COSMOS

Cultivate resilient smart Objects for Sustainable city applications

Grant Agreement Nº 609043

D6.2.1 Reliable and Smart Network of Things: Software Prototype (Initial)

WP6: Reliable and Smart Network of Things

Version: 1.0
Due Date: 30 June 2014
Delivery Date: 15 February 2015
Nature: Report
Dissemination Level: Public
Lead partner: University of Surrey
Authors: F. Carrez (editor) & A. Akbar (Unis), A. Marinakis & P. Bourelos (NTUA), J. Krempasky (ATOS)
Internal reviewers: ATOS

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	07/08/2014	F.Carrez, Adnan	UniS	Initial draft
V0.2	13/09/2014	F. Carrez & A. Akbar, A. Marinakis, J. Krempasky	UniS, NTUA, ATOS	Contributions from all the authors
V0.3	26/09/2014	F. Carrez & A. Akbar, A. Marinakis, J. Krempasky	UniS, NTUA, ATOS	Final version after incorporating reviews
RE_SUBMIT V0.4	10/02/2015	Adnan Akbar	UniS	Align chapter 2 with state of the art methods
RE_SUBMIT V1.0	12/02/2015	Juan Sancho	ATOS	Final Review

Table of Contents

Table of Figures

Acronyms

Acronym	Meaning
API	Application Programming Interface
CEP	Complex Event Processing
EM	Expectation Maximization
GMM	Gaussian Mixture Model
GPS	Global Positioning System
HMM	Hidden Markov Model
IoT	Internet of Things
IP	Internet Protocol
KNN	K Nearest Neighbor
ML	Machine Learning
MLE	Maximum Likelihood Estimation
PIR	Passive Infrared
SVM	Support Vector Machine
VE	Virtual Entity

1 Introduction

This document is intended to demonstrate the prototype development for WP6 “Reliable and Smart Network of things” based on the initial deliverable D6.1.1. 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 methods and statistical analysis methods are implemented for inferring high-level knowledge; and methods based on Complex Event Processing (CEP) engines are explored to contribute towards situational awareness models. The implemented module developed so far may not be in mature form, but it reflects the concept and line of action for future progress.

The outline of the report is as follows. Section 2 explains the use and implementation of machine learning methods and statistical analysis techniques for extracting high-level knowledge from raw IoT data. It also highlights the limitations and drawbacks of applying these methods. Section 3 is focussed on the real time situational analysis in order to cope with the dynamic scenarios in real world. Section 4 gives an oversight of how experience sharing can be exploited for autonomous solutions. Finally, a brief conclusion is given to reflect the progress of the work package so far.

2 Reliability of information

2.1 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 have the potential to extract high level knowledge from raw IoT data and highlight the major factors which limit the widespread usage of these algorithms for practical applications. The development in the following years will be built on the work of year 1 to address these issues. The background knowledge and further details about the scenario can be find in initial deliverable D6.1.1. The focus of this deliverable is on the implementation of the methods discussed in D6.1.1.

2.2 Use-case description

2.2.1. Smart Energy Management Scenario

One of the core objective 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.

2.3 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.

2.3.1. 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 1.

