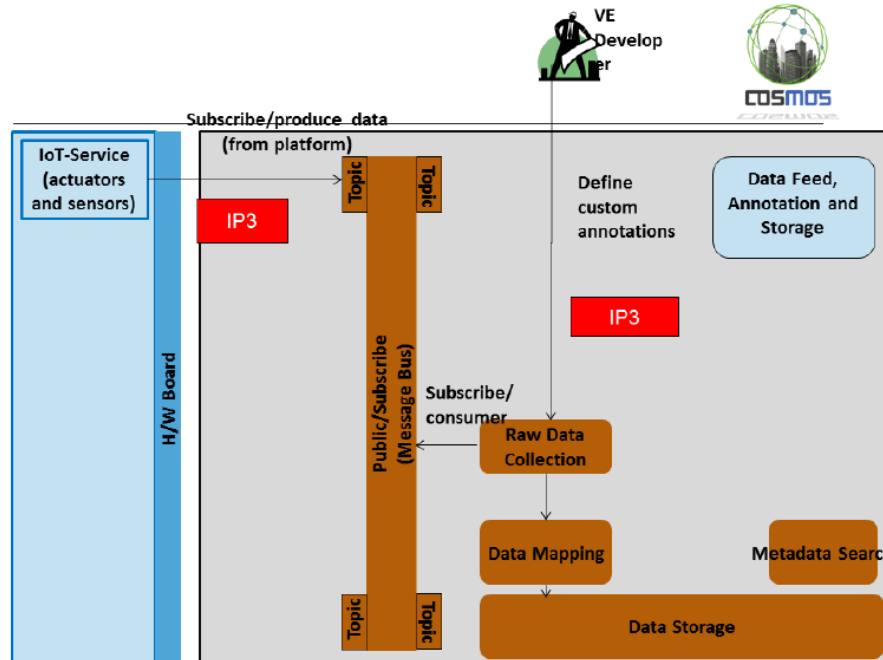


Figure 84: Data Feed, Annotation and Storage Subsystem

### 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-traffic-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.

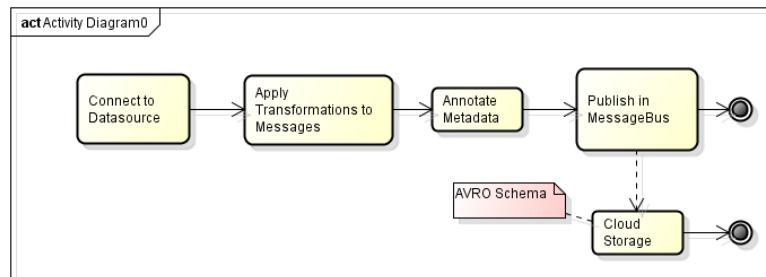


Figure 86: Madrid data feed Activity Diagram

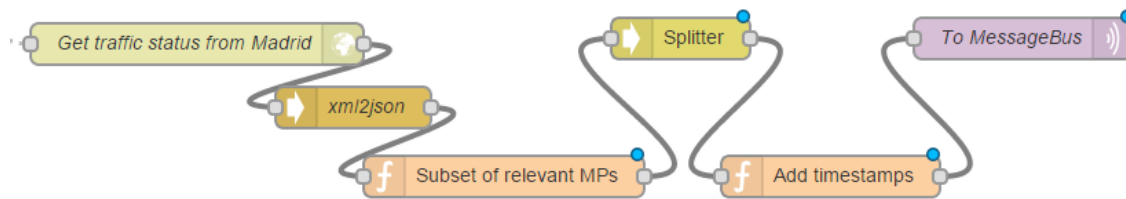


Figure 87: Madrid data feed Flow

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.

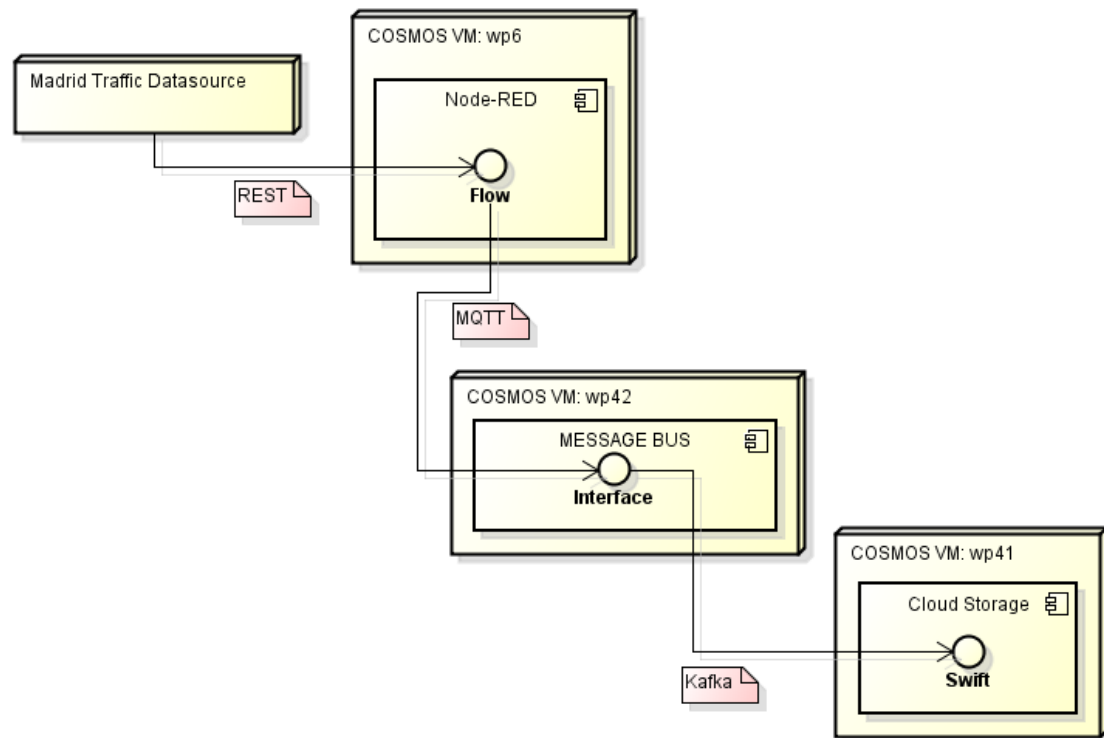


Figure 88: Madrid Data Feed Deployment Diagram

### 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

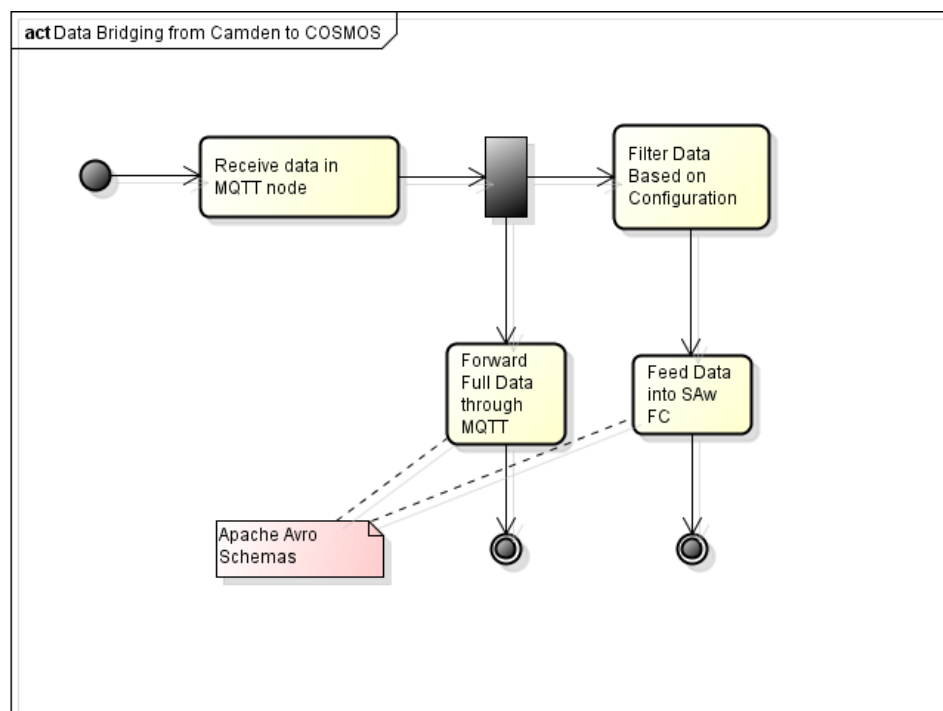


Figure 89: Data Bridging Activity Diagram

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.

