COSMOS

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement Nº 609043

D6.1.2 Reliable and Smart Network of Things: Design and Open Specification (Updated)

WP6: Reliable and Smart Network of Things

Version: 1.0
Due Date: 30th April 2015
Delivery Date: 30th April 2015
Nature: Report
Dissemination Level: PUBLIC
Lead partner: 5 (UNIS)
Authors: F. Carrez (editor-UNIS) & A. Akbar (UNIS), P. Bourelos (ICCS/NTUA), J. Sancho (ATOS), J. Rico (ATOS), B. Târnaucă (SIEMENS)
Internal reviewers: Achilleas Marinakis (NTUA), Shelly Garion (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		F.Carrez, Adnan Akbar	UNIS	Initial version
V0.2	22/02/2015	Adnan Akbar	UNIS	Inputs on Data Analysis Section
V0.3	18/03/2015	Panagiotis Bourelos	ICCS/NTUA	Input on Experience Sharing
V0.4	1/04/2015	Adnan Akbar	UNIS	Input on Event Detection and Pre-processing
V0.5	1/04/2015	Bogdan Târnaucă	SIEMENS	SA: General structure for SA section + Functional overview and initial content
V0.6	15/04/2015	Adnan Akbar	UNIS	Organize the structure of document with components description and add new topics.
V0.7	15/04/2015	Juan Sancho	ATOS	Input in Event Detection FC
V0.8	16/04/2015	Adnan Akbar	UNIS	Input in SA for ML and CEP
V0.9	16/04/2015	Panagiotis Bourelos	ICCS/NTUA	Experience Sharing update
V0.91	16/04/2015	Juan Sancho	ATOS	SA: intro + CEP + MB
V0.92	17/04/2015	Juan Sancho	ATOS	SA: use case
V0.93	23/04/2015	Bogdan Târnaucă	SIEMENS	Restructuring of SA section. SA interfaces description.
V0.95	24/04/2015	François Carrez	UNIS	Finalization
V0.97	27/04/2015	Achilleas Marinakis	ICCS/NTUA	Internal Review
V0.98	28/04/2015	Shelly Garion	IBM	Internal Review
V0.99	29/04/2015	Juan Sancho	ATOS	Misc. revision
V1.0	29/04/2015	François Carrez	UNIS	Ready for Submission

Table of Contents

Table of Figures

Table of Acronyms

Acronym	Meaning
ANN	Artificial Neural Network
API	Application Programming Interface
CB	Case Base
CBR	Case-based Reasoning
CEP	Complex Event Processing
CRUD	Create/Read/Update/Delete
EM	Expectation Maximization
FC	Functional Component
GMM	Gaussian Mixture Model
GPS	Global Positioning System
GUI	Graphical User Interface
HMM	Hidden Markov Model
HTTP	Hyper-Text Transfer Protocol
IGR	Information Gain Ratio
IoE	Internet of Everything
IoT	Internet of Things
IP	Internet Protocol
JDK	Java Development Kit
KNN	K-Nearest Neighbour
MaL	Maximum Likelihood
MAP	Maximum A Posteriori
MAPE-K	Monitor/Analyse/Plan/Execute - Knowledge

MD	Model Developer
ML	Machine Learning
MLP	Multi-Layer Perceptron
MTBF	Mean-Time Between Failure
NILM	Non-Intrusive Load Monitoring
OWL	Ontology Web Language
PA	Predictive Analysis
PAA	Piece-wise Aggregation Approximation
PCA	Principle Component Analysis
PMML	Predictive Model Markup Language
QoS	Quality of Service
RBF	Radial Basis Function
RDD	Resilient Distributed Datasets
RDF	Resource Description Framework
SA	Situation Awareness
SAX	Symbolic Aggregation approXimation
SPARQL	SPARQL Protocol and RDF Query Language (Recursive acronym)
SQL	Simple Query Language
SSID	Service Set IDentifier
SVM	Support Vector Machine
SVR	Support Vector Regression
UC	Use-case
URL	Unified Resource Locator
VE	Virtual Entity

1 Introduction

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 methods based on machine learning and statistical analysis techniques will be explored and developed for extracting events from raw data. The focus of Year 1 was to explore state of the art methods and highlight the limitations of these methods whereas in year 2 we have addressed the limitations and extended the methods beyond the state of the art. Complex Event Processing engines will be deployed for knowledge inference in complex deployment situations based on near real-time analysis of complex events. We intend to explore Machine Learning techniques in conjunction with *Complex Event Processing* (CEP) in order to provide adaptive solutions for dynamic scenarios to optimize the performance of CEP for situation awareness. Finally different methods for experience sharing between virtual entities will be investigated so that things can be made smarter and able to make decisions using the experience of entities, which have already faced identical situations. All of the above mentioned objectives are elaborated in this document with relevant examples.

This document is intended to provide a generic approach to meet the high-level objectives which are summarized below:

1) To address the limitations of existing knowledge extraction techniques and extend it to provide solutions for large-scale IoT applications;
2) To provide adaptive methods for inferring complex events from raw data streams;
3) To provide means for Situation Awareness for IoT networks;
4) To develop Experience sharing mechanisms between virtual entities and thus making them autonomous.

What is new in this deliverable compared to previous D6.1.1 version.

- Full-restructuring of the document which is now following the functional decomposition (and not the WP task structure);
- Fully reworked section on Situation Awareness aligned with the conclusion of the State-of-the-Art document D2.2.2 [1];
- All other Functional Components come in their second iteration aligned with Year 2 objectives.

The document is organized on the basis of description of WP6 components which are shown with thick edges within the COSMOS Function View shown in Figure 1 below:

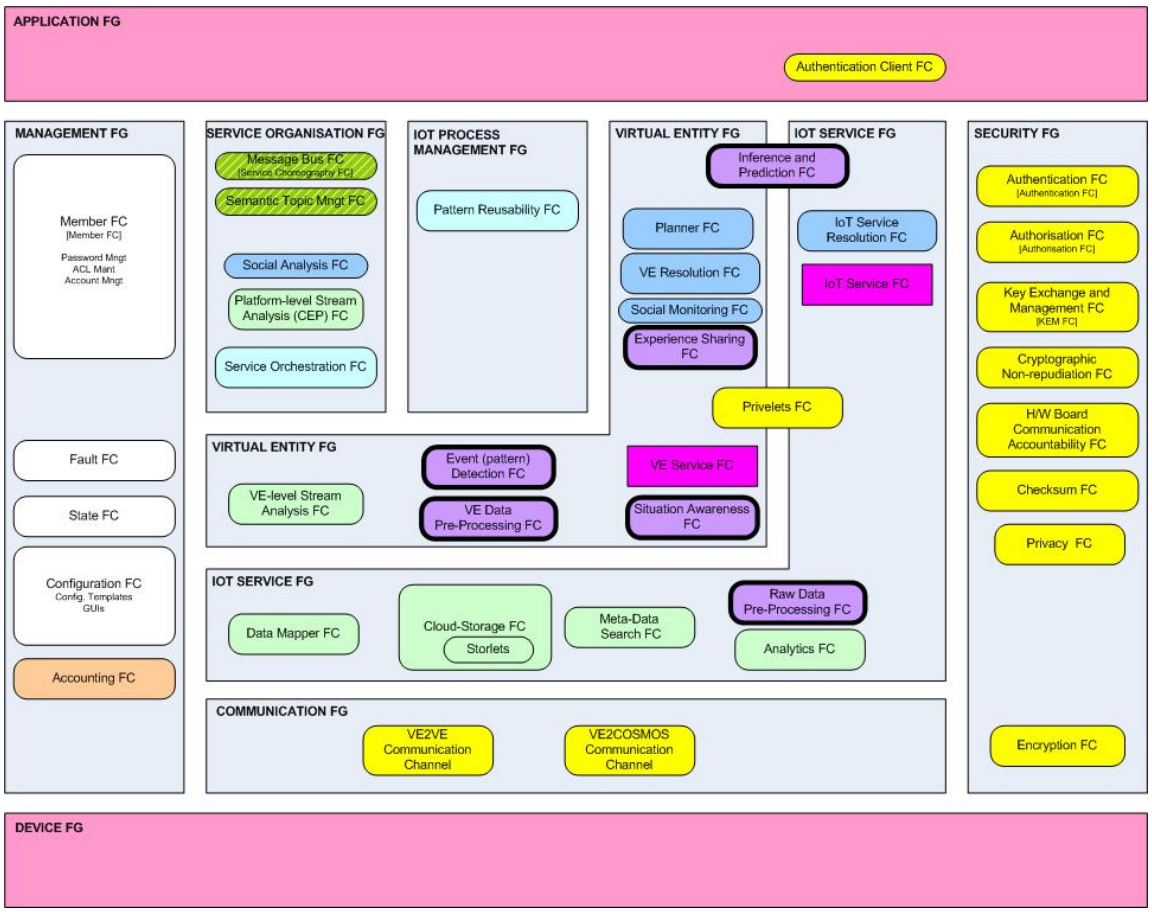


Figure 1: COSMOS Function View (w/ WP6 FC emphasis)

2 Inference/Prediction Functional Component

2.1 Introduction

In the world of *Internet of Things* (IoT), devices and sensors are deployed or used in varying conditions and different situations. Mostly they are deployed in remote places and are connected using less reliable wireless links. In order to prolong their battery life, data provided by these devices may be sporadic, less reliable and incomplete. Data itself is of no value until it is processed intelligently to extract high-level knowledge which can be used to make decisions. Data mining methods based on machine learning and statistical analysis techniques have the potential to extract knowledge from unreliable and incomplete data. Pattern recognition techniques based on data mining methods have long been used in speech and image processing applications but their use in IoT world is still in its early years.

This section explains different state of the art data mining methods which were explored for extracting high level knowledge from raw IoT data and we have demonstrated how these methods have the potential to contribute for novel applications. A high-level knowledge can be any event, some actionable knowledge or some statistical property of the data depending on the application. Data from different sensors in IoT form patterns, and different patterns represent different events. However, only certain events are interesting depending on the context of application and require further action. Pattern recognition methods based on data mining algorithms enable to extract these interesting events which have the potential to form the basis for many interesting applications. We have demonstrated in our work, how these algorithms can be applied in IoT domain for novel and innovative applications.

The performance of most of the Machine Learning models deteriorates when applied on real-time dynamic environments. One of the main factors contributing in deterioration of *Machine Learning* (ML) models is concept drift which occurs when the statistical properties of the target variable or the statistical distribution of the observation data changes over time. Secondly, the complexity of machine learning algorithms increases exponentially with the amount of data. Most of the innovative solutions found in literature based on ML and statistical analysis techniques are implemented on small data sets and are not usable for large scale IoT applications. In this regard, the aim of Year 2, leveraging on Year 1 experiments, is to focus on ML solutions which are able to cope with the dynamic nature of underlying IoT data in real time and also able to deal with the largeness of IoT data for smart city applications.

2.1.1. Machine Learning

ML represents any algorithm or process which learns from the data. The availability of data from large number of diverse devices which represent real-time events has motivated the researchers to explore more intelligent solutions using ML algorithms. Whether it is in the context of a health department, supply chain management system, transportation or the environment, methods based on ML enables to provide more intelligent and autonomous solutions by analysing and providing further insight. Different data mining algorithms have been used in very diverse contexts, detecting traffic congestion [2], predicting the energy demand of the buildings [3] and predicting the behaviour of customers in an online shop [4] are few examples of using data mining algorithms in context of IoT. There are many ML algorithms found in the literature, but in a broader context they fall into following three main categories.

- Classification Analysis;
- Clustering Analysis;
- Regression Analysis.

The focus of this section will be on the classification analysis as the focus of year 1 development was based on these techniques. We intend to explore using other possibilities in year 2.

An efficient implementation of any ML algorithm involves different steps ranging from filtering, interpolation, and aggregation on one end to optimization methods on the other. A proper understanding of a problem is mandatory to apply the right choice of steps. The different steps involved in machine learning algorithms with possible options are shown in Figure 2. A brief introduction to few of these techniques is given in this section.

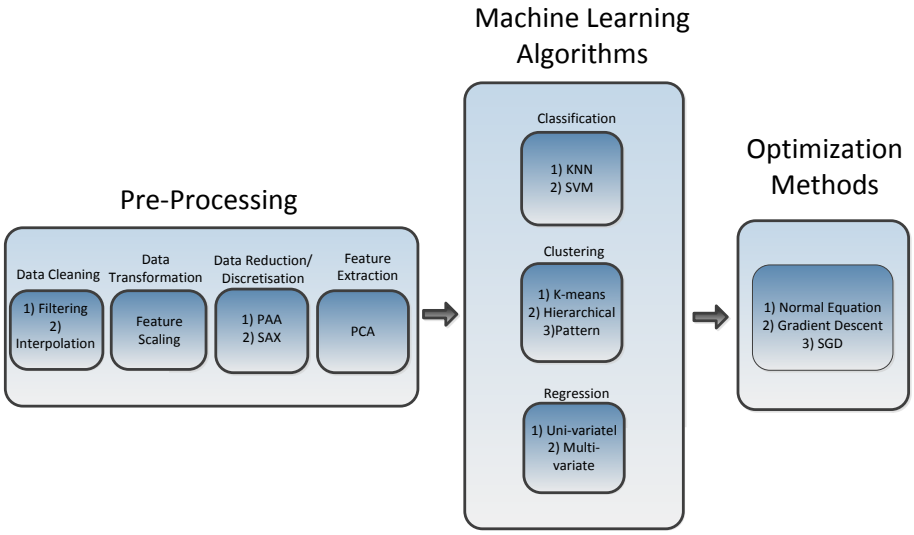


Figure 2: Efficient Implementation of Machine Learning

2.1.2. Classification Analysis

Classification is a supervised machine learning technique which requires labeled data in learning phase used widely for pattern recognition and categorization of new observations. The classification of emails as SPAM or NOT-SPAM (a.k.a. spam filters) represents a well-known example of classification analysis [5]. The server learns from the user's behavior, whenever the user marks (label) emails as spam. It looks for the key words in the marked spam email and if it detects the repetition of those keywords in new mails, it will mark them as spam. There are many variants of classification tools found in the literature. The authors in [6] included several classification algorithms in top 10 data mining algorithms including *Support Vector Machine* (SVM), *K-Nearest Neighbour* (KNN), decision trees and Naive Bayes'. Each classifier has certain advantages and disadvantages and the selection of any particular classifier depends on the data characteristics.

SVM is one of the most widely used classification algorithm. The two main advantages which give SVM an edge on others are:

1. Its ability to generate nonlinear decision boundaries using kernel methods;
2. It gives a large margin boundary classifier.

SVM requires a good knowledge and understanding about how they work for efficient implementation. SVM works by mapping the training data into a high dimensional feature space, and then separates the classes of data with a hyper plane, and maximizes the distance which is called the margin. It is possible to maximize the margin in feature space by using kernels into the method, which result into non-linear decision boundaries in the original input space. The decisions about pre-processing of the data, choice of a kernel, and setting parameters of SVM and kernel influence the performance of SVM greatly and incorrect choices may severely reduce the performance of SVM as discussed in [7]. It is also possible to use mixture of different kernel functions for optimized performance and one such example is given in [8], where authors used SVM for image classification with kernel function which is mixture of *Radial Basis Function* (RBF) and polynomial kernel. The SVM algorithm requires extensive time in training but once the model is trained, it makes prediction on new data very fast.

Another efficient and simple classification algorithm is KNN, which is one of the simplest and instance-based learning techniques 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 predicts the class with the highest votes. KNN memorizes labeled data in the training set and then compares the new data features to them. 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 [9]. KNN is also called lazy algorithm as it takes zero effort for training. However, it requires full effort for predicting for new data points [9].

2.1.3. Regression Analysis

Regression analysis is one of the most widely used ML tool; it is used to define a relation between variables so that the values can be predicted in future using the defined relation. In general, the dependent variable is called a *target variable* and the set of independent variables which form the input are called *predicted variables*. In order to elaborate a bit more, let us consider one simple and yet practical example of regression analysis e.g. to identify a relation between energy use and weather in order to predict the energy demand according to weather changes. In this example, energy usage is a dependent variable (target variable) and weather parameters such as outside temperature, wind and humidity are independent variables (predicted variables) that act as the input. It is a general observation that if the weather is cold, the energy consumption will be higher due to usage of heating systems as compared to mild or warm weather. If the dependent variable is relying on only one factor, it is called uni-variate model whereas like in our example if it depends on more than one variable (temperature, wind and humidity) it is called multi-variate model. The energy demand can vary linearly along the weather parameters; or there can be a non-linear relation between them. Non-linear regression models are generally more accurate but they come with the increased cost of complexity. Over fitting is also a common problem in non-linear models.

If one of the dependent variable is time, then the regression analysis can also be called *time series analysis*. Traditionally, statistical methods like ARMA and ARIMA were used for time series regression but recently the trend has shifted towards more sophisticated ML models such as *Support Vector Regression* (SVR) and *Artificial Neural Networks* (ANN) because of their robustness and ability to provide more accurate solutions.

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 many examples found in the literature where SVR has been successfully applied for prediction such as predicting stock market feeds [10], electricity demands [11] and traffic flow [12]. The prediction of travel-time is an essential aspect of intelligent transportation systems. Different algorithms from statistical theory and recently from ML domain have been applied for predicting the travel time as the random nature of traffic makes it difficult to predict. In [13], the authors applied SVR for the same problem and demonstrate that it outperforms other statistical methods performance-wise. Recently, the hybrid regression models have also been in use in IoT and one such example is given in [14] where a combination of several regression algorithms was made to predict the short-term sensor readings.

2.1.4. Statistical Inference:

Classification Analysis methods described in previous section is an example of supervised ML model where parameters of the model are estimated from training data which consist of pairs of input and annotations of the output. The performance of supervised models is often quite good but the labeling of data poses an extra task. Data is often labeled manually, and although the process of labeling may be simple but it limits the amount of examples that can be classified using manual methods. In addition, as the annotation task becomes more complicated such as for labeling data for traffic congestion, annotations become far more challenging as well. In this regards, we explored a statistical inference approach based on *Hidden Markov Models* (HMM) for predicting an event in scenarios where labeled or annotated data is not available. For such scenarios, the training of a model for predicting output without the availability of annotated output data seemed counter intuitive, but it is possible using Expectation Maximization algorithms.

2.1.4.1. Hidden Markov Model

HMM is an extension of Markov model in which the states of the system are not directly observable as contrast to simple Markov models like Markov chain where state is visible to the observer. Markov model is an example of statistical model, which assumes Markov property to define and model the system. In simple words, Markov property states that the future state of the system only depends on the present state and does not depend on the past values. Each state has some probability distribution related to the output, and therefore sequence of outputs generated gives indirectly information about the sequence of states as shown in Figure 3 where at every sampling time instant, we have an observation which is related to hidden state. HMM is widely used for temporal pattern recognition.

There are many applications in literature where Markov model and HMM have been applied to find temporal relations within data. One such example is given in [15], in which authors showed the use of HMM to find the possible transition of events (states) depending on their temporal relation. The authors demonstrated their approach on the energy data which they gathered using the test bed installed in their research centre. They first applied clustering to find the possible hidden concepts or states and name it as weekday or weekend; and then applied HMM on top of it in order to show the possible state sequence and probability of transition from one state to another. For example, if there were consecutive four weekdays registered, then there will be very high probability that the next day will be a weekend. HMM can also be used as a classification tool for inferring events. For example, the authors in [16] applied it for accident detection. It is a quite generic tool and used extensively for classification

purpose in different fields as evident by their use in [17]. The main difference from the classification analysis techniques discussed in previous section is that HMM is a parametric technique which is based on the statistical properties of the underlying data with respect to time. HMM can also be used for predicting the optimal and most probable sequence of states or outcomes using *Expectation Maximization* (EM) algorithm as used by authors in [18].

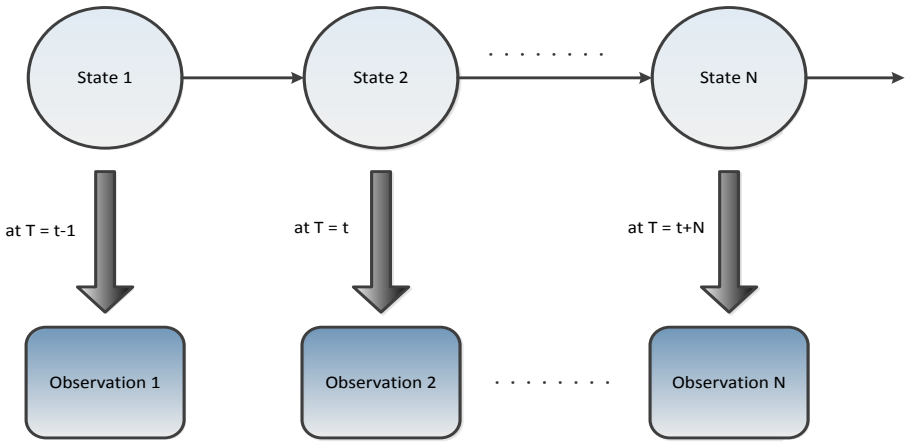


Figure 3: Hidden Markov Model

A complete description of HMM requires specification of total number of hidden states N, and the specification of three probability measures represented by $\lambda = (A, B, \pi)$, where:

- π represents the set of prior probabilities for every state;
- A represents the set of transition probabilities a_{ij} to go from state i to state j: $a_{ij} = P(q_t = S_j | q_{t-1} = S_i)$ and collected in matrix A;
- B represents emission probabilities which is the probability of observation in a particular state: $b_{i,k} = P(x_n = v_k | q_n = S_i)$, the probability to observe v_k if the current state is $q_n = S_i$. The number $b_{i,k}$ is collected in a matrix B.

There are three basic problems of HMM in order to apply for real world applications. They are:

- **Problem 1**: Given the observation sequence $O = O_1, O_2, \dots O_t$ and a model $\lambda = (A, B, \pi)$, how to efficiently compute the probability of observation sequence given the model $P(O|\lambda)$?
- **Problem 2:** Given the observation sequence $O = O_1, O_2, \dots O_t$ and a model $\lambda = (A, B, \pi)$, how do we choose a corresponding state sequence $Q = q_1, q_2, \dots q_t$ which best explains the state sequence?
- **Problem 3:** For the given observation sequence $O = O_1, O_2, \dots O_t$, how to find a model parameters which maximize $P(O|\lambda)$?

The problem 3 is of particular interest for unsupervised learning in which only observations are given. In this case, the hidden states are estimated using Expectation Maximization (EM) algorithms based on Maximum Likelihood theory.

2.1.5. Large-Scale IoT Data

Big data is no more a fantasy and has now become a necessity for many applications. Across different applications in IoT, we have seen more data results into more effective algorithms and more meaningful abstraction of knowledge. Most of the applications in IoT are of large scale: smart buildings, intelligent transportation system and real-time management of supply

chain logistics are just few examples dealing with vast repositories of data. Therefore, any practical application must fulfil the requirement of scaling up to datasets of interest.

The ever-increasing data has resulted into paradigm shift for finding new optimized techniques in contrast to traditional methods. Distributed computing is one research area which is explored to carry complex analytics in parallel on multiple machines and form the basis of MapReduce [19]. Whereas, cloud storage becomes the main candidate for providing scalable solutions for managing large data sets. In our work, we have explored the use of Apache Spark [20] for providing parallel, scalable and distributed platform for carrying analytics on large IoT data and integrated it with Openstack Swift Object storage and used storlets [21] for carrying analytics close to the storage.

In the following section, we briefly describe the requirements which drive our design decisions for using Spark and storlets.

2.1.5.1. Requirements

The conventional approaches of dealing with large data problems are proving to be insufficient and require a distinct approach. In this section, we briefly summarized the requirements for dealing with large data problems:

- **Time latency:** Time latency is an important requirement for large data processing solutions. The proposed method should be able to finish processing data in a given time frame. The value of insights gained from the data may lose its importance if the underlying process takes longer time. The statistical properties of the data underlying may change over longer period of time analysis.
- **Scaling out:** For large data problems, scaling out (using large number of low-end servers) is the preferred choice as compared to scaling up (using small number of high-end servers). The latter approach of using machines with very high processing capabilities is not an optimum choice as the cost of such machines does not scale linearly i.e. a machine with twice as memory and processing capabilities will be of significantly more than the double of machine).
- **Reliability:** A large data programming model/framework should be able to deal with the failures of hardware underlying. In big data applications, failures of hardware underlying are inevitable due to large number of machines lying. Consider a simple example; a large cluster is made of 1000 high reliable machines with *Mean-Time Between Failures* (MTBF) of 1000 days. Even with these reliable servers, the system will experience roughly one failure/day.
- **Analytics close to data:** It represents one of the fast gaining requirements for large scale data processing applications. The ability to gather and store data is developing overwhelmingly as compared to the ability to process it. Big data is no more a fantasy and has now become a necessity for many applications. Across different applications in IoT, we have seen more data results into more effective algorithms and more meaningful abstraction of knowledge. Most of the applications in IoT are of large scale; smart buildings, intelligent transportation system and real time management of supply chain logistics are just few examples dealing with vast repositories of data. Therefore, any practical application must fulfil the requirement of scaling up to datasets of interest.

2.1.5.2. Distributed and Scalable Machine Learning using Apache Spark

In many ML algorithms, the optimization algorithm (such as gradient descent) involves accessing same dataset iteratively to optimize certain parameters for finding optimum decision boundary or an appropriate function. MapReduce [22] enables to represent each iteration as a MapReduce job, but each job must reload the data from a disk which incurs a significant amount of performance penalty.

Spark is designed to address the limitations of MapReduce in order to overcome the bottleneck issues caused by disk I/O retrieval and to improve the performance for iterative algorithms. Spark provides the ability to run computations in the memory which enables it to provide much faster computation times for complex and iterative applications as compared to systems based on traditional MapReduce, such as Hadoop etc. Spark is designed to be fast and general purpose at the same time. It is designed for different data analysis applications which include iterative algorithms such as ML applications, batch applications and real-time streaming applications. By providing a generic optimized framework, it provides a generic data analysis platform which makes it suitable for wide IoT applications.

Spark [20] achieved fast distributed computing with the help of *Resilient Distributed Dataset* (RDD), which represents a read only collection of objects which can be split into many partitions. The different partitions of RDD can be computed on the different machines across the cluster. The programs can cache an RDD in memory which can be reused in multiple parallel operations like in MapReduce, which is the main reason for the fast performance of Spark. RDDs are quite generic and can contain objects of type Scala, Python and Java. It can run on Hadoop Yarn manager and can read data from Hadoop Distributed File System thus making it extremely portable for running on different systems.

2.1.5.3. Analytics Close to data using Storlets

Although object storage can store objects, manage them, protect them in a highly scalable way, it does not itself dramatically increase the rate at which we can extract value from objects. A new research prototype developed by IBM called storlets is developed on the concept of converting the software-defined object store into a smart storage platform. It aims at greatly increasing the value of the data that can be retrieved from the storage and the speed at which we can access what we need. Storlets allow the object storage to move the computation close to the data, as oppose to traditional approach of moving the data to the server by the system to run computation.

The impact of storlets is of considerable importance. Stored data can be processed locally, and no longer needs to be transferred over the network to a remote computer, processed and then put back onto the storage server, all of which incurs both network transfer latencies and ok then extra costs. Storlets aim at reducing costs, providing enhanced flexibility and improving security at the same time by turning the object store into a platform and extending the functionality of the object store using software. Saving bandwidth by avoiding unnecessary data transfers is not the only advantage which storlets provide; storlets provide perfect ways to introduce new services: storlets can analyze each object and extract its metadata including size, subject, format and more.

2.2 Functional Overview

This component is responsible for providing high-level knowledge from raw IoT data using different pattern recognition techniques. In this context, we have explored several supervised ML techniques including different variants of SVM and KNN and statistical techniques such as HMM which were explained earlier. In short, it provides the two following main functionalities.

1) If labelled historical data is available (raw data with labelled high-level knowledge), it provides the functionality to train the model and provides capability to deploy the model in order to predict the output for real-time data;
2) If the labelled historical data is not available (incomplete data), it exploits the temporal patterns of the data and learns using statistical properties of the data to train the model which can be used to predict the output for real-time data.

Both functionalities are provided in the context of large-scale IoT data.

2.3 Connection with other Components

The inference and prediction FC can take both raw data and VE data as input depending on the scenario and the usage required by application developer. Figure 4 shows the needed inter-component interactions steps for carrying inference/prediction on raw data. 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.

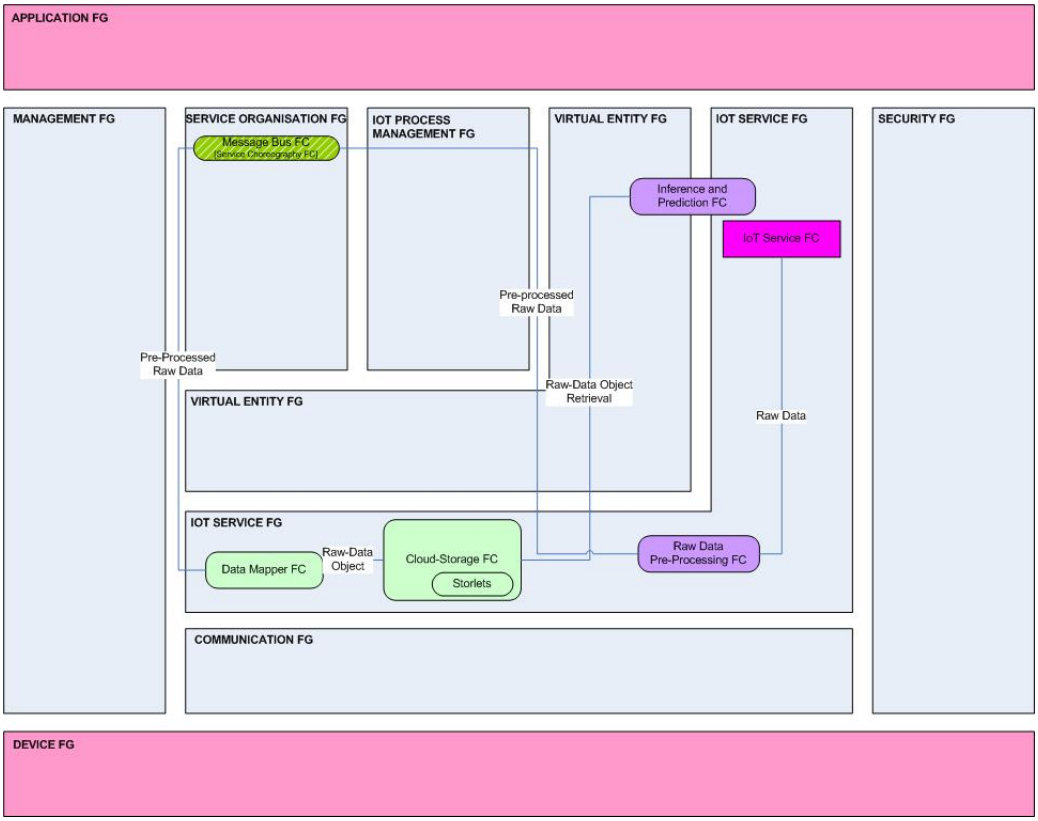


Figure 4: Inference/Prediction on raw data

Figure 5 below shows the inter-component interactions steps needed for inference and prediction on VE data. The flow of data is almost the same with the additional capability of running storlets for pre-processing on the object storage and within Spark for complex ML tasks. Pre-processing becomes an important step when running analytics on large-scale complex IoT data.

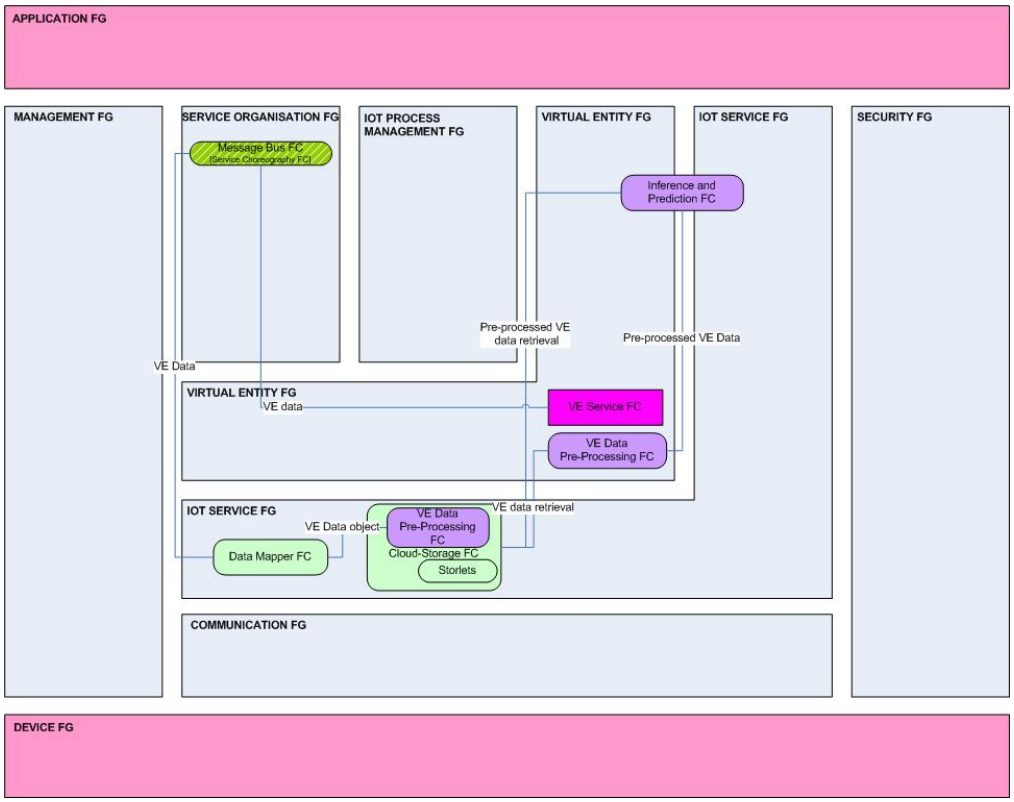


Figure 5: Inference/Prediction on VE Data

2.4 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 application developer is interested in. 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 the model retrievable for later use. Using this approach, the semantic description of VEs and IoT services or data 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:

An API will be provided to connect the application developer to Inference/Prediction block for the following purposes:

1) In order to select the particular prediction model and to define the input and output for the model;
2) To select the specific type of pre-processing required for the application

Client/VE-Inference/Prediction Block API:

An interface will be provided between the client/VE and inference/prediction block which will serve the following two purposes:

3) 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);
4) 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.

Storage-Modeling Block Interface:

All the historical data is stored in the form of objects in object storage. Machine Learning models require an access to historical data for training purpose. In this regards, modeling block should be connected to the object storage.

2.5 Use-Case Scenario Extension from Year 1

This use-case demonstrates the use of pre-processing techniques and knowledge extraction techniques at the same time. We have extended the use-case scenario from year 1 for detecting occupancy state from electricity consumption data with the use of storlets for pre-processing and providing the means for extracting knowledge from large-scale IoT data in a distributed way using apache Spark. The work done in this section is based on the idea of moving computations to the storage in order to avoid the bottleneck caused by the bandwidth limitations of the network. We have proposed a novel solution based on our initial findings which show encouraging results and are summarized in this section. It should be noted that the results and discussions presented in this section are preliminary as it is still a work under progress.

2.5.1. Background

The amount of data in IoT is increasing exponentially and the trends indicate further increase in data size. The ability to store data is developing overwhelmingly as compared to the ability to process it. The fact that the improvements in the storage have outmatched the improvements in processing data to the extent where even the ability to read could not match what we store is itself distressing. The tendency of data storage has increased from tens of megabytes to magnitude of terabytes in last couple of decades whereas the increase in latency and bandwidth performance has merely improved to the factor of tens or perhaps hundreds.

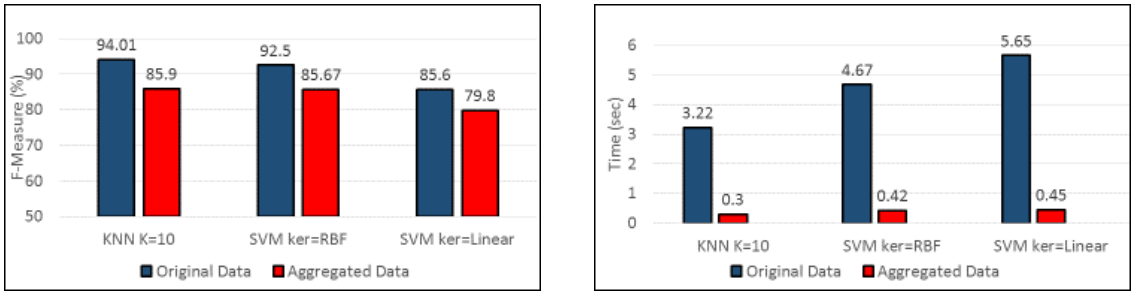
It is a common practice for most of the existing data analysis platforms to have separate processing nodes and storage nodes such as amazon platform which has EC2 for computing and S3 for storage. Many data-intensive workloads are not very much processing demanding and the transfer of data from storage nodes to the processing nodes proves to be a bottleneck in such scenarios. In our work, we have explored the possibility of moving computations to the storage in order to reduce the amount of data transferred over the network. The initial results have shown significant potential towards reduced complexity and providing scalable solutions.

2.5.2. Initial results and explanations

We have extended the same scenario of occupancy detection discussed in the previous section. The 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 which were discussed in 2.1. Figure 6 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

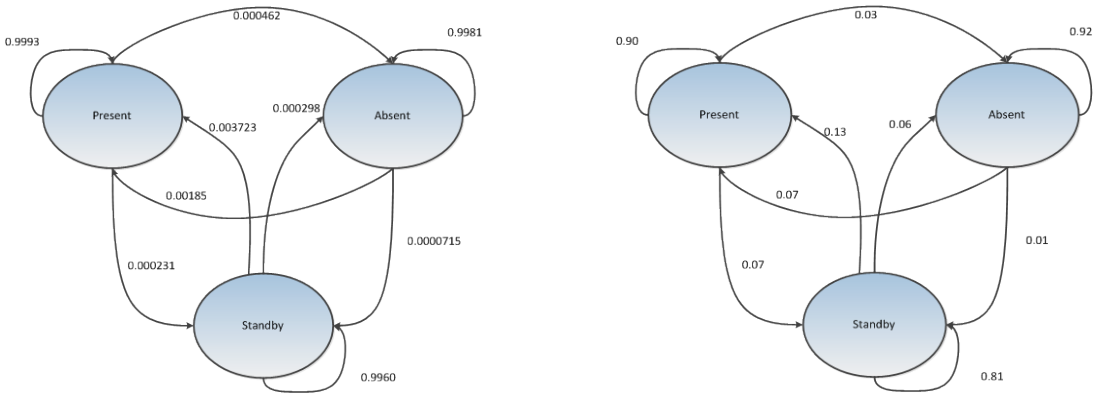
Whereas, Figure 7 shows the results for the Hidden Markov Model (HMM). Although the accuracy of algorithms has dropped a little but the gain in reducing time complexity is huge.



(a) Accuracy

(b) Time Complexity

Figure 6: Aggregated data comparison with non-aggregated data for Machine Learning algorithms



Accuracy = 82%, Ts = 10 sec

Accuracy = 70%, Ts = 10 min

Figure 7: Aggregated data comparison with non-aggregated data for HMM

One might argue that applying pre-processing tasks before the data is stored using middleware platform is 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) [23], for detecting which devices are being used and requires very high sampling rate. In such a 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. The two main advantages of our approach are:

1) The amount of data traveling over the network will be reduced to the factor of aggregation. In the example above, we applied *Piece-wise Aggregation Approximation* (PAA) with the fixed window size of 6 samples which directly resulted into the six times less data transfer over network.
2) The storlets can also be used extracting basic form of knowledge directly, thus reducing the time and data over network and increasing the web services provided by the system. For example, if we are interested in the readings at particular day and time, a simple storlet can be written which takes day and time as input and will return only the required data. Whereas, in a conventional method, system will have to load the file in memory and scan it to get the required information. For large data files, it results in latency and large amount of data transferred over network.

2.6 Conclusion

We have explored supervised machine learning models for extracting high-level knowledge from raw IoT data based on pattern recognition techniques and statistical inference methods based on Hidden Markov Model. We have integrated Apache Spark with the object storage to run the same machine learning algorithms in a scalable and distributed manner. The use of storlets for pre-processing shows encouraging results for reducing the amount of data which travels over the network and hence reducing the time complexity.

3 Pre-Processing Functional Component

3.1 Introduction

Pre-processing is an important step for applying ML algorithms in IoT due to many reasons. Most of the devices are connected with wireless links in a dynamic environment and resource constraint nature of these devices affects the communication link and their performance. The deployment of cheap and less reliable devices is a common practice in IoT for bringing the overall cost of a system down with the unfortunate resulting flaw of missing values, out of range values or impossible data contributions. The sentence "*garbage in garbage out*" fits perfectly to many ML algorithms. The amount of data is increasing exponentially in IoT and the processing of such large data with minimum time latency is an important factor which can be optimized by the use of proper pre-processing methods. Several aggregation techniques are commonly used in IoT for reducing the total amount of data travelling through the network. In this context, *Piecewise Aggregation Approximation* (PAA) and *Symbolic Aggregation approXimation* (SAX) are the most common techniques. Data collected in IoT can be a combination of many features. The number of features plays an important role in the complexity and the performance of a ML algorithm. In few scenarios, these features are correlated with each other and it is possible to reduce the features by representing the data using new uncorrelated features using statistical analysis such as *Principal Component Analysis* (PCA). A brief introduction about the most commonly used pre-processing techniques is given below.

3.1.1. Data Cleaning

In real time dynamic environment, a faulty or a missing sensor reading may occur due to bad communication channel or loss of service. The missing values can result in irregular time series or incompatible vector size as compared to other devices connected. A simple data cleaning method involves then filtering out-of-range values and filling out missing values. Missing values can be filled by the mean value of the sensor over some time window, by last recorded value or by simple interpolation methods using historical data.

3.1.2. Data Transformation

Data transformation involves transforming the data into the form that is optimum for ML process. Feature scaling is an important example which is used extensively as a pre-processing step for ML algorithms [24]. The range of values of different features is on different scales. If one of the features has considerably wider range as compared to other features, the optimization function for the ML algorithm will be governed by that particular feature and will affect the performance of algorithm. Also, it will take much longer time for the optimization objective to converge in such cases. Feature scaling is therefore a method that brings all the features on the same scale making sure they contribute equally to classification algorithm. Standardization is most commonly method used for feature scaling which involves having each feature represent as a Gaussian like curve with zero mean and unit variance.

3.1.3. Data Reduction

Data reduction is perhaps the most important pre-processing step when dealing with very large data sets. Several variants of aggregation techniques are used in order to reduce the data size without any loss of information. PAA and extended version of PAA called SAX are the most commonly used aggregation techniques in IoT for data reduction.

PAA is a simple dimensionality reduction method for time series analysis. In order to reduce time series from n dimensions to m dimensions, the data is divided into m equal sized frames and the mean value is calculated for the data lying in that frame. The vector of mean values calculated will represent new series with m dimensions.

Feature extraction is another technique used widely for data reduction where the number of features of data is large and mostly correlated to each other. Feature extraction enables to extract most relevant and uncorrelated features in order to perform optimum analysis. PCA [25] is one of the most famous feature extraction technique which form new uncorrelated features based on the statistical properties in order to represent the same data with less dimensions. PCA is widely used in image processing and pattern recognition systems.

3.2 Functional Overview

This component provides several functions at VE and platform level which are summarized below:

1) It provides the capability for basic pre-processing on raw data streams including filtering, selecting or aggregation on real-time data using light weight CEP running on VE;
2) It provides the capability to run pre-processing on the object storage using storlets. Storlets can be both generic and domain specific as well. For domain specific storlets, an application developer can write his own storlets which can be run using restful web interface;
3) It provides the functionality to run different pre-processing techniques including aggregation, interpolation, and data cleaning and feature scaling using Apache Spark on the platform level.

3.3 Connection with other Components

Pre-processing FC provides the capability to pre-process raw data using light weight CEP running at VE level and to pre-process VE data using storlets and Spark. Both connections are shown in the Figure 8. 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.

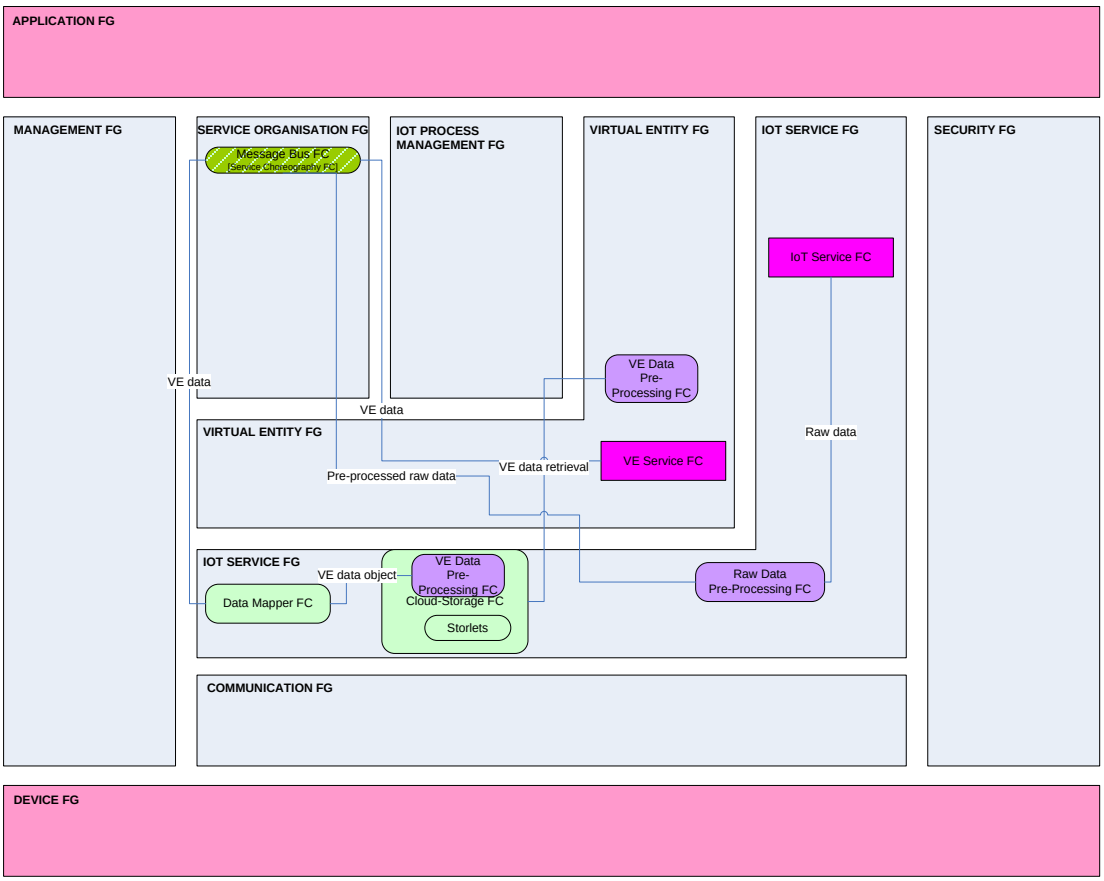


Figure 8: Connection of pre-processing FC with other components

3.4 Interfaces

As shown in the Figure 9, Pre-processing component is present at three different levels and the interfaces for every level are described below:

VE Level using CEP

An interface will be provided for application developer to define the pre-processing using DOLCE language for the CEP. The output will be published to under the specific topic on message bus.

Object Storage using Storlets

An application developer can code their 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.

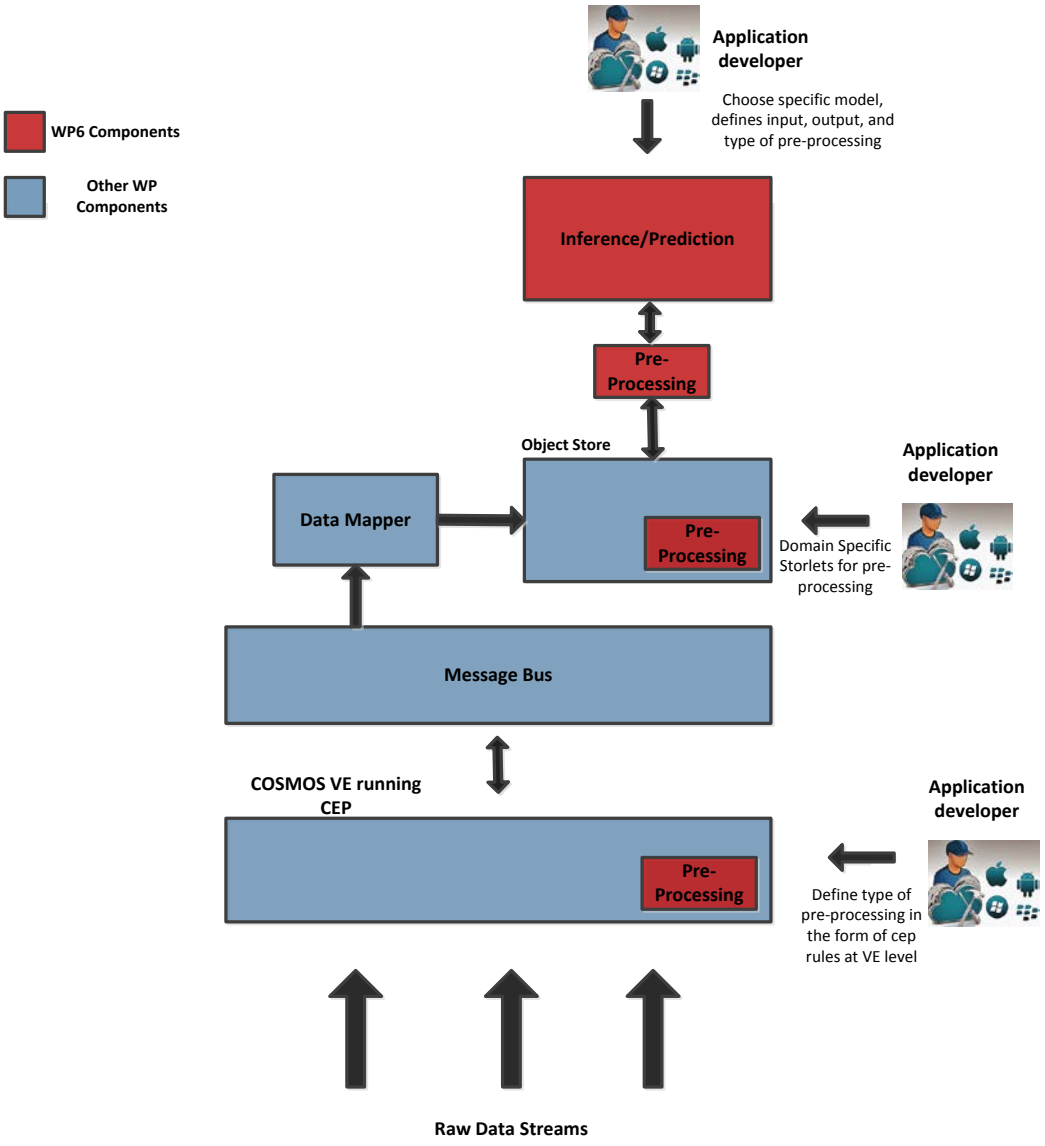


Figure 9: Pre-Processing component

3.5 Use-case scenario

A use-case scenario for the pre-processing has already been described in section 2.5 where we discussed using aggregation as a pre-processing step using storlets for reducing the amount of data travelling over the network and hence reducing the time complexity for machine learning algorithms.

3.6 Conclusion

Pre-processing is an important step for running analytics on large IoT data sets. It is possible to carry pre-processing at different levels within the platform depending on the application. Pre-processing on raw data can filter the data and only store the relevant data, while pre-processing after storage can be useful for running different types of complex analytics.

4 Event (Pattern) Detection Functional Component

4.1 Introduction

There are many applications in IoT which requires real-time processing of data such as intelligent transportation systems and smart buildings. As an example, consider an intelligent transportation system: analytics on the historical data can enable the researchers to have an idea about the average traffic flow which can assist when taking important decisions like managing traffic signals and the location of important buildings such as hospitals and fire stations (as they should be located around the places where less dense traffic provides easier access). But real-time traffic flow is a random and dynamic process and any incident or external factor such as bad weather or rain can affect the traffic flow and result into traffic congestion. Historical data analysis does not provide the mean to detect traffic congestion in real-time. Real-time monitoring of traffic system is a very important aspect of smart city applications. Traffic has to be managed in an optimized way by analysing parameters like traffic flow and traffic density in real-time. There are thousands of sensors deployed in smart cities generating gigabytes of data per day. The process of analysing, inferring and correlating these data streams in real-time requires therefore alternate solutions.

The Research area of *Complex Event Processing* (CEP) includes processing, analyzing and correlating event streams from different data sources to infer more complex events in real-time. The main objective of CEP was to provide the processing capability of big data engines which enables it to analyze the data and extract patterns on the run in real-time with distributed architecture. The main difference from the traditional big data engines was that CEP can handle multiple events which are seemingly unrelated and can correlate them in order to provide a desired and meaningful output. In the early years of data processing, data was at rest but now data processing is taking place as the information is in motion. Sensors data, stock market feeds and GPS data from vehicles produce data in the form of real-time data streams.

CEP engines require rules or patterns to detect an event from data streams which have to be given manually by the administrators of the system. Based on this, there is an assumption that administrators have the required background knowledge which sometimes is neither available nor so precise. The manual setting of rules and patterns limits the use of CEP only for expert's domain and poses a weak point and even though with prior expertise and knowledge, experts are prone to make errors in choosing optimized parameters for dynamic systems. Systems based on CEP deploy static rules and there is no means to update the rules automatically. In real-time dynamic scenarios, parameters of a system may change and performance of CEP may deteriorate in such dynamic scenarios. In our work, we have addressed the above mentioned limitations of CEP using our novel approach based on adaptive clustering in order to exploit historical data and find the optimized parameters for CEP rules automatically for extracting complex events from real-time data streams. The parameters for CEP rules depend on the threshold values in order to differentiate between different events. We explored adaptive clustering approach for finding threshold values and update the rules accordingly. Our approach is able to follow the changes in the distribution of data and adapt to it in run time. Our proposed architecture is able to infer complex events from raw data streams in a distributed manner and is able to provide adaptive solutions.

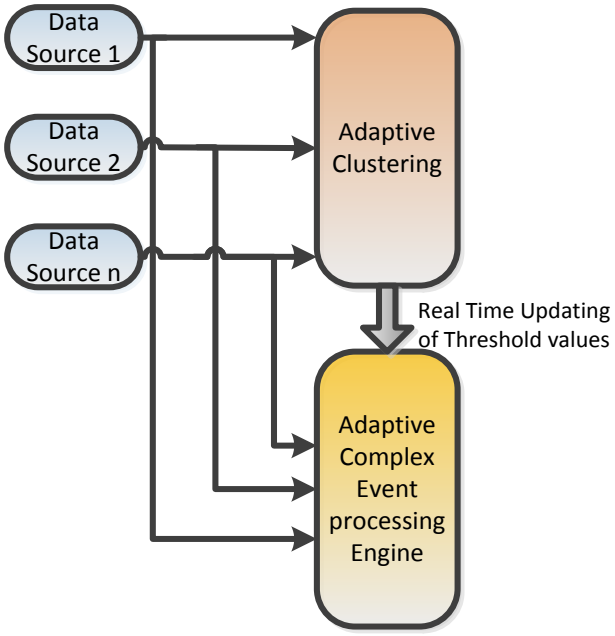


Figure 10: Proposed solution for Event Detection

4.1.1. Clustering

Clustering refers to the unsupervised ML technique which is used for grouping similar objects on the basis of pre-defined metrics 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 generates a hypothesis about the given data, whether the data under observation has distinct classes, or overlapping classes or classes with fuzzy nature. With the advent of big data, clustering applications have even increased. Clustering serves as the basic of many data mining techniques and finds applications in almost every industry. For example market researchers apply clustering techniques to group the customers into different segments, such that customers with common interests are in a same group. In this way, they can target different groups with different focused offers which are more related to them. Social networks apply clustering to group similar people into communities in order to help in recognizing people with similar interests from a larger set of people. Another interesting application of clustering is used by insurance companies to group different areas on the basis of crime rates so that they can offer different rates for insurances to the different areas depending on the level of insecurity.

4.1.1.1. K-means

K-means [26] is an iterative process which forms the clusters by finding centroids such that the sum of the squares of distance from centroids to data is minimized. For a data set X with n number of samples, it divides them into k number of disjoint clusters, each described by the centroid value. In the first initialization step, it assigns initial centroids, most commonly by selecting k number of samples randomly from the data set X. In a second step it assigns each

sample to its nearest centroids and forms k clusters. In the third and final step, it creates new centroids by calculating the mean value of the samples assigned to each previous centroid and calculates the difference between the old and new centroids. It keeps on iterating the process until the difference is less than a threshold value. K-means algorithm finds the local optimum solution which depends on the initial positions of centroids it has chosen. Hence usually it is run multiple times with different random initializations and chooses the best result from multiple runs. The number of clusters k have to be specified in advance and is a major drawback in the approach as the distribution of data may be unknown.

4.1.1.2. GMM

Gaussian Mixture Models (GMM) [27] is a probabilistic clustering technique. GMM is a probabilistic model which assumes that all the observation points are generated from a mixture of finite number of Gaussian distributions and the parameters of Gaussian distribution are not known. It can also be taken as the generalization of k-means clustering which also incorporate the information about the covariance structure of the data. K-means is an example of hard clustering whereas GMM is an example of soft clustering where assignments of underlying data points to particular Gaussian distribution is done in terms of probability.

The source of generation of different data points is unknown which makes it harder to learn GMM for the data observations as one does not know which points belong to which source. This problem can be solved using Expectation Maximization algorithm which is well known statistical algorithm and can solve this problem by iteratively optimizing the observation data set over different distributions.

It works in two steps. In the first step, it assumes Gaussian distributions centered around random points in the data set and then it computes the probability for each point being generated by every component of the model. It then maximizes the likelihood of data for those parameters. It keeps repeating it until the likelihood reaches the local optimum.

4.1.2. Complex Event Processing

A number of COSMOS components are using the µCEP engine developed by the *Internet of Everything* (IoE) Lab of ATOS Research & Innovation division. Due to the fact that this component is served as an enabler for the real-time analysis of streaming data, it is already described in deliverable D4.1.2 - Information and Data Lifecycle Management [28]. The following picture represents the architectural view of this component.

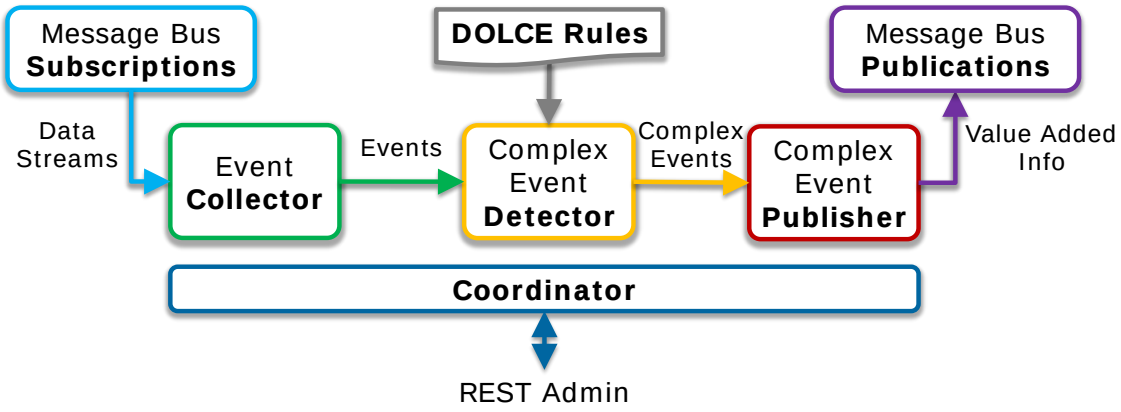


Figure 11: µCEP interfaces

4.2 Functional Overview

This component provides the following functionalities.

1) It enables to correlate the data from different data sources in real time to infer complex event.
2) It exploits the historical data to automatically calculate the parameters for CEP Rule so the administrators do not require domain knowledge.

4.3 Connection with other components

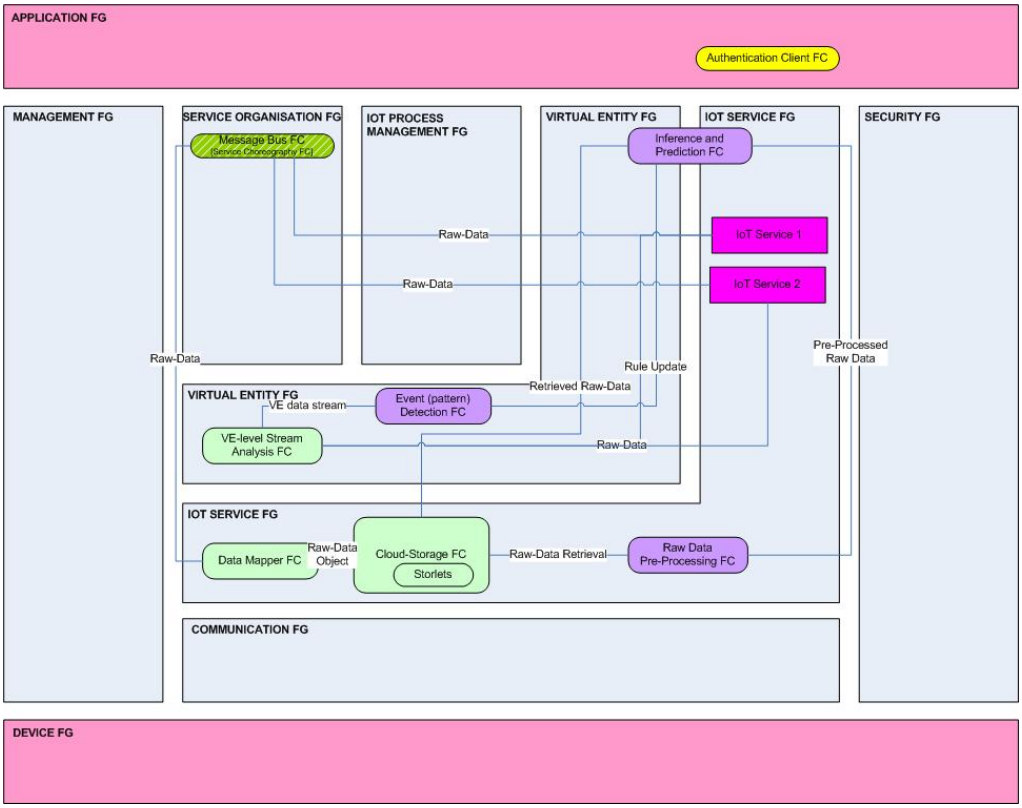


Figure 12: Connection of Event Detection FC

Event Detection FC is using VE-level stream analysis capability provided by COSMOS. As it can be seen from the Figure 12, raw data is generated by IoT Services and Event Detection block correlates the data using DOLCE rules provided by application developer in 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 access historical data and run machine learning algorithms to estimate optimized parameters for CEP rules which will be updated automatically through interface provided between Inference and Prediction FC with Event Detection FC.

4.4 Interfaces

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, what 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, what is possible to be done using a specific API function over the *REST Administration* interface. In this sense,

Event Detection-Inference/Prediction

An interface will be developed which connects the Event detection component with the Inference/Prediction component for updating of threshold values for the CEP rules automