COSMOS

Cultivate resilient smart Objects for Sustainable city applications

Grant Agreement Nº 609043

D6.2.2 Reliable and Smart Network of Things: Software Prototype (Updated)

WP6: Reliable and Smart Network of Things

Version: 1.0
Due Date: 30 June 2015
Delivery Date: 30 June 2015
Nature: Report
Dissemination Level: Public
Lead partner: University of Surrey (UNIS)
Authors: A. Akbar (editor) & F. Carrez (Unis), P. Bourelos (NTUA), Juan Sancho (ATOS)

Internal reviewers: Achilleas Marinakis (NTUA), Paula Ta-Shma (IBM)

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
V0.1	08/05/2015	F. Carrez, A. Akbar	UniS	Initial draft
V0.2	01/06/2015	Adnan Akbar	UniS	Update Ch 2, 3, 4
V0.3	09/06/2015	Juan Sancho	ATOS	SAw section
V0.4	19/06/2015	Adnan Akbar	UniS	Add results in Ch 2 and ch 3
V0.5	19/06/2015	Juan Sancho	ATOS	SAw update
V0.6	25/06/2015	Achilleas Marinakis	NTUA	Review
V0.7	26/06/2015	Juan Sancho	ATOS	Update based on review
V0.8	29/06/2015	Panagiotis Bourelos	NTUA	Updates based on Review
V0.9	30/06/2015	Adnan Akbar	UniS	Updates based on review
V1.0	30/06/2015	Juan Sancho	ATOS	Release version

Table of Contents

Table of Figures

Acronyms

Acronym	Meaning
API	Application Programming Interface
CBR	Case Base Reasoning
CEP	Complex Event Processing
EM	Expectation Maximization
FC	Functional Component
GMM	Gaussian Mixture Model
GUI	Graphical User Interface
HMM	Hidden Markov Model
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
KNN	K Nearest Neighbour
ML	Machine Learning
RBF	Radial Basis Function
REST	Representational State Transfer
SAw	Situational Awareness
SVM	Support Vector Machine
SVR	Support Vector Regression
ttl	Time-To-Live
UUID	Universally unique identifier
VE	Virtual Entity
XML	eXtensible Markup Language
XP	Experience

1 Introduction

This document is intended to demonstrate the prototype development for WP6 “Reliable and Smart Network of things” based on the deliverable D6.1.2 [1]. The main objective of this work package is to provide methods for inferring high-level knowledge from raw IoT data, to provide the means for situational awareness and understanding how things have behaved in comparable situations previously. Different data mining techniques based on Machine Learning (ML) methods and statistical analysis methods are implemented for inferring or predicting high-level knowledge; and methods based on Complex Event Processing (CEP) engines are explored to contribute towards situational awareness models.

This document follows the approach of earlier submitted document D6.1.2: Design and Open specification [1] from implementation of view. In short, following changes have been made in this iteration as compared to previous version.

- Full-restructuring of the document which is now following the functional decomposition (and not the WP task structure);
- Fully reworked section on Situational Awareness according to D6.1.2 [1].
- All other Functional Components come in their second iteration aligned with Year 2 objectives.
- Added use-case for prediction in Chapter 2

The outline of the report is as follows. Chapter 2 explains the use and implementation of machine learning and statistical methods for inference/prediction functionality. Chapter 3 explains the use of Pre-processing FC in different use-case scenarios. Chapter 4 explains our proposed method for adaptive event detection using ML and CEP and support our proposed solution with the help of results. Chapter 5 explains different implementation steps involved in Situational Awareness process. Chapter 6 is focussed on Experience Sharing for autonomous behaviour of things. Finally, a brief conclusion is given in Chapter 7.

2 Inference/Prediction Functional Component

This component is responsible for analysing raw data in order to provide high-level knowledge which can be used for automated, proactive and intelligent applications. In this regard, we have implemented several pattern recognition algorithms on the use-case scenarios such as different variants of Support Vector Machines (SVM) and Hidden Markov Models (HMM) for inferring high-level knowledge.

We have also explored several regression mechanisms for time series prediction of data for providing proactive solutions for smart city applications. In this chapter, we briefly explain the architecture, interfaces of the component and finally the application of the component with the help of use-case scenario.

2.1 Architecture

Figure 1 below shows Inference/Prediction FC and its connections with other components in the overall architecture of COSMOS platform. The inference and prediction FC can take both raw data and VE data as input depending on the scenario and the usage required by the application developer. Raw data can be pre-processed before publishing on the Message Bus FC (WP4) under specific topic, and then stored on the Cloud Storage FC (WP4) using Data Mapper FC (WP4). Inference and Prediction FC retrieves the data from the Cloud Storage FC using the provided interface.

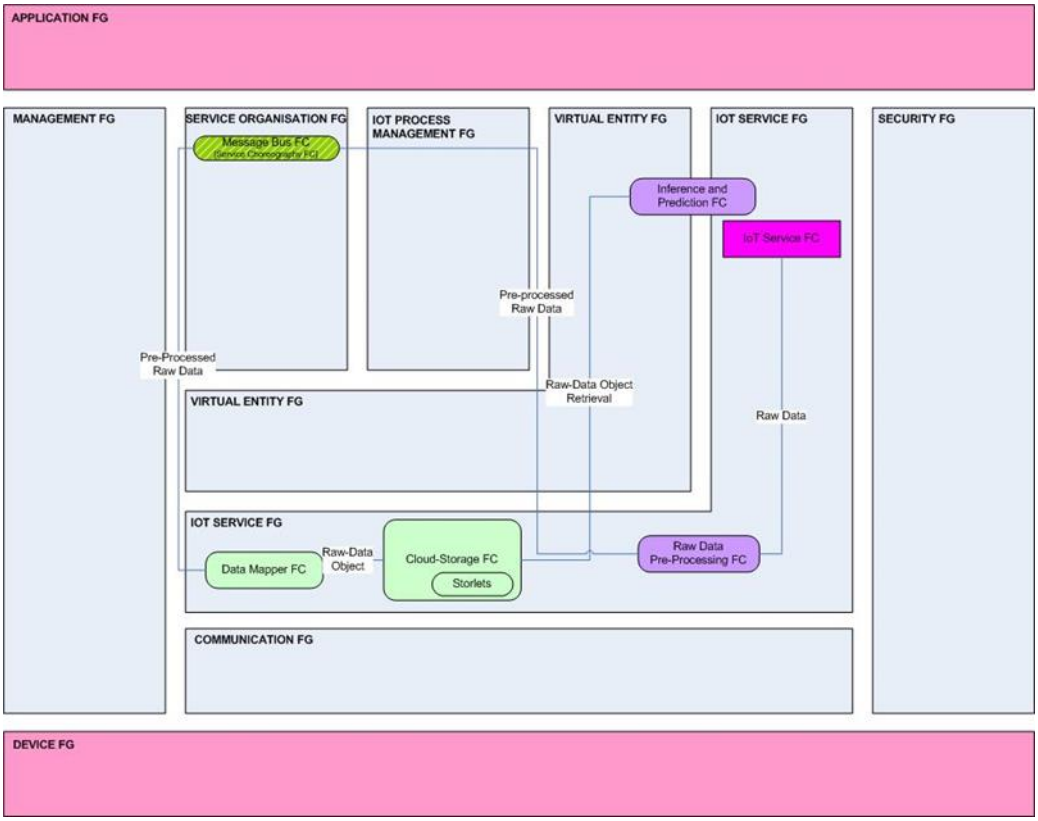


Figure 1: Inference/Prediction FC

2.2 Interfaces

The application developer can use the models provided by COSMOS for inference of high-level knowledge and for predicting events. In order to use the existing models, the application developer will have to define the input features and output entity in which he/she is interested. In order to achieve this, the application developer will use the COSMOS storage services in order to access historical data for the construction of off-line model. Then when a model is available, on-line update of the model can be done under certain circumstances. Once enough data is collected, and therefore a model available, the Prediction component/functionality will be instructed for making a prediction based on the available model. The resulting model will be persisted and semantically annotated in order to make it retrievable for later use. Using this approach, the semantic description of VEs and IoT services or Message Bus topics, could also include a reference to a prediction model (if available).

Since COSMOS is intended to be used by different actors and forge cooperation and reuse, prediction models can be built by different parties (for instance in the case of public data) provided that they are semantically described and linked to the data sources. Once stored and annotated, other actors will be able to query the semantic store for prediction models and use them according to their needs. The interfaces are summarized below:

Application Developer-Inference/prediction Block API:

In the current version of deliverable, we have provided the following interfaces for Inference/prediction block.

1) Any particular model can be selected depending on the application requirement or the type of available data (labelled or not).
2) Historical data will be provided and act as input features for machine learning algorithms. These data can be either from the COSMOS cloud storage or external, depending on the application.
3) Specific type of pre-processing can be selected at run-time depending on the application requirement.
4) In some cases the particular type of pre-processing needed is not known ahead of time. In that case, one could use storlets to perform pre-processing close to the data (especially if the pre-processing reduces the data size).
5) For applications, where labelled data is available for supervised machine learning algorithms as we discussed in our energy management scenario (occupancy detection), it will be provided as input.

These interfaces are summarized in the Figure 2.

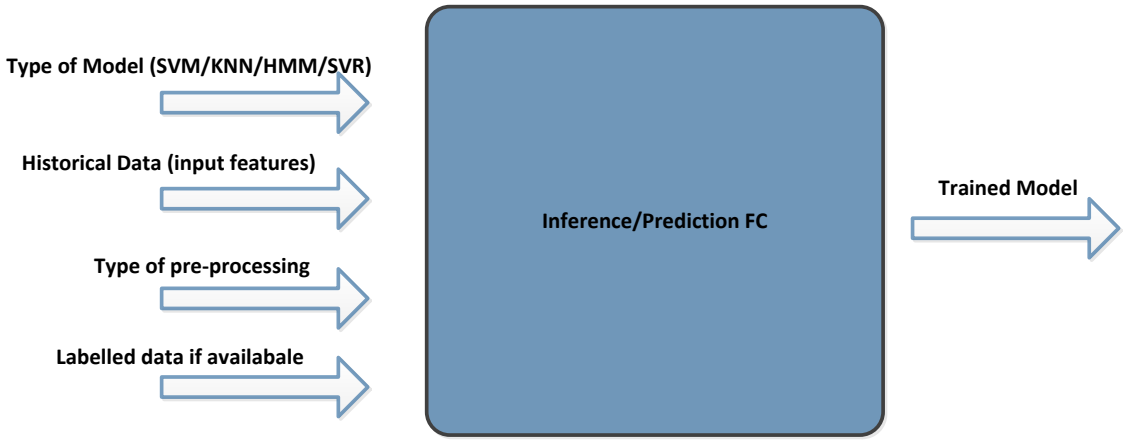


Figure 2: Summary of Inference/Prediction FC interfaces

In the future, we plan to provide an interface for connecting clients or VEs to the inference/prediction block for the following two purposes.

1) A Client or a VE will send a request using the API to register its interest in particular topic stating the interested characteristics/services (Occupancy state of a room, Traffic conditions on a road, predicted traffic state);
2) A Client or VE can also use API to get required prediction value at particular instant. An example can be a client sending a query to prediction block to find the traffic state at particular location.

2.3 Scenario Synopsis

This component of COSMOS will act on the top of other components in order to provide intelligent solutions. The purpose of this demo is to demonstrate how machine learning and statistical data analysis methods can be exploited to extract high level knowledge from raw IoT data and provide proactive solutions.

We have used the following two use-case scenarios in order to demonstrate the application of this FC.

1) Use-case scenario 1 involves inference of occupancy state from electricity consumption data for optimized energy usage for smart buildings
2) Use-case scenario 2 is focussed on providing proactive solutions for intelligent transportation systems using time series prediction mechanism. It can also be used for providing level 3 Situational Awareness.

The background knowledge and further details about the scenarios can be found in initial deliverable D6.1.2 [1]. The focus of this deliverable is on the implementation of the methods discussed in D6.1.2.

2.4 Use-case 1 : Smart Energy Management Scenario

2.4.1. Introduction

One of the core objectives of COSMOS is to provide truly smart building capabilities by contributing towards more automated and innovative applications. In this regard, occupancy detection plays an important role in many smart building applications such as controlling heating, ventilation and air conditioning (HVAC) systems, monitoring systems and managing lighting systems. Most of the current techniques for detecting occupancy require multiple sensors fusion for attaining acceptable performance. These techniques come with an increased cost and incur extra expenses of installation and maintenance as well. All of these methods are intended to deal with only two states; when a user is present or absent and control the system accordingly. In our work, we have proposed a non-intrusive approach to detect an occupancy state in a smart office using electricity consumption data by exploring pattern recognition techniques. The contextual nature of pattern recognition techniques enabled us to introduce a novel concept of third state as standby state which can be defined as:

"A state when a user leaves his work desk temporary for a short period of time and switching off HVAC and other equipment associated with occupancy state is not optimal choice"

The intuition behind our approach is that the pattern of electricity data of user will be different when the user will be at his desk and using his appliances as compared to other states. And if our algorithm recognize the pattern, it can infer the state of user as well. Furthermore, our proposed solution does not require extra equipment or sensors to deploy for occupancy detection as smart energy meters are already being deployed in most of the smart buildings.

2.4.2. Implementation and Results

We have implemented both machine learning methods and statistical inference methods for the same problem. Both techniques have their advantages and disadvantages which were analysed in our work. We have used classification analysis methods such as support vector machine (SVM) and K-nearest neighbour (KNN) which are the examples of supervised machine learning methods. They provide excellent performance but the downfall of using classification analysis is it requires labelled data which poses an extra work and additional cost. And it is an example of discriminative models which does not exploit temporal patterns formed by data. On the other hand, statistical methods like hidden markov model (HMM) can be used for temporal pattern recognition to infer hidden events using maximum likelihood theory. It does not require labelled data for inferring events from raw data. The accuracy of such methods are not too high and their complexity also increases with the increasing data size. But as stated earlier, these algorithms do not require labelled data which give these methods an edge on classification analysis methods which require labelled data for training purpose. The focus of this section is on the implementation of different models and the results. More details about the scenario can be found in the deliverable D6.1.1 [2].

2.4.3. Machine Learning Model Implementation

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 labeled data to learn and recognize the patterns. In

our case, electricity consumption data with the ground truth reality acts as training data. The total gathered data was divided in the ratio 70:30, where the larger dataset was used for training the classifier and the smaller dataset was used for validating the model. A high level architecture of the model is shown in the Figure 3.