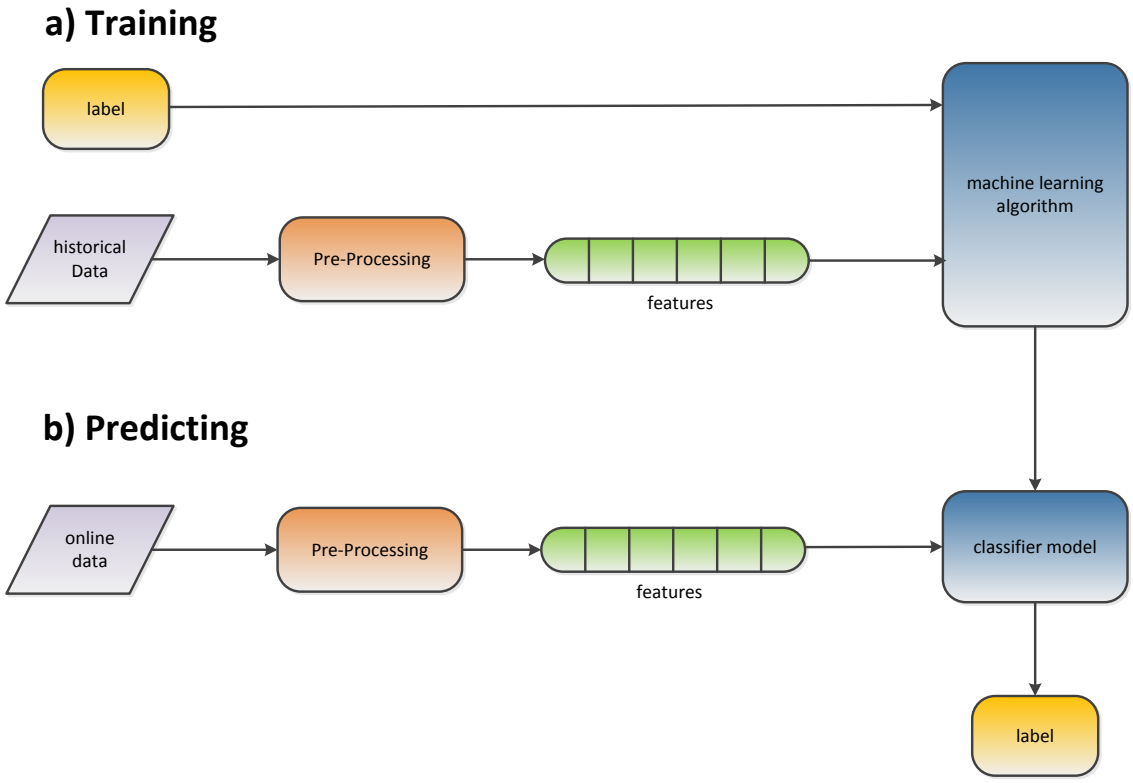

Figure 1: A general architecture of implementation of Machine Learning Model

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 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.

2.3.1.1. 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.

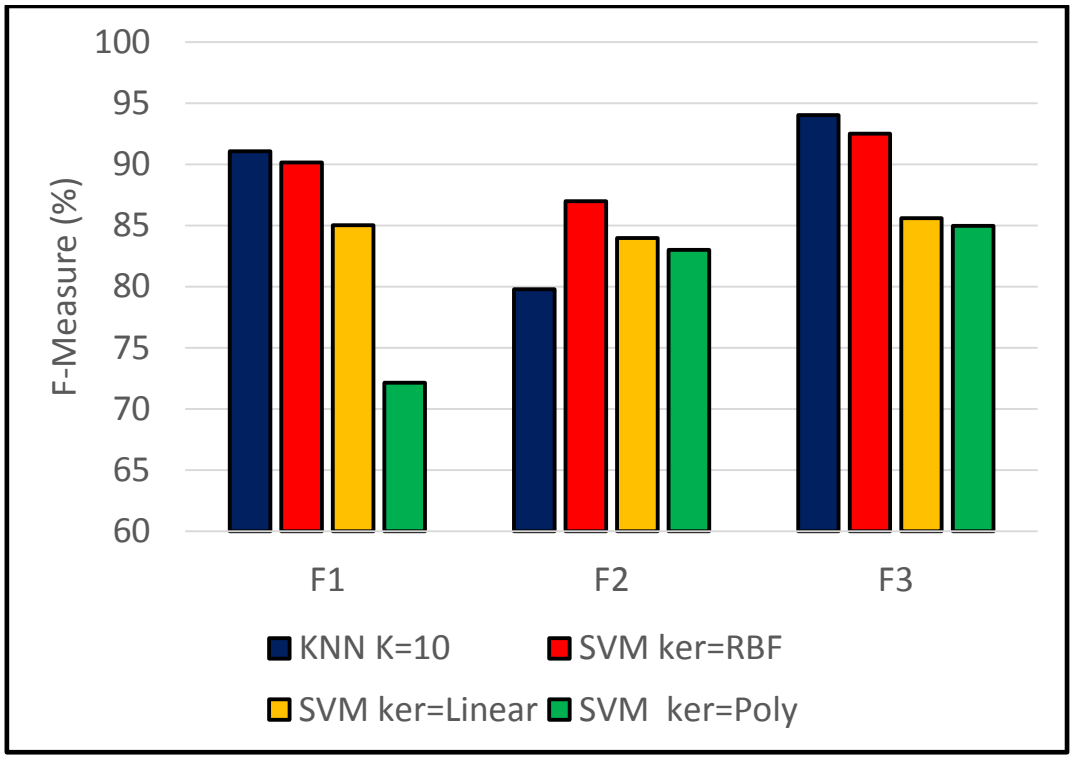

Figure 2: Performance of Classifiers for Different Feature Sets

Figure 2 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.