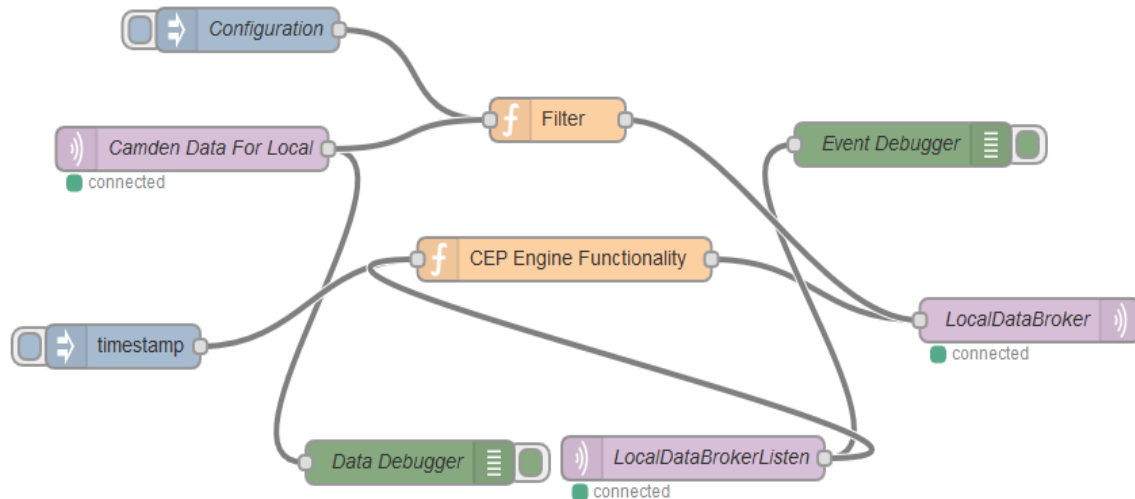


Figure 90: Data Bridging Flow and Connectivity

### 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

<b>Test Case Number Version</b>	<b>CAM_01</b>
<b>Test Case Title</b>	Camden Data Extraction
<b>Module tested</b>	Node-Red flow, Java MQTT libraries
<b>Requirements addressed</b>	4.1, 4.2, 4.9, 6.1, 6.5
<b>Initial conditions</b>	Node-Red flow, Hildebrand Server MQTT Broker, VE code
<b>Expected results</b>	Receive JSON formatted data in Node-Red flow Successfully filter fields Forward Message to Planner FC and CEP representing function Succeed in reading data from Planner FC
<b>Owner/Role</b>	VE developer
<b>Steps</b>	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)

<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	None

### 3.2.5.2.6 Deployment Diagram

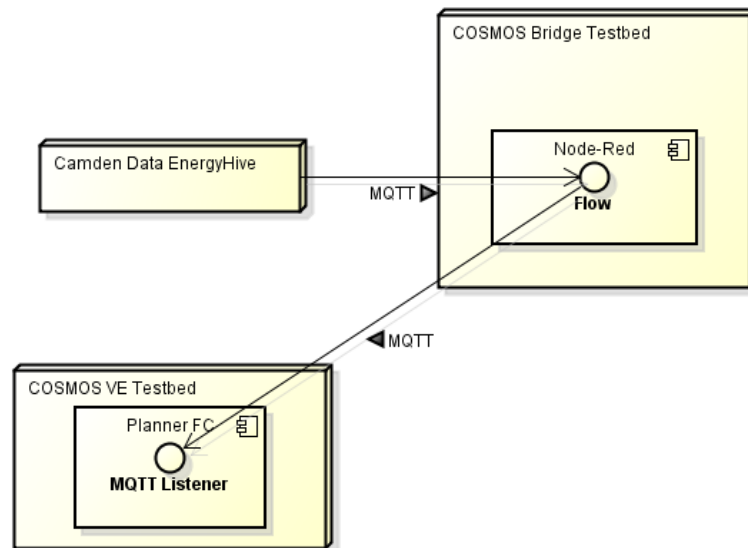


Figure 91: Deployment Diagram for Camden Data Feed

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
<b>dev_id</b>	Unique identifier for the socket
<b>gw_dev_id</b>	Unique identifier of the gateway with which socket is connected
<b>Info_id</b>	Identifier for the data feature (such as “3” for Active Power)
<b>Info_value</b>	Actual measure of the data feature being reported
<b>Info_desc</b>	Name of the data feature
<b>Report_time</b>	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.

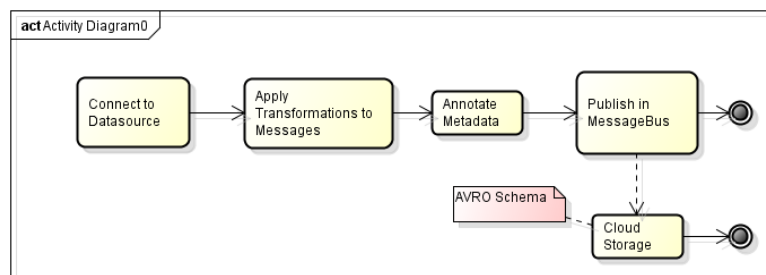


Figure 92: Taipei data feed Activity Diagram

### 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

<b>Test Case Number Version</b>	III_01
<b>Test Case Title</b>	III feed storage
<b>Module tested</b>	Message Bus, Cloud Storage, Node-RED Flow
<b>Requirements addressed</b>	4.1, 4.2, 4.9, 6.1, 6.5
<b>Initial conditions</b>	A functional Message Bus AVRO Schema shared with Cloud Storage Node-RED instance
<b>Expected results</b>	Electricity consumption data made available in real-time through Message Bus Electricity consumption data stored for later analysis in Cloud Storage
<b>Owner/Role</b>	VE developer
<b>Steps</b>	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
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	Desynchronization of data source causes data corruption
<b>Required changes</b>	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.

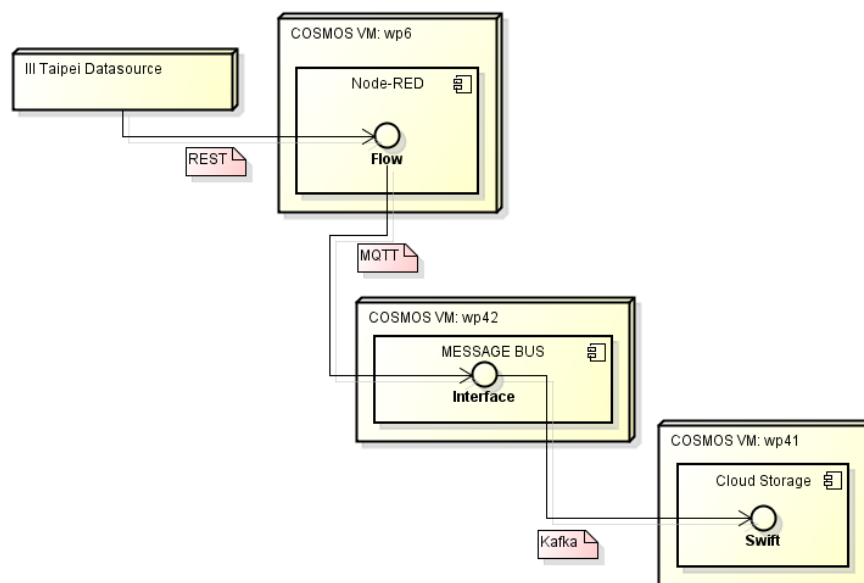


Figure 93: Deployment diagram for III data feed

### 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 CEP, 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 CEP 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.



