COSMOS

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement Nº 609043

D7.7.1 Integration of Results (Year 1)

WP7: Use cases Adaptation, Integration and Experimentation

Version: 1.6
Due Date: 30/09/2014
Delivery Date: 15/02/2015
Nature: Report
Dissemination Level: Public

Lead partner: ICCS/NTUA
Authors: George Kousiouris, Achilleas Marinakis, Panagiotis Bourelos, Orfefs Voutyras, Vassilis Psaltopoulos (ICCS/NTUA), Jozef Krempasky (ATOS), Paula Ta-Shma (IBM), Francois Carrez, Adnan Akbar (UniS), Bogdan Târnaucă, Leonard Pitu (SIEMENS)

Internal reviewers: Juan Sancho (ATOS), NTUA

www.iot-cosmos.eu

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

Version Control:

Version	Date	Author	Author's Organization	Changes
0.1	28/07/2014	Achilleas Marinakis	ICCS/NTUA	Circulating the ToC
0.2	13/08/2014	Paula Ta-Shma	IBM	Adding IBM contribution
0.3	29/09/2014	Achilleas Marinakis, Panagiotis Bourelos	ICCS/NTUA	Updating Test Cases and Demos Scenarios
0.4	01/10/2014	Achilleas Marinakis and co-authors	ICCS/NTUA, ATOS, UniS	Updating Unit Tests, Model Validation and Future Work
0.5	02/10/2014	Achilleas Marinakis and co-authors	ICCS/NTUA, UniS	Updating Testbed Instantiation and Future Work
0.6	09/10/2014	Achilleas Marinakis and co-authors	ICCS/NTUA, UniS, SIEMENS	Version ready for internal review
1.0	15/10/2014	Achilleas Marinakis	ICCS/NTUA	Version for submission
RE_SUBMIT 1.05	23/1/2015	George Kousiouris	ICCS/NTUA	New ToC adapted to D7.6.1 template
RE_SUBMIT 1.1	05/02/2015	Paula Ta-Shma	IBM	Added section 2.2.1.2
RE_SUBMIT 1.2	06/02/2015	Achilleas Marinakis	ICCS/NTUA	Added section 2.2.1.1
RE_SUBMIT 1.3	08/02/2015	Adnan Akbar	UniS	Added section 2.2.1.4
RE_SUBMIT 1.4	09/02/2015	Panagiotis Bourelos	ICCS/NTUA	Added section 2.2.2.1 and 2.2.2.2
RE_SUBMIT 1.5	10/02/2015	George Kousiouris	ICCS/NTUA	Adapted inputs and redistributed content
RE_SUBMIT 1.6	13/02/2015	Juan Sancho	ATOS	Final Review

Table of Contents

Executive Summary 8
1. Introduction 10
1.1. Purpose of the Document 10
1.2. Integration Periods within the scope of this document 10
1.3. Document Structure 10
2. Integration Stage 1 (M10-M16) 11
2.1. Overview from Integration plan 11
2.2. Performed Tests 12
2.2.1. Subgoal: Data Management and Analytics Tests 12
2.2.2. Subgoal: Autonomous Behaviour of VEs 34
2.2.3. Component Tests 46
2.2.4. Occupancy detection model validation 46
2.3. Observed Deviations and Applied Mitigation Strategies 49
2.4. Future steps with regard to the advancements of this period 50
2.4.1. Security aspects 50
2.4.2. Data Analytics and Storage 50
2.4.3. Autonomous Behaviour of VEs 50
2.4.4. Modelling Aspects 50
3. Conclusions and Future Steps 51
References 52
Annex A: Unit/Component Tests 53
Data Mapping 53
Message Bus 54
Cloud Storage – Metadata Search 54
Cloud Storage – Storlets 54
Event Detection & Situational Awareness @Platform 54
Prediction 55
Semantic Description and Retrieval 55
Privelets 57
Planner 58
Event Detection & Situational Awareness @VE 58
Experience Sharing 59
Hardware Security Board 59

Table of Figures

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

Figure 34: Input data format for Modelling case 32
Figure 35: Model Training using Spark MLlib, Storlets and Swift..... 32
Figure 36: Runtime Prediction using the trained model 32
Figure 37: Sequence Diagram for Modelling and Storage Analytics Subsystem..... 33
Figure 38: Modelling and Storage Analytics Deployment Diagram 34
Figure 39: Autonomous Behavior of VEs with minimum platform integration subsystem 36
Figure 40: Autonomous Behavior of VEs application GUI..... 36
Figure 41: Components of the main flat-VE..... 37
Figure 42: Case Base and Friend List examples..... 37
Figure 43: Visual representation of all scenario possibilities..... 38
Figure 44: Autonomous Behavior of VEs- Automated Event Detection and Incorporation of COSMOS Platform 40
Figure 45: Visual representation of the platform integration scenario 40
Figure 46: Sample DOLCE configuration file of freezing event detection..... 41
Figure 47: Example of a data stream as input to the CEP 41
Figure 48: Example of code for MB subscription for listening. 42
Figure 49: Autonomous Behaviour of VEs Deployment Diagram 43
Figure 50: Autonomous Behaviour of VEs System Sequence Diagram..... 43
Figure 51: Enrich Case Base Content Sequence Diagram 44
Figure 52: React to Event Sequence Diagram 44
Figure 53: RequestSolution(Problem) Sequence Diagram..... 45
Figure 54: UpdateSocialMetricsLocal(Evaluation_of_Solution) Sequence Diagram 45
Figure 55: Performance of Classifiers..... 48
Figure 56: F-measure relation with training data 48
Figure 57: Time Complexity..... 49

List of Tables

Table 1: Data Feed, Annotation and Storage Test Case 17
Table 2: Planner with minimum integration to the COSMOS Platform Test Case 38
Table 3: Planner full integration with COSMOS Platform 42
Table 4: Results for Data Mapping Unit Test #1 53
Table 5: Results for Data Mapping Unit Test #2 53
Table 6: Results for Message Bus Unit Test #1 54
Table 7: Results for Event Detection @Platform Unit Test #1 54
Table 8: Results for Prediction Unit Test #1 55
Table 9: Results for Semantic Description and Retrieval Unit Test #1 55
Table 10: Results for Semantic Description and Retrieval Unit Test #2 56
Table 11: Results for Privelets Unit Test #1 57
Table 12: Results for Privelets Unit Test #2 57
Table 13: Results for Planner Unit Test #1 58
Table 14: Results for Experience Sharing Unit Test #1 59
Table 15: Results for Hardware Security Board Unit Test #1 59
Table 16: Results for Hardware Security Board Unit Test #2 60
Table 17: Results for Hardware Security Board Unit Test #3 60
Table 18: Results for Hardware Security Board Unit Test #4 61

Acronyms

Acronym	Meaning
API	Application Programming Interface
CB	Case Base
CBR	Case Based Reasoning
CEP	Complex Event Processing
CRS	Coordinate Reference System
D	Deliverable
GPS	Global Positioning System
GUI	Graphical User Interface
GVE	Group Virtual Entity
HSB	Hardware Security Board
HTTP	Hyper Text Transfer Protocol
ID	Identifier
IoT	Internet of Things
IP	Integration Point
JSON	Java-Script Object Notation
KNN	K-Nearest Neighbours
MB	Message Bus
ML	Machine Learning
PM	Person Month
REST	Representational State Transfer
SVM	Support Vector Machine
UC	Use Case
UI	User Interface
UTM	Universal Transverse Mercator
VE	Virtual Entity
VM	Virtual Machine
WP	Work Package
XP	Experience

Executive Summary

This report presents the results of the integration process carried out in year 1 of the COSMOS project. We started the process by 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.

Following, and by receiving input from the Integration Plan, the subsystems to be tested have been defined. The tests have been centered around six demo scenarios, all deriving from the aforementioned subsystems. Relative sequence diagrams have been extracted, indicating the testing process. Through the demos, we aim to highlight and test the COSMOS functionalities such as:

- Data flow from different types of sources, adaptively annotated, grouped and stored on a Cloud based object storage;
- Metadata search and visualization, based on the capabilities of created specific Storlets that work directly on the used data. This functionality can be adapted per case and based on the needed computational transformations and executed near the source of information (the data storage);
- Increased security, achieved through the incorporation of a hardware based encryption component, for the transfer of information from the source to the COSMOS platform;
- Increased privacy, applied also as a Storlet concept, to enable a multi-level access policy on the stored data, conceiling per case sensitive information (e.g. image facial blurring);
- Decentralised and autonomous management and knowledge acquisition of VEs, enriched with social interactions;
- Increased modelling capabilities, incorporating data preprocessing and integrating Apache Spark with object storage (Openstack Swift) for data analysis and modelling using historical data.

For each of these cases, the defined scenario implementations and test results 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 following years of the project. Specifically for the testing of the modelling approach, we have developed a model for predicting events by inferring high-level knowledge from raw IoT data by pattern recognition. Different Machine Learning algorithms were developed for pattern recognition purposes. In section 2.2.4, we validate the model by comparing the prediction results to the recorded events at particular times. We used an Occupancy Detection scenario for the validation purpose in which we are inferring occupancy states of users by observing electricity patterns of users. We used several variants of Support Vector Machine (SVM) [2] and K Nearest Neighbors (KNN) [3] in our work. The validation of results showed that the efficiency of KNN is highest (more than 90%) as compared to SVM variants.

We developed Unit/Components Tests for the modules that are involved in the system sequences at this time. The results of these Tests are provided in the template that was introduced in D7.6.1, including the requirements (COSMOS requirements are listed in the relevant annex of D2.2.1 [4]) that are addressed, either partially or fully, from them. The component tests are included in the Annex, since one component may participate in more

than one subsystem. Thus, in order not to repeat information, we crossreference them directly from that location.

As a conclusion, the goals identified for this first integration period have been met, resulting in an initial running instance of a COSMOS Platform Provider, offering increased applied functionalities that were tested through scenarios. Those functionalities may be reused in the context of the actual COSMOS applications that are going to be created in the near future and indicating the path of adding new functionalities. These aspects are expected to be combined in the near future in order to further fulfill the project's scope and objectives.

1. Introduction

1.1. Purpose of the Document

D7.7.1 has the purpose of consolidating the results from the work performed in the technical tasks, aiming to provide the integrated view of COSMOS outcomes. In order to achieve this goal, a process has been already defined in the context of D7.6.1 [1], that is concretized in specific actions in this document, describing the concrete integration points and necessary functionalities in them.

This includes the definition of component and 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. 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 can be reused or instantiated appropriately in the project UC scenarios, which are expected to be realized in the following months of the project.

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 correspoding to this document are:

- Integration period 1, with duration from PM10 to PM16

In the next iteration of the document (PM25), the two intermediate integration periods will also be included (PM17-22 and PM23-25).

1.3. Document Structure

The document is structured as follows:

- In Chapter 2, the integration period fields are described, 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. Sequence diagrams are also included, to indicate the concrete interactions between the components.
- In Chapter 3, conclusions are portrayed with relation to this period and its outcomes.
- In the Annex we have included the unit tests on a component level, for the components that are part of the defined subsystems for this period. They have been included at that location in order not to repeat information, since a component may participate in more than one subsystem.

2. Integration Stage 1 (M10-M16)

2.1. Overview from Integration plan

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

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

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

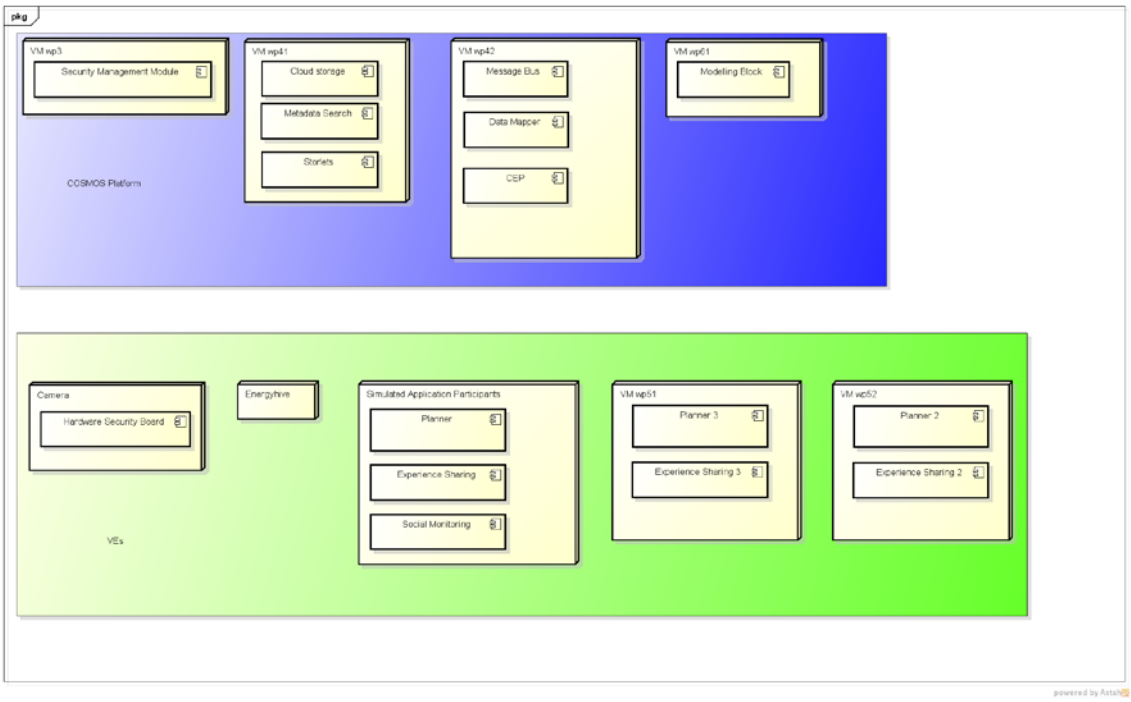

Figure 1: COSMOS Platform Overall Deployment Diagram

2.2. Performed Tests

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

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

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

2.2.1. Subgoal: Data Management and Analytics Tests

This subgoal involves the following components:

Message Bus

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

Data Mapper

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

Swift Cloud Storage

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

Metadata Indexing

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

Storlets

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

Analytics

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

Security Framework

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

The aforementioned components are combined in the following scenarios.