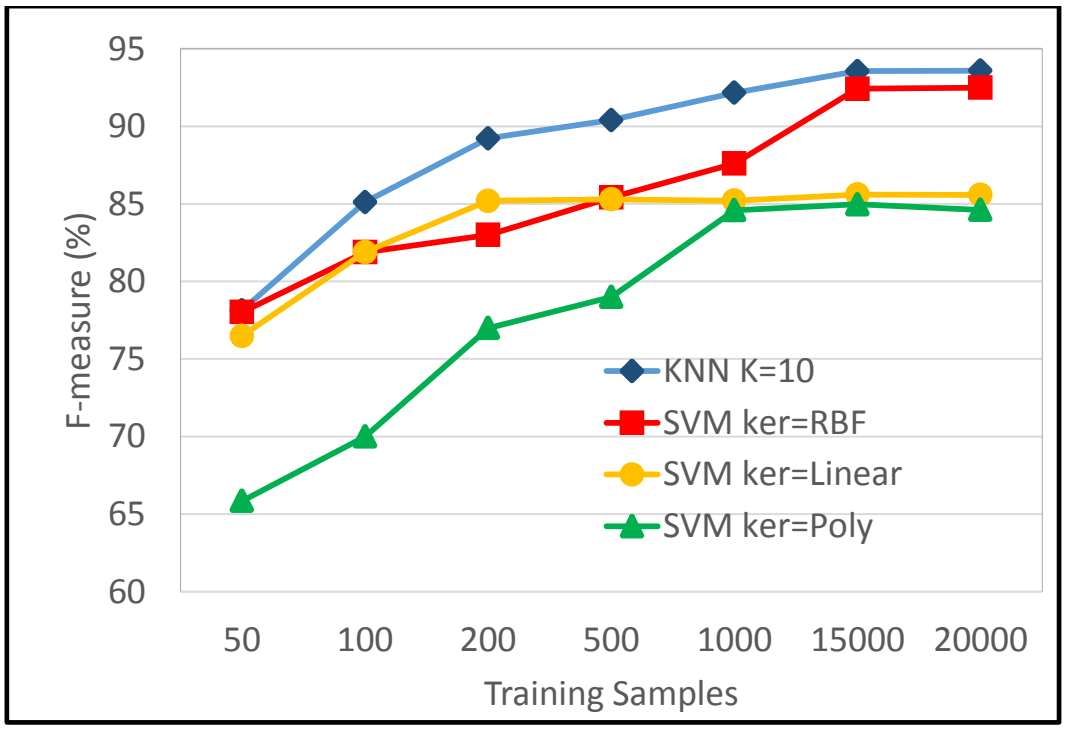

Figure 3: F-measure relation with training data set

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 3 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 4. 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.

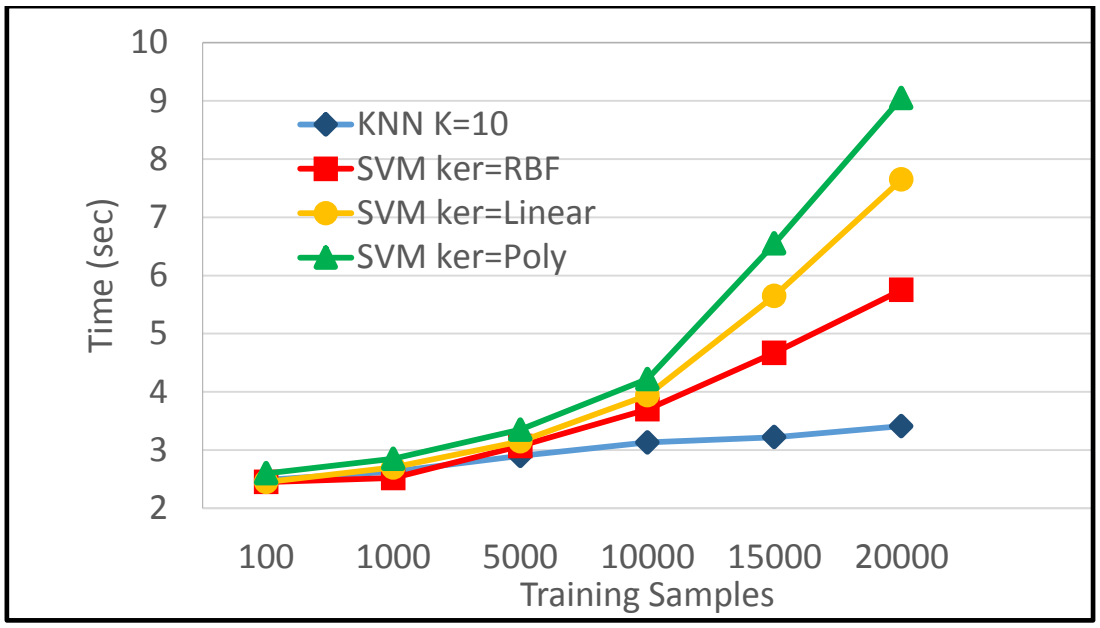

Figure 4: Time Complexity Comparison

2.3.2. 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 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 5. 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.