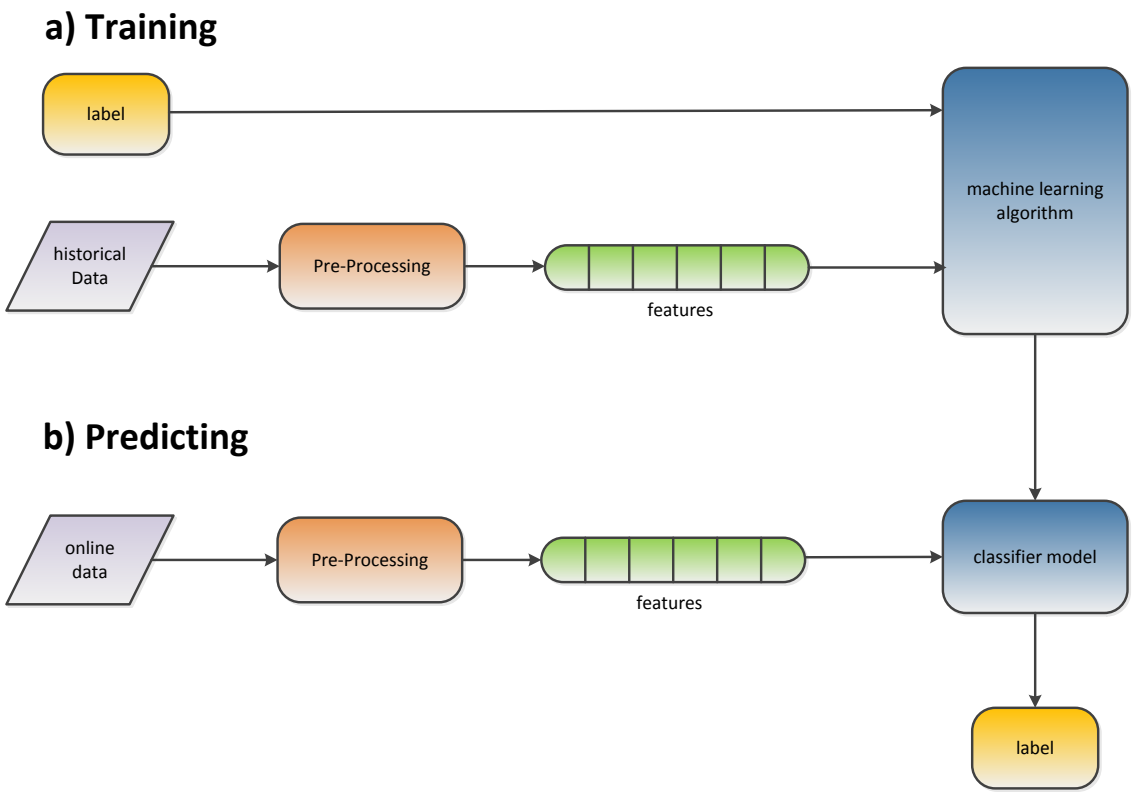


Figure 3: A general architecture of implementation of Machine Learning Model

We implemented four state of the art classification algorithms for pattern recognition which are shown in Table 1.

No.	Technique	Parameters	Abbreviation
1	K Nearest Neighbor	K = 10	knn
2	Support Vector Machine	Kernel = RBF	SVM-RBF
3	Support Vector Machine	Kernel = Linear	SVM-Lin
4	Support Vector Machine	Kernel = Polynomial	SVM-Poly

Table 1: Classification Algorithms

Support Vector Machines (SVM) is an efficient classification algorithm which is widely used for pattern recognition because of its 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 SVM. 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 Neighbor (KNN) is one of the simplest learning technique used for classification. It is a non-parametric algorithm which means it does not make any prior assumptions on the data set. It works on the principle of finding predefined number of labeled samples nearest to the new point, and predict the class with the highest votes. The advantage of KNN lies in simple implementation and reduced complexity. Despite its simplicity, it works quite well 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.

2.4.4. Results and Analysis

F-measure represents an accurate and widely used metric for comparing the performance of multi-class classifiers. We have also used F-measure to compare the performance of implemented algorithms. The equation for calculating F-measure is below

$$F - measure = \frac{2P.R}{P+R}$$

where P represents Precision and R represents Recall. And can be calculated as follows,

$$Precision = \frac{TP}{TP+FP} \; ; \quad Recall = \frac{TP}{TP+FN}$$

where TP is true positive, FP is false positive and FN is false negative.

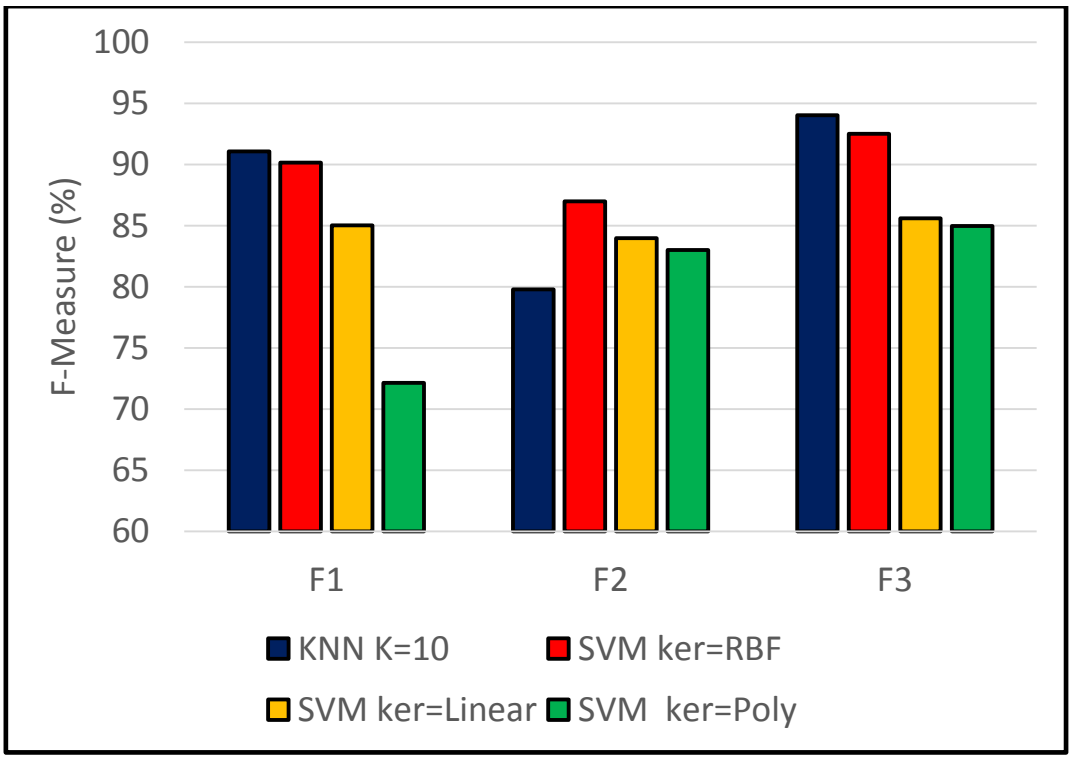


Figure 4: Performance of Classifiers for Different Feature Sets

Figure 4 shows the F-measure plot of different classification algorithms implemented for all three feature sets. From the figure, it is obvious that KNN performs best for F1 and F3 and achieves accuracy up to 94% 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 an 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 this report.

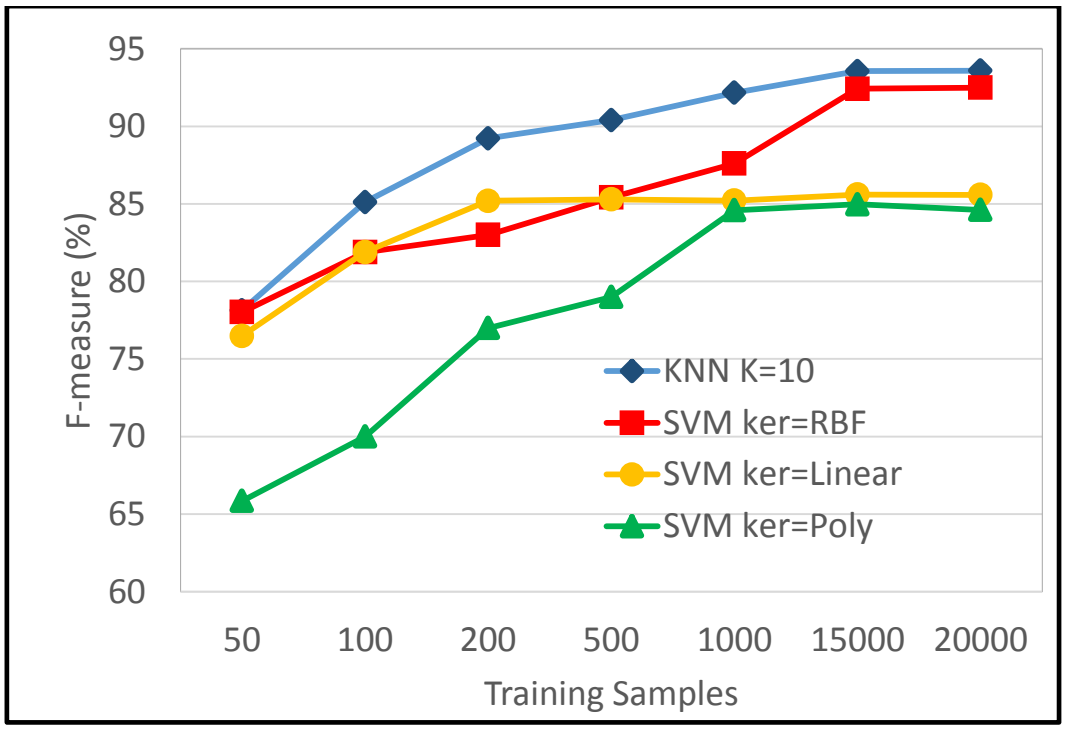


Figure 5: F-measure relation with training data set

The number of training samples plays an important role in the performance of a classifier. As the training samples increase, the decision boundary becomes more accurate and the performance of classifier improves. Figure 5 shows how the F-measure of different classifiers improve 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 non-linear 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.

A large commercial building can have hundreds of nodes and the amount of data can be huge. In order to provide 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 on a personal laptop (Intel i7-4510, 12GB RAM, Ubuntu 64 bit operating system) is plotted in Figure 6. 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 memorizes the all 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 optimization algorithm) and results in exponentially increasing time with increased number of training samples.

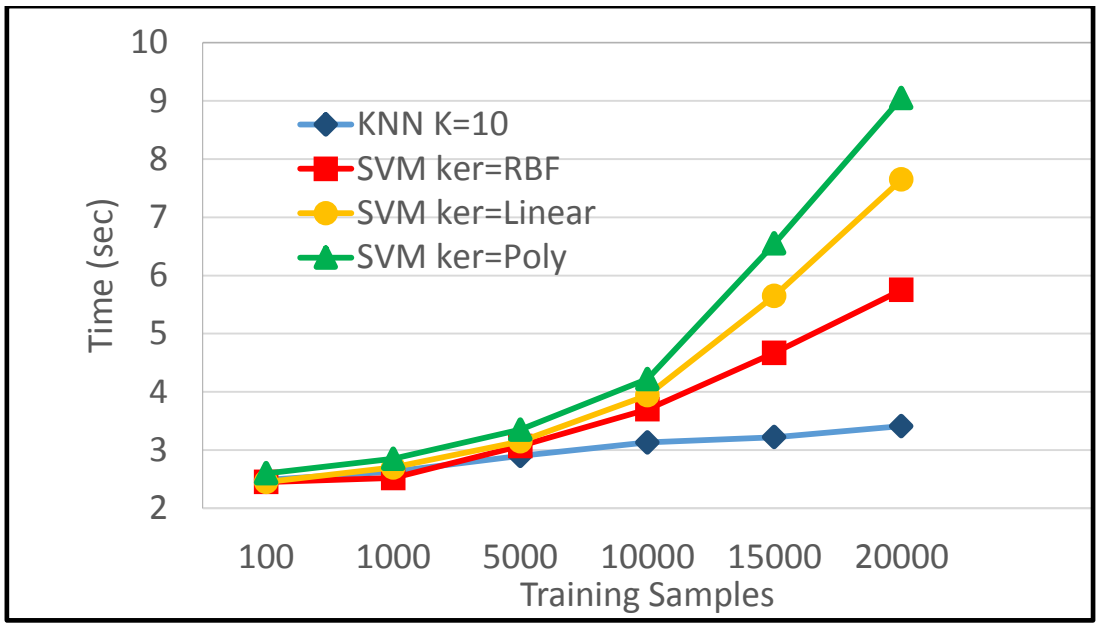


Figure 6: Time Complexity Comparison

2.4.5. Statistical Approach

Occupancy detection follows a temporal relation between its states. For example a standby state will always occur after present state and it is not possible to go directly into standby state from absent state. We have discussed in D6.1.1 [2] that HMM can be used for inferring events when labelled annotations of output are not available using expectation maximization algorithms. In this scenario, electricity consumption data will be our observations and occupancy states will be hidden states of Markov Model.

We have used the electricity consumption data to train the Hidden Markov Model. The EM algorithm assumes that the observations are discrete in nature but in our case the different features of electricity consumption data can take continuous values. In order to overcome this issue, we have assumed that the observations follow Gaussian distribution and applied GMM on the observation data set to make it discrete.

The maximum efficiency we got using HMM was 82%. A transition probabilities from one state to other is shown in the Figure 7. It is clear from the figure that the transition probability in any state is more optimum to be remain in that state for next sampling instance. The reason is that the sampling instance is approximately 10 seconds and it is highly probable that the user will be in that state for the next sampling instance as well.

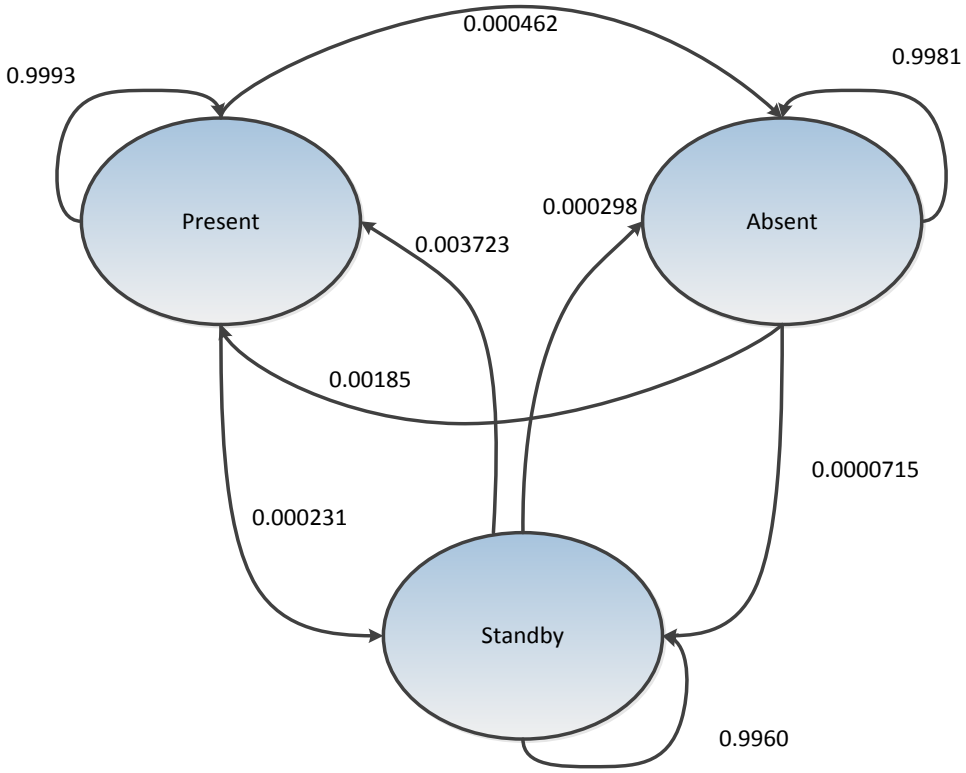


Figure 7: Transition Probabilities for HMM

2.5 Use-case 2: Intelligent Transportation System

Intelligent transportation system represents one of the most important aspect of smart cities with a strong impact on the potential development of the city and the quality of life. In this regard, the city of Madrid has deployed thousands of sensors which measure different features of traffic. They help the city administrators to understand the traffic flow and take reactive measures in order to ensure smooth traffic flow. Data from different sensors can be combined using Complex Event Processing (CEP) for detecting complex events such as congestion or any accident.