2.2.1.1 Data Feed, Annotation and Storage

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

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

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

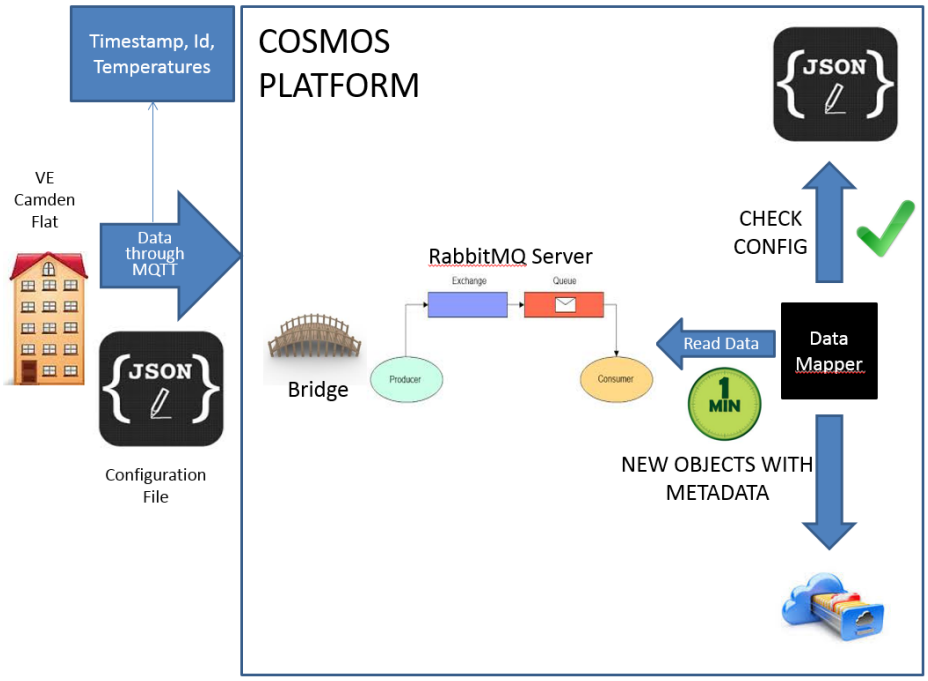

Figure 2: Data Feed, Annotation and Storage Scenario

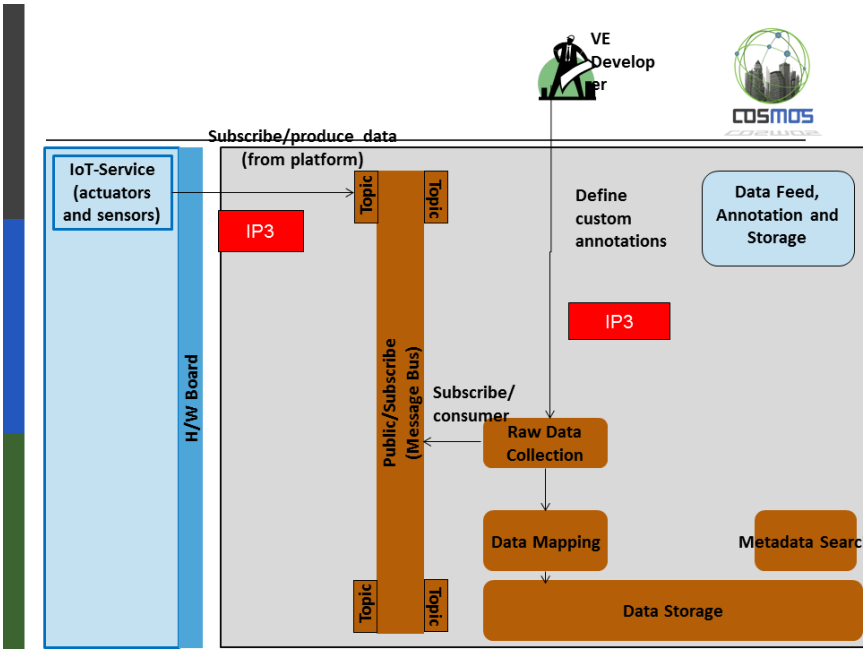

Figure 3: Data Feed, Annotation and Storage Subsystem

Link with COSMOS Message Bus

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

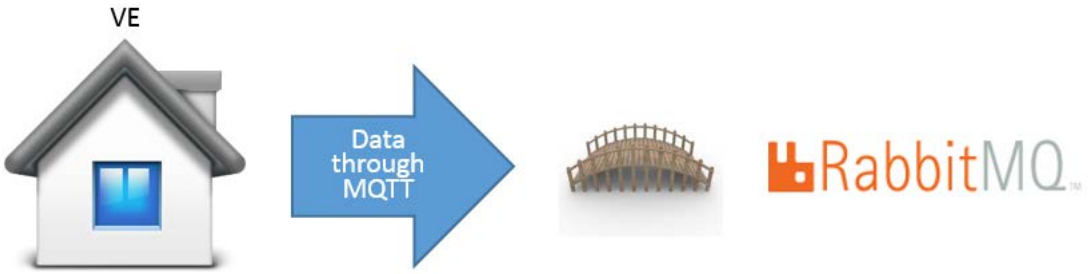

Figure 4: Subscribe/produce data adaptation

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

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

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

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

Figure 5: Bridge Configuration

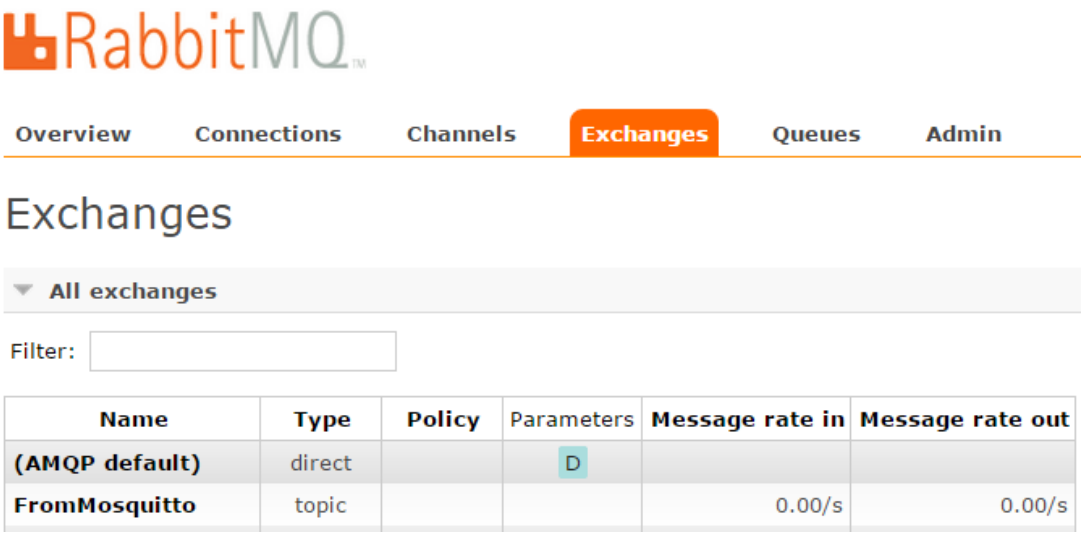

Figure 6: RabbitMQ HTTP-based API - Topic "FromMosquitto"