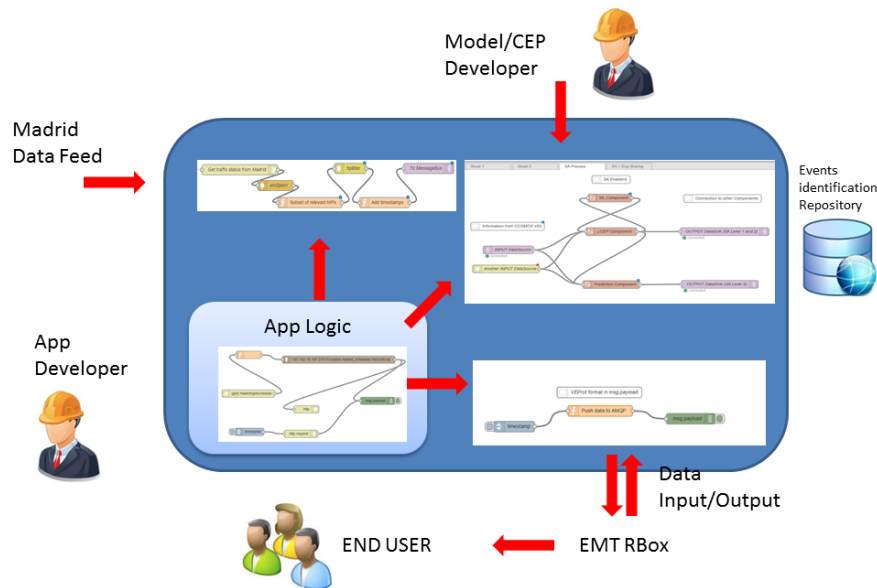


Figure 94: Madrid Scenario Integration

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 CEP), 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 subflows, which implies their iterated usage in generic and arbitrary application scenarios. As an 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.

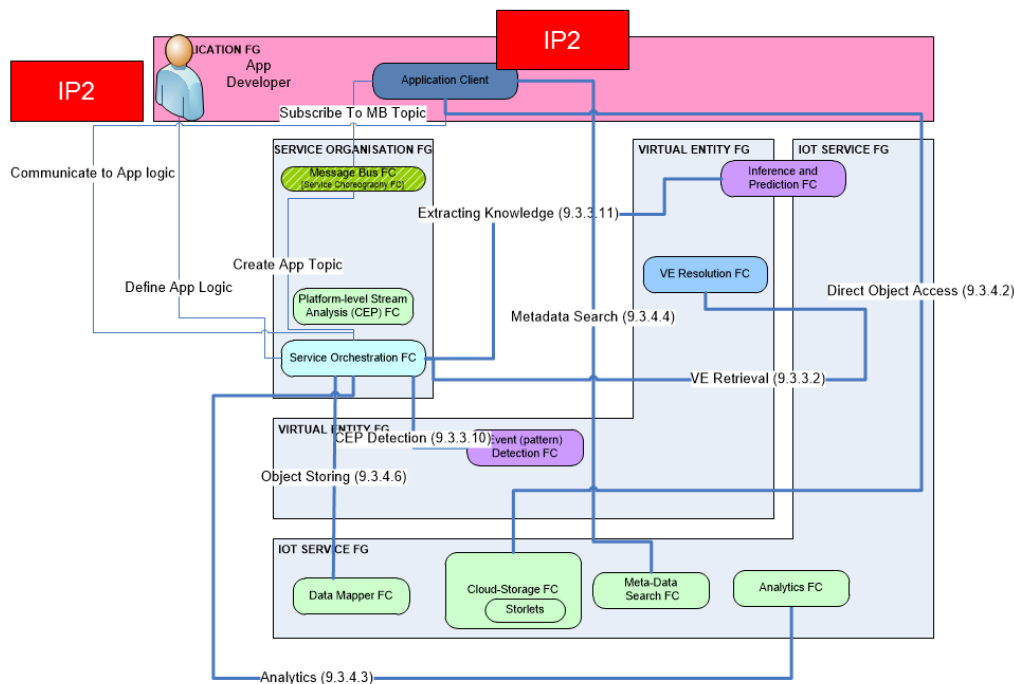


Figure 95: Centralized archetype subsystem as used in the Madrid Scenario Application

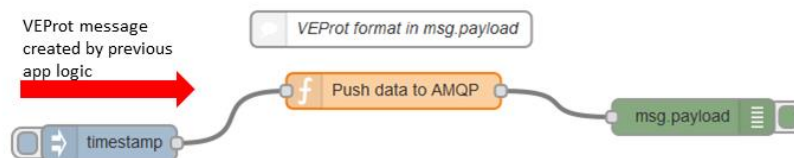


Figure 96: Generic Data Output Flow towards the RBox system

### 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
    ]
  },
  "idSession" : "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",
  "vcp_data": [
    {
      "dataLayer": {
        "layer": "ROUTESMAD.alarms",
        "idRoute": "55d1801ffb954f08f8f1489f",
        "idSession": "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. "Vcp\_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. "idSession" -> Using usrtrack idsession. 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.

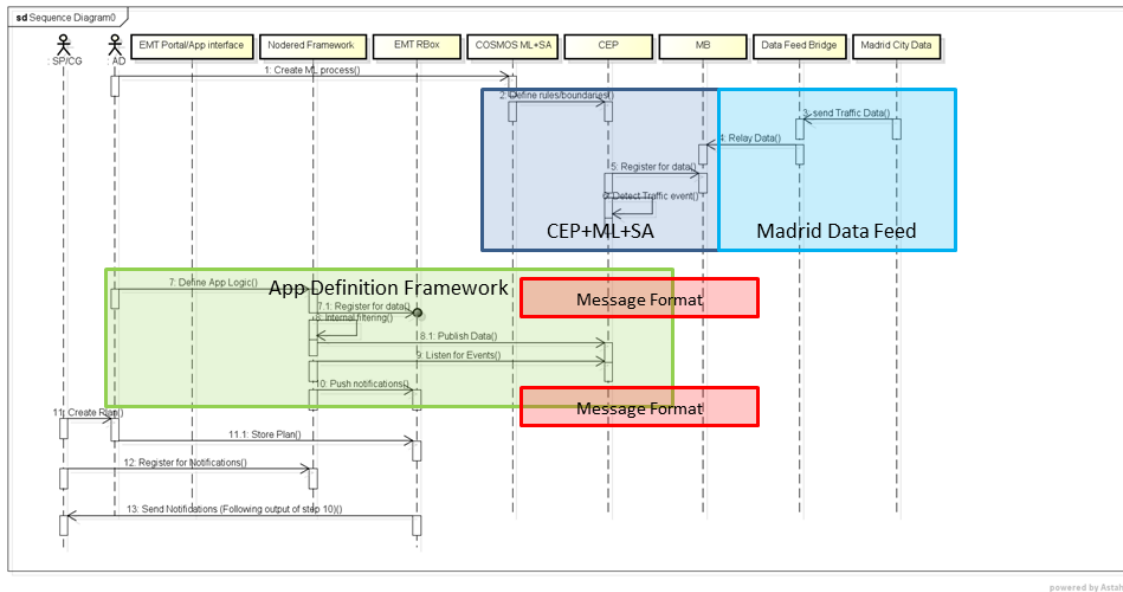


Figure 99: Sequence diagram for Madrid Scenario Application

### 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 notifications 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	<ol style="list-style-type: none"> <li>1. The actor logs in the EMT portal and creates the route plan</li> <li>2. The actor logs in the respective flow and inserts the SP id (this is triggered manually for testing purposes)</li> <li>3. The events identified from the COSMOS CEP are relayed through the MB to the App logic and from there to the</li> </ol>

	SP/CG interface (portal or mobile app)
<b>Passed</b>	
<b>Bug ID</b>	N/A
<b>Problems</b>	None
<b>Required changes</b>	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 CEP 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 CEP 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 end the User input. The Application 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.



Both of the above images therefor demonstrate that the main points of 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.

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.

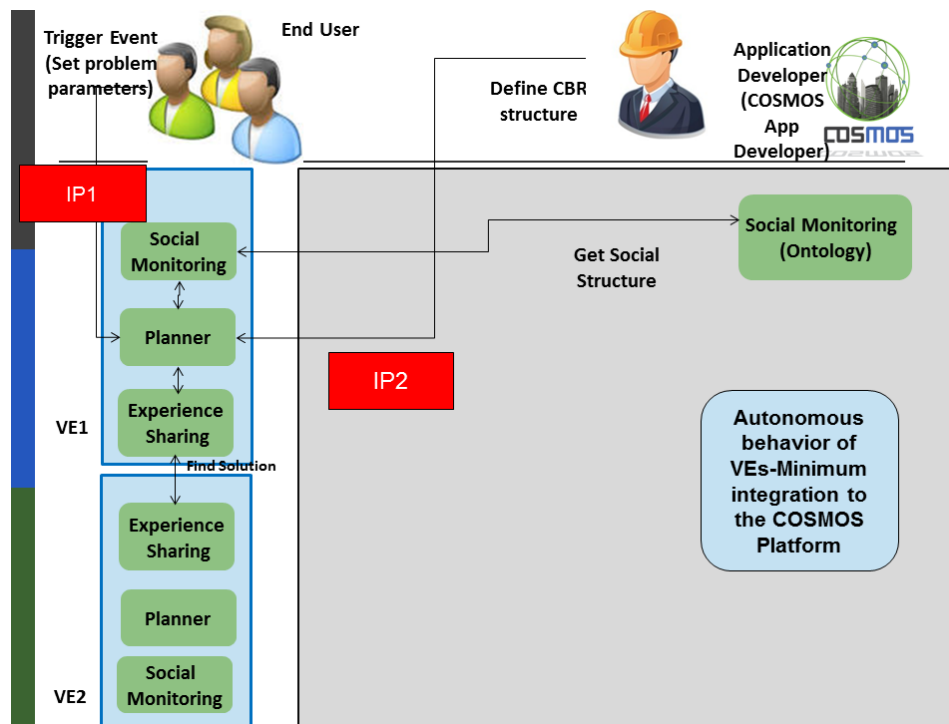


Figure 101: Autonomous Behavior of VEs with minimum platform integration subsystem

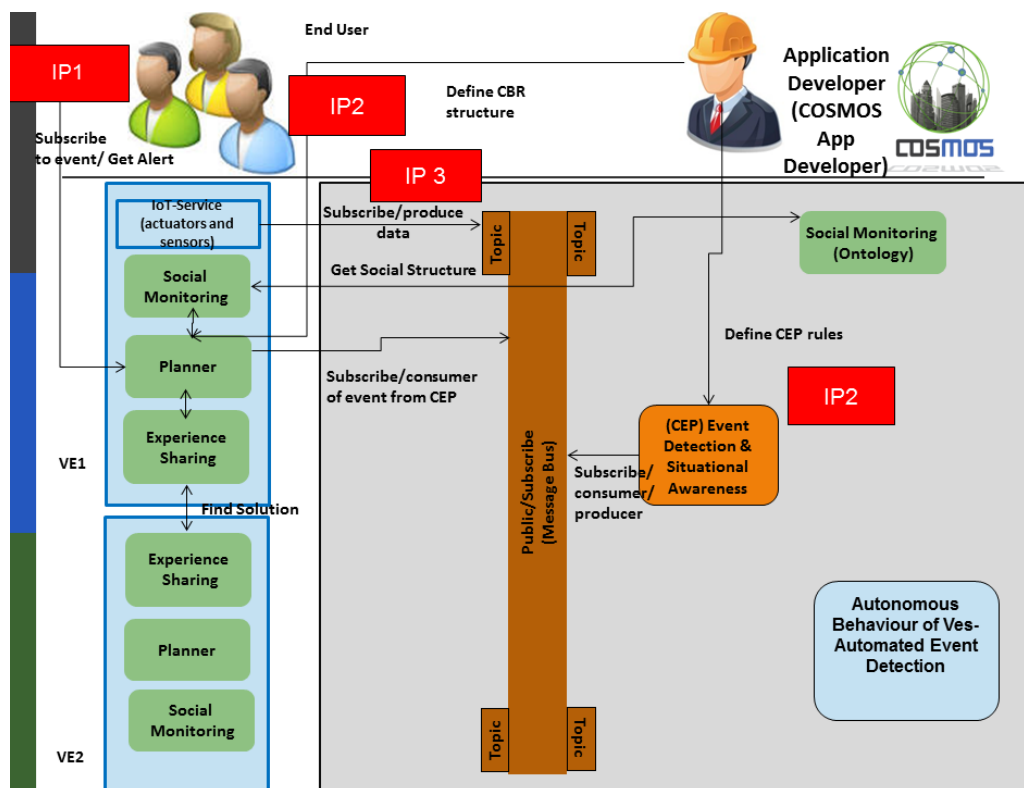


Figure 102: Autonomous Behavior of VEs- Automated Event Detection and Incorporation of COSMOS Platform

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 CEP 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 CEP 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 CEP 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.


### Heating Scheduler Application

Please provide a continuous time period of heating.  
Additionally please add a desired temperature.  
Finally specify your available budget.









Submit Data

Back to the Main Menu

Figure 103: Heating Scheduling App GUI

### 3.2.7.4 Sequence Diagram

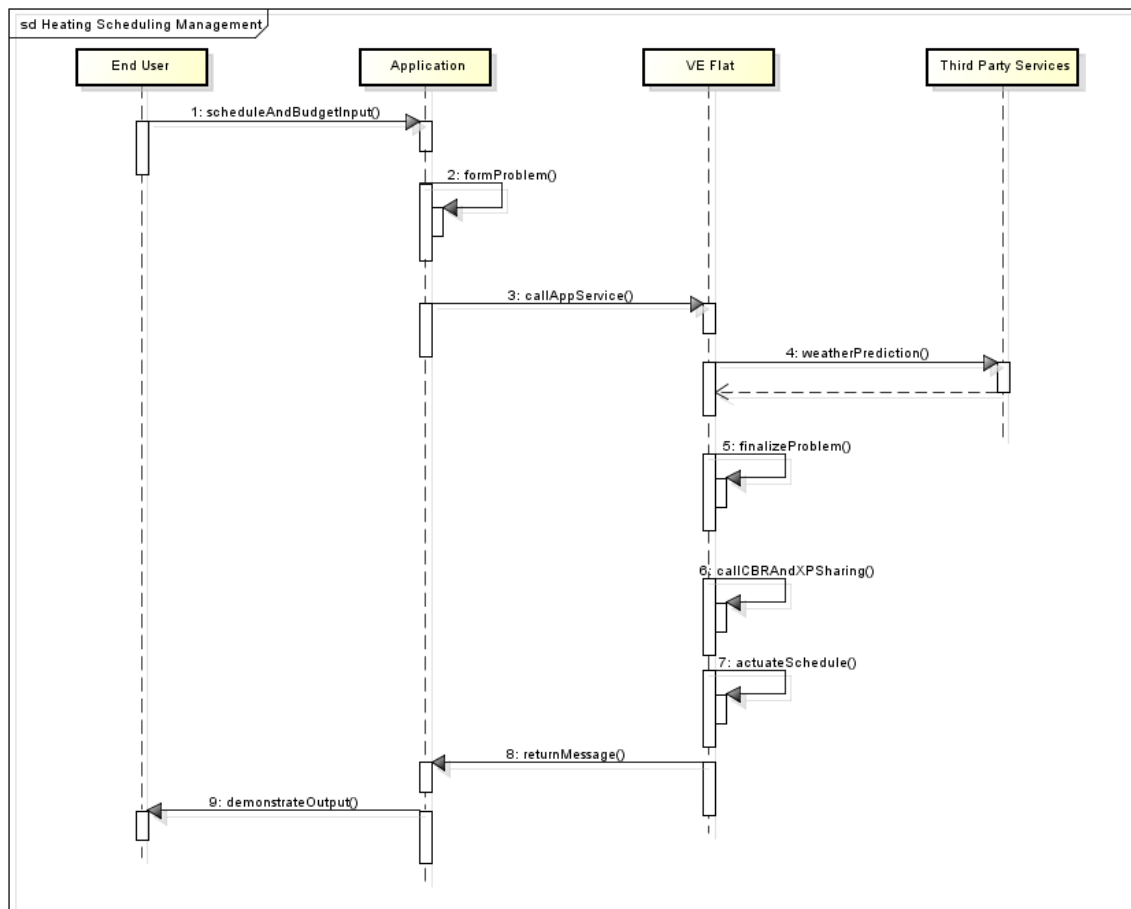


Figure 104: Steps for the Heating Schedule Management Application (General View)

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 communicate 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.