Most of the current methods are focussed on detecting congestion once it has occurred, but it is a common observation that traffic does not become bad from good at once. It follows certain patterns and if we are able to predict them, we can take proactive measures to avoid traffic congestion before it happens. In this section, we explain our proposed solution with the initial results.

2.5.1. Implementation

We have explored several regression methods for predicting time series data of traffic features. Regression analysis is one of the most widely used machine learning tools, which is used to define a relation between variables so that the values can be predicted in future using the defined relation. In our case, Support Vector Regression (SVR) with RBF (Radial Basis Function) kernel outperforms other methods in terms of accuracy.

SVR is an extension of SVM which is widely used for regression analysis. The main idea is the same as in SVM, it maps the training data into higher feature space using kernel functions and find the optimum function which fits the training data using hyper plane in higher dimension.

There are several challenges which have to be addressed when the models are trained using historical data. Models trained on historical data may not be accurate when deployed on real situations and in case of erroneous readings, the error propagates and it keeps on increasing. In order to overcome this issue, we proposed to use a moving time window for training models and update the models using latest readings.

The city of Madrid council publishes traffic data every 5 minutes. It includes traffic features such as average traffic speed and average traffic intensity which are direct indicatives of traffic state. We used a moving time window of 2 hours i.e. use the data of only 24 (120min/5 min per reading) latest readings and train the model using these readings to predict the traffic values for next 10 minutes (2 readings). We can also predict further, but as we increase the prediction time, accuracy of predicted values decrease. Hence, we started with 10 minutes prediction and we plan to build on these results to predict further ahead with more accuracy in future.

2.5.2. Results

Figure 8 below shows the initial results of our implemented prediction algorithm on average traffic speed readings. As it can be seen, predicted values are tracking the actual data quite accurately. The reason behind this, is that we update the model after every new reading. In this way, if there is an error in the predicted value, it is incorporated and model is updated accordingly and hence it stops the error from propagating. First 24 data points were used in training the model initially.

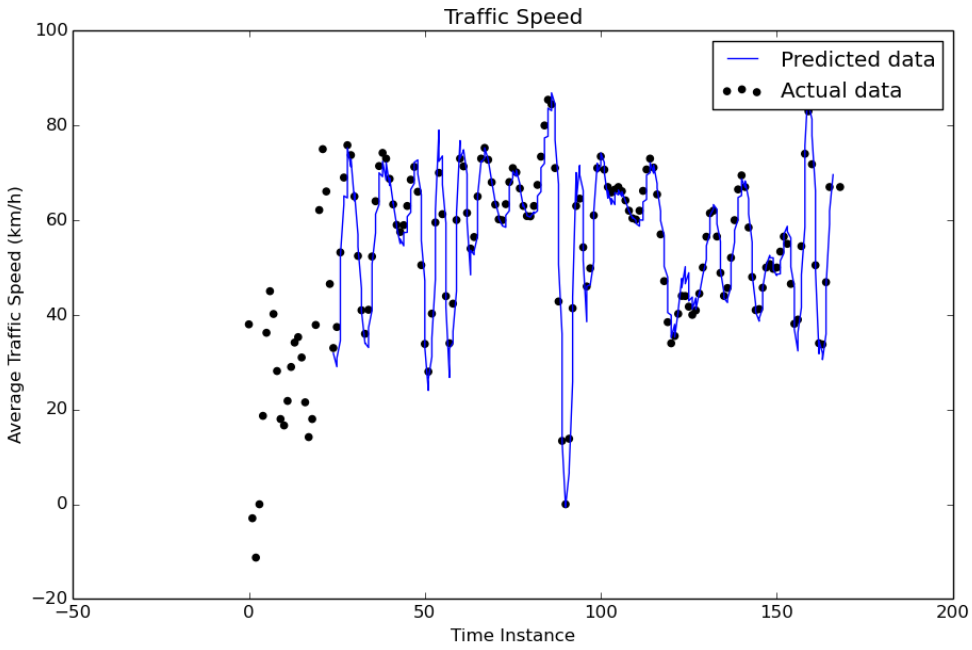


Figure 8: Traffic Speed Prediction

Similarly, Figure 9 shows the initial results for prediction on average traffic intensity. The size of moving window is the same and it also predicts for next 2 readings. In our future work, we intend to explore prediction further ahead.

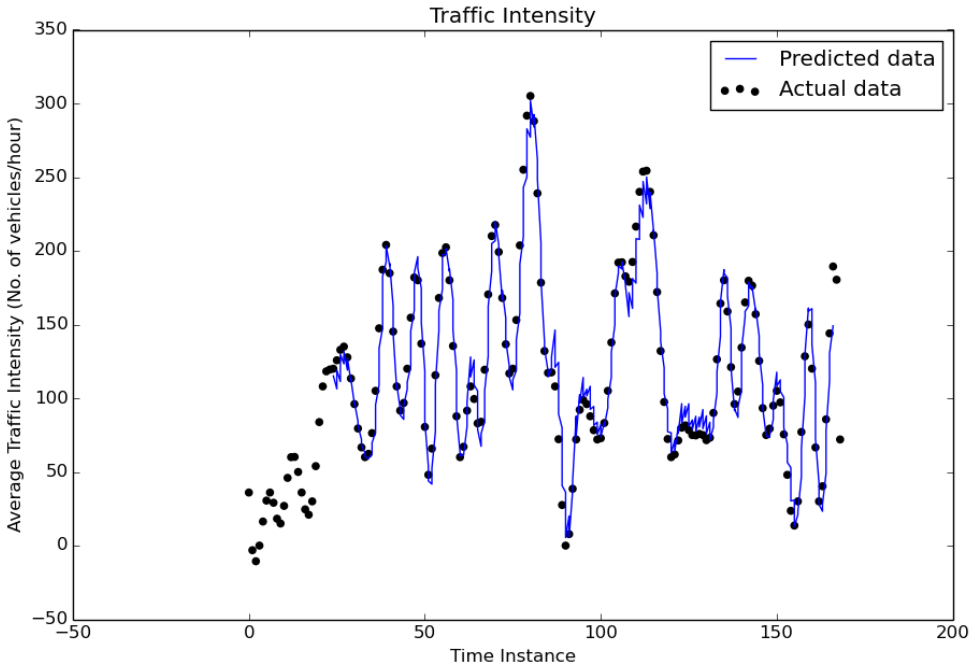


Figure 9: Traffic Intensity Prediction

2.6 Sequence chart

Figure 10 shows a sequence chart of model creation and updating of model using real time values, while Figure 11 shows the sequence of functions for calling prediction block.

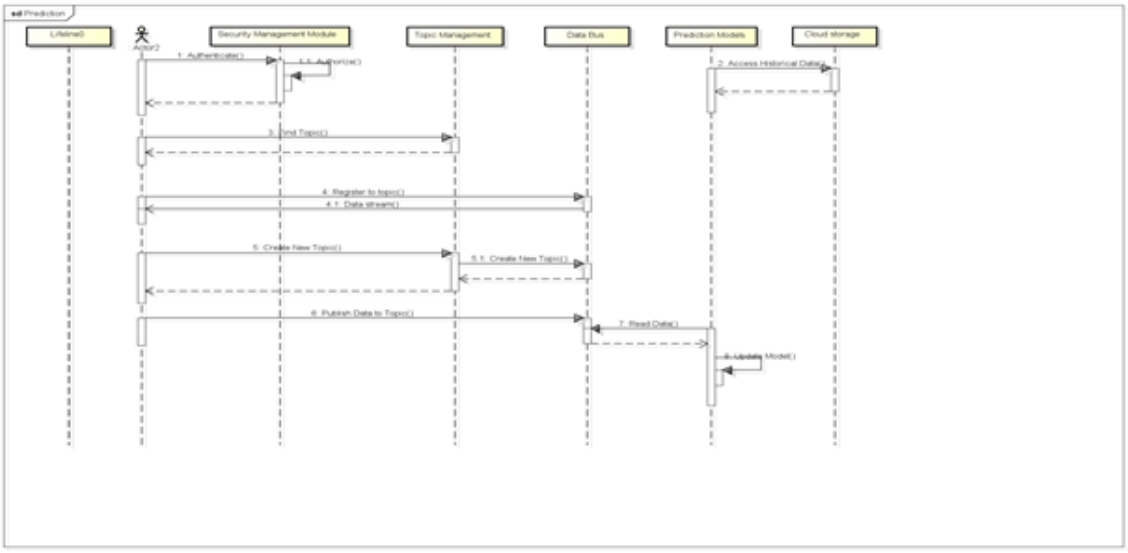


Figure 10: Updating Prediction Model

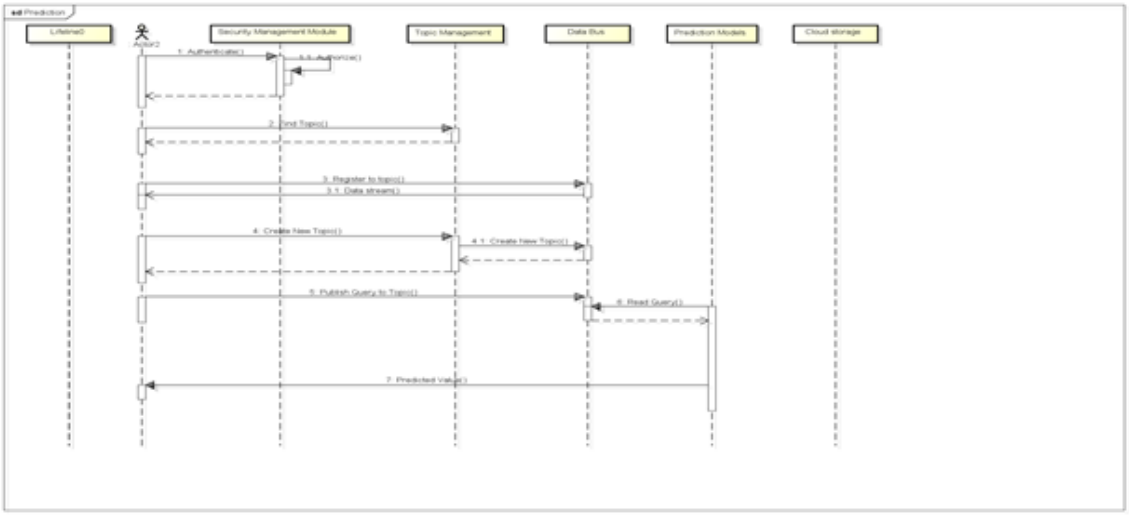


Figure 11: Calling Prediction Model

2.7 Delivery and Usage

Package Information:

It includes two folders, one for each application, accompanied with source code and data files. The code is written in Python 2.7 and is with .py extension, while the data file is in .csv format.

Pre requisites

Python 2.7 and the following open source libraries are required:

1) Numpy
2) Scikit
3) Pandas
4) Datetime
5) SciPy

Occupancy Detection App:

1) The source code for all the algorithms implemented can be found in the folder occupancy_detection with the following names;
    I. occupancy-svm-lin-kernel.py
    II. occupancy-svm-poly-kernel.py
    III. occupancy-svm-rbf-kernel.py
    IV. occupancy-k-nearest.py
    V. occupancy.hmm.py
2) Interested algorithm can be run in a python console by first giving the path of the data file.
3) Every algorithm will first divide the data file into training data and test data. Then it will train the models using training data set and will then evaluate the model using test data set and will print the confusion matrix.
4) Classifier divides the output state into three states, as described previously i.e. present, absent and standby state.

5) Confusion Matrix shows the efficiency of classification results.
6) Future versions of the application will be provided with a GUI in order to assist the running protocol.

Intelligent Transportation Prediction App:

1) The source code for the prediction algorithm is uploaded on the SVN and can be found in the folder prediction.
2) Set the path for historical data file which is used for training and validating the prediction model.
3) Mention the name of input feature vector in which the user is interested to find predicted values.

Implementation on Spark:

We implemented the use-case scenario 1 on Spark platform which is being developed in WP4. Use-case scenario 2 is still a work under progress but we plan to implement it on Spark platform as well in order to make it scalable for large amount of data available in COSMOS.

Licensing Information:
All the codes are developed in Python, which is an open source software.

3 Pre-processing Functional Component

Pre-Processing is a generic component and different components such as inference/prediction FC and Event (Pattern) detection can use it according to their requirements. It involves several functions ranging from simple data cleaning mechanisms to more sophisticated mechanisms such as data aggregation or feature scaling.

3.1 Architecture

Pre-processing FC is present at following three levels to provide pre-processing depending on the context of application.

1) On VE for processing raw data streams using μcep for filtering and aggregation
2) On object storage using storlets for carrying simple pre-processing tasks
3) On platform level using data analytics platform

Pre-processing on raw data provides simple functionalities like filtering or aggregating the data whereas VE data pre-processing is more complex and provides a fundamental step for ML algorithms. Also, application developers can write domain specific pre-processing using storlets. Figure 12 below shows the pre-processing component and its connections with other components in COSMOS architecture.

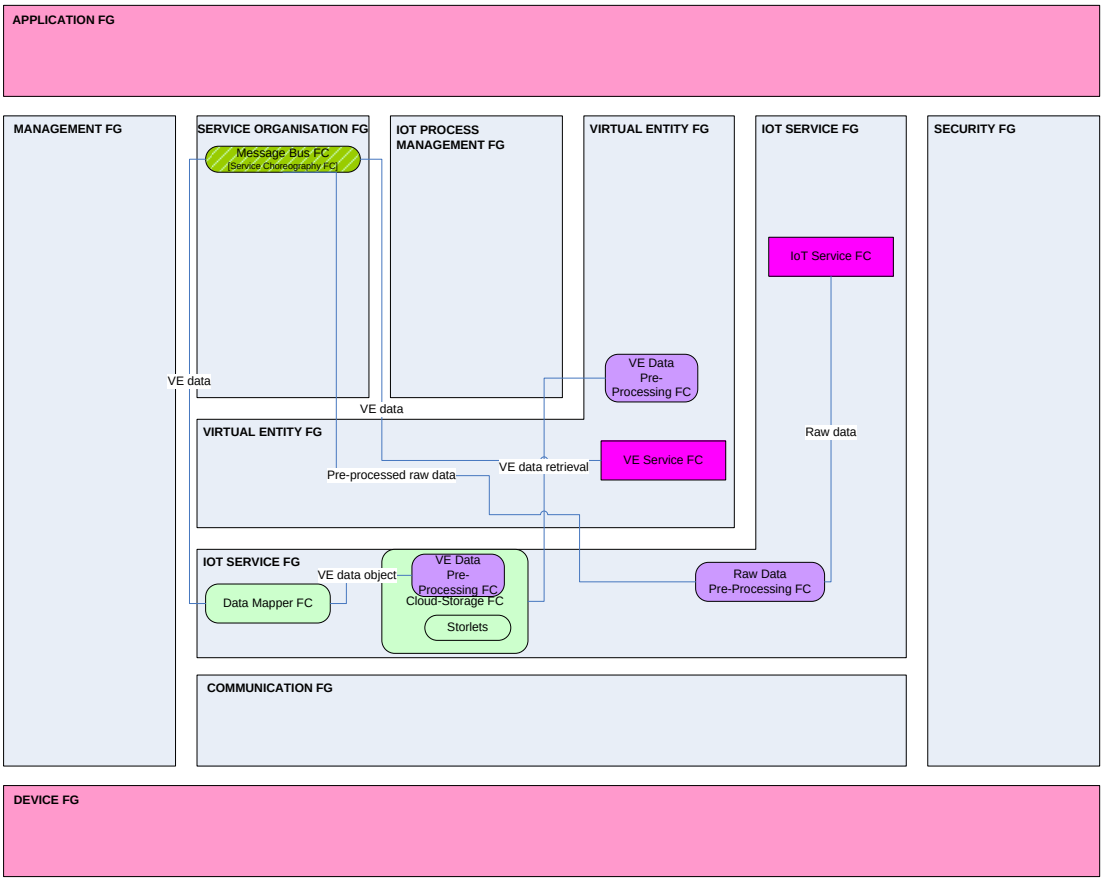


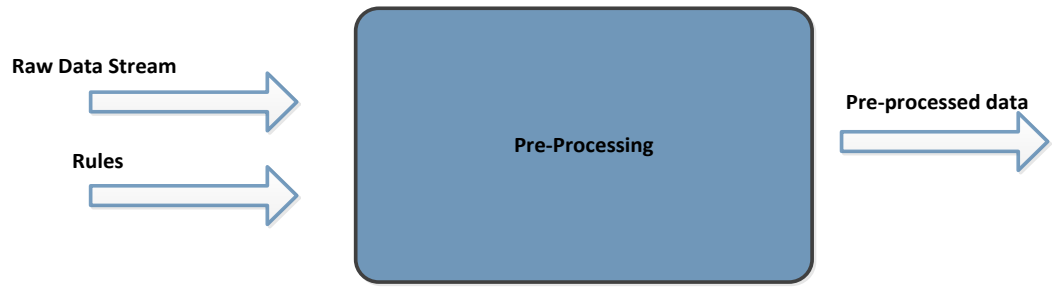
Figure 12: Pre-Processing Fc Architecture

3.2 Interfaces

As shown in Figure 12, Pre-processing component is present at three different levels and the interfaces for every level are described below:

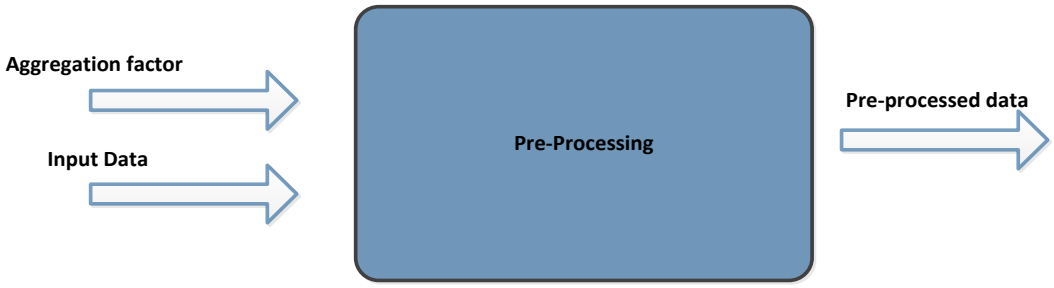
VE Level using CEP

An interface is provided to define rules for μcep using DOLCE language for simple pre-processing such as filtering or aggregation at VE level. The output will be published to Message Bus under the specific topic.



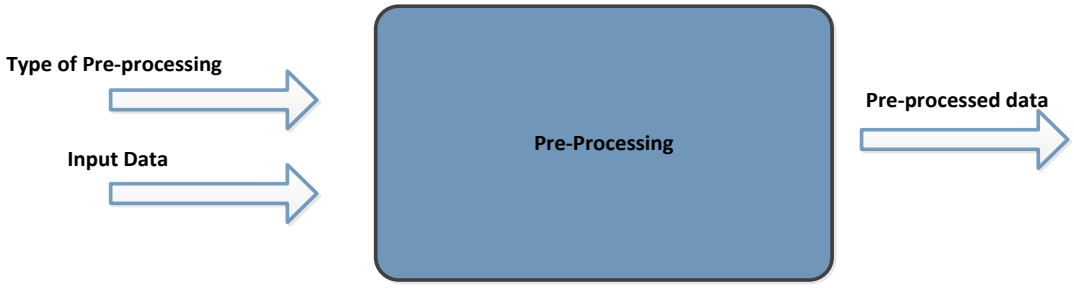
Object Storage using Storlets

An application developer can code its own storlets in java or can use generic storlets provided by COSMOS for aggregation. Aggregation storlet will take the aggregation factor and the input features as input.



Platform level using Apache Spark

Different pre-processing methods are provided which can be specified when choosing a particular data mining method.



3.3 Scenario Synopsis

As discussed earlier, this component can be used at the platform level inside spark or at the object storage using storlets. The advantage of carrying pre-processing at object storage is of immense importance as it will decrease the amount of data travelling over the network.

3.4 Use-case Description

We have extended the use case scenario 1 from Section 2.4 for Smart energy management in order to demonstrate the application of pre-processing component for optimizing the inference functionality.

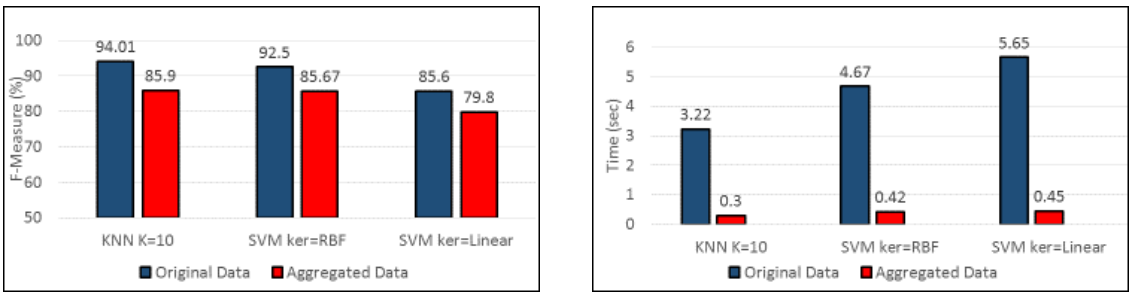
3.4.1. Implementation and Results

Sampling rate of the data gathered for occupancy detection scenario was 10 seconds. The sampling rate of 10 seconds might seem redundant for occupancy detection as it is highly unlikely that the state of the user will change in the period of 10 seconds. We explored the possibility of applying aggregation techniques and analyse their effect on the accuracy and time complexity of different data mining algorithms. We aggregated the data for one minute into a single reading by using Peacewise Aggregation Approximation. Figure 13 shows the comparison of accuracy and time complexity of the following classification algorithms.

1) K Nearest Neighbor (KNN)
2) Support Vector Machine (SVM) with Radial Basis Function (RBF) kernel
3) Support Vector Machine (SVM) with linear kernel

Although the accuracy of algorithms has dropped a little but the gain in reducing time complexity is huge.

Figure 14 shows the results for the Hidden Markov Model (HMM). The left figure shows the earlier results without aggregation as we have already explained in Section 2.4.5. We aggregated the data for 10 minutes and applied HMM model on aggregated data which is shown in the right figure. Although the accuracy of the model has dropped to 70% but as less data have to be processed by HMM, it results in computationally less expensive algorithm. There is a trade-off between the complexity and accuracy of algorithm as the training data is decreased or increased.



(a) Accuracy

(b) Time Complexity

Figure 13: Performance of aggregated and original data for ML

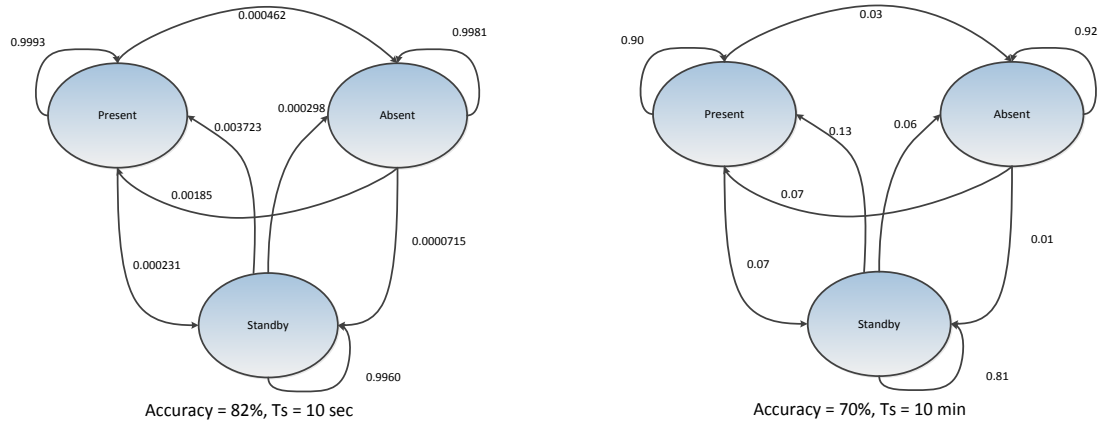


Figure 14: Performance of aggregated and original data for HMM

One might argue for applying pre-processing tasks before the data is stored using middleware as a better option. In our case this is not an optimum choice as occupancy detection is one application which can be inferred from raw data at low sampling rate. There are other applications such as *Non-Intrusive Load monitoring* (NILM) [3], for detecting which devices are being used and requires very high sampling rate. In such scenario, if we apply pre-processing at middleware platform, it will limit the use of data to specific applications and valuable information would be lost. In this regard, we proposed to store all raw data and apply pre-processing techniques at the object storage side.

3.5 Delivery and Usage

Pre requisites

It uses the same libraries as mentioned in Section 2.7 alongside python 2.7. The libraries are mentioned again below:

1) Numpy
2) Scikit
3) Pandas
4) Datetime
5) SciPy

Usage:

Most of the pre-processing tasks were done using pre-built libraries and functions but nevertheless we have to write our own codes for pre-processing using storlets. There is no separate code for pre-processing as all of the use-case for pre-processing was done with other components. We have uploaded the code for use case scenario described in Section 3.4 in the pre-processing folder and it can be run in python shell by just giving the path of historical data file (electricity data).

4 Event (Pattern) Detection Functional Component

This component is intended to provide the functionality for near real-time processing of data for event detection by providing a hybrid solution based on Complex Event Processing (CEP) and Machine Learning (ML) methods. CEP provides the distributed and scalable solution for analysing data stream in near real-time manner, but it does not exploit historical data and the manual setting of rules is a major drawback. Rules for CEP are static and hence solutions provided by CEP are static as well. Though ML methods exploit historical data and provide more automatic solutions, they are unable to provide near real time solutions and scalability is a major issue associated with them. In our proposed solution, we exploit both approaches and combined them in order to provide a near real time solution which is more context aware, adaptive and exploits historical data as well.

4.1 Architecture

Event Detection FC is using VE-level stream analysis capability provided by COSMOS. As it can be seen from Figure 15, raw data is generated by IoT Services and Event Detection block correlates the data using DOLCE rules provided by application developer in near real-time to infer a complex event. Also, at the same time raw data is published on the Message Bus and stored in the cloud storage. Inference/Prediction FC exploits this historical data and run machine learning algorithms to estimate optimized parameters for CEP rules which will be updated automatically through an interface provided between Inference/Prediction FC and Event Detection FC.

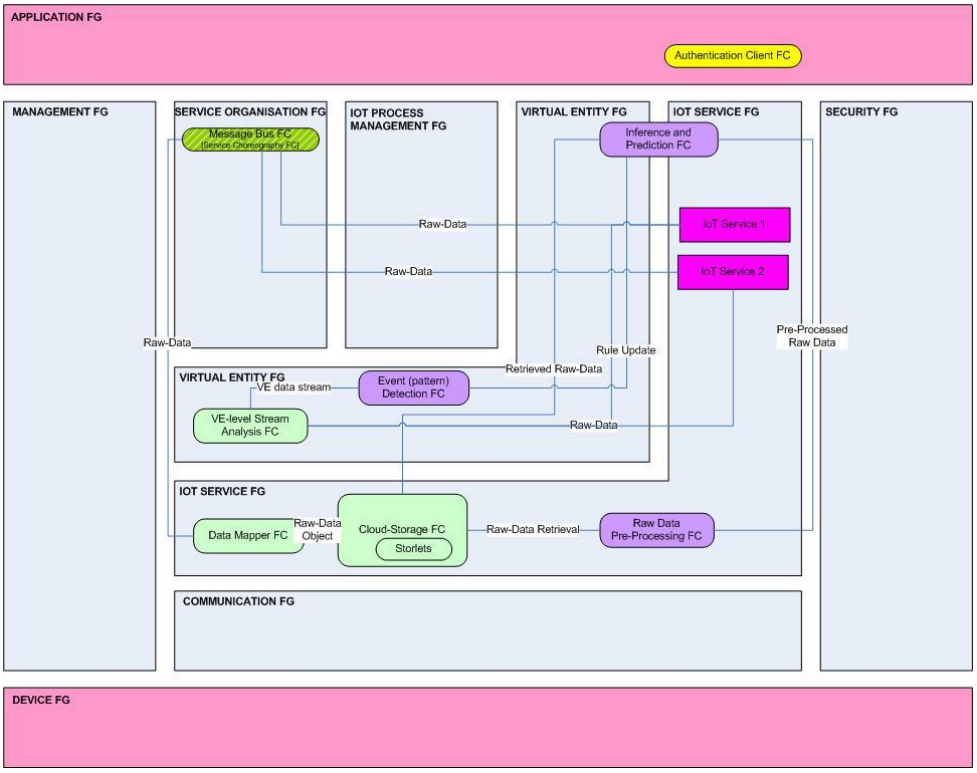


Figure 15: Event Detection FC Architecture

4.2 Interfaces

Event (Pattern) detection functional component has two main components which are:

1) Machine Learning exploiting historical data
2) Event Processing Network based on CEP for real-time analysis of data stream

The interfaces for both components are shown in Figure 16 below.