Data Format

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

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


Figure 7: JSON Message from the Camden UC

Metadata Annotation

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

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


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


Figure 8: Configuration File

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

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

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

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

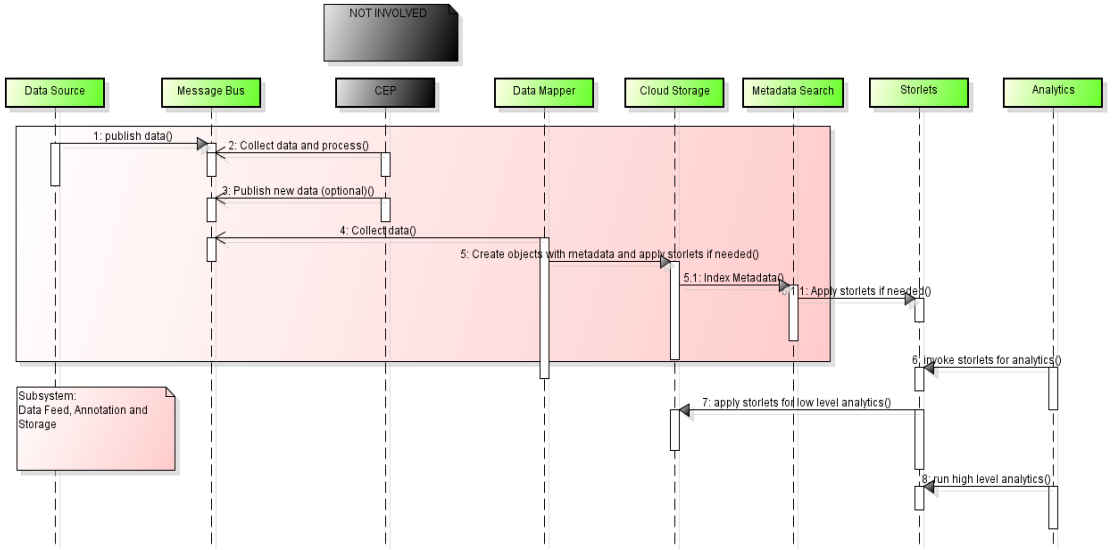

Figure 9: Sequence Diagram for Data Feed, Annotation and Storage Subsystem

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

Table 1: Data Feed, Annotation and Storage Test Case

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

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

Figure 10: Data Object Body

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

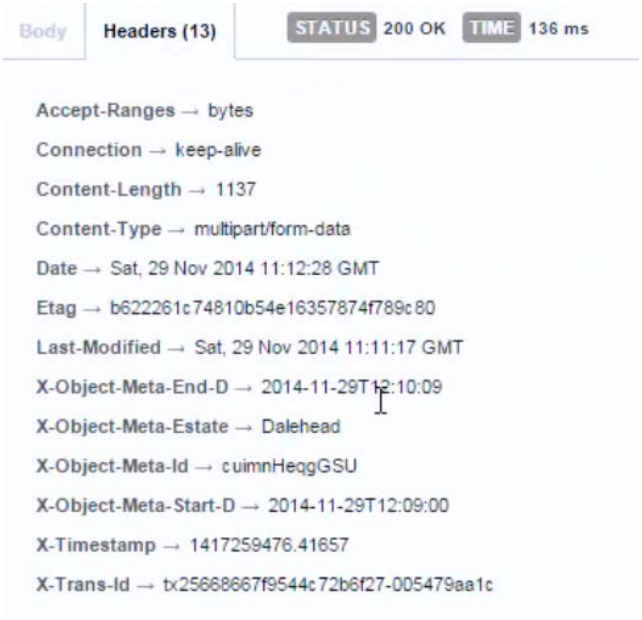

Figure 11: Data Object Metadata

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

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

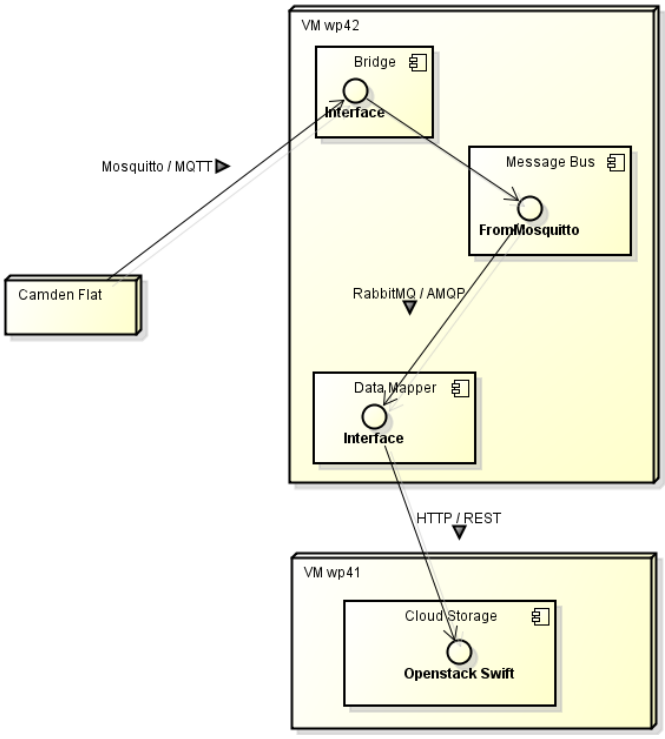

Figure 12: Data Feed, Annotation and Storage Deployment Diagram

2.2.1.2 Storage and Analytics on Metadata

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

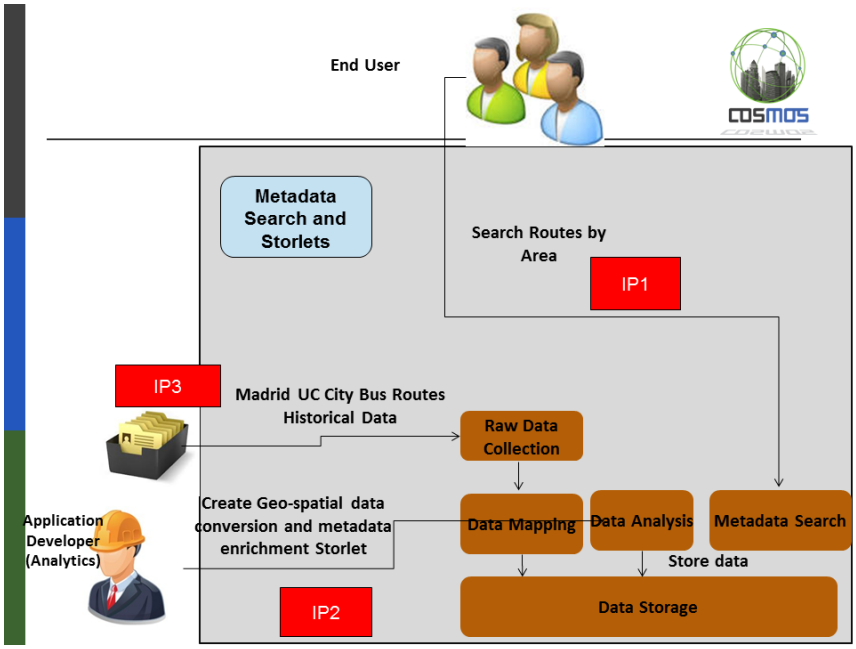

Figure 13: Metadata Search and Storlets Subsystem

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

Scenario

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

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

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

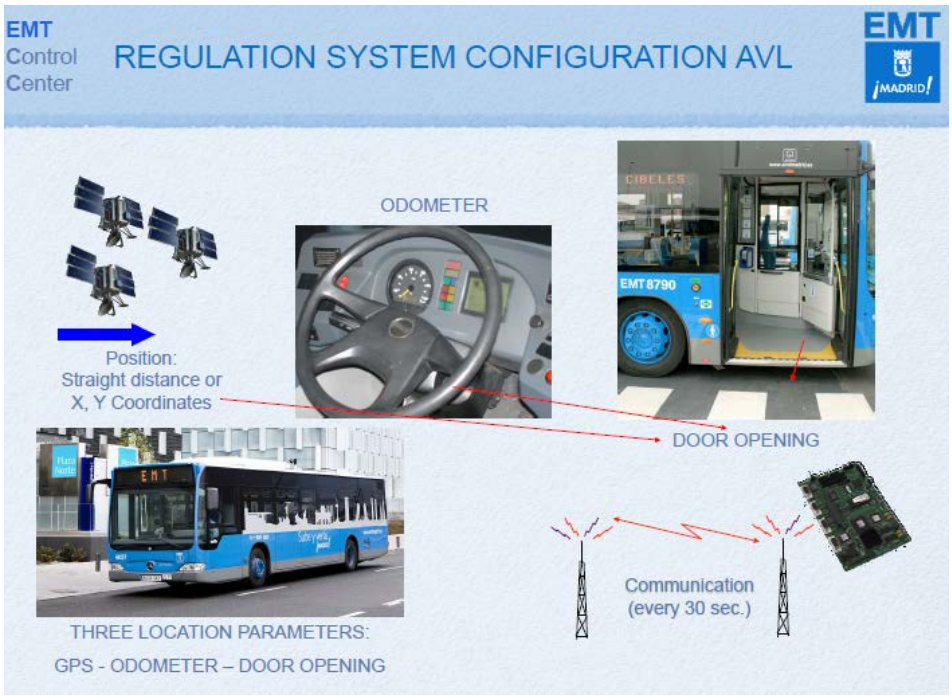

Figure 14: Overview of EMT available data characteristics

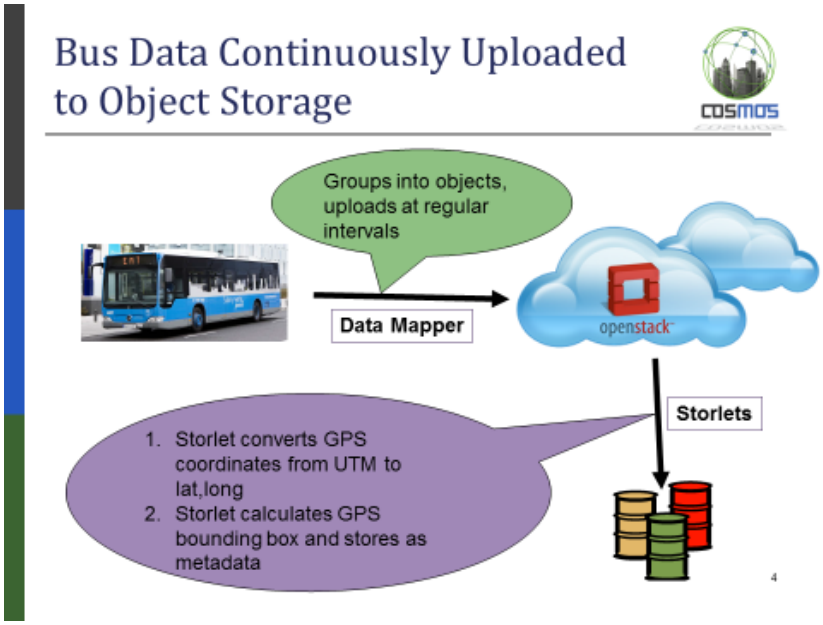

Figure 15: Overview of Metadata and Storlets Scenario

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