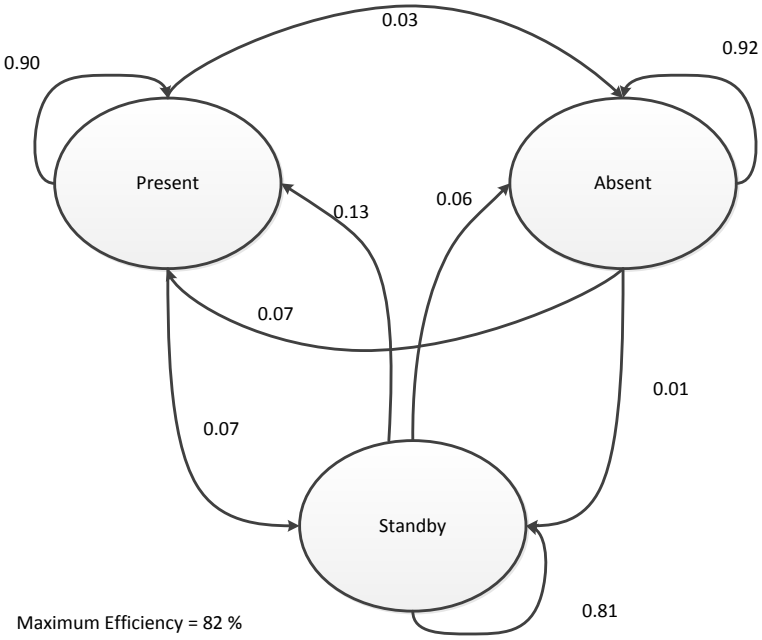

Figure 5: Transition Probabilites for HMM

2.4 Sequence chart

Figure 6 shows a sequence chart of model creation and updating of model using real time values, while Figure 7 shows the sequence of functions for calling prediction block.