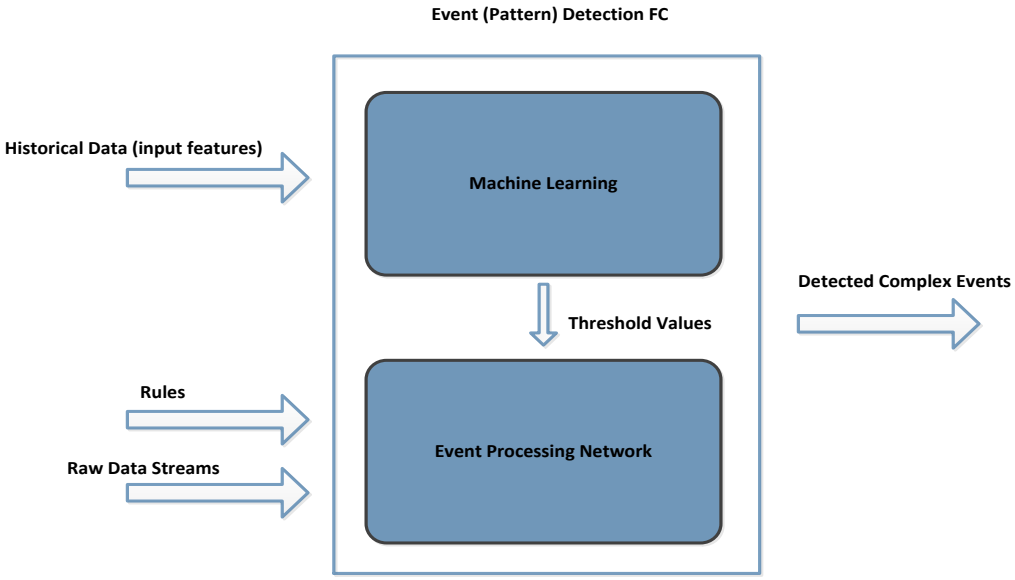


Figure 16: Interfaces for Event (pattern) detection FC

VE - µCEP

The µCEP engine that is being used in COSMOS utilizes the DOLCE domain language for the definition of rules. A *DOLCE Rule* file must have at least two clauses: `Events` and `Complex Events`. The former represents the data streams received by the engine, while the later represents the data that will be outputted in case a rule is triggered –i.e., when a *Detect clause* is evaluated to *True.*

The injection of data streams into the µCEP is done using the Message Bus, and the same applies for outputting the complex events as *Value Added Info*. The *Event Detector* module will be subscribed to various *Topics,* and the *Complex Event Publisher* will be publishing to certain *Topics*.

Application Developer - µCEP

Apart from the data streams that are willing to be analysed, there is an additional entrance point to the engine, the *DOLCE Rules* file. Application Developers are able to modify it in two different ways:

1. Editing a **.dolce* file on their own, using their preferred Text Editor. This approach provides full control over the definition of the rules
2. Using a specific Web Graphical User Interface (GUI) developed by COSMOS project, which facilitates the creation of a *DOLCE Rules* file in an easier manner, following a guided wizard

In either case, the file has to be deployed into the µCEP running instance, which is possible to be done using a specific API function over the *REST Administration* interface.

Event Detection-Inference/Prediction
An interface will be developed to connect the Event detection component with the Inference/Prediction component for updating the threshold values for the CEP rules automatically using ML methods.

4.3 Use-case description – Intelligent Transportation System

We have used Madrid traffic scenario in order to demonstrate our approach. The city of Madrid council have thousands of static traffic sensors deployed in the Madrid which are measuring different traffic features such as average traffic speed (km/h) and average traffic intensity (No. of vehicles per hour).

The use of speed sensors is becoming common nowadays for managing traffic. It gives the overall idea about the traffic state but the obvious can be misleading sometimes and the "grandma" example illustrates nicely- *"if old grandma is driving down an empty highway at low speed at night, the traffic speed will be a poor indicator of traffic status (e. g. slow moving instead of free flow)".* Another indicative of traffic state is the average traffic intensity (No. of vehicles per hour). High traffic intensity indicates that large number of vehicles are flowing which can indicate busy traffic state but at the same time can indicate free flow traffic. And the low traffic intensity means less busy traffic hours or may be a bad traffic state which is limiting free flow of traffic.

Both of these parameters are indicative of traffic state but can be misleading when used alone. We proposed to combine these parameters using CEP rules in order to detect traffic state with more reliability. For CEP rules, we need threshold values to put limits on average speed or average traffic intensity before classifying it as a bad traffic state or a good traffic state. In conventional systems it is done manually using administrator prior knowledge or his decision power which is a major drawback. But as discussed earlier, we applied clustering techniques in order to collect these threshold values automatically for detecting traffic state. We exploit historical data for the specific measuring point and threshold values are optimized for that measuring point. Our solution is general as different measuring points will have different threshold values according to the context and behavior of that particular measuring point.

The novelty of our approach will be to find the threshold values for CEP rules using clustering techniques from ML domain and update it iteratively.

4.4 Implementation and Results

High-level block diagram of our approach is shown below in Figure 17. It consists of two main components which are:

1) Event Processing Network
2) Adaptive Clustering

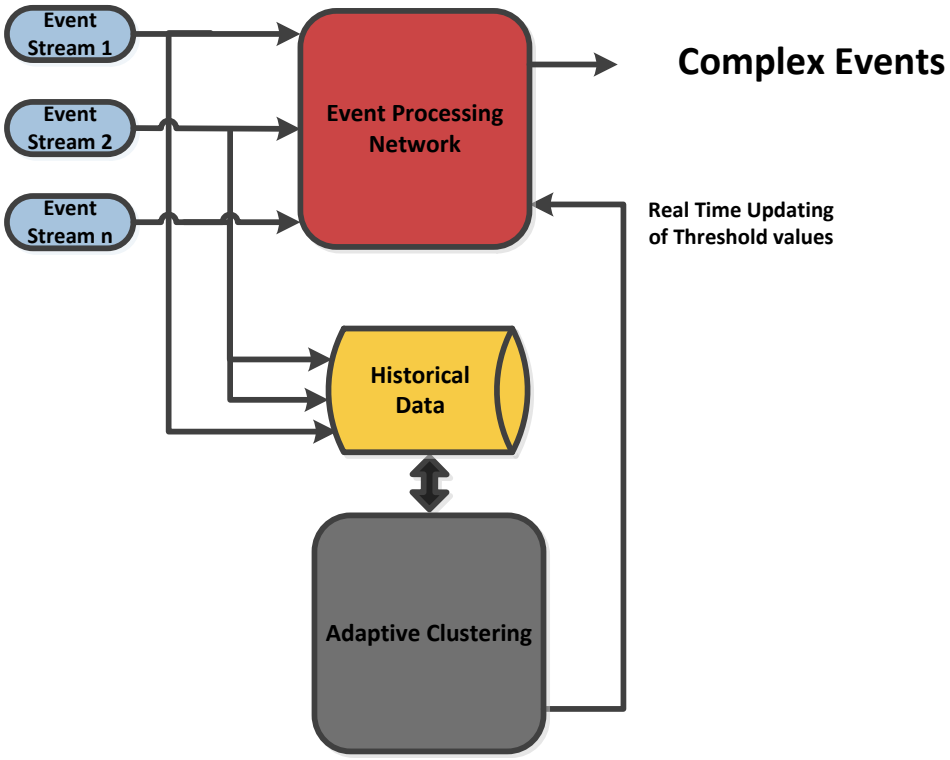


Figure 17: Block diagram for Event Detection

4.4.1. Adaptive Clustering

In this section, we explain how clustering can be exploited to find threshold values for CEP rules automatically. Clustering refers to the unsupervised machine learning technique which is used for grouping similar objects on the basis of pre-defined metric such as distance or density. Cluster analysis is a general technique used widely for knowledge discovery in big data sets and several variants of algorithms are found in the literature. The choice of a particular clustering algorithm and parameters (such as type of metric, threshold and number of clusters) is governed by the problem scenario and the data sets involved in the problem.

Cluster analysis separates the data into different possible events and group the data points from the same event together. Cluster analysis can help to find the boundaries between different events which can serve as threshold values for CEP rules.

In general, threshold values for CEP rules are static and is a major drawback when deployed in real-life dynamic environments. Threshold values may become inaccurate with the passage of time. Statistical properties of the underlying data may change over time. Automatic methods are required to find threshold values which are able to adapt automatically. In our proposed method, we update the clusters regularly using predefined metrics and hence update the threshold values accordingly.

In real-time scenarios, the definition of context is changing with time. In morning hours, traffic intensity is usually more with low average speed and hence the definition of bad traffic is different from the night hours when traffic is smoother. The response of traffic is different at different time periods and hence we proposed to have different threshold values for different time periods. Silhouette index is used for accessing the quality of clustering. Silhouette index is calculated for new values and if its value is less than threshold, we retrain the model and find new parameters.

4.4.2. Implementation

We have implemented clustering in Spark using MLlib library. Historical data is being stored in the object storage and it is accessed using Spark SQL. We extracted and filtered data using Spark SQL queries. More details about the storage and accessing the data can be found in D 4.1.2 [4]. For applying machine learning algorithms on Spark SQL data, spark introduces a relatively new library called spark ML, which is still a work under progress offering only few basic algorithms. In order to use existing machine learning libraries from Spark MLlib, we need a wrapper in order to convert Spark SQL data frame into RDDs. Figure 18 shows different steps involved in the implementation of adaptive clustering. In the current version of prototype, feature scaling is done in Spark but in future, we intend to explore the possibility of using storlets for pre-processing.

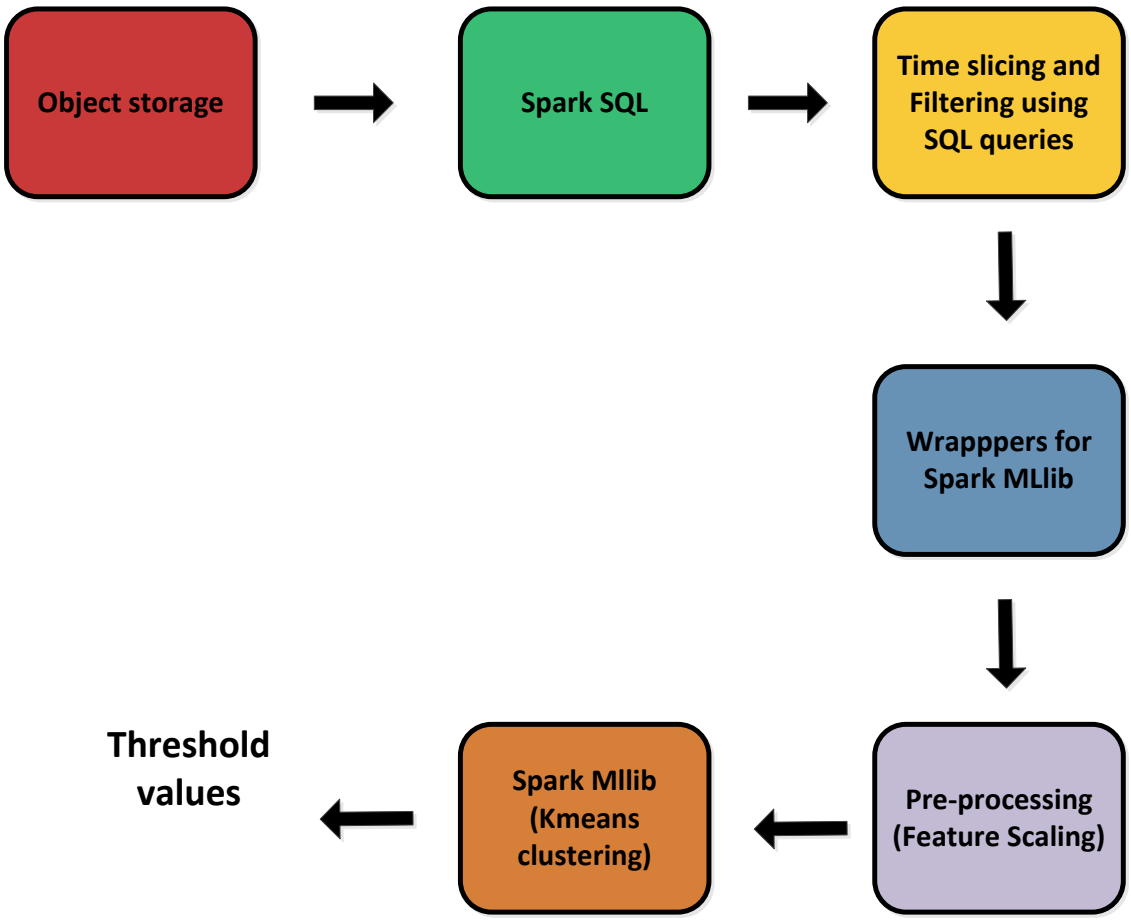


Figure 18: Implementation of Clustering for finding Threshold Values

Spark SQL queries enabled us to carry functions like time slicing or feature selection in a simple manner. For example, if we want to extract data only for traffic intensity and traffic velocity and only between 8am to 12 pm from all historical data for specific measuring point PM10001, the resulting SQL query will be:

df = sqlContext.sql("SELECT code, intensity, velocity FROM madridtraffic WHERE tf >= '08:00:00' AND tf <= '12:00:00' AND code = 'PM10001'")

Where madridtarffic is an SQL table registered for Madrid historical data.

4.4.2.1. Results

Figure 19 shows the clustering result for different traffic hours as described in the Table 2 below. As can be seen from the Table 2, we define the morning hours as 8 am to 12 pm. We load the historical data of the last month between the time range mentioned and apply k-means clustering with k = 2. It results into two clusters as can be seen in the figure below. Blue cluster represents high average traffic speed and high traffic intensity means traffic is flowing smoothly. It represents good traffic state. Whereas red cluster represents traffic points with low average traffic speed and low traffic intensity which represents bad traffic state. Midpoint between both cluster centres represents the boundary separating both states and we used this boundary to define threshold values for detecting complex Events.