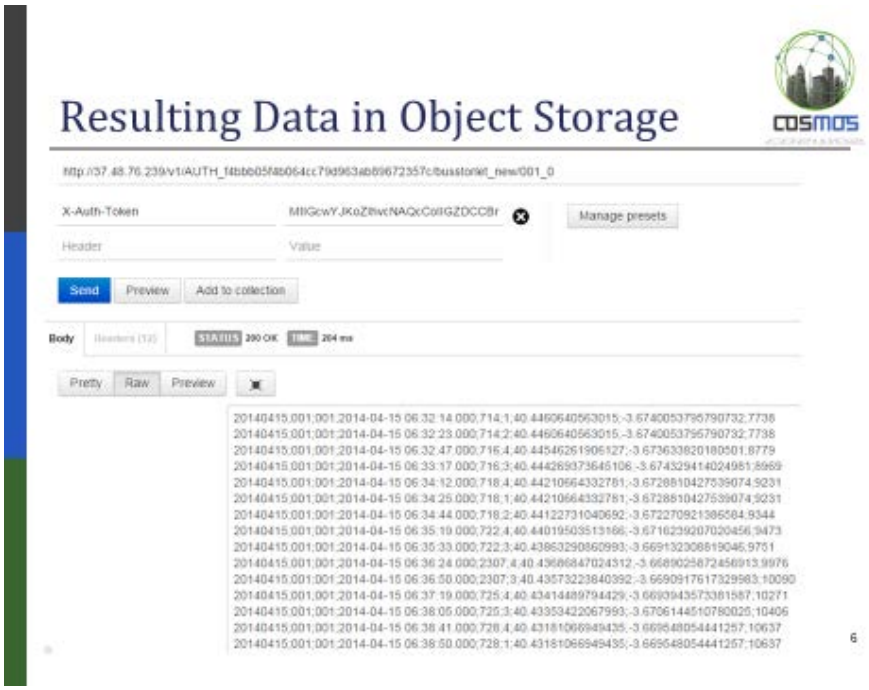

Figure 16: Data Format in Object Storage

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

Figure 17: Metadata insertion in Object Storage

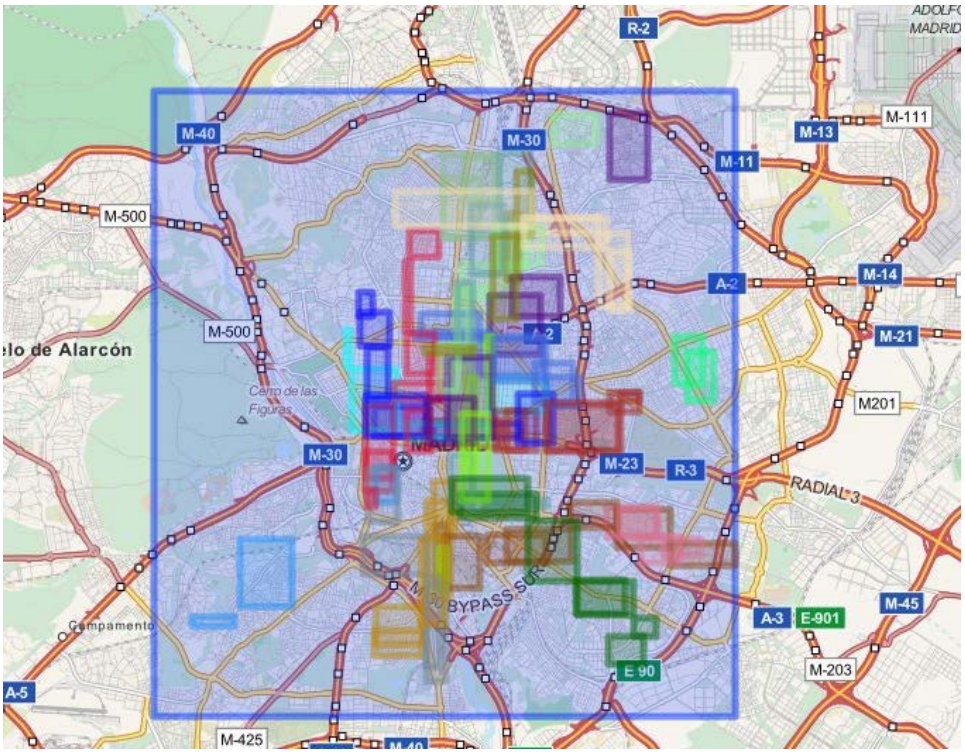

Figure 18: End user interface for the Metadata Search subsystem

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

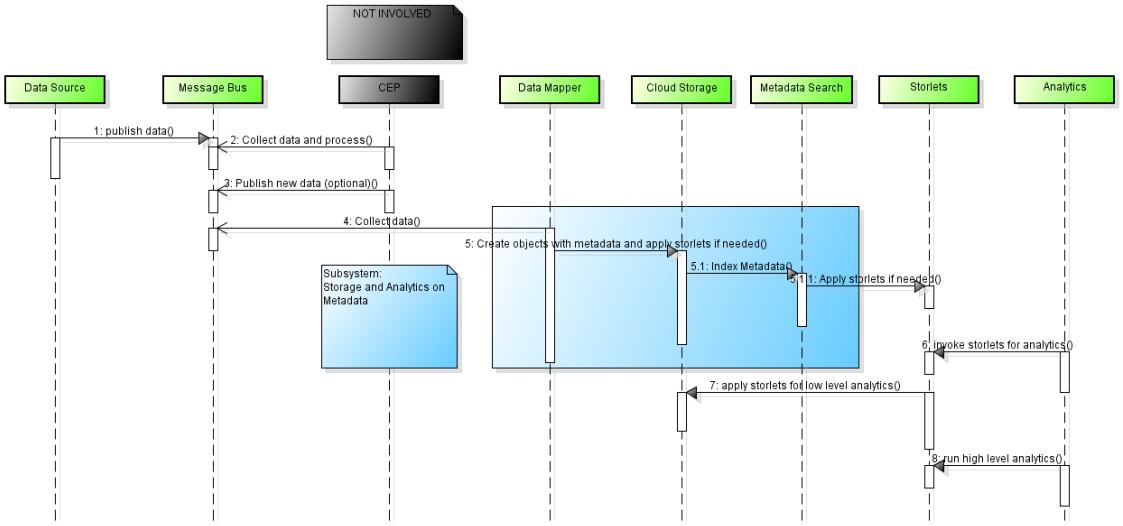

Figure 19: Sequence Diagram for Storage and Analytics on Metadata Subsystem

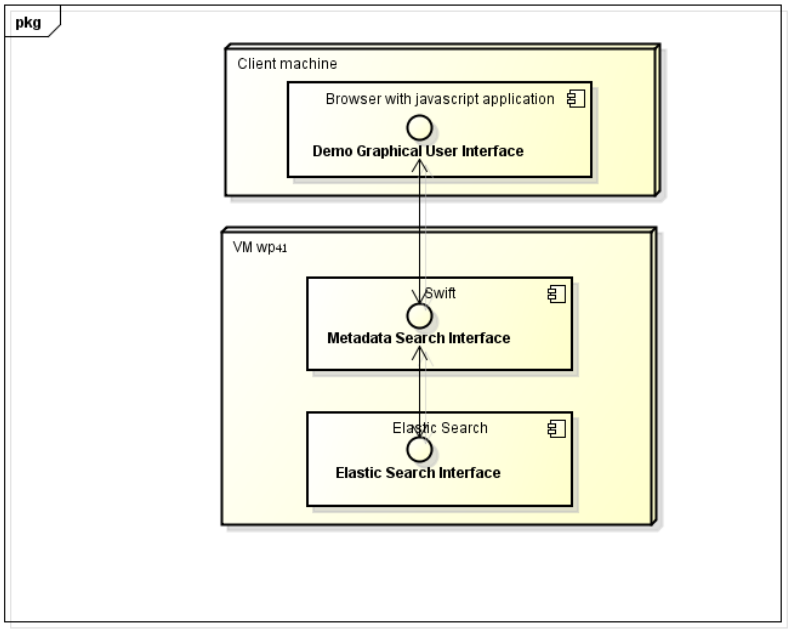

Figure 20: Deploymnet Diagram for the Metadata Search

2.2.1.3 Security, Privacy and Storage

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

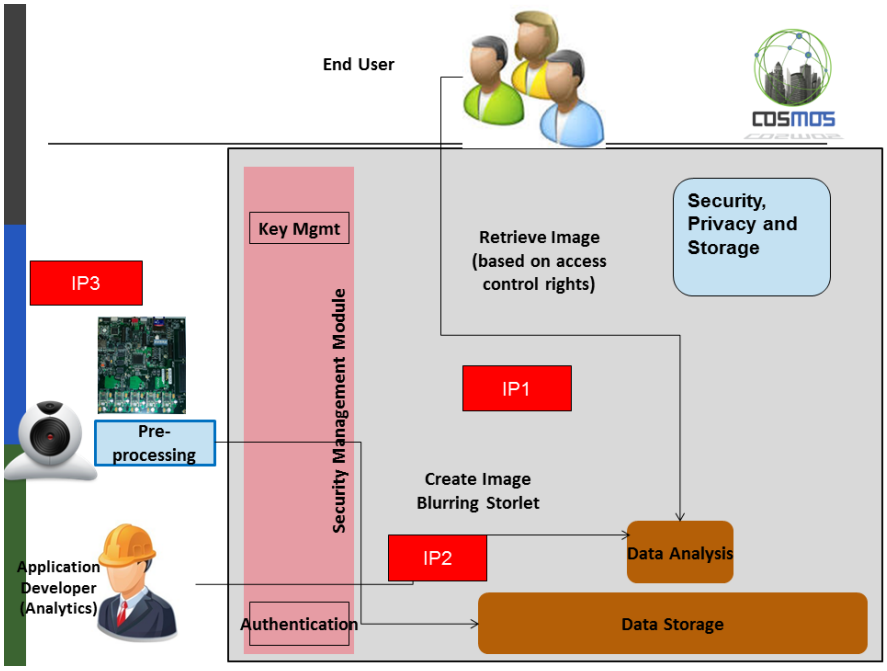

Figure 21: Security, Privacy and Storage

Scenario

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

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

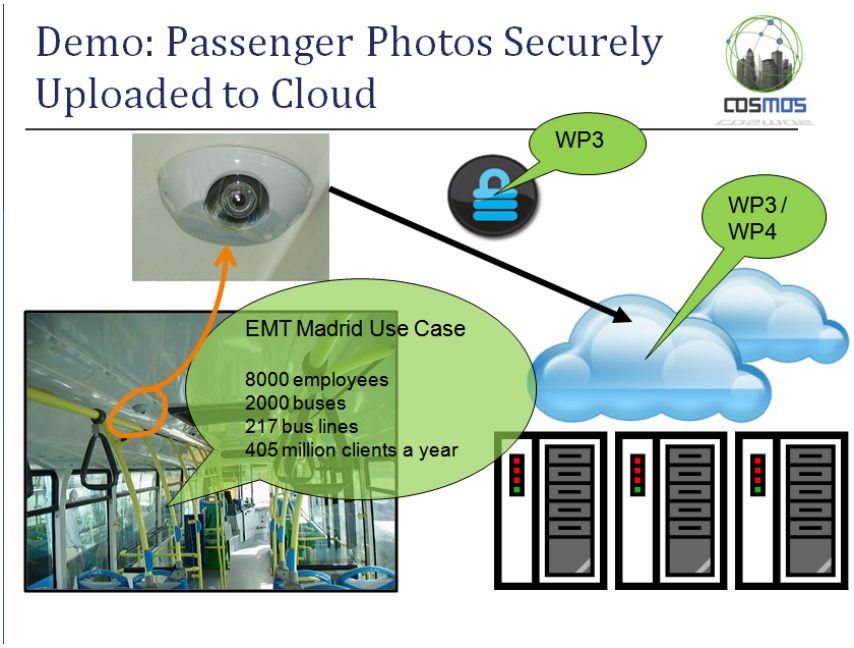

Figure 22: Security Demonstrator Show-Case

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

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

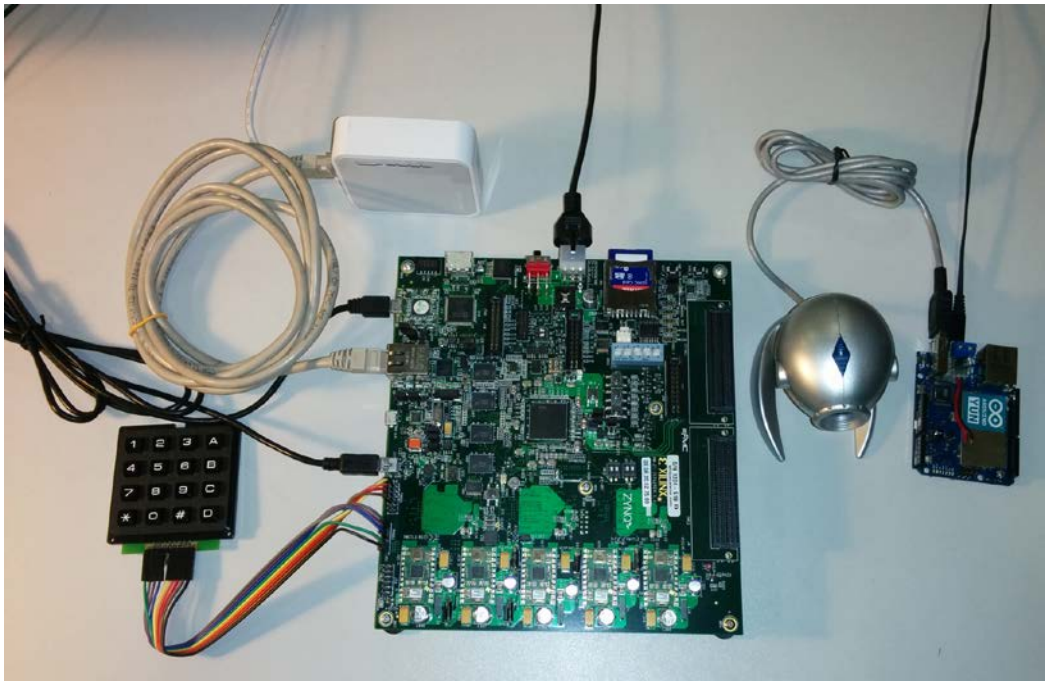

Figure 23: Hardware Security Board

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

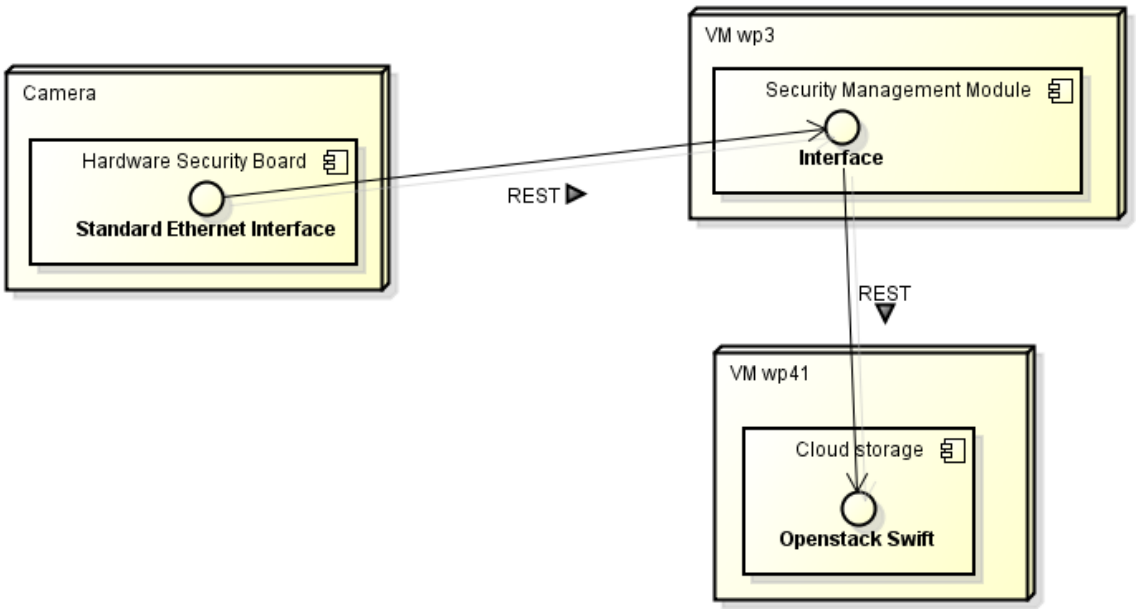

Figure 24: Security, Privacy and Storage Deployment Diagram

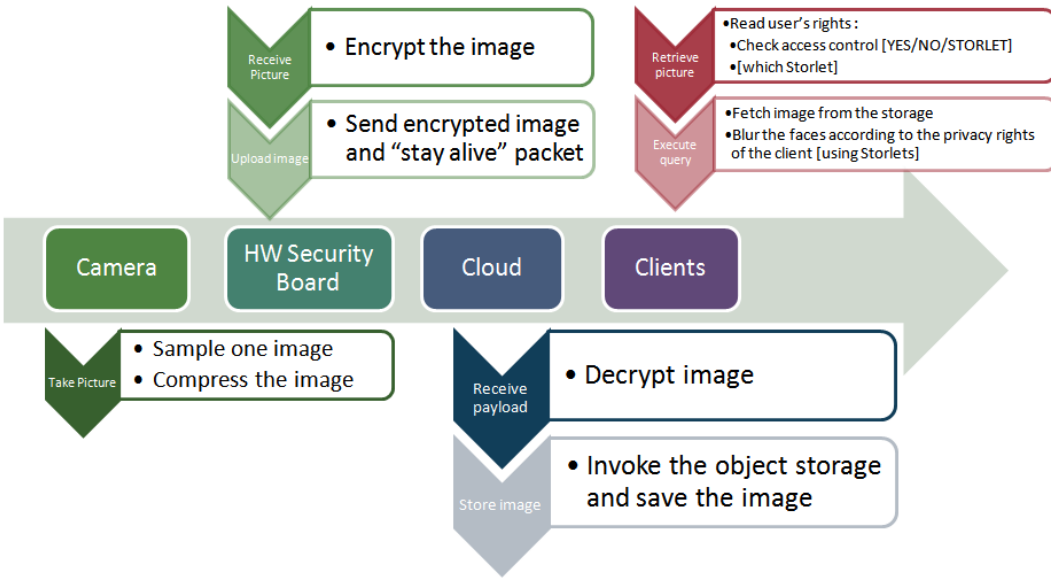

Figure 25: Demonstrator Flow

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

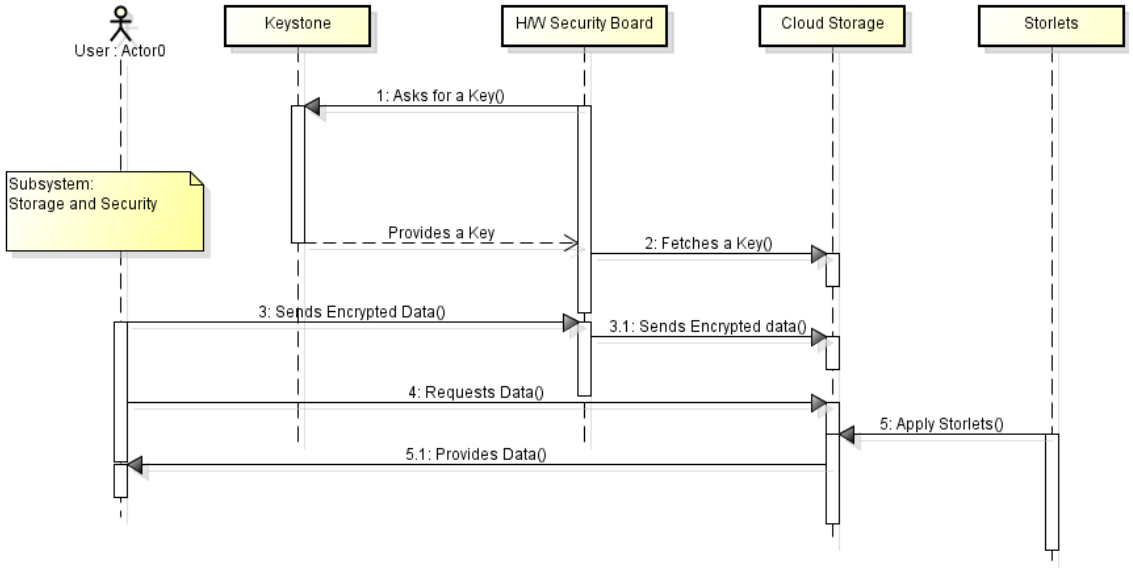

Figure 26: Sequence Diagram for Storage and Security Subsystem

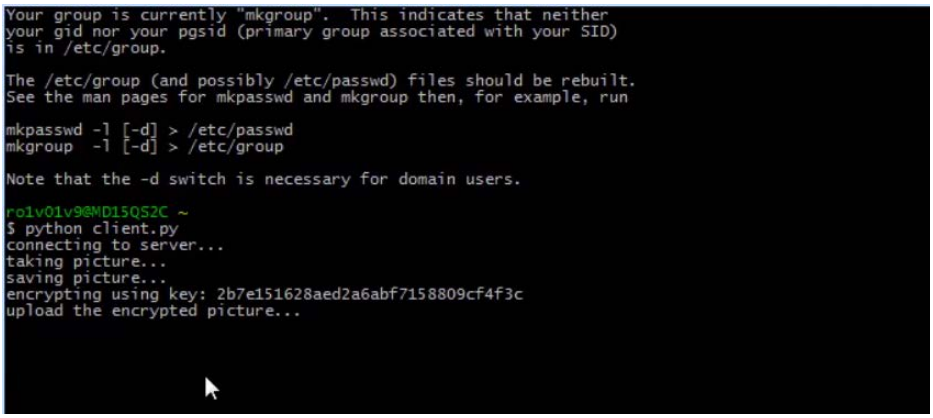

Figure 27: Client side image encryption

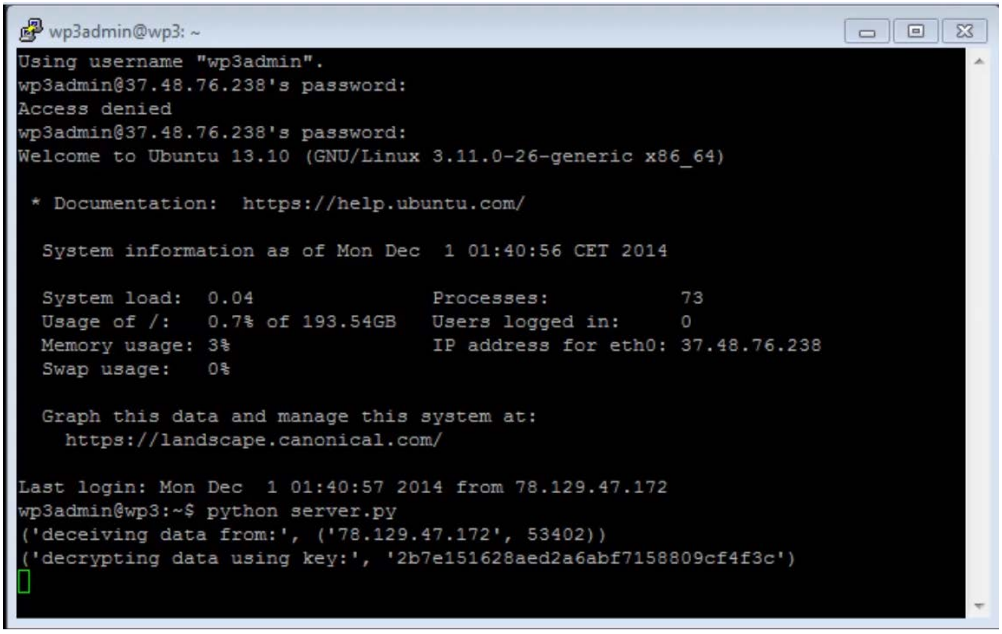

Figure 28: Platform Gateway for receiving and decrypting encrypted image

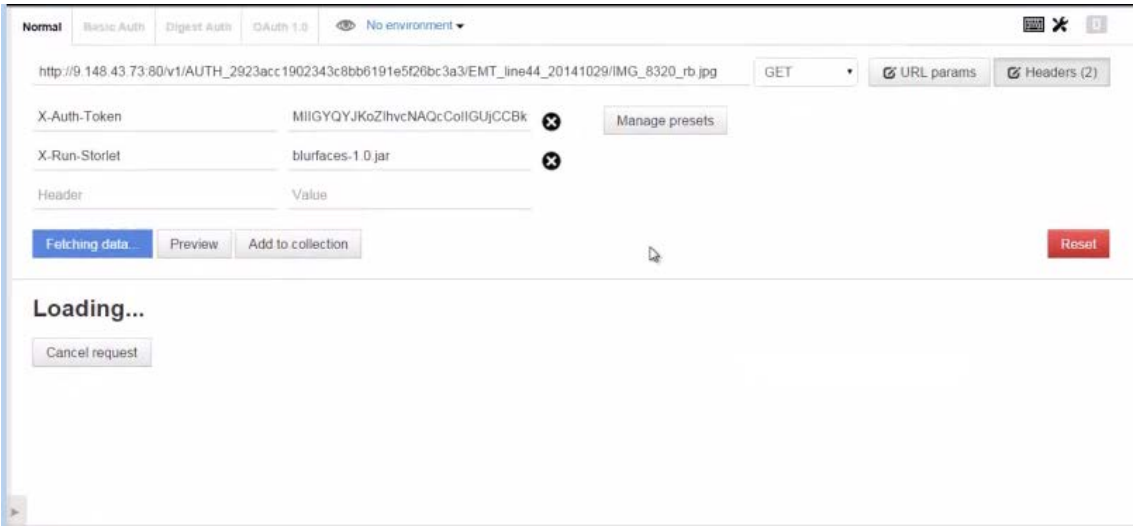

Figure 29: Facial Blurring Storlet Usage

Figure 30: Storlet Output (Blurred Image)

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

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

This corresponds to IP1, regarding end user involvement.