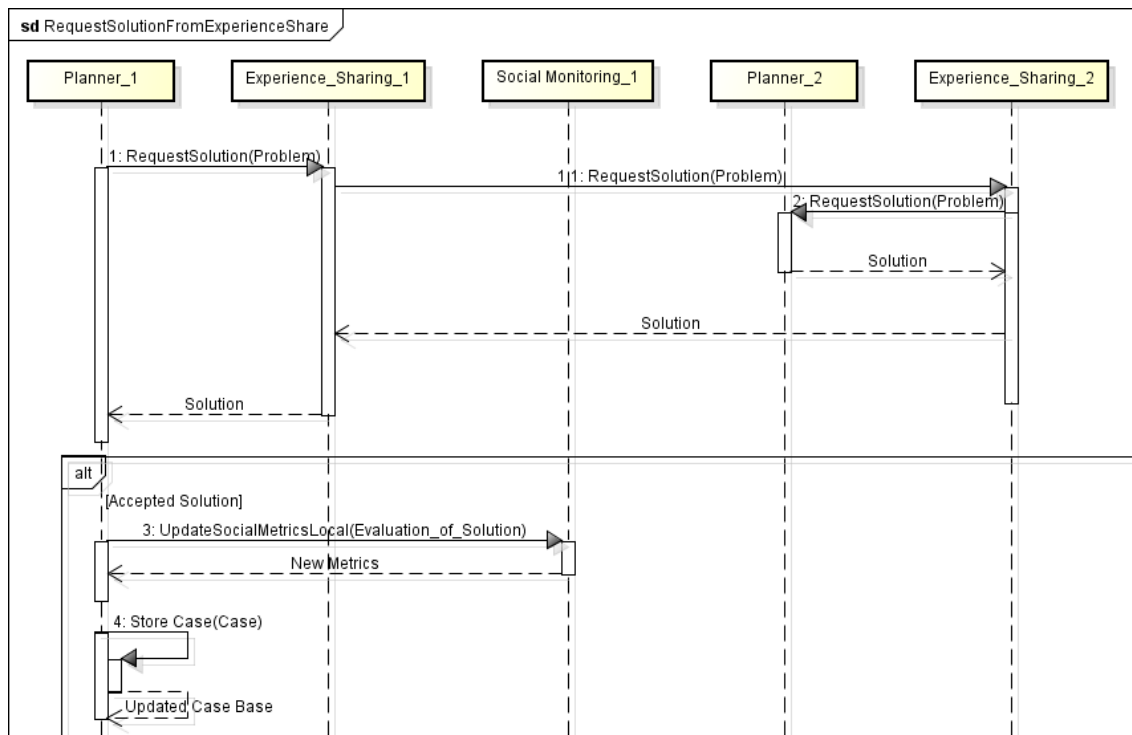


Figure 105: Steps for the Heating Schedule Management Application (CBR and XP Sharing parts)

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**

<b>Test Case Number Version</b>	<b>CAM_UNI_01</b>
<b>Test Case Title</b>	Complex Event handling
<b>Module tested</b>	Data Bridging, SAW FC's $\mu$ CEP engine, Planner FC, Experience Sharing FC, Social Monitoring FC
<b>Requirements addressed</b>	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]
<b>Initial conditions</b>	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)
<b>Expected results</b>	Handle irregularities in sensor operation by sharing alerts in remote VEs and by filtering affected sensors
<b>Owner/Role</b>	Application Developer
<b>Steps</b>	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)
<b>Passed</b>	Yes
<b>Bug ID</b>	N/A
<b>Problems</b>	None
<b>Required changes</b>	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. Therefore we decided to trigger it manually for testing purposes.

**Table 15: Heating Schedule Test Case**

<b>Test Case Number Version</b>	<b>CAM_UNI_02</b>
<b>Test Case Title</b>	Heating Schedule
<b>Module tested</b>	Data Bridging, Planner FC, Experience Sharing FC, Social Monitoring FC
<b>Requirements addressed</b>	[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
<b>Initial conditions</b>	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
<b>Expected results</b>	Form a Problem and associate a Solution Remote Solution acquisition (Experience Sharing) Alert User with text box pop up
<b>Owner/Role</b>	Application Developer
<b>Steps</b>	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
<b>Passed</b>	Yes
<b>Bug ID</b>	N/A
<b>Problems</b>	None
<b>Required changes</b>	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

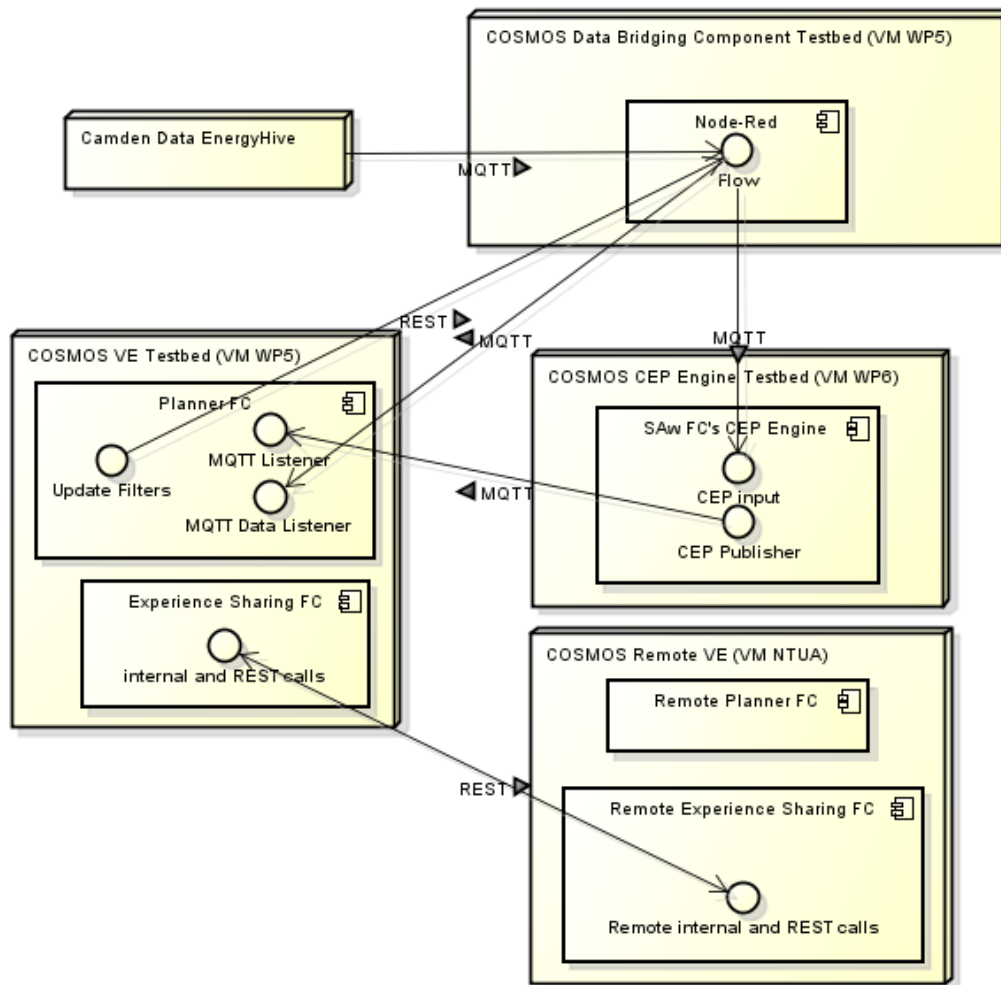


Figure 106: Camden Scenario Deployment Diagram

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®

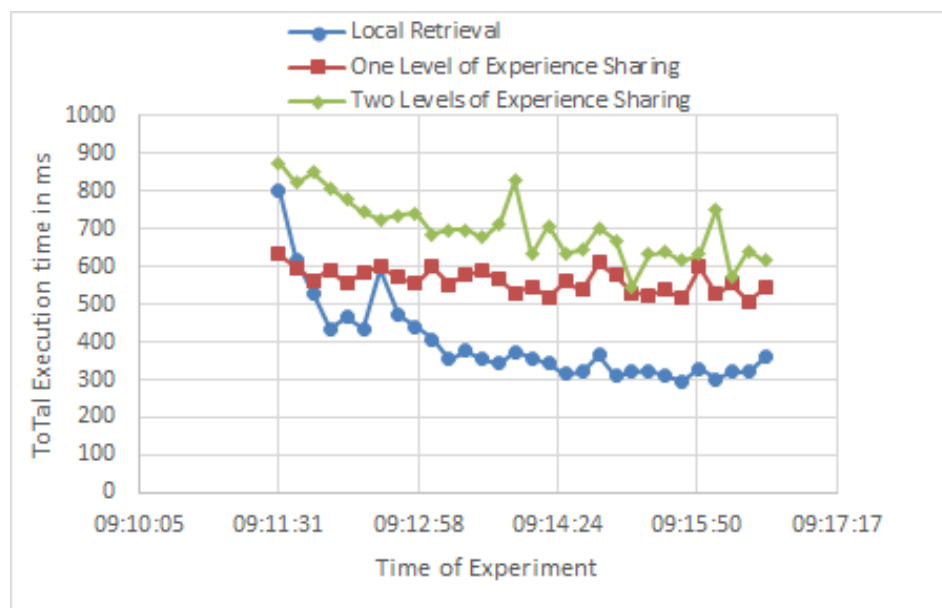


Figure 107: Low volume simulations

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.

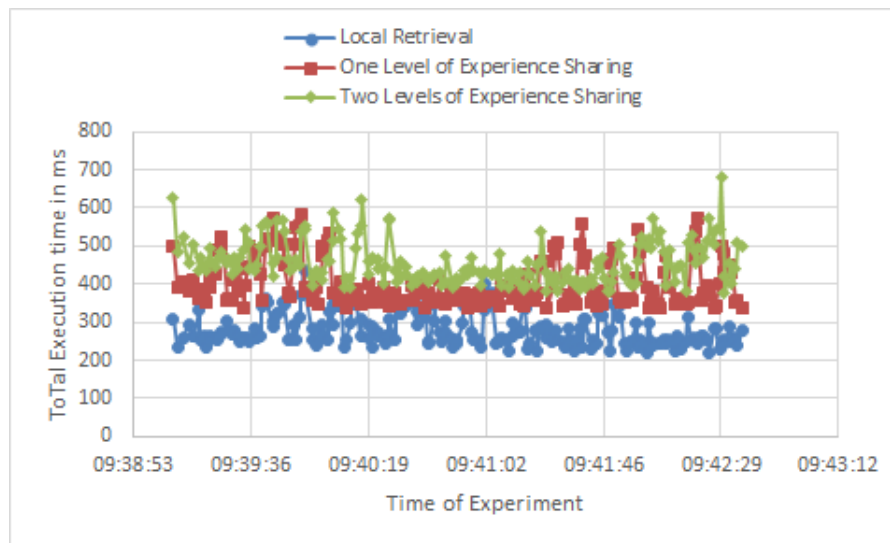


Figure 108: Medium volume simulations

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.

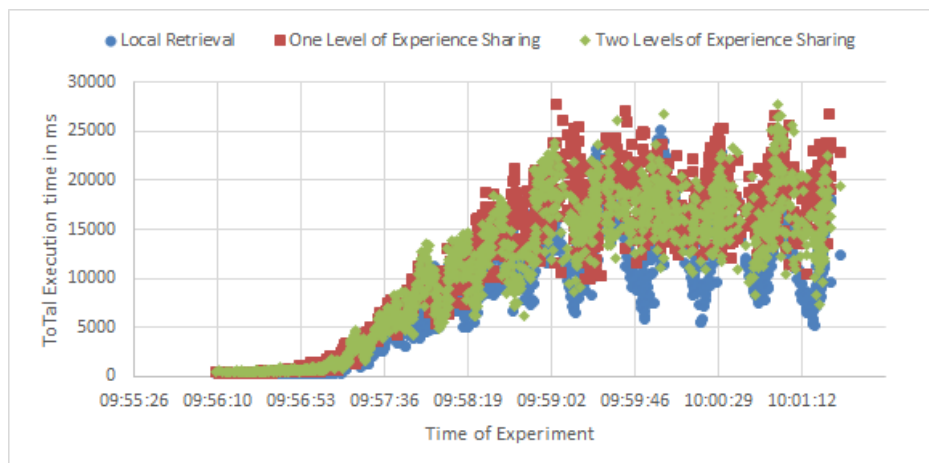


Figure 109: High volume simulations

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 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 ( $\mu$ CEP) 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  $\mu$ CEP 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.

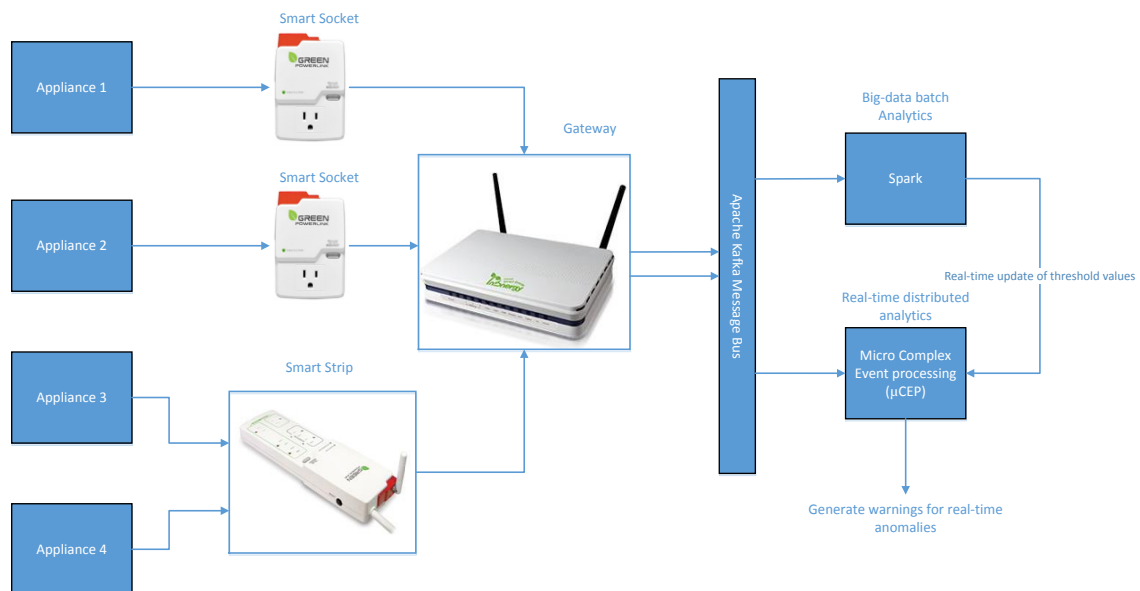


Figure 110: Real-time anomaly detection

### 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) CEP, Situational awareness and Machine Learning (3.2.4.1)

### 3.2.8.4 Message formats and configuration

Complex Event Anomaly is detected by CEP 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 CEP. CEP applies the pre-defined rules and detects an anomaly which is notified to the application developer through III gateway.

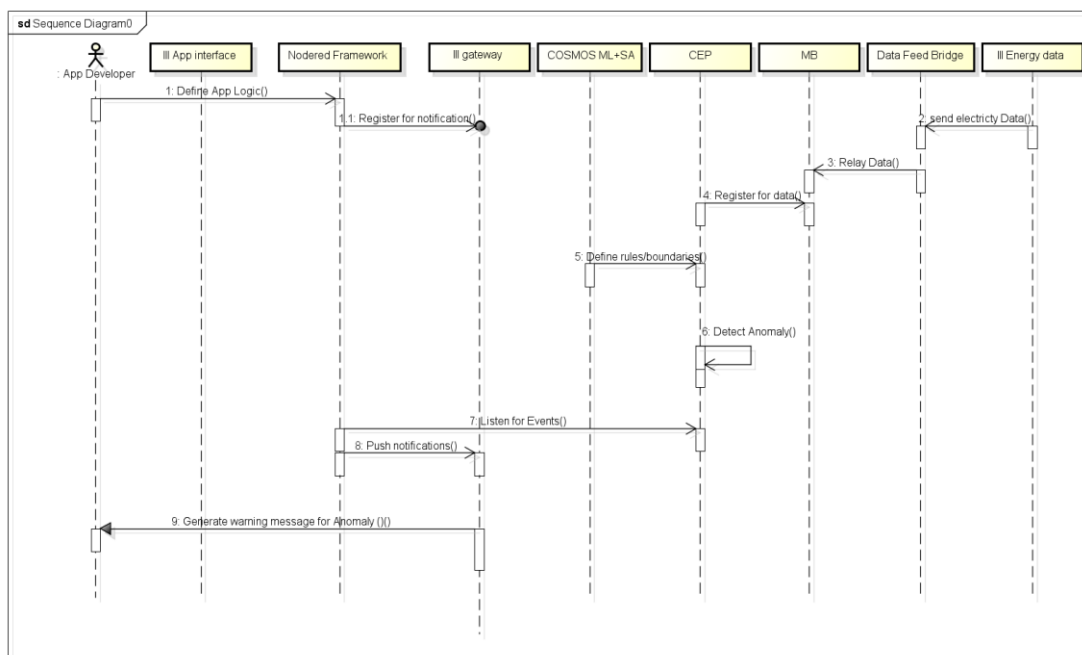


Figure 111: Sequence diagram for Taipei Scenario

### 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	<ol style="list-style-type: none"> <li>1. User connects the appliance to a smart plug</li> <li>2. The user registered for anomalies or unusual behavior</li> <li>3. The events identified from the COSMOS CEP are relayed through the MB to the App logic and from there to the III gateway.</li> </ol>
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  $\mu$ CEP. Node-RED provides an interface connect III Taipei data source to COSMOS message bus.

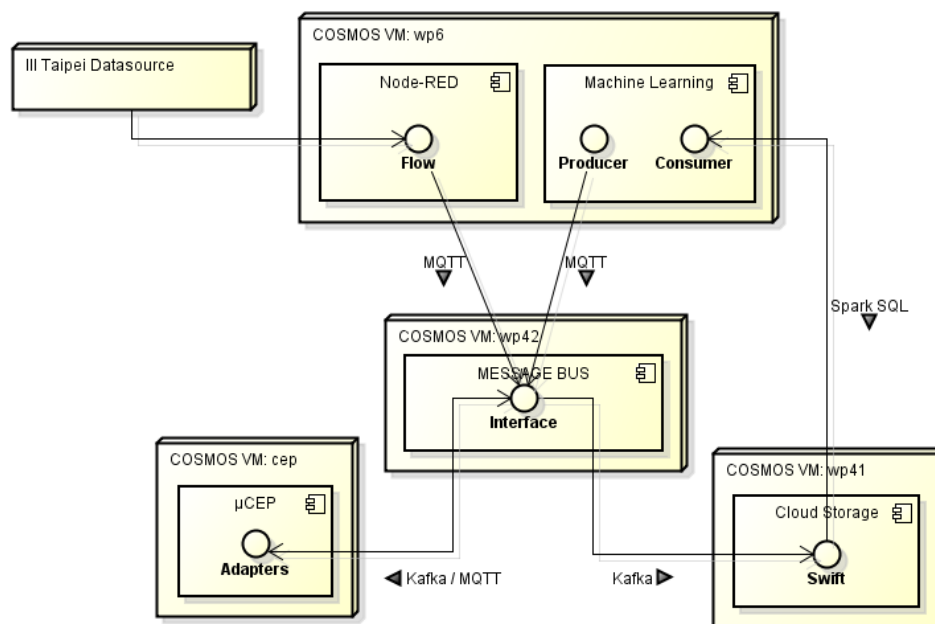


Figure 112: Deployment diagram for III Scenario

### 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. Conclusions

As a conclusion and following the implementation of the integration plan and process of Y1, as revised in the beginning of Y2, we can conclude that significant progress has been made during Y2 and in accordance with the goals of the plan.

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, the logic and complexity of which will be enhanced in Y3, while data feeds have been incorporated for all cases, 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, that is expected to continue, 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, which is one of the goals of the following period in terms of more support e.g. for events in the case of the Madrid UC. Furthermore, it has helped clarify and hide integration points, thus allowing for added underlying complexity to be achieved.

Finally, in relation to the integration plan, we have achieved a very satisfactory percentage of maturity levels, with 12 out of 14 subgoals achieved (while the other 2 are in progress), and 4 of them going beyond the anticipated level of maturity. Minor corrections have also been identified in terms of (3) subgoal extensions over integration periods as well as subgoal substitution in one case, thus feeding back to the integration plan and based on our experiences during this period.

## 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](http://en.wikipedia.org/wiki/Support_vector_machine)
- [3] K-Nearest Neighbours (KNN): [http://en.wikipedia.org/wiki/K-nearest\\_neighbors\\_algorithm](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 7.6.2 “Integration Plan (Updated)”, ICCS/NTUA and other partners, April 2015
- [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>

## Annex A: Unit/Component Tests

### Data Mapping

Table 18: Results for Data Mapping Unit Test #1

<b>Test Case Number Version</b>	<b>4_DM_1</b>
<b>Test Case Title</b>	Storing live data, coming from the Message Bus, in the Cloud Storage.
<b>Module tested</b>	Data Mapping
<b>Requirements addressed</b>	4.1, 4.2
<b>Initial conditions</b>	RabbitMQ service is installed and running Hildebrand data stream is available
<b>Expected results</b>	Data are stored as objects with timestamps, metadata and a lower threshold for the size.
<b>Owner</b>	Achilleas Marinakis
<b>Steps</b>	Execute: /NTUA/DataMapping/target java -jar DataMapping-Y1.jar Hildebrand
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	None

Table 19: Results for Data Mapping Unit Test #2

<b>Test Case Number Version</b>	<b>4_DM_2</b>
<b>Test Case Title</b>	Storing historical data, coming from a static file, in the Cloud Storage
<b>Module tested</b>	Data Mapping
<b>Requirements addressed</b>	4.1, 4.2
<b>Initial conditions</b>	The path to the static file is provided.
<b>Expected results</b>	Data are stored as objects with timestamps and id metadata and a lower threshold for the size.
<b>Owner</b>	Achilleas Marinakis
<b>Steps</b>	Execute: /NTUA/DataMapping/target java -jar DataMapping-Y1.jar Surrey
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	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 20: Results for Prediction Unit Test #1

<b>Test Case Number Version</b>	<b>6_ML_1</b>
<b>Test Case Title</b>	6_ML_1
<b>Module tested</b>	Prediction
<b>Requirements addressed</b>	6.2, 6.3, 6.6, 6.22, 6.23
<b>Initial conditions</b>	Python installed with numpy, scikit, pandas and statsmodel. Time series historical data file.
<b>Expected results</b>	Machine Learning model which can predict the user Occupancy state.
<b>Owner</b>	University of Surrey
<b>Steps</b>	Install the required libraries. Store the historical data file. Run the code.
<b>Passed</b>	Yes
<b>Bug ID</b>	N/A
<b>Problems</b>	No
<b>Required changes</b>	No

## Semantic Description and Retrieval

Table 21: Results for Semantic Description and Retrieval Unit Test #1

<b>Test Case Number Version</b>	<b>5_SDR_1</b>
<b>Test Case Title</b>	VE semantic description and retrieval
<b>Module tested</b>	Triple store access
<b>Requirements addressed</b>	5.0, 5.2, UNI.414, UNI.416, UNI.425, UNI.426
<b>Initial conditions</b>	First version of the semantic model available. Test triple store installed. Test query API available.
<b>Expected results</b>	Successful recording of the VE triples and matches between the retrieval results and the expected VE descriptions.
<b>Owner</b>	SIEMENS
<b>Steps</b>	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.
<b>Passed</b>	Yes
<b>Bug ID</b>	
<b>Problems</b>	
<b>Required changes</b>	

**Table 22: Results for Semantic Description and Retrieval Unit Test #2**

<b>Test Case Number Version</b>	<b>5_SDR_2</b>
<b>Test Case Title</b>	VE description validation
<b>Module tested</b>	Triple store access
<b>Requirements addressed</b>	5.0, 5.2, UNI.414, UNI.416, UNI.425, UNI.426
<b>Initial conditions</b>	First version of the semantic model available. Test triple store installed. Test query API available.
<b>Expected results</b>	Inconsistent VE descriptions should not be stored into the triple store.
<b>Owner</b>	SIEMENS
<b>Steps</b>	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.
<b>Passed</b>	Yes
<b>Bug ID</b>	
<b>Problems</b>	
<b>Required changes</b>	

## Privelets

Table 23: Results for Privelets Unit Test #1

<b>Test Case Number Version</b>	<b>3_PR_1</b>
<b>Test Case Title</b>	Privacy
<b>Module tested</b>	Privelets
<b>Requirements addressed</b>	UR13
<b>Initial conditions</b>	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
<b>Expected results</b>	VEs exchange public data with anonymity
<b>Owner</b>	Achilleas Marinakis
<b>Steps</b>	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
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	None

## Planner

Table 24: Results for Planner Unit Test #1

<b>Test Case Number Version</b>	<b>5_PL_1</b>
<b>Test Case Title</b>	Notification Service and CBR
<b>Module tested</b>	Planner
<b>Requirements addressed</b>	UNI. 704, 706, 708, 715, 719 5.31 5.29, UNI.010 5.21 5.10 5.9, UNI.251
<b>Initial conditions</b>	Have a CB inside the test bed. Have the VE and app simulation jar inside the test bed.
<b>Expected results</b>	The answer to the notification of the User Generated Event in the GUI. Command Line or Terminal Log with similarities retrieved and queries used.
<b>Owner</b>	Panagiotis Bourellos, Orfefs Voutyras
<b>Steps</b>	Place the CB in the localstore folder of the home directory. Open command line or terminal.



	<p>Navigate to the jar folder.</p> <p>Execute “java –jar DemoProjectMaven-1.0-SNAPSHOT.jar”.</p> <p>Keep the terminal or cmd open to view log info and locate the GUI.</p> <p>Enter a set of temperature and time that exist locally (time: 3535 and temp: 26).</p> <p>Press “Send Notification”.</p>
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	None

## Experience Sharing

Table 25: Results for Experience Sharing Unit Test #1

<b>Test Case Number Version</b>	<b>6_ES_1</b>
<b>Test Case Title</b>	Experience Sharing and returned Solution
<b>Module tested</b>	Planner, Experience Sharing
<b>Requirements addressed</b>	Same as those in 5_PL_1 plus: 5.11, 5.12, 6.8, 6.18, 6.19, 6.34, 6.35
<b>Initial conditions</b>	<p>Have a CB and Friend List inside the test bed.</p> <p>Have the VE and app simulation jar inside the test bed.</p> <p>Have one or both auxiliary XP sharing VEs active in the VMs, along with their CBs and Friend Lists.</p>
<b>Expected results</b>	<p>The answer to the notification of the User Generated Event in the GUI</p> <p>Command Line or Terminal Log with similarities retrieved and queries used</p> <p>The Logs of auxiliary VEs in the VMs with the Planner result and the Experience Sharing steps</p>
<b>Owner</b>	Panagiotis Bourellos, Orfefs Voutyras
<b>Steps</b>	<p>Place the CB and Friend List in the localstore folder of the home directory;</p> <p>Open command line or terminal;</p> <p>Navigate to the jar folder;</p> <p>Execute “java –jar DemoProjectMaven-1.0-SNAPSHOT.jar”;</p> <p>Keep the terminal or cmd open to view log info and locate the GUI;</p> <p>Boot the VEs in their respective VMs;</p> <p>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);</p> <p>Press “Send Notification”</p>
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	None

## Hardware Security Board

Table 26: Results for Hardware Security Board Unit Test #1

<b>Test Case Number Version</b>	<b>3_HSB_1</b>
<b>Test Case Title</b>	Communication over standard interfaces
<b>Module tested</b>	Communication module
<b>Requirements addressed</b>	3.1
<b>Initial conditions</b>	Configure standard communication interfaces (I2C, SPI, USB, Ethernet)
<b>Expected results</b>	Data flow can take place over the specified interfaces. Sensors and the Ethernet network can be accessed.
<b>Owner</b>	Leonard Pitu
<b>Steps</b>	<ol style="list-style-type: none"> <li>1. Load the hardware design (*.bit file)</li> <li>2. Load the firmware</li> <li>3. Boot the firmware</li> <li>4. Obtain an IP address</li> <li>5. Read the sensors</li> <li>6. Make the data available over ETH.</li> </ol>
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	None

Table 27: Results for Hardware Security Board Unit Test #2

<b>Test Case Number Version</b>	<b>3_HSB_2</b>
<b>Test Case Title</b>	Secure the data flow (simulation)
<b>Module tested</b>	Cryptographic accelerator
<b>Requirements addressed</b>	3.3
<b>Initial conditions</b>	Configure the FPGA with the hardware design
<b>Expected results</b>	The cryptographic accelerator shall meet the FIPS-180 standard.
<b>Owner</b>	Leonard Pitu
<b>Steps</b>	<p>Standard test vectors</p> <ol style="list-style-type: none"> <li>1. Load the hardware module in the simulator (ModelSIM)</li> <li>2. Load the test cases according to FIPS-180</li> <li>3. Run the tests</li> </ol> <p>Random test vectors</p> <ol style="list-style-type: none"> <li>1. Load the hardware module in the simulator (ModelSIM)</li> <li>2. Load private test vectors generated with Crypto++</li> <li>3. Run the tests</li> </ol>
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	None

Table 28: Results for Hardware Security Board Unit Test #3

<b>Test Case Number Version</b>	<b>3_HSB_3</b>
<b>Test Case Title</b>	Secure the data flow (board)
<b>Module tested</b>	Cryptographic accelerator
<b>Requirements addressed</b>	3.3
<b>Initial conditions</b>	Configure the FPGA with the hardware design.
<b>Expected results</b>	The cryptographic accelerator shall meet the FIPS-180 standard.
<b>Owner</b>	Leonard Pitu
<b>Steps</b>	Standard test vectors <ol style="list-style-type: none"> <li>1. Load the hardware design (*.bit file)</li> <li>2. Load the test cases according to FIPS-180</li> <li>3. Run the tests</li> </ol> Random test vectors <ol style="list-style-type: none"> <li>1. Load the hardware design (*.bit file)</li> <li>2. Load private test vectors generated with Crypto++</li> <li>3. Run the tests</li> </ol>
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	None

Table 29: Results for Hardware Security Board Unit Test #4

<b>Test Case Number Version</b>	<b>3_HSB_4</b>
<b>Test Case Title</b>	HSB enrolment
<b>Module tested</b>	Cryptographic accelerator
<b>Requirements addressed</b>	3.7
<b>Initial conditions</b>	Configure the FPGA with the hardware design Load the software (Linux + drivers + test application)
<b>Expected results</b>	Use the ECDH to securely exchange the key
<b>Owner</b>	Leonard Pitu
<b>Steps</b>	<ol style="list-style-type: none"> <li>1. Load the hardware design (*.bit file)</li> <li>2. Load the software</li> <li>3. Boot the system</li> <li>4. Run the test cases and fetch the key from Keystone</li> </ol>
<b>Passed</b>	Yes
<b>Bug ID</b>	None
<b>Problems</b>	None
<b>Required changes</b>	None