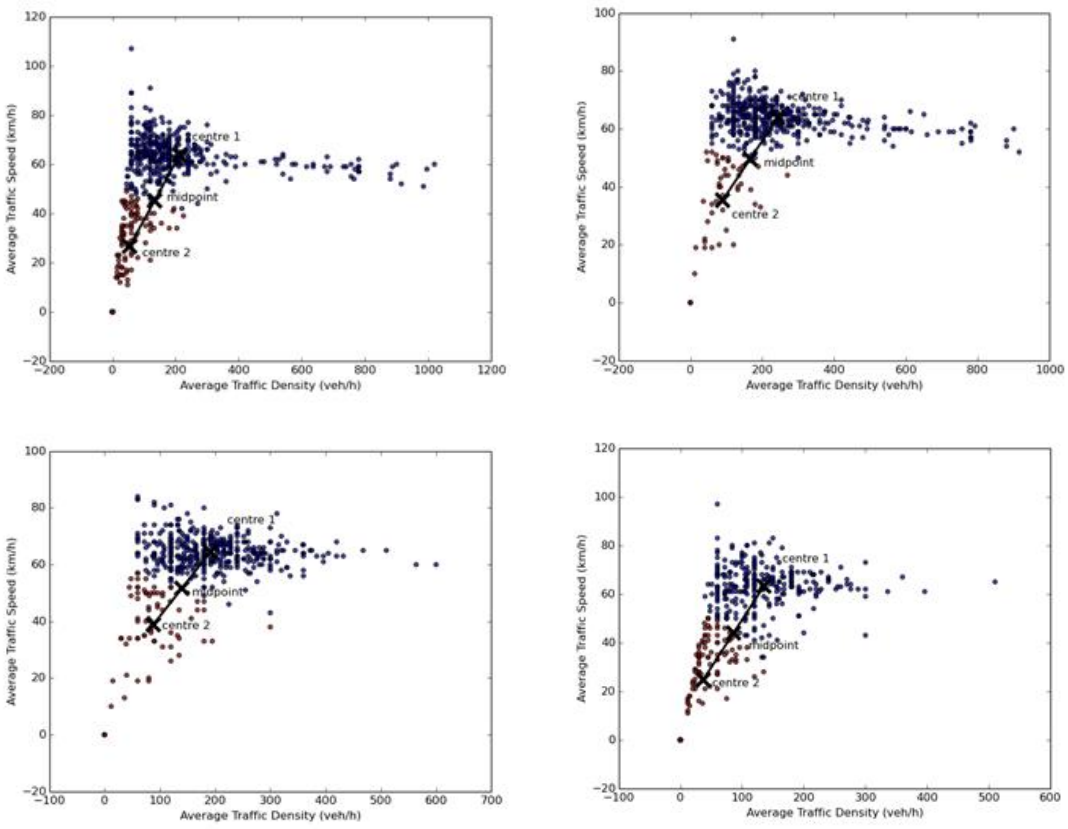


Figure 19: Clustering Models a) Morning traffic hours b) Afternoon traffic hours c)Evening traffic hours d) Night traffic hours

No.	Traffic Period	Time Range	Threshold Values (midpoint)	silhouette index
1.	Morning traffic hours	8 am to 12 pm	(133 veh/h, 45 km/h)	0.53
2.	Afternoon traffic hours	12 pm to 4pm	(166 veh/hr, 49 km/h)	0.56
3.	Evening traffic hours	4pm to 8pm	(133 veh/hr, 58km/h)	0.52
4.	Night traffic hours	8pm to 12 am	(86 veh/h, 44km/h)	0.51

Table 2: Threshold Values for different time periods

4.4.3. Example Rules for CEP

Different combination of rules can be applied to infer temporal and casual pattern using CEP. In this section, we describe an example set of rules for inferring bad traffic state using the data sources and threshold values explained earlier.

In this example, μCEP detects an event when the average traffic speed and average traffic flow is less than the threshold values and then checks if the values are continuously decreasing with respect to previous recorded values. Different steps involved are

1) Use sliding tuple window for 3 readings (i.e. take last 3 readings) for traffic speed and traffic flow as *TrafficSpeed1, TrafficSpeed2,* and *TrafficSpeed3; TrafficFlow1, TrafficFlow2* and *TrafficFlow3*.
2) Apply rules as; *TrafficSpeed3* < ***ThresholdSpeed*** and *TrafficFlow3* < ***ThresholdFlow***, *TrafficSpeed2* < *TrafficSpeed3* and *TrafficFlow2* < *TrafficFlow3* (which indicates that the next reading is worse than the previous reading), *TrafficSpeed1* < *TrafficSpeed2* and *TrafficFlow1* < *TrafficFlow2* where *TrafficSpeed1* and *TrafficFlow1* are the latest readings.

Generate Notification: Critical warning of bad traffic or congestion.

This is just one example in order to demonstrate how CEP rules can be exploited to find more complex events. We are exploring other options to exploit rules to infer more detailed information from the data available.

4.5 Delivery and Usage

There are two main functionalities involved in this component. Firstly, finding threshold values using ML methods and publishing the values on Message Bus; secondly using these threshold values in μCEP rules for detecting complex event.

4.5.1. Machine Learning component

This part of the component is implemented using Spark SQL and Spark mlLib libraries

Pre requisites:

1) Download spark latest version
2) Configure it to access object storage
3) Install mqtt publisher

Usage:

Code is uploaded on the SVN under folder Event detection with the name “clustering-demo.py”. It can be run by opening python shell in spark and executing it. It will access historical data, apply clustering, find suitable threshold values and publish them on the mqtt Message Bus. It will update the threshold values as more data is stored in the cloud storage.

5 Situational Awareness (SAw) Functional Component

5.1 Architecture

From the amount of possibilities to design a SAw architecture, in this prototype we have designed one that takes advantage of –if not all— a good number of already built COSMOS functional components. The following diagram represents the architecture used by the prototypes of Year 2.

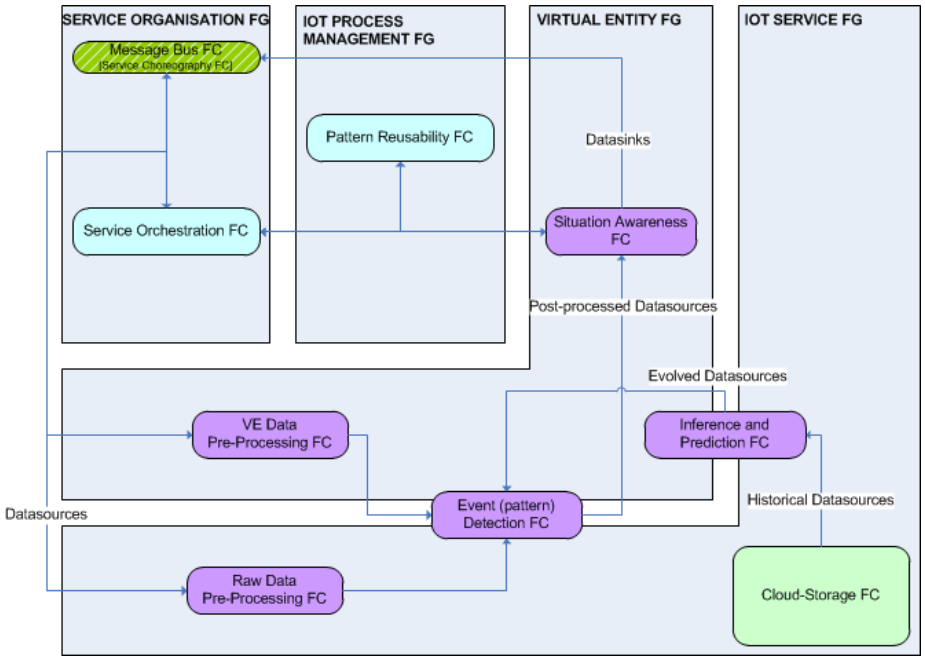


Figure 20: Situational Awareness Architecture

Apart from representing the process of data collection, understanding and projection, this diagram includes the necessary components that enable a generic instantiation of a SAw project in an application. While some of the FCs have been already described in previous sections, others like the Service Orchestration FC complete the automation of the data gathering step from the Message Bus, through the Pre-processing FCs and the Event Detection FC, up to their ingestion into the Message Bus again. In addition, it is in charge of handling the correct analysis that will be applied to the data sources at the Event (pattern) Detection FC. With respect to evolution of the data –SAw Level 3—, the Inference and Prediction FC is fed by the necessary historical data sources. In brief, the SAw component includes:

- The collection of raw data from the Message Bus and Cloud Storage
- Applying different reasoning techniques to such data to synthesise higher-level context information
- Population of the value-added generated information to make it available when required
- Coordination of related information between different components of the system and provision of a process for building, hosting or enabling context-aware applications and services

5.2 Interfaces

The insights related to the interfaces between the Functional Components that are involved in the SAw process have been explained in their specific sections across this document. As it has been said, what is particular for the SAw FC is the automation of the selection of the *channels* or *topics* to subscribe to, the provision of the analytic routines to be applied, and the population of the value added generated information for those applications that will take advantage of it. The way of facing this issue will be achieved at the Service Orchestration FC. The features that implement this component comprise the parsing of the analytic rules in order to extract the metadata information related to the data sources; once obtained, the information of these data sources will be ingested into the Message Bus so that the components involved in SAw FC are able to use them; finally, additional *channels* will be provisioned again based on the parsed metadata. Additionally, a specific endpoint may be made available as a single point of contact to monitor the runtime conditions of the SAw component.

5.3 Scenario Synopsis

Some of the use cases already presented in COSMOS –or part of them— are directly or indirectly following a Situational Awareness process. For instance, both the detection of a traffic jam and the smart energy management are cases in which the context of the entities in place is being collected and their evolutions analysed. On top of that, the underlying SAw process should be as generic as to be instantiated in an additional scenario with the less effort needed. In this sense, the following describes a particular use case that, even if not fully implemented due to limitations in the technologies in hand, showcases a perfect situation in which a Situational Awareness process is part of the solution: Mobile Sensing.

The usage of sensors implemented in mobile VEs such as smartphones, cars, bicycles or buses is being known lately as Mobile Sensing. In brief, the idea is to collect as much as possible environmental measurements of a place within certain boundaries for later processing and usage. Taking advantage of the city scenarios available in COSMOS, we can collect an amount of environmental parameters of several Madrid streets captured by the buses that traverse the city every day. With that information, we can understand the status of the city instantly, but also predict future evolution taking advantage of analytics techniques applied on the historical data available.

5.4 Use-case Description

In connection with the Use Case about a person with special needs, there is a group of people that endures heart diseases. Environmental monitoring is critical for this group because under certain conditions they are prone to suffer atrial fibrillation[1], which may lead to more problematic cardiac arrhythmias. Michalkiewicz et al [5] have suggested that meteorological factors such as temperature, relative humidity, and atmospheric pressure are implicated in the occurrence of atrial fibrillation in 87% of patients. Other studies reported independent associations of atmospheric temperature and pressure, increasing relative air moisture, and wind speed and direction with the occurrence of ventricular tachyarrhythmia [6].

[1] http://en.wikipedia.org/wiki/Atrial_fibrillation

With the aid of a SAw process, an application would trigger alarms noticing a conjunction of environmental parameters that may potentially lead to increase notably the risk of a heart attack, both with instant sensor values and also with their predicted evolution. For instance, given a person with special needs that goes every day to work by bus, a mobile application may be able to trigger an alarm in advance in case a predicted condition is detected at the arrival bus stop. Even if the ability to capture all the necessary environmental factors is not completely available with the actual fleet of buses, the purpose of our work is to showcase how a SAw process is being taken into consideration in order to support future studies on this matter.

All in all, the actual use case dealing with the testing phase of our prototype is represented in the following Figure 21. In this scenario there is a person with special needs that is travelling from one place of the city to another one using a previously established bus route. A route comprises one or more buses, bus lines and bus stops, which can be understood as checkpoints. At a certain time of a day the person with special needs must be found in those checkpoints, which would mean that the trip is going well. While the EMT system is able to provide information about the bus fleet and the special person trip, it is COSMOS duty to collect, process and understand that context in order to derive alarms to a caregiver in case an anomaly is detected, such as a taking a wrong bus, arriving late due to a traffic jam in the city, or detecting a high level of pollution in the destination, to name a few ones.

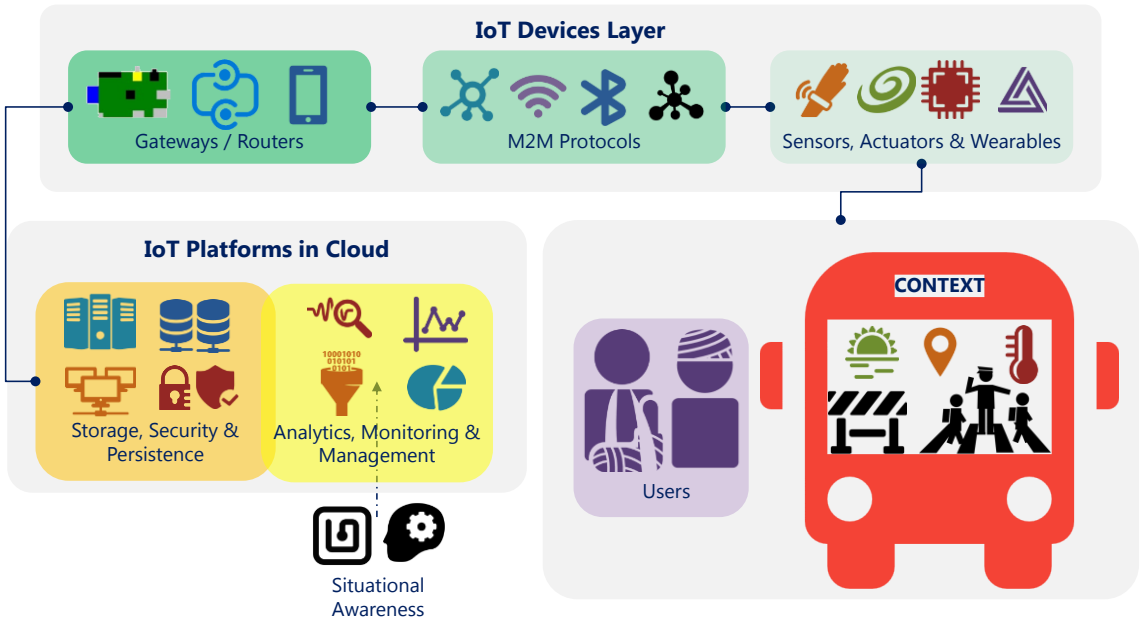


Figure 21: Assisted Mobility Use Case

Not only the information collected –the data sources— that is involved in this use case comes from the EMT bus system, but also other city services are included into the scenario in order to have a better understanding of the pulse of the city.

5.4.1. Implementation

The SAw process under consideration has been already presented in deliverable D6.1.2 [1], so basically the idea is to implement the 3 SAw steps, which can been done with the aid of Node-RED. A top-down methodology has been followed, resulting in a very generic, simple *Flow* composed by a bunch of *nodes* as presented in Figure 22.

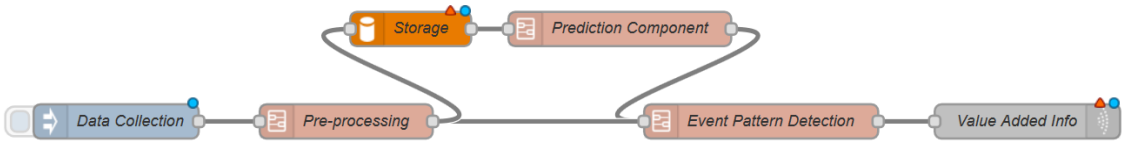


Figure 22: Generic SAw process Flow

If we go inside the 'Data Collection' *Subflow* (Figure 23, left side), we may find a number of *input nodes* –what highlights that heterogeneous interfaces are taken into consideration– connected to a Message Adapter function, which is doing nothing but transforming the message's format into the one needed by the rest of the chain: JSON formatted messages.

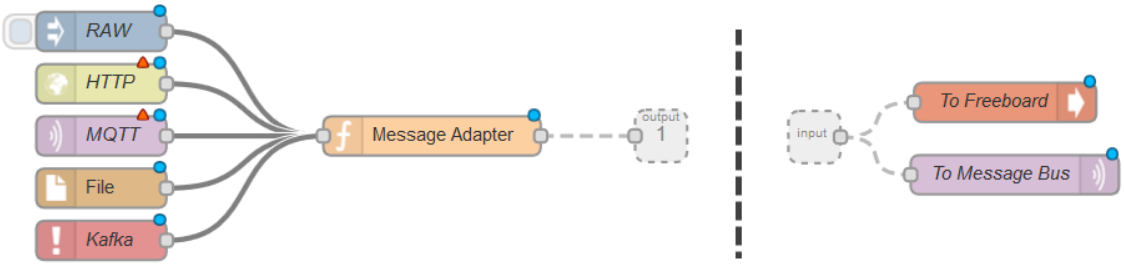


Figure 23: Data Collection Subflow

In an IoT ecosystem we usually deal with different protocols and communication interfaces, thus provisioning this kind of *Subflow* is commonplace.

With respect to the intermediate *nodes*, they encapsulate the required functionalities as explain in their respective descriptions; its usage can be understood as a series of connected *black-boxes* with the necessary input and output interfaces to do the job. Firstly, the *Pre-processing* node is in charge of filtering those environmental parameters that fold inside a specific area based on its geoposition information. In addition, a grouping rule can be set up so as to aggregate those relevant parameters within a specific time window. The output of this node is then sent both to the Storage and the *Event Pattern Detection*. The latter is in charge of analysing the coming events and detecting certain situation based on rules and thresholds. In fact, the same rules may be applied to Instant Data and Predicted Data, it is only needed to specify in the outputted *ComplexEvents* which alarms are triggered from one data or the other.

Finally, the latest *node* in the chain (*Value Added Info*) is used to redirect the output data from the SAw process to the desired destination, either it is a Message Bus or a representation in a Web panel as Freeboard (Figure 23, right side).

5.4.2. Service Wrappers

A service wrapper is a computer application that wraps arbitrary functionalities enabling them to run in the background and start them automatically at boot time. In our demo we are using PM2 –which stands for Project Manager 2— as the application that wraps Node-RED, and thereby the Flows. More information about this tool can be found in deliverable D3.2.2 [7].

Attending to the scenario under consideration, several environmental parameters can be captured from a number of city services, such as said mobile sensing buses, local weather channels, newly OpenData portals or even personal human data captured from user's own wearable devices. As an example, Figure 24 shows a *service wrapper* of the traffic status in Madrid City implemented with a flow that connects to an HTTP service, splits up an XML response in various JSON elements and transmits them all to the COSMOS Message Bus for later storage. Worth saying that the flow below can be instantiated in a completely different machine than the SAw process flow, leveraging to the underlying Message Bus the data transmission in a really easy manner.

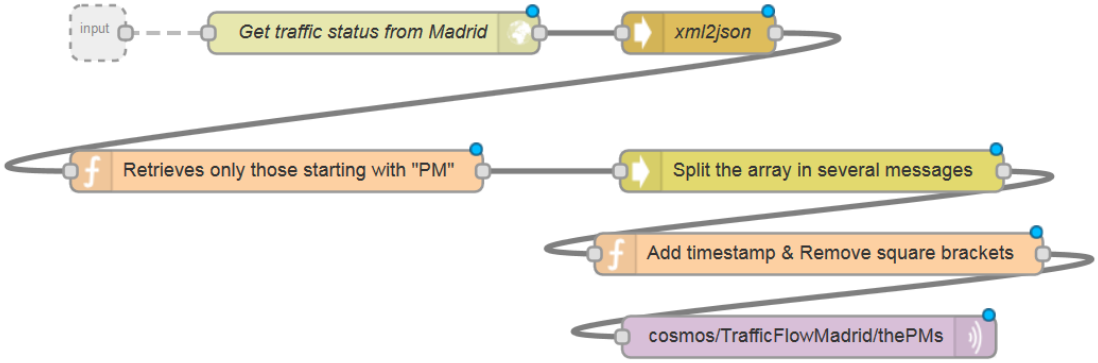


Figure 24: Traffic state service to COSMOS Message Bus

We are providing and additional *service wrapper* that really facilitates the required steps to acquire the position of the buses in the city. Using the OpenData portal of the EMT we can make use of the GetArriveStop method that serves our purposes. With the aid of the following flow an application developer can easily retrieve this info, in this case presented as HTTP service endpoint in https://flows.iot-cosmos.eu:{port}/ep/busInfo?idStop={value}.

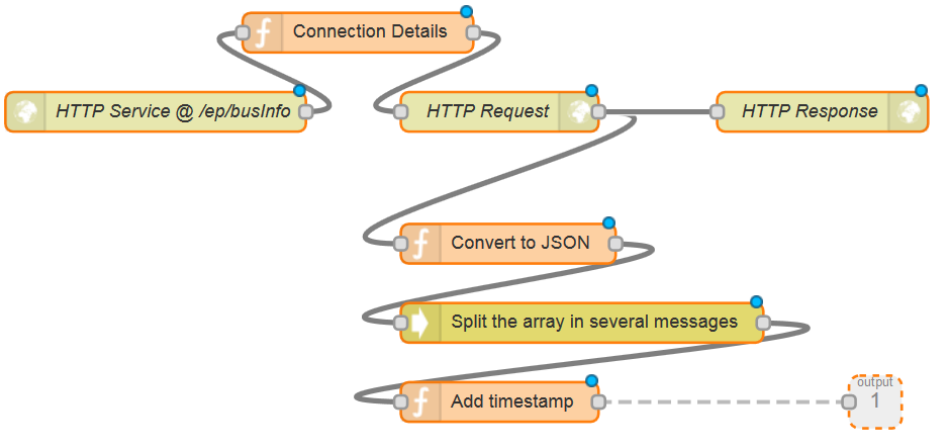


Figure 25: EMT Opendata to Message Bus

Additional data collection processes may require slightly different implementations, but in essence they are likely to look similar to the ones presented above, even if programmed in a different program language (Java, C, Python …) rather than a Flow Based Programming method (Node-RED).

5.5 Conclusion

The current shape of the SAw prototype is made up of an appropriate selection and interaction of several, specific functional components that behave together in order to provide the necessary "real word knowledge" to a use case that lacks it. The scenario in which a person with special needs travels through the city by using EMT buses is a clear example of the suitability of our solution. While the EMT system is able to report the presence or not of an entity –a person in this case— inside a bus, the understanding of this situation and its implications relies on COSMOS. In turn, the SAw functional component ensures that the information is being detected, recognized, classified and projected.

6 Experience Sharing Functional Component

6.1 Architecture

The first use of the Experience Sharing component is the acquisition of Knowledge in the form of Cases. Beginning with the analysis of the Request Handling, there exists the possibility of using the outcome of Social characteristic extraction into refining the way requests are handled. Based on Social criteria of Knowledge Exchange ranking of a VE, the request handler of an Experience Sharing request, may decide on certain occasions (perhaps connected with high resource utilisation), on whether to actively handle the requests or drop them.

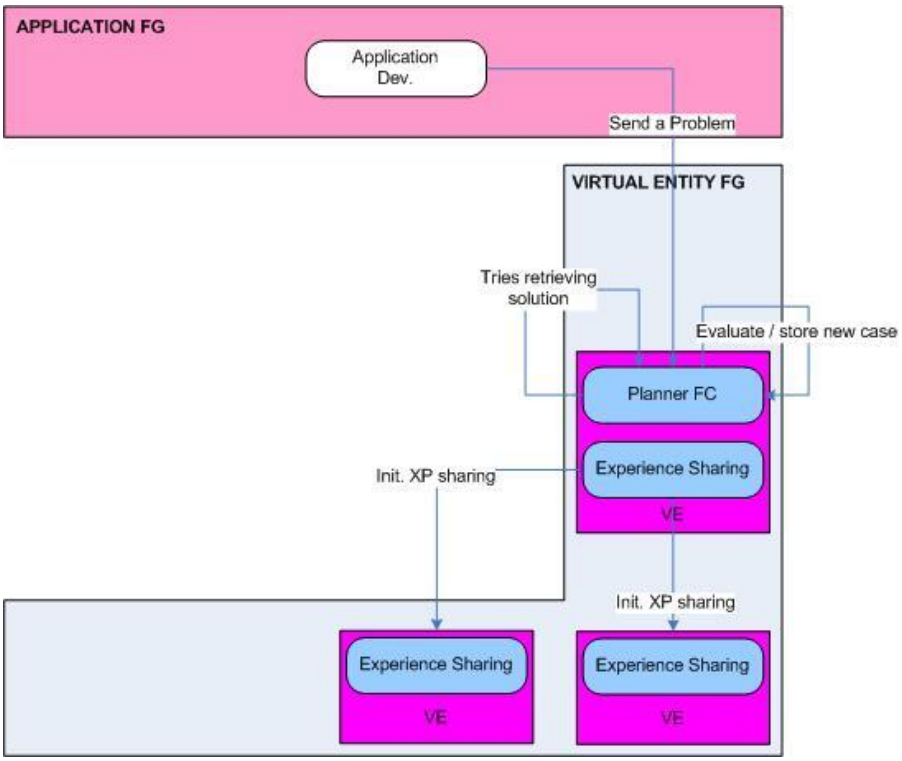


Figure 26: Interaction diagram for Experience Sharing (Case seeking)

Proactive Experience is connected with the capability of a VE to detect a change in its condition which may affect neighbouring or similar VEs. Therefore the VE must start “Sharing” this new condition detected locally to a number of VEs, whose description, or profiles, or types of relationships with the originator VE, provide indication that they may be affected adversely.

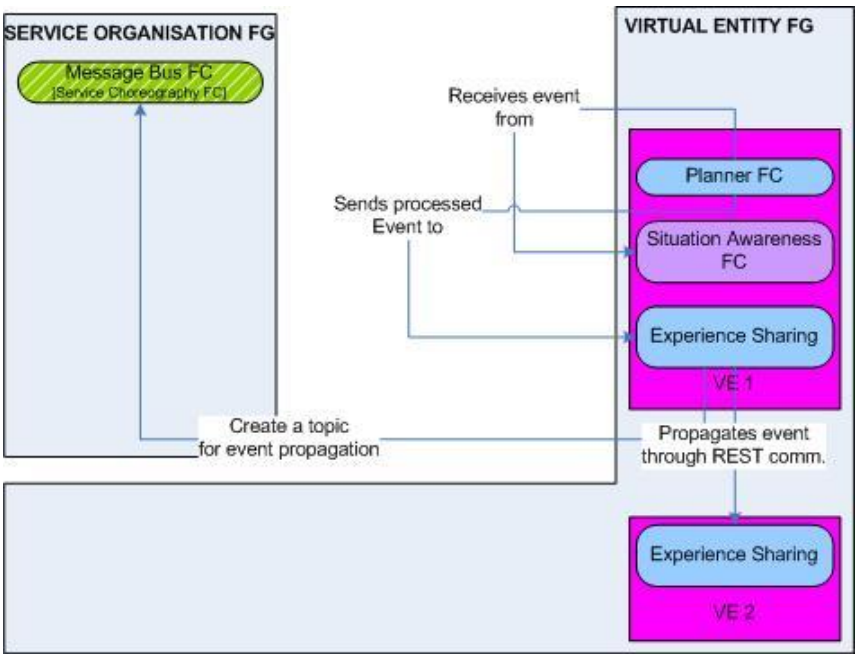


Figure 27: Interaction diagram for Experience Sharing (Proactive)

6.2 Interfaces

The interface for the Experience Sharing component remains stable within the context of the newer implementation [1] as far as triggered Experience Sharing is concerned. The steps in this case are once again the initiation of the XP Sharing by the Planner on a per Application basis. The interface method itself is agnostic meaning that any description of the structure of the Cases to be retrieved is possible. Therefore the remote endpoint servicing the request for Experience calls upon the Planner provided APIs also in an agnostic manner.

The newest addition to the actual implementation is the overlaying of privacy and authorisation handling capabilities so as to safeguard the data being shared. Additionally the code will, after the initial security provisions, maintain checks on total load of computation, in a way suitable for the safeguarding of the limited resources of most VE implementations in an embedded, IoT-enabled ecosystem. Finally the requests have been provided with dedicated IDs so that in conjunction with Year 1's ttl, requests are handled more efficiently.

The interface for the Proactive Experience Sharing will be a separate API, with a differentiated request handling endpoint, running as a parallel Service to the normal Experience Sharing. The interface function is specifically formed, so as to take advantage of the pre-specified structure of an incoming Complex Event, which will be recognised as a set of expected actions by CBR utilising functions running on the Planner. The reasoning is to transmit said Events in relative VEs in a sense of alert activity, providing both the actual Event and the actions to be performed, in a worst case scenario of zero pre-stored Knowledge.

6.3 Scenario Synopsis

The Scenarios considered for implementation are as follows. First is the implementation of a Heating Schedule Management system based on Decentralized Knowledge Diffusion (Experience Sharing in the reactive sense). In this scenario the Users of Flats will input parameters for the retrieval of a Heating Schedule matching their needs. As such, Experience

Sharing will be used to retrieve the relative Solutions, along with the improvements made on the mechanism as described in D6.1.2 [1].

The second scenario will involve the use of the SAw component as the place of origination of a Situation which will have to be Proactively Shared among peers, either through REST or through the Message Bus component.

6.4 Use-case description

The Use case being used to showcase the capabilities of the Experience Sharing FC is the Camden housing area, along with two specific implementations carefully chosen for their ability to demonstrate functionality.

6.4.1. Implementation

6.4.1.1. Heating Scheduling for Carbon Reduction Scenario

The proposed scenario takes into account that the End-User desires an increased amount of cost efficiency, without having to be manually acting in order to provide feedback or actuation to a heating schedule of his/her flat.

For the purposes of the scenario, each flat possesses a management and monitoring tablet which can act as the Gateway to the entire network. The VE code will be located on this tablet, as well as all COSMOS applications the End User may choose to install. The Heating Scheduling application will provide a Graphical User Interface for ease of access with a minimal of complexity and required options.

The End User is only to be engaged during the early phase of the scenario actions. The first step is to plan a program, stating the desired temperature value for his/her flat, for specific time intervals of the planning period. Additionally the End User must input his/her desired budget.

Given the possibility that the VE itself may not possess suitable Knowledge (Experience), it will initiate its own Experience Sharing mechanism, which targets suitable remote VEs the flat VE has knowledge of. These VEs will, in turn, search their own Case Bases for a suitable Solution and return their answers to the original VE. At this point the application will evaluate the monetary requirements of the returned Solution and actuate the Schedule or modify the input if the End User's budget is overshot.

6.4.1.2. Proactive Experience Sharing for Fire in Flat detected Event

The proposed scenario takes into account that in many cases the VE will have to act with no User input in order to achieve its objectives of continuous resource management by utilising some form of self-correcting features.

In this Scenario the VE is monitoring in real time the readings being provided by the REST Service exposing its internal temperature. The monitoring process is based on the SAw component which uses its µCEP in a way of detecting persistent abnormal values which can be used to indicate a fire in the dwelling. What is classified as abnormal is given as a rule setting to the µCEP by the Machine Learning component which has performed ML techniques on the historical temperature readings of the flat VE.

If an actual Event is detected the publisher will summon the Planner in order to actuate on the event and after that, the Planner will call on the XP Sharing component in order to propagate the event in any VEs that may also be affected. The propagation will lead to the remote VEs,

also receiving the Event, and handling it in similar fashions, so as to overcome the unexpected conditions presenting themselves.

6.5 Sequence and use case charts

6.5.1. Sequence chart for normal Experience Sharing

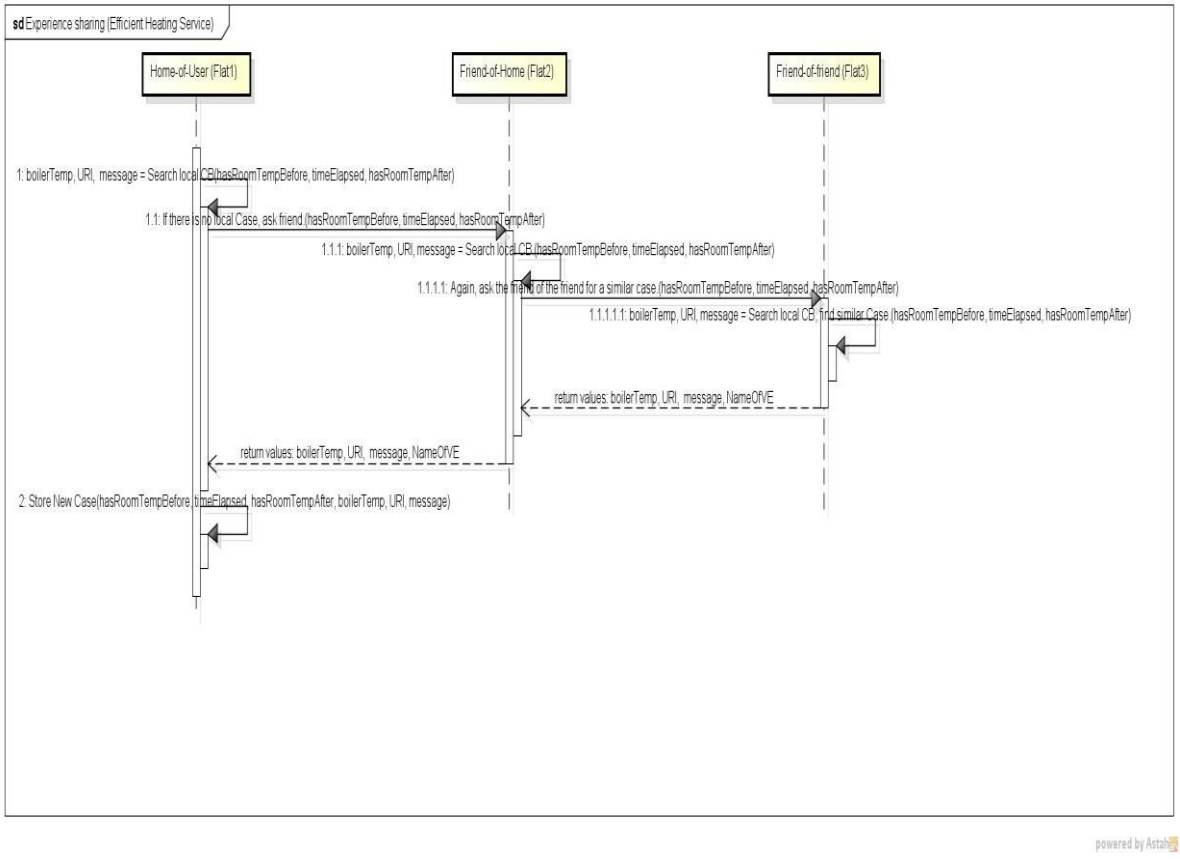


Figure 28: Sequence chart for normal Experience Sharing

6.5.2. Sequence chart for Proactive Experience Sharing

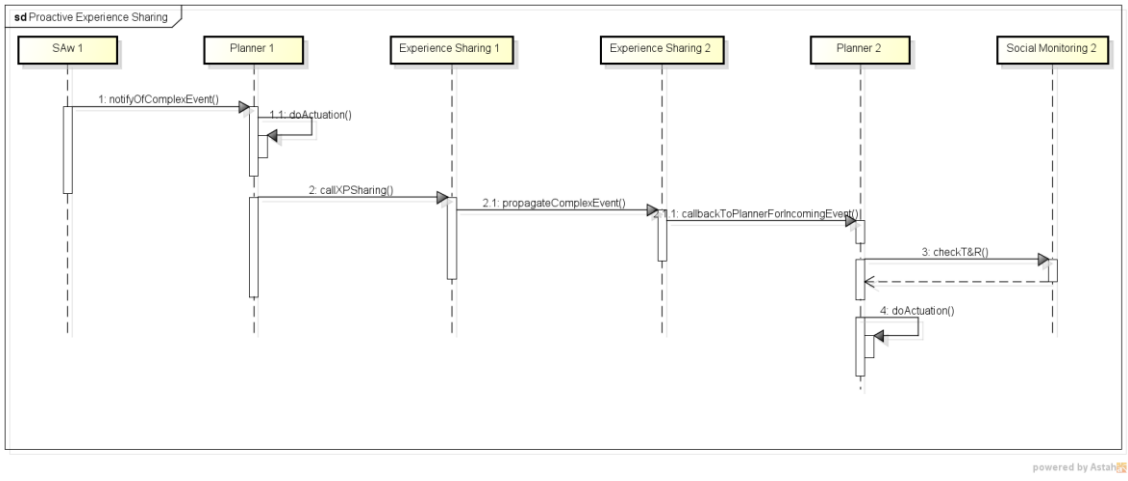


Figure 29: Steps taken in Proactive Experience Sharing

6.5.3. Use case charts

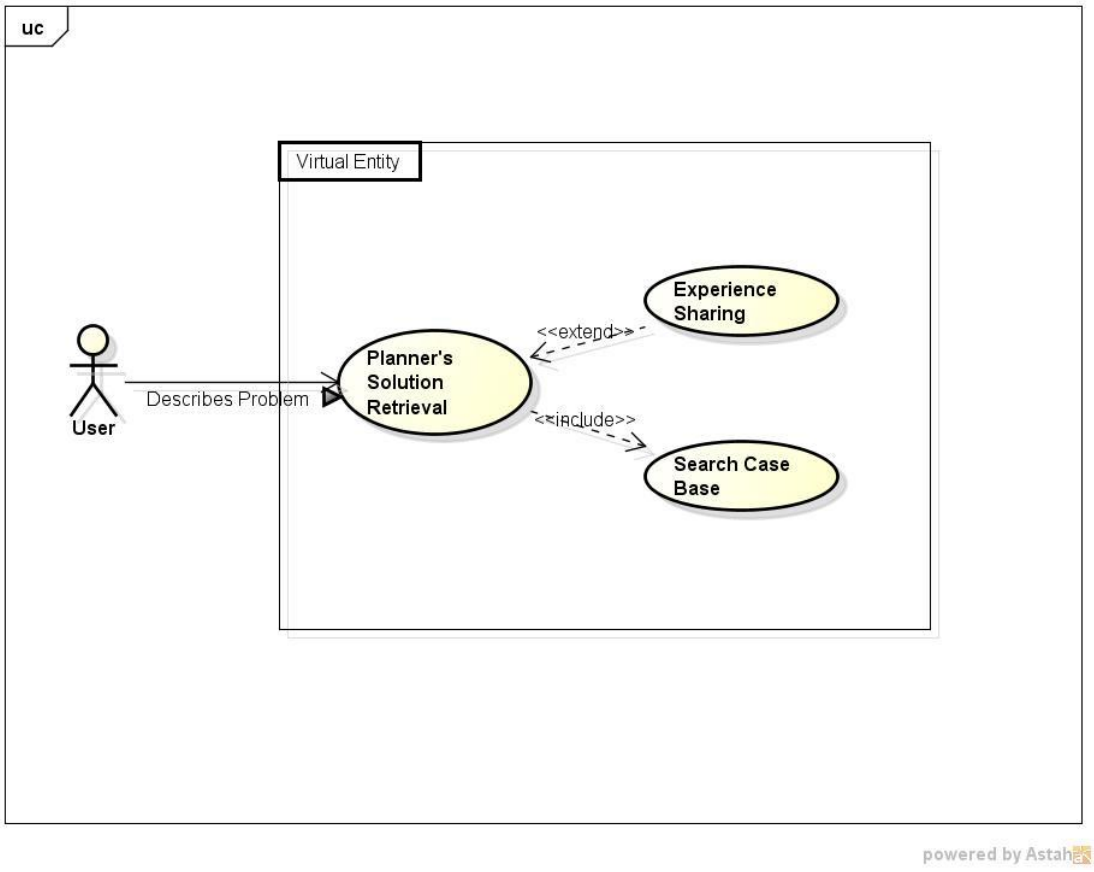


Figure 30: Use case chart for normal Experience Sharing

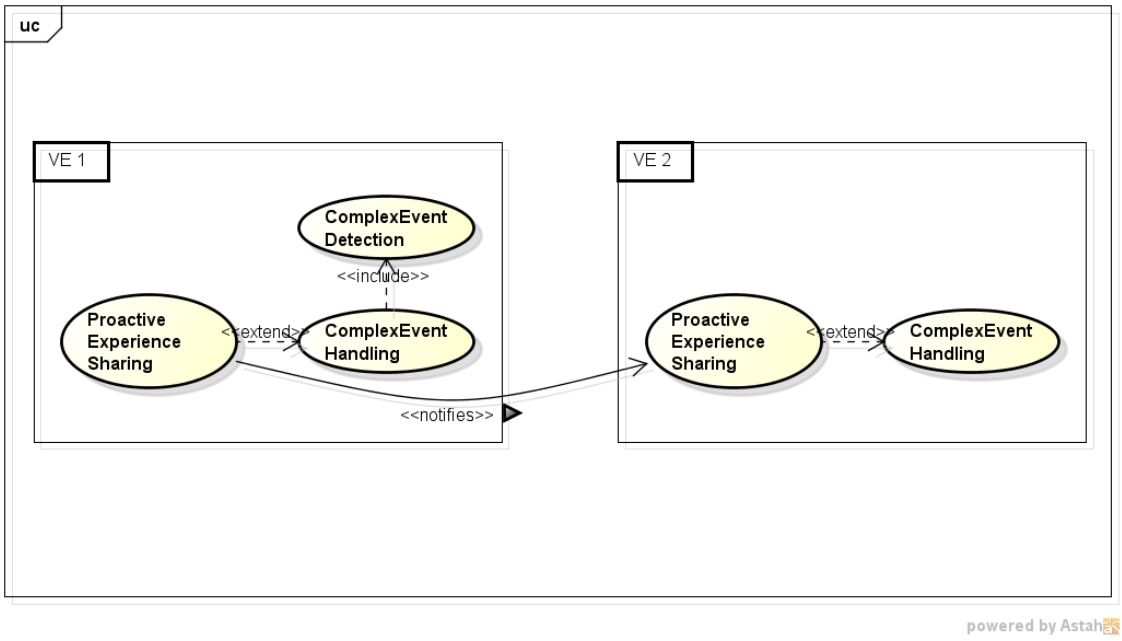


Figure 31: Use Case for Proactive Experience Sharing

6.6 Delivery and usage

6.6.1. Pre requisites for the Experience Sharing Component

The basic pre requisite is JDK 1.8 along with the Jetty Server libraries for RESTful communication between VEs. More specifically, the Jetty version used is jetty-servlet-9.1.5.v20140505.

The Experience Share class is a separate java class that is called or calls the Planner component of WP5 as needed. It contains an experienceSharePOST method for RESTful communication through HTTP POST and the overriding doPost method to receive and handle POST calls. Additionally it contains a similar method for Proactive Experience Sharing, again by implementing HTTP POST communication, targeting a different Service and also relying on information retrieved by the Planner and the Social Monitoring Functional Components. In their newer versions, both functionalities of the Experience Sharing FC, use the Java native UUID library for generating unique request IDs, as well as Java native Atomic Numbers for load measuring purposes.

Also the FC uses the external JSON Simple library for creating and diffusing JSON structured Strings through HTTP communication.

6.6.2. Code use of the FC within the product lifecycle

As the Experience Sharing FC is implementing an integral part of the VE functionalities, its presence is a prerequisite for the implementation of a physical entity virtualisation as a VE, in the COSMOS ecosystem. This in effect means, that for each deployment of COSMOS code on any physical entity being registered as a VE with the COSMOS Registry FC, the Experience Sharing code will be included in the downloaded functionality bundle. Eventual ways of deploying the code are yet to be considered and are outside the scope of this Deliverable.

7 Conclusions

In this document, we explained different components which were developed and implemented in order to fulfil the objectives of work package. Most of the functionalities offered by different components are generic in nature, so that it can be applied to different use case scenarios. Few components are still a work under progress which requires further research and will be developed over the course of project period.

8 References

[1] C. COSMOS, "COSMOS Project D 6.1.2: Reliable and Smart Network of Things," .

[2] C. COSMOS, "COSMOS Project D 6.1.1: Reliable and Smart Network of Things," .

[3] A. Zoha, A. Gluhak, M. A. Imran and S. Rajasegarar. Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey. *Sensors 12(12),* pp. 16838-16866. 2012.

[4] C. COSMOS, "COSMOS Project D 4.1.2: Information and Data Lifecycle Management," .

[5] D. Michalkiewicz, J. Chwialkowski, M. Dziuk, R. Olszewski, G. Kaminski, A. Skrobowski and M. Cholewa. The influence of weather conditions on the occurrence of paroxysmal atrial fibrillation. *Pol. Merkur Lekarski 20(117),* pp. 265-269. 2006.

[6] V. Èuliæ, N. Siliæ and D. Miriæ. Triggering of ventricular ectopic beats by emotional, physical, and meteorologic stress: Role of age, sex, medications, and chronic risk factors. *Croat. Med. J. 46(6),* pp. 894-906. 2005.

[7] C. COSMOS, "D 3.2.2: End-to-End Security and Privacy," .