2.2.1.4 Modelling and Storage Analytics

Introduction

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

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

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

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

Integration of New Models

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

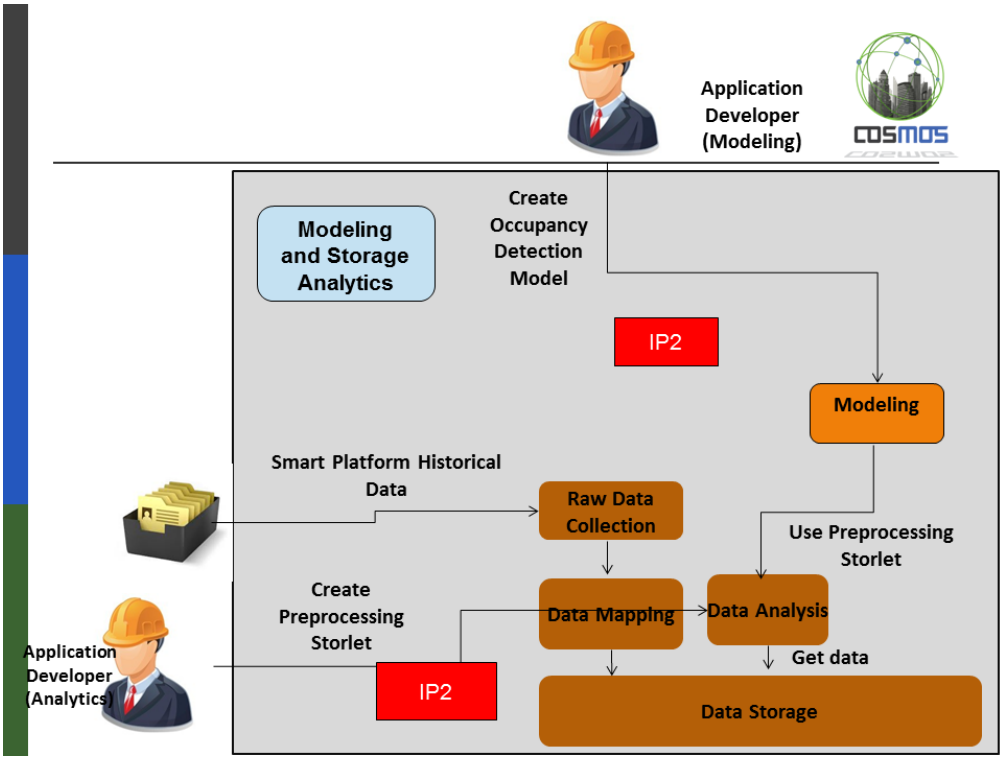

Figure 31: Storage and Modelling Subsystem

Data Flow

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

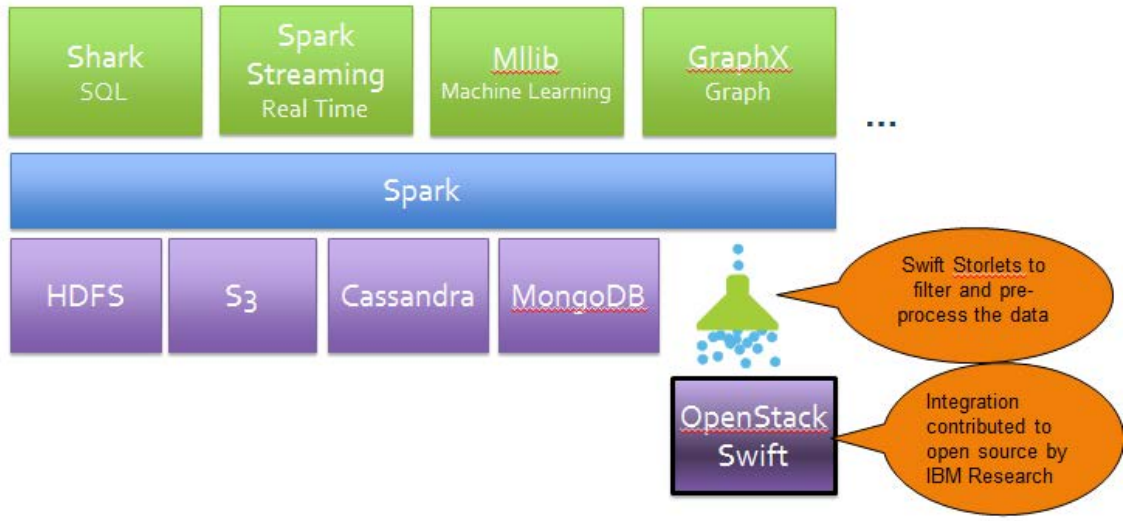

Figure 32: Overview of integration between Spark and Swift

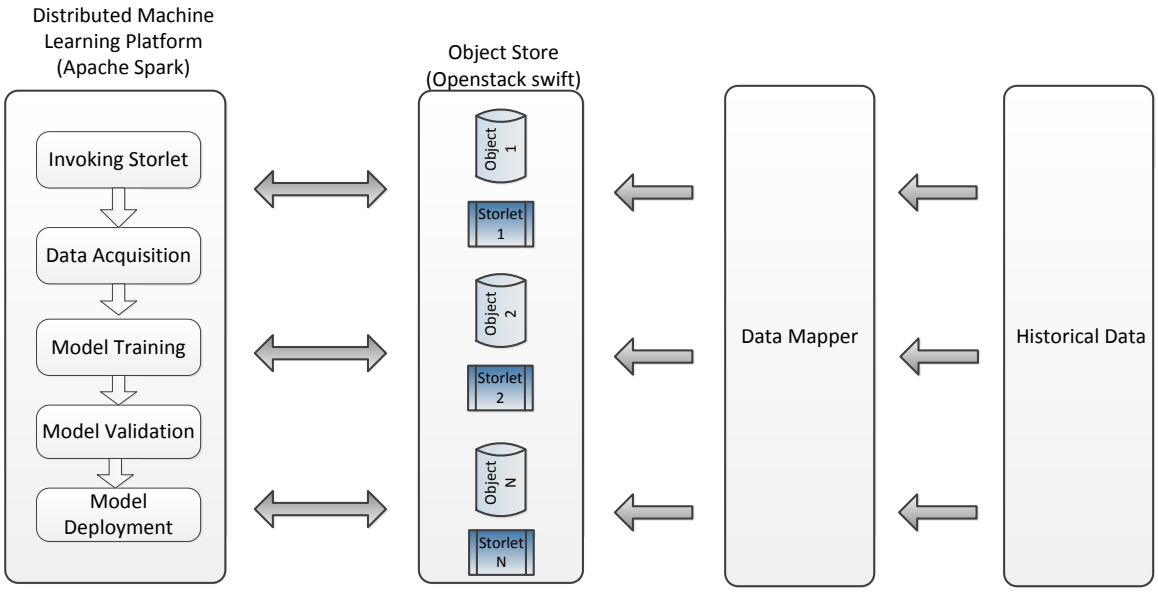

Figure 33: Data Flow across the Subsystem

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

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

Figure 34: Input data format for Modelling case

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

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

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

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

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

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

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

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


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


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

Figure 36: Runtime Prediction using the trained model

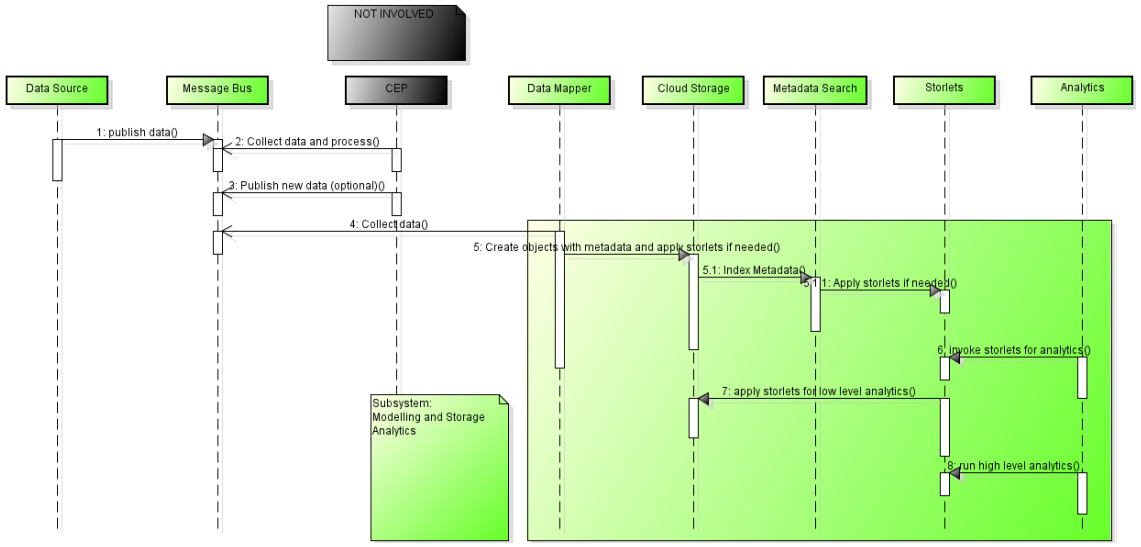

Figure 37: Sequence Diagram for Modelling and Storage Analytics Subsystem

Subsystem Test Case

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

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

Deployment Diagram

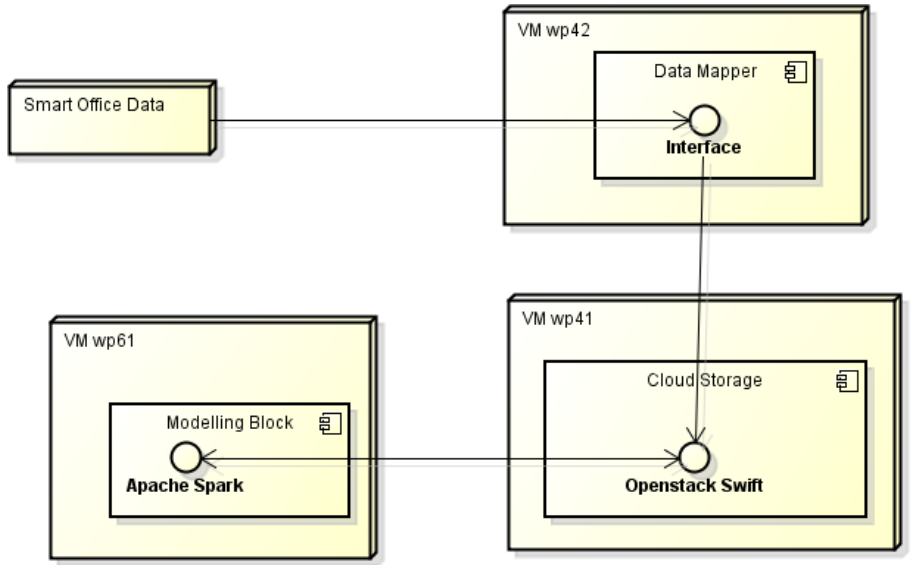

Figure 38: Modelling and Storage Analytics Deployment Diagram

2.2.2. Subgoal: Autonomous Behaviour of VEs

The subgoal includes the following componnets:

Message Bus

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

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 correspons to the IP1 point and is performed through the GUI presented in Figure 40.

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

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

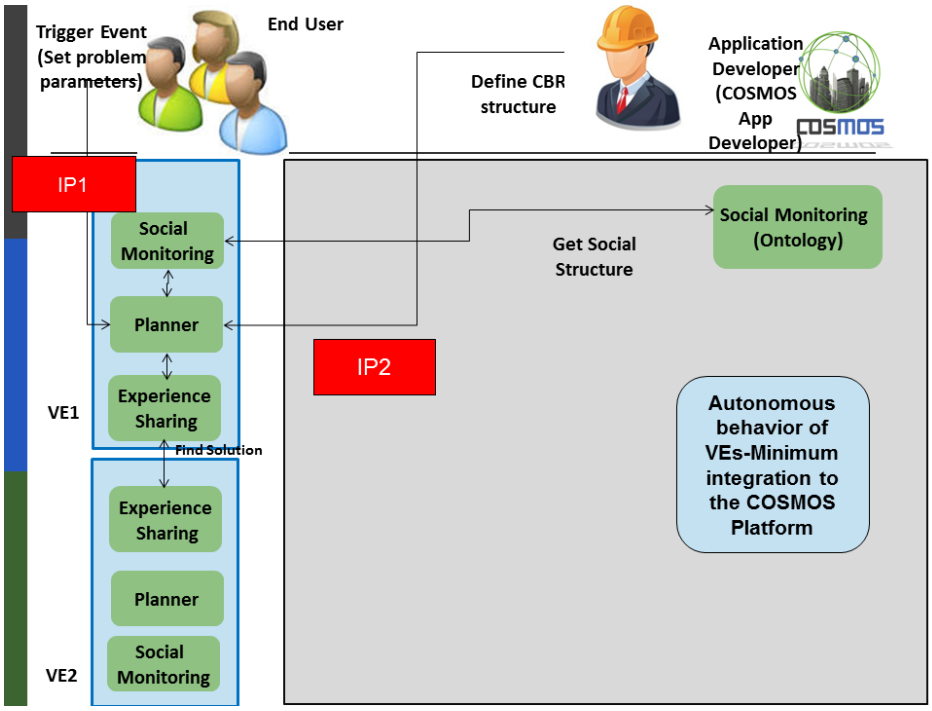

Figure 39: Autonomous Behavior of VEs with minimum platform integration subsystem