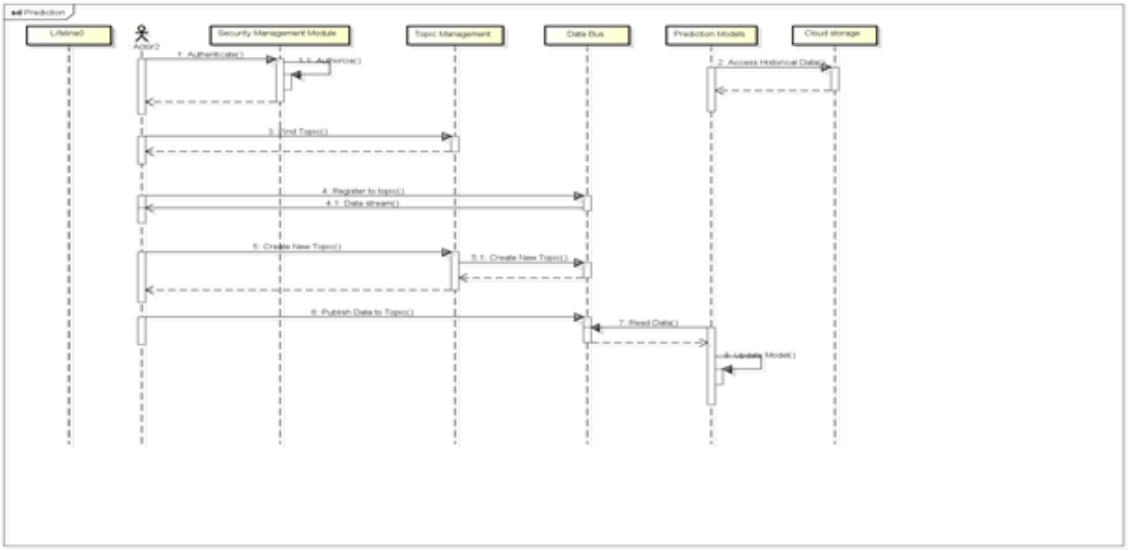

Figure 6: Updating Prediction Model

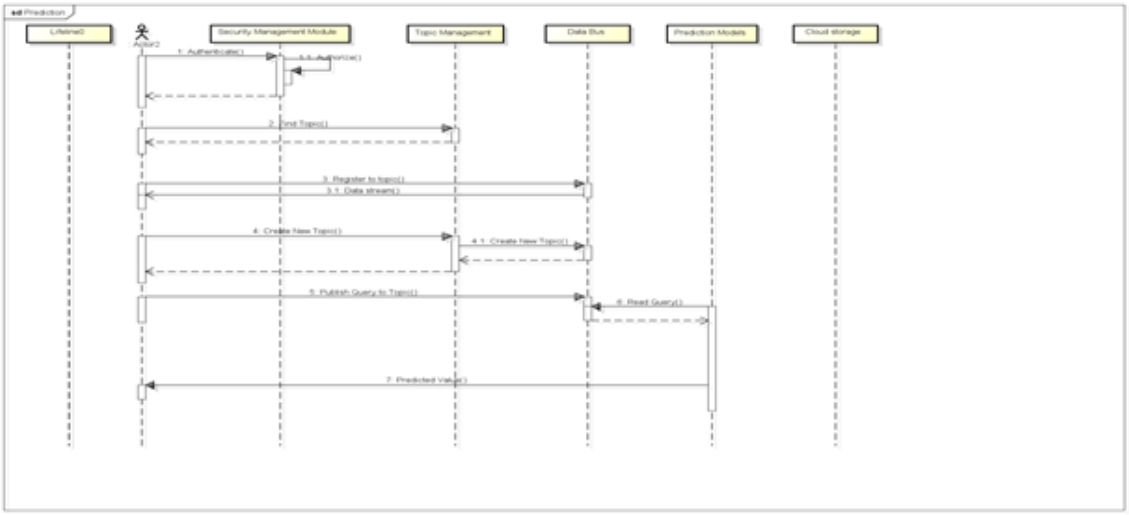

Figure 7: Calling Prediction Model

2.5 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.

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

3 Situational knowledge acquisition and analysis

3.1 Scenario Synopsis

The Situational knowledge acquisition component enables clients to dynamically change the situation monitoring, analysis and acquisition performed by CEP described in D4.1.1 deliverable. This new functionality enables a wide range of new possibilities to explore in the next years.

The demo will demonstrate run-time adaptation of situation acquisition based on external input, as well as the possibility to assign additional contextual information to the detected situation such as historical occurrences and/or actions taken in order to further support decision making processes.

3.2 Architecture

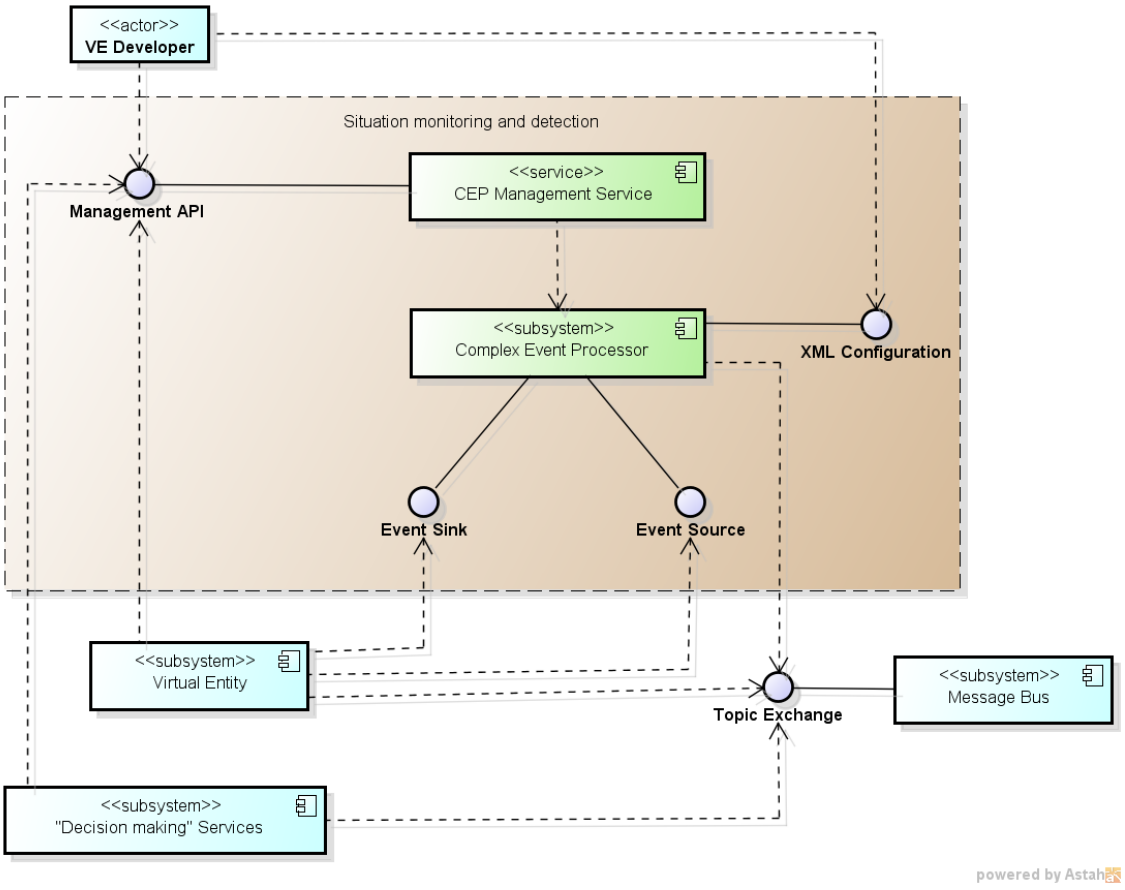

Figure 8: Situational knowledge acquisition COSMOS subsystem

Figure 8 shows a high level contextual view of situational knowledge acquisition in the COSMOS subsystem architecture. A more detailed technical description of this subsystem is described in D4.1.1 and D4.2.1 documents. The ability to dynamically change the situation analysis and detection is provided by the Management API exposed through the central CEP Management Service.

3.3 Use-case description

Situational knowledge acquisition component tracks streams of data from multiple sensors on buses such as the bus identifier, position and door state in order to identify meaningful or enhanced information like unusual delay or high density of buses per stop, as described in D6.1.1.

The demo will be based on real stream of data provided by EMT. For example, the detection of a bus delay situation can be demonstrated in the following way:

1. Define contextual model and situation analysis.
   1.1. Define rush hours.
   1.2. Define calculation of average traveling time during rush hours for subsequent stops for every bus line.
   1.3. Define detection of a bus delay situation when actual traveling time of a bus to a particular bus stop exceeds the average traveling time of the specific bus line by 20%.
   1.4. Define additional information that will be attached to the detected bus delay event to enable more sophisticated decision making such as traveling times of previous buses from the same bus line, etc.
2. Upload the defined model to the CEP component either through the static configuration file or the provided Management service.
3. Interconnect information about buses with the CEP using a message bus (Rabbit MQ).
4. Subscribe to the detected situation on a message bus.
5. Make any decision based on received information about a bus delay.
6. Update contextual model in case of bus line route change.
7. Upload updated model to the CEP component without breaking continuous evaluation of unchanged model (i.e. other bus lines).
8. Receive updated information based on the new model.
9. Make decisions based on received information according to updated evaluation.

3.4 Sequence charts

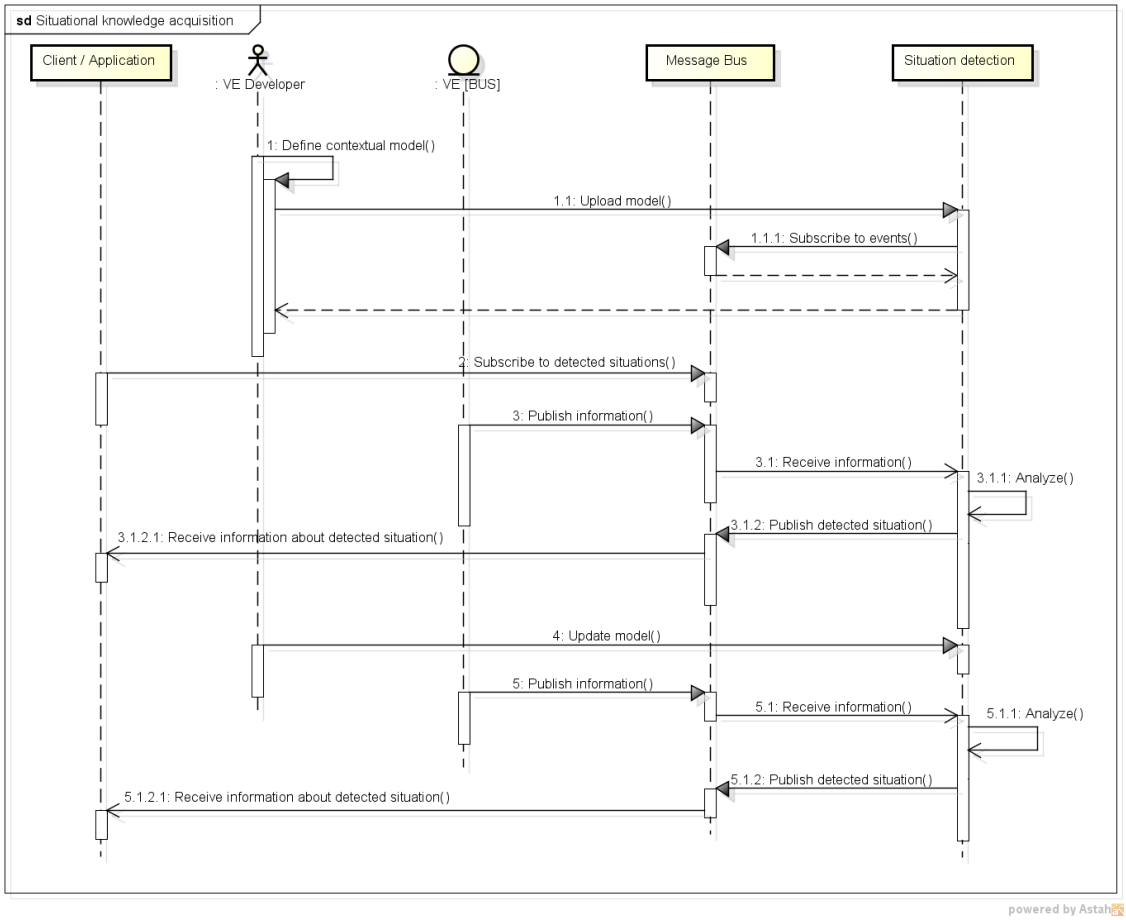

Figure 9: Situational knowledge acquisition data flow

The Figure 9 depicts a general collaboration between situational knowledge acquisition components and other involved components.

3.5 Delivery and usage

The information about packaging, prerequisites, component dependencies, deployment options as well as user and administration guide is available in D4.2.1.

The context modelling and event detection definition is described in D6.1.1.

The model definition for demo purposes will be deployed as part of the package described in D4.2.1.

4 Experience and Experience Sharing

4.1 Scenario Synopsis

The purpose of this demo is to indicate the VE's capability to act in a more autonomous way, by retrieving Experiences (cases) from its friends VEs. It also demonstrates how COSMOS can be used in order to conserve energy in a flat.

4.2 Use-case description

Autonomous Behaviour and Experience Sharing:

1. Every flat is modelled as a VE.
2. Every flat contains a thermometer (sensor) and a boiler whose temperature can be measured (sensor), as well as set (actuator).
3. During the creation of an application, COSMOS offers the developer the opportunity to define how cases relevant to his application are going to be stored, created and maintained.
4. The application is installed in the flat VE.
5. The flat VE continuously records the actions of a human user regarding the heating of the flat. For example it records that at a room temperature of 6^{o} Celsius, the user has turned the heating on for a total of 10 minutes (600 seconds) and stopped at 20^{o} using hot water at 70^{o}
6. The VE builds a case of {6, 600, 20, 70}
7. In a similar fashion, the VE builds a case base.
8. Cases are considered a type of experience. Thus the following process describes experience sharing.
9. The user alerts the house of his return and requests a certain target temperature in a certain time limit.
10. If the VE is unable to locate a similar case inside its local case base, then it contacts its friends VEs.
11. The problem, i.e. the set of attributes is then be answered by one or more VEs with similar or exact problem descriptions, allowing the house VE to build a new case.

4.3 Implementation

Efficient Heating Scenario:

The Efficient Heating scenario, as described in D5.2.1, requests the solution value of a boiler actuator. By providing the Experience Share service of the VE with the parameters of the problem, the parameters of the solution, as well as the values of the problem parameters and a ttl variable to specify depth (time-to-live), the experience sharing mechanism is initiated with targets specified by the Planner component. The expected output that the service receives is the actuator value, the URI of the recommended service, any messages contained in the solution and finally the name of the VE that provided the solution.

4.4 Architecture

Involved actors: flat VE, user

Involved functional components: Planner, Experience Sharing

4.5 Sequence and use case charts

4.5.1. Sequence chart

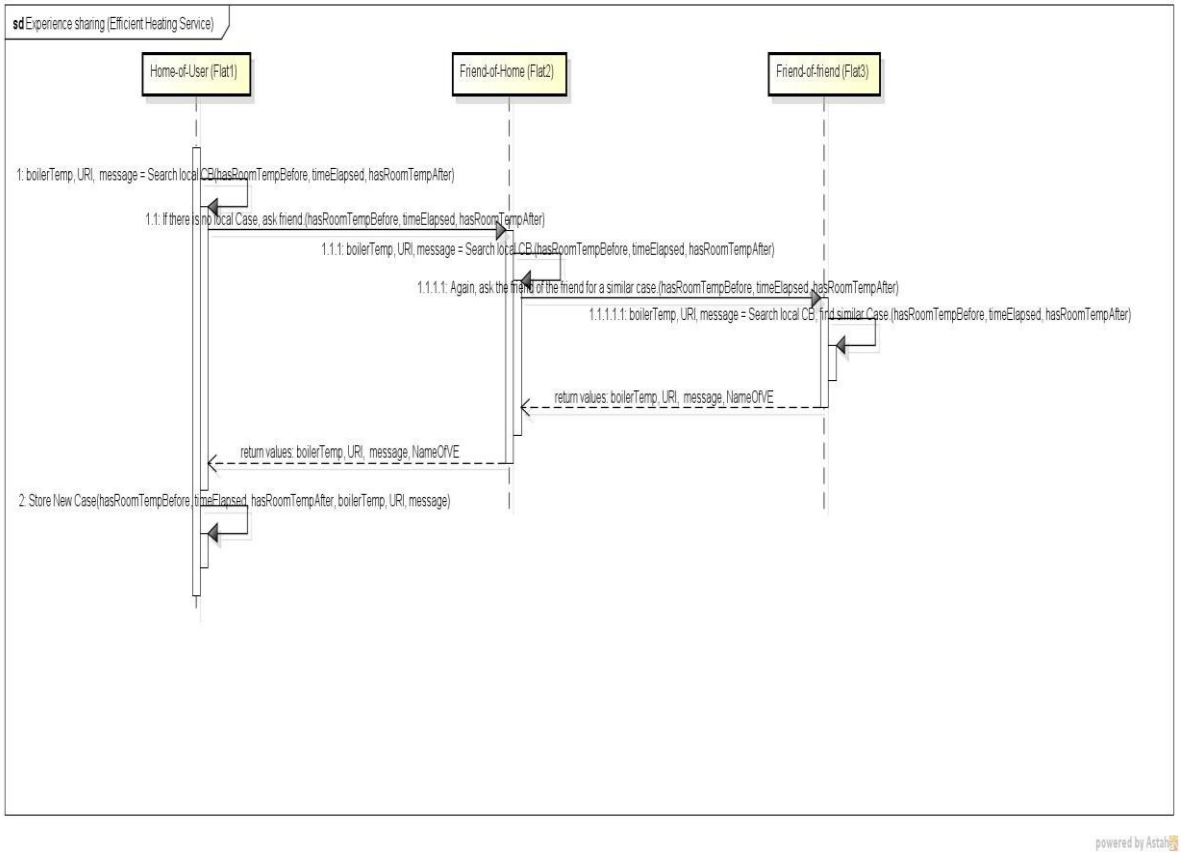

Figure 10: Sequence chart for Experience Sharing

4.5.2. Use case chart

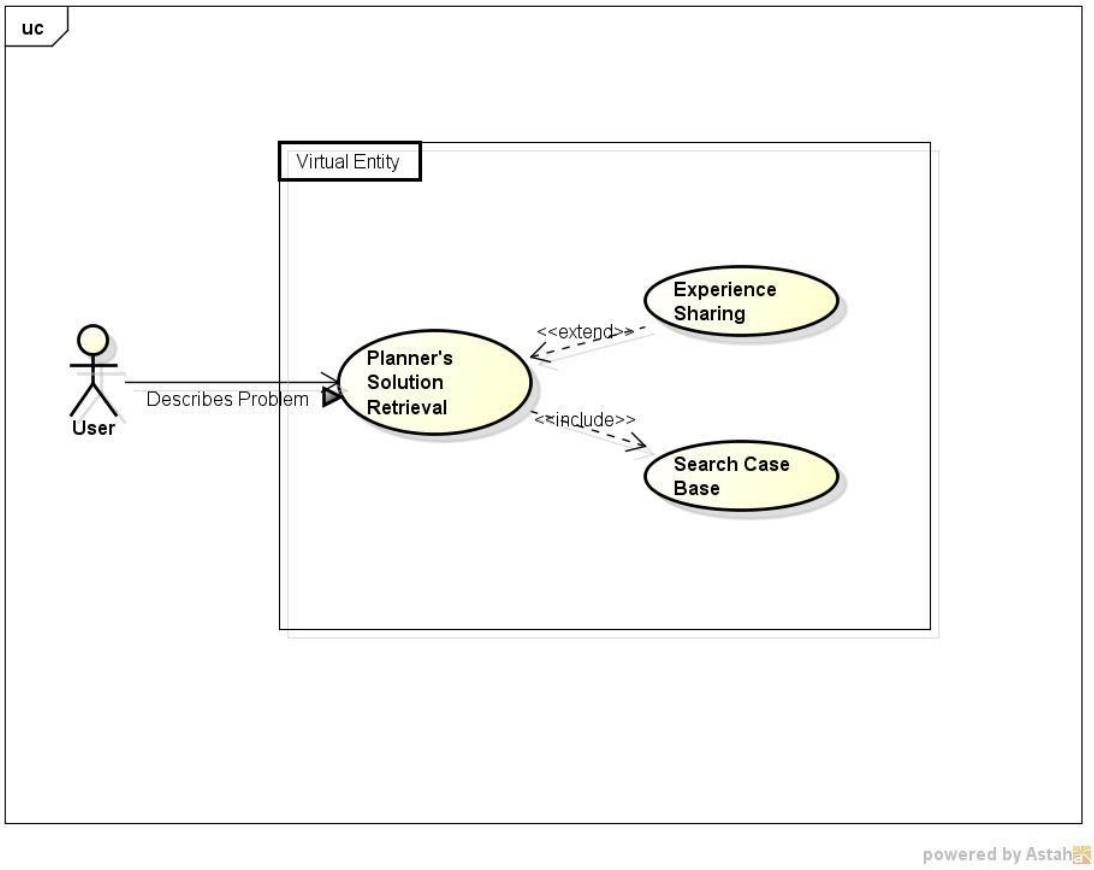

Figure 11: Use case chart for Experience Sharing

4.6 Delivery and usage

Pre requisites for the Experience Share 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 calls on 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.

5 Conclusions

In the first year, the main focus was on the various options which can be explored. In this regard, the prototype is intended to give an oversight on the possible techniques. In the following years, the focus will be shifted on applying these techniques in the context of big data for large-scale real time applications to provide intelligent and autonomous solutions.