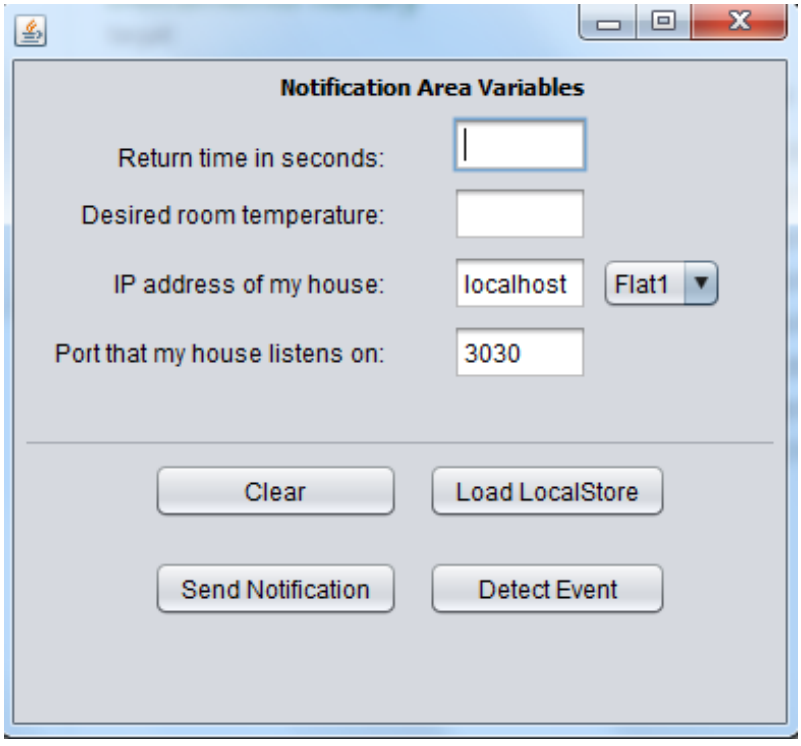

Figure 40: Autonomous Behavior of VEs application GUI

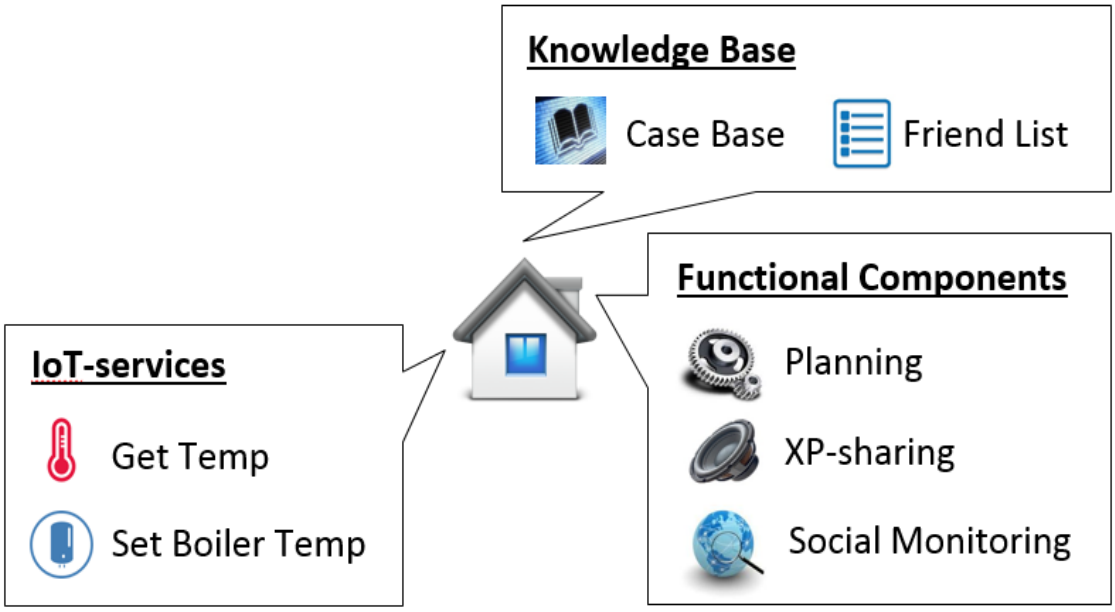

Figure 41: Components of the main flat-VE

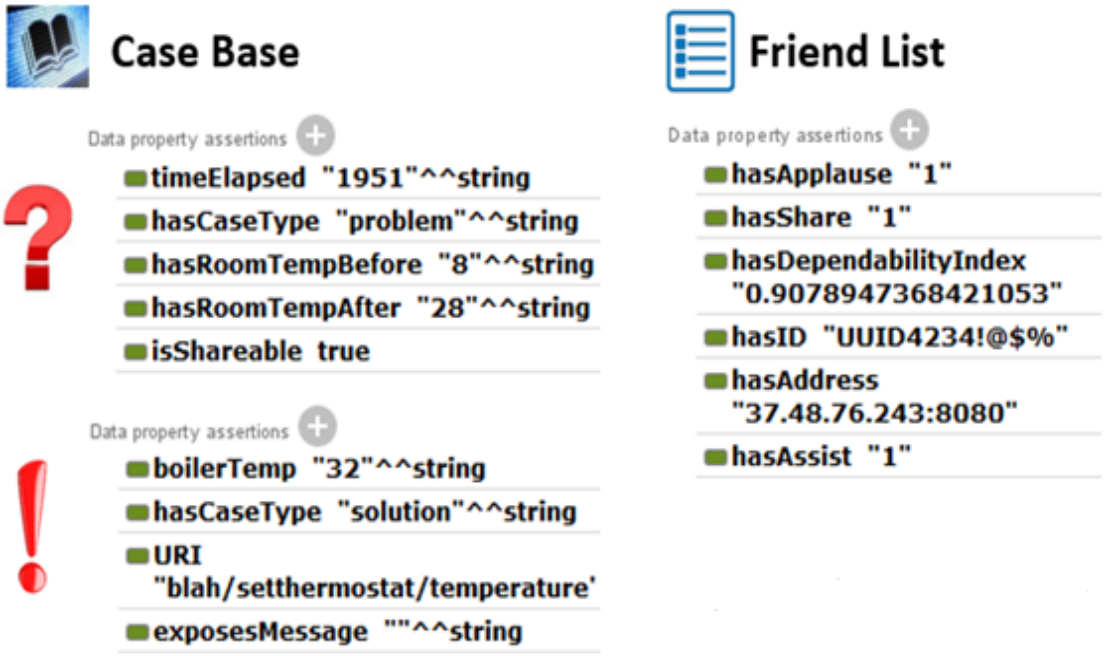

Figure 42: Case Base and Friend List examples

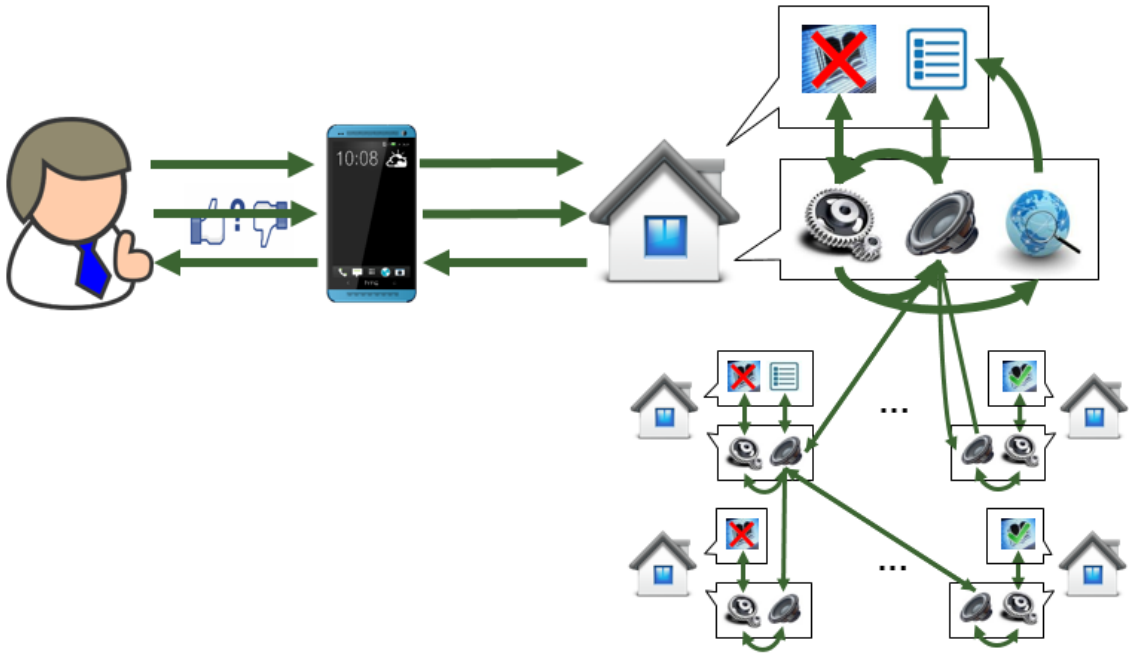

Figure 43: Visual representation of all scenario possibilities

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

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

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

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

Test Case Number Version	**AB_1**
Test Case Title	Reaction to User Generated Event
Modules tested	Planner, Experience Sharing, Social Monitoring
Requirements addressed	UNI. 704, 706, 708, 715, 719 5.9, UNI.251 5.10 5.11 5.12 5.29, UNI.010 5.31 6.8 6.9 6.18

Initial conditions	Have a CB and appropriate Friend lists inside the test bed Have the VE and application simulation jars inside the test bed
Expected results	The answer to the notification of the User Generated Event in the GUI A prompt for marking the helpfulness of the returned Solution, if Experience Sharing was used Changes in the CB and Friend list, depending on circumstances Command Line or Terminal Log with extra input, eg. requests made, similarities retrieved and queries used
Owner/Role	End User, Application Developer
Steps	Place the CB in the localstore folder of the home directory Open command line or terminal Navigate to the jar folder Execute “java –jar DemoProjectMaven-1.0-SNAPSHOT.jar” Keep the terminal or cmd open to view log info and then locate the GUI Enter a set of temperature and time Press “Send Notification” Grade the Solution as helpful or unhelpful if needed
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

2.2.2.2 Planner full integration with Platform

In this specific scenario our aim is to demonstrate that there is a clear chain of integration between the various developed components, especially in regards to “event detection” as a trigger from the Platform for further Planner actions. The expected chain of observed actions is the Data Source providing a stream of data in a topic of the Message Bus, the VE Planner to use this stream in order to extract Cases and at a later time the 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.

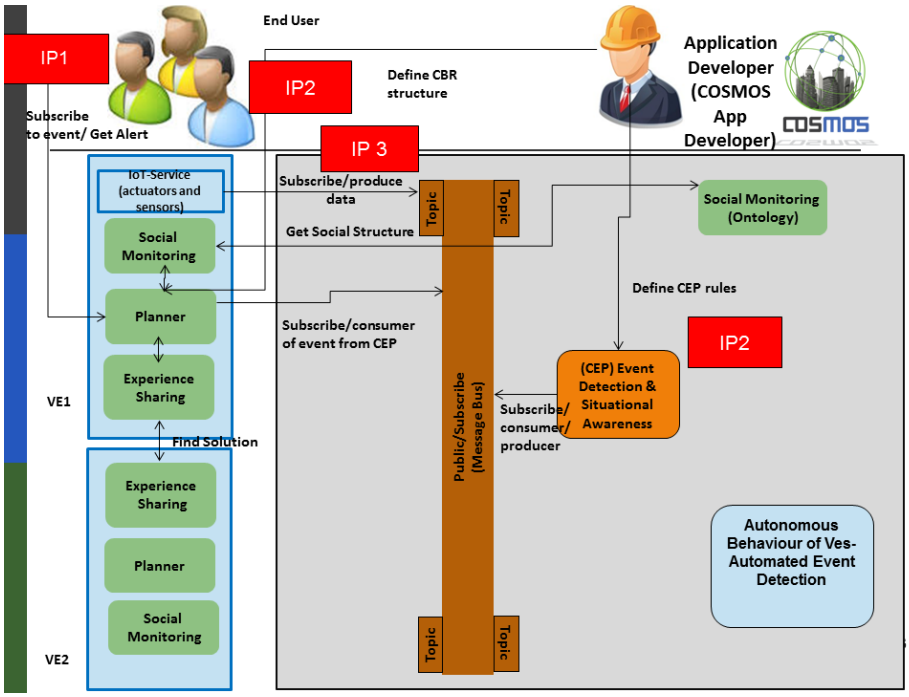

Figure 44: Autonomous Behavior of VEs- Automated Event Detection and Incorporation of COSMOS Platform

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:

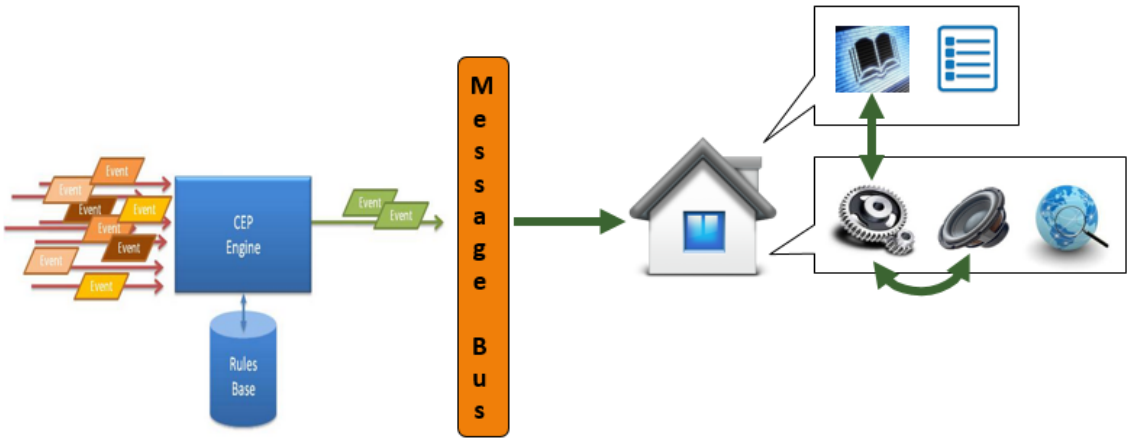

Figure 45: Visual representation of the platform integration scenario

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 me
channel.exchangeDeclare(exchangeName, "topic"); //there are a few exchan
channel.queueDeclare(queueName, false, false, true, null);
String bindingKey = "#";
channel.queueBind(queueName, exchangeName, bindingKey);
//System.out.println("Waiting for messages to be published..." + "\n");
QueueingConsumer consumer = new QueueingConsumer(channel);
channel.basicConsume(queueName, true, consumer);
```

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

Table 3: Planner full integration with COSMOS Platform

Test Case Number Version	**AB_2**
Test Case Title	Reaction to CEP generated event
Module tested	CEP, Message Bus, Planner, Experience Sharing, Social Monitoring
Requirements addressed	Same as **AB_1** 5.20
Initial conditions	Have a CB and appropriate Friend lists inside the test bed Have the VE and application simulation jars inside the test bed A functional CEP in the test bed A functional Message Bus
Expected results	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
Owner/Role	End User, Application Developer, VE developer
Steps	Place the CB in the localstore folder of the home directory Open command line or terminal Navigate to the jar folder Execute “java –jar DemoProjectMaven-1.0-SNAPSHOT.jar” Keep the terminal or cmd open to view log info and then locate the GUI Press button “Detect Event” The terminal/cmd will show whether the listening Planner detects an event send by the CEP Grade the Solution as helpful or unhelpful if needed
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

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

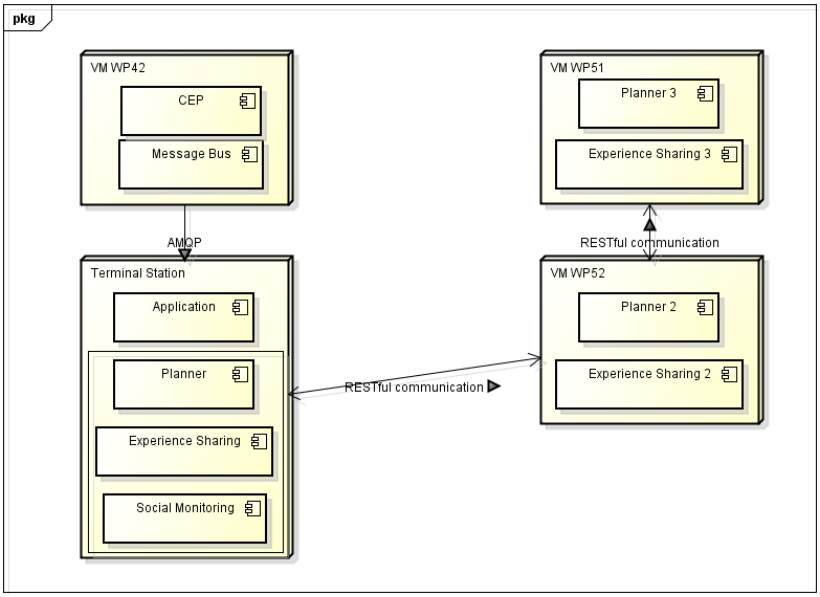

Figure 49: Autonomous Behaviour of VEs Deployment Diagram

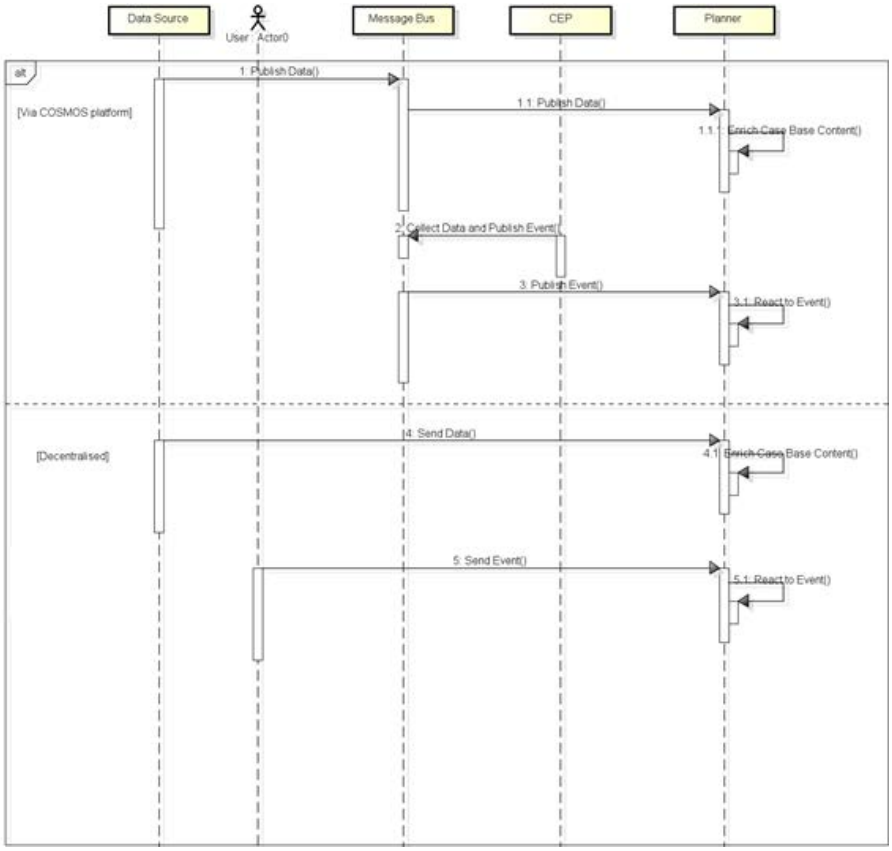

Figure 50: Autonomous Behaviour of VEs System Sequence Diagram

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

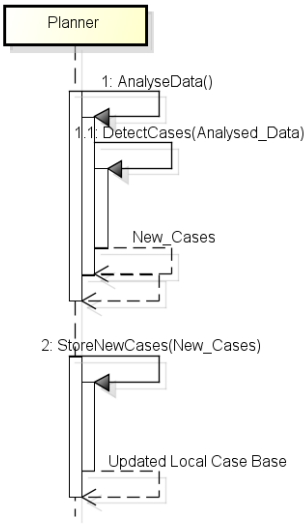

Figure 51: Enrich Case Base Content Sequence Diagram

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

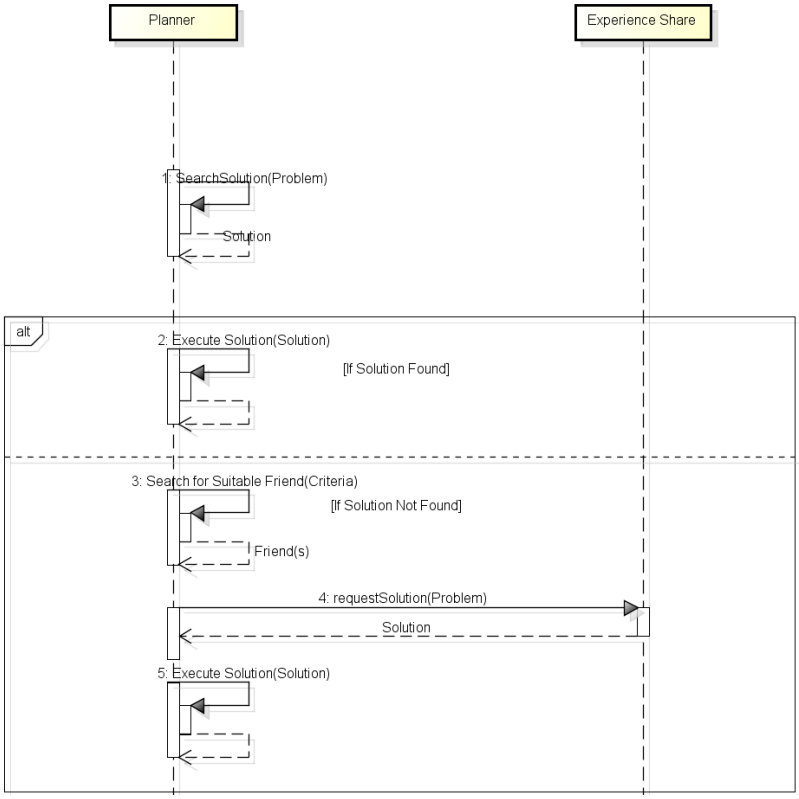

Figure 52: React to Event Sequence Diagram

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

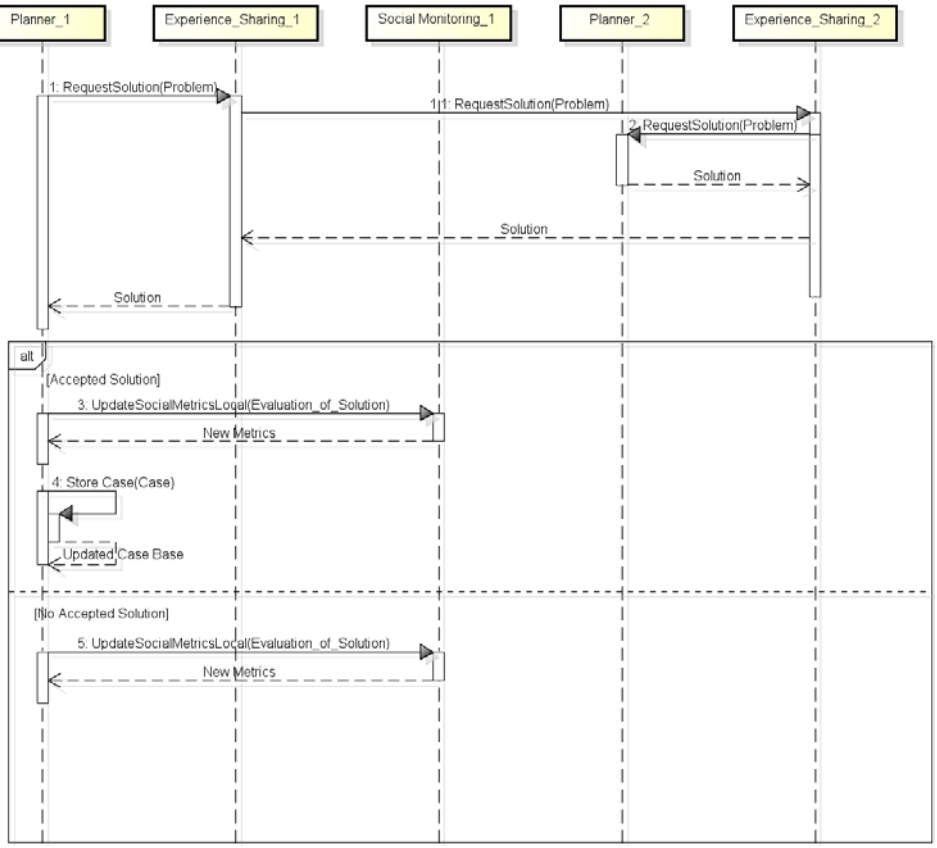

Figure 53: RequestSolution(Problem) Sequence Diagram

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

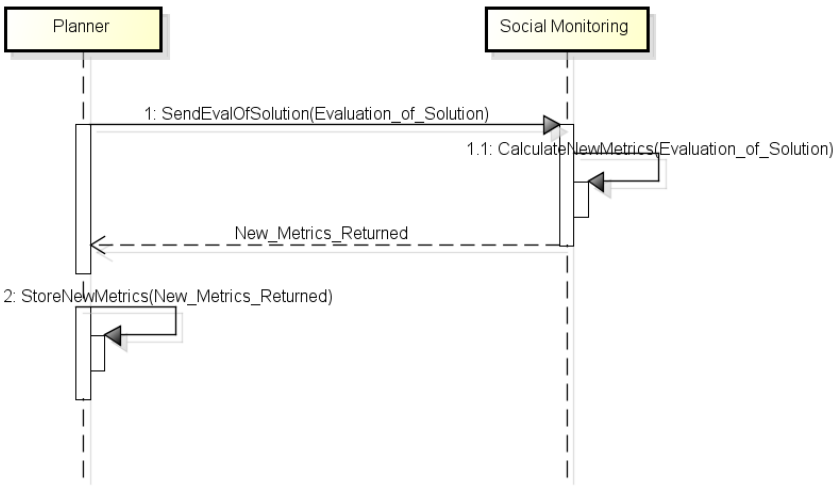

Figure 54: UpdateSocialMetricsLocal(Evaluation_of_Solution) Sequence Diagram

2.2.3. Component Tests

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

2.2.4. Occupancy detection model validation

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

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

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

2.2.4.1 Accuracy of Model

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

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

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

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

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

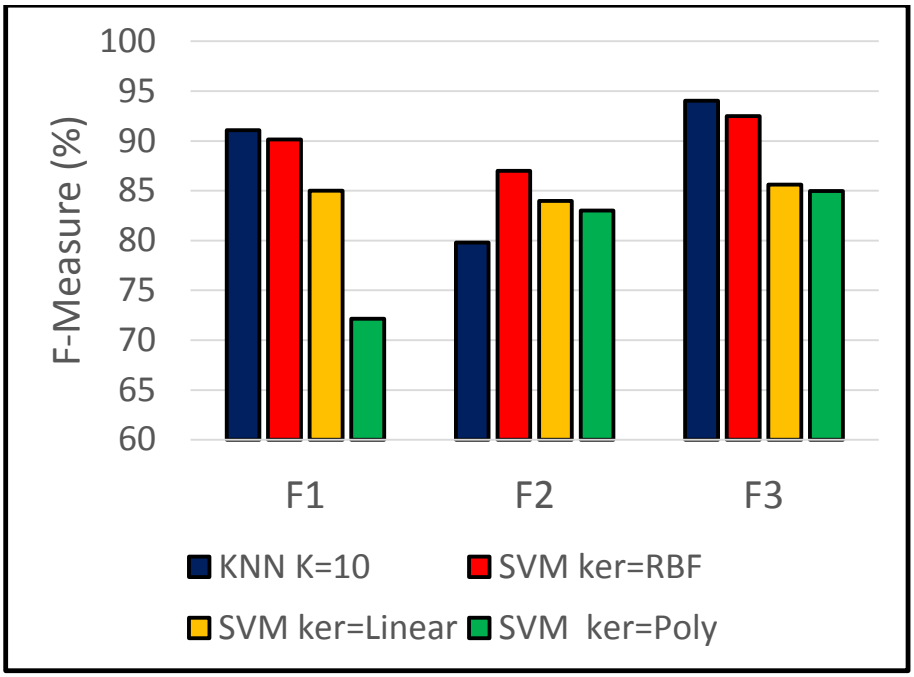

Figure 55: Performance of Classifiers

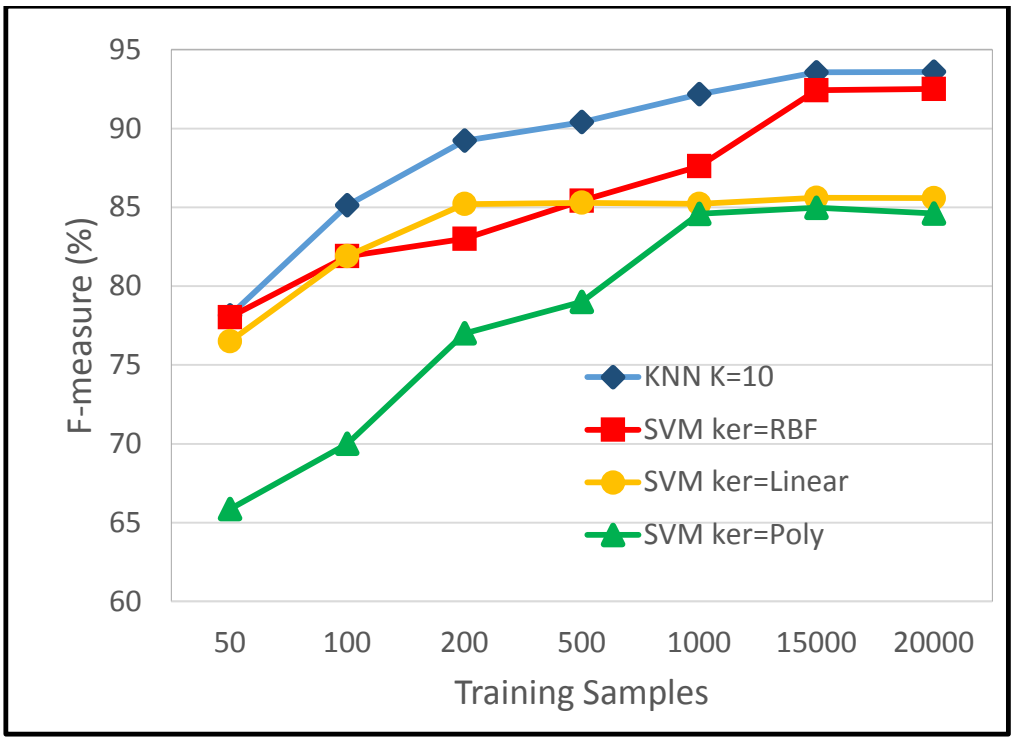

Figure 56: F-measure relation with training data

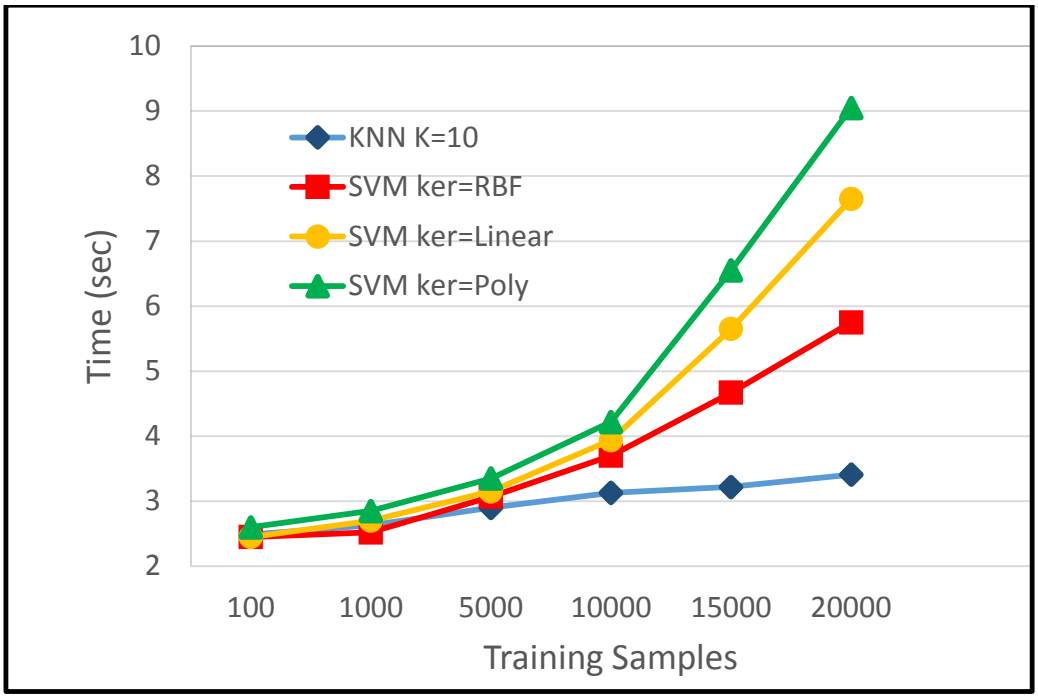

Figure 57: Time Complexity

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

2.3. Observed Deviations and Applied Mitigation Strategies

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

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

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

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

2.4.1. Security aspects

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

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

2.4.2. Data Analytics and Storage

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

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

2.4.3. Autonomous Behaviour of VEs

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

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

2.4.4. Modelling Aspects

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

3. Conclusions and Future Steps

As a conclusion and following the implementation of the integration plan and process, in the end of Y1 of the project we have 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). This is also one of the main points for the near future, the adaptation of these functionalities on the overall scenarios, resulting in COSMOS applications that fulfill the vision of the project and apply the concepts presented in the technical WPs in the realm of a concrete application.

Regarding the semantic incorporation needed for the project, one of the major goals is the semantic description of VEs, IoT-services and possibly of the applications and human users, thus forwarding to their incorporation in the aforementioned application context and overall integration with the platform, which results in having an organized, semantically enriched and reusable description of the available VEs in the context of the project.

References

[1] COSMOS Deliverable 7.6.1 “Integration Plan (Initial)”, ICCS/NTUA and other partners, June 2014

[2] Support Vector Machine (SVM): http://en.wikipedia.org/wiki/Support_vector_machine

[3] K-Nearest Neighbours (KNN): http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[4] COSMOS Deliverable 2.2.1 “State of the Art Analysis and Requirements Definition (Initial)”, UniS and other partners, February 2014

[5] COSMOS Deliverable 5.1.1 “Decentralized and Autonomous Things Management: Design and Open Specification (Initial)", ICCS/NTUA and other partners, April 2014

[6] COSMOS Deliverable 4.1.1 “Information and Data Lifecycle Management: Design and Open Specification (Initial)", IBM and other partners, April 2014

Annex A: Unit/Component Tests

Data Mapping

Table 4: Results for Data Mapping Unit Test #1

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

Table 5: Results for Data Mapping Unit Test #2

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

Message Bus

Table 6: Results for Message Bus Unit Test #1

Test Case Number Version	**4_MB_1**
Test Case Title	4_MB_1
Module tested	Message Bus
Requirements addressed	4.9
Initial conditions	RabbitMQ service is installed and running. Hildebrand data stream is available.
Expected results	Messages are propagated to subscribers.
Owner	Atos
Steps	1.) Execute message publisher ~/Atos/Mosquitto/Bridge.exe 2.) Execute /Atos/Mosquitto/VeSub.exe 3.) Messages are correctly routed to the subscriber.
Passed	Yes
Bug ID	
Problems	
Required changes	

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.

Event Detection & Situational Awareness @Platform

Table 7: Results for Event Detection @Platform Unit Test #1

Test Case Number Version	**4_ED_1**
Test Case Title	4_ED_1
Module tested	Complex Event Processing
Requirements addressed	4.7, 4.10, 4.11, 4.12, 5.14
Initial conditions	Data streams provided by Message Bus are available. Provided CEP configuration file contains definition of the “Sensor Malfunction” situation.
Expected results	Detected situation is published on the specific Message Bus topic.
Owner	Atos
Steps	1.) Connect CEP to the event stream by invoking:

	~/Atos/Cep_Cos20Sa/RmqCollector/RmqCollector.exe –hild 2.) Start event detection service: ~/Atos/ Cep_Cos20Sa/SCep 3.) Forward detected events back to the message bus by invoking ~/Atos/Cep_Cos20Sa/RmqPublisher/ RmqPublisher.exe –[topicName] 4.) Observe that detected situation is published on the message bus under specified “topicName”.
Passed	Yes
Bug ID	
Problems	
Required changes	

Prediction

Table 8: Results for Prediction Unit Test #1

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

Semantic Description and Retrieval

Table 9: Results for Semantic Description and Retrieval Unit Test #1

Test Case Number Version	**5_SDR_1**
Test Case Title	VE semantic description and retrieval
Module tested	Triple store access
Requirements addressed	5.0, 5.2, UNI.414, UNI.416, UNI.425, UNI.426
Initial conditions	First version of the semantic model available. Test triple store installed. Test query API available.
Expected results	Successful recording of the VE triples and matches between the

	retrieval results and the expected VE descriptions.
Owner	SIEMENS
Steps	1)Use a template form to fill in the description of a VE; 2)Call the API to store the description of the VE into the triple store; 3) Repeat steps 1 and 2 for a number of VEs with different attributes; 4)Query the triple store in order to retrieve the description of different VEs based on various search criteria; 5)Compare the results with the expected outcome.
Passed	Yes
Bug ID	
Problems	
Required changes	

Table 10: Results for Semantic Description and Retrieval Unit Test #2

Test Case Number Version	**5_SDR_2**
Test Case Title	VE description validation
Module tested	Triple store access
Requirements addressed	5.0, 5.2, UNI.414, UNI.416, UNI.425, UNI.426
Initial conditions	First version of the semantic model available. Test triple store installed. Test query API available.
Expected results	Inconsistent VE descriptions should not be stored into the triple store.
Owner	SIEMENS
Steps	1)Use a template form to fill in the description of a VE with erroneous or inconsistent fields; 2)Call the API to store the description of the VE into the triple store; 3) Repeat steps 1 and 2 for a number of VEs with different attributes; 4)Whenever the VE description does not meet the consistency requirements, and error condition must be signalled and no commit to the triple store should be made; 5)Query the triple store in order to retrieve the description of different inconsistent VEs whose description and storage attempt was made; 6)No complete or partial VE description should be retrieved.
Passed	Yes
Bug ID	
Problems	
Required changes	

Privelets

Table 11: Results for Privelets Unit Test #1

Test Case Number Version	**3_PR_1**
Test Case Title	Privacy
Module tested	Privelets
Requirements addressed	3.1, 3.8, 3.15
Initial conditions	Existing jsonFile.json in root directory that follows precise pattern. RESTful Client (or any mechanism) to exchange JSON files.
Expected results	Data exchange filtering procedure
Owner	NTUA
Steps	1) Generate a Json file on the root directory of our device “C:\\” and call it “jsonFile.json”. Pre-complete or leave it empty. 2) Run the component as a java application and switch to the REST client and/or the UI page. 3) Use Google Chrome and its add-on called “Advanced REST Client”, (or any other way) to send/receive JSON files. 4) Perform @POST service insertData with groups of tags on the call’s payload. (data is now filtered according to the configuration) 5) Perform @GET service getPublicJasonFileTags to get the filtered JSON file.
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Table 12: Results for Privelets Unit Test #2

Test Case Number Version	**3_PR_2**
Test Case Title	Configuration
Module tested	Privelets
Requirements addressed	3.1, 3.8, 3.15
Initial conditions	Existing jsonFile.json in root directory that follows precise pattern. RESTful Client (or any mechanism) to exchange JSON files.
Expected results	View and edit configuration file.
Owner	NTUA
Steps	1) Check about the existing configuration file by calling the @GET service called retrieveExistingConfigurationFile. 2) Update the existing configuration file, and add or remove tags considered as private by calling the @POST service, updateExistingConfigurationFile and giving in the payload a Json file.3) Check the link http://localhost:8080/JAXRS-

	COSMOS/faces/home.xhtml, the UI of that has to do with the instance of the data to be exchanged or repeat step 1.
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Planner

Table 13: Results for Planner Unit Test #1

Test Case Number Version	**5_PL_1**
Test Case Title	Notification Service and CBR
Module tested	Planner
Requirements addressed	UNI. 704, 706, 708, 715, 719 5.31 5.29, UNI.010 5.21 5.10 5.9, UNI.251
Initial conditions	Have a CB inside the test bed. Have the VE and app simulation jar inside the test bed.
Expected results	The answer to the notification of the User Generated Event in the GUI. Command Line or Terminal Log with similarities retrieved and queries used.
Owner	Panagiotis Bourelos, Orfefs Voutyras
Steps	Place the CB in the localstore folder of the home directory. Open command line or terminal. Navigate to the jar folder. Execute “java –jar DemoProjectMaven-1.0-SNAPSHOT.jar”. Keep the terminal or cmd open to view log info and locate the GUI. Enter a set of temperature and time that exist locally (time: 3535 and temp: 26). Press “Send Notification”.
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Event Detection & Situational Awareness @VE

Not applicable for year 1.

Experience Sharing

Table 14: Results for Experience Sharing Unit Test #1

Test Case Number Version	6_ES_1
Test Case Title	Experience Sharing and returned Solution
Module tested	Planner, Experience Sharing
Requirements addressed	Same as those in 5_PL_1 plus: 5.11, 5.12, 6.8, 6.18, 6.19, 6.34, 6.35
Initial conditions	Have a CB and Friend List inside the test bed. Have the VE and app simulation jar inside the test bed. Have one or both auxiliary XP sharing VEs active in the VMs, along with their CBs and Friend Lists.
Expected results	The answer to the notification of the User Generated Event in the GUI Command Line or Terminal Log with similarities retrieved and queries used The Logs of auxiliary VEs in the VMs with the Planner result and the Experience Sharing steps
Owner	Panagiotis Bourelos, Orfefs Voutyras
Steps	Place the CB and Friend List in the localstore folder of the home directory; Open command line or terminal; Navigate to the jar folder; Execute “java –jar DemoProjectMaven-1.0-SNAPSHOT.jar”; Keep the terminal or cmd open to view log info and locate the GUI; Boot the VEs in their respective VMs; Enter a set of temperature and time that exist in another VE (time: 1800 and temp: 26 for VE Flat 2/ time: 3535 and temp: 28 for VE Flat 3); Press “Send Notification”
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Hardware Security Board

Table 15: Results for Hardware Security Board Unit Test #1

Test Case Number Version	3_HSB_1
Test Case Title	Communication over standard interfaces
Module tested	Communication module
Requirements addressed	3.1
Initial conditions	Configure standard communication interfaces (I2C, SPI, USB, Ethernet)

Expected results	Data flow can take place over the specified interfaces. Sensors and the Ethernet network can be accessed.
Owner	Leonard Pitu
Steps	1. Load the hardware design (*.bit file) 2. Load the firmware 3. Boot the firmware 4. Obtain an IP address 5. Read the sensors 6. Make the data available over ETH.
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Table 16: Results for Hardware Security Board Unit Test #2

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

Table 17: Results for Hardware Security Board Unit Test #3

Test Case Number Version	**3_HSB_3**
Test Case Title	Secure the data flow (board)
Module tested	Cryptographic accelerator
Requirements addressed	3.3
Initial conditions	Configure the FPGA with the hardware design.
Expected results	The cryptographic accelerator shall meet the FIPS-180 standard.
Owner	Leonard Pitu
Steps	Standard test vectors

	1. Load the hardware design (*.bit file) 2. Load the test cases according to FIPS-180 3. Run the tests Random test vectors 1. Load the hardware design (*.bit file) 2. Load private test vectors generated with Crypto++ 3. Run the tests
Passed	Yes
Bug ID	None
Problems	None
Required changes	None

Table 18: Results for Hardware Security Board Unit Test #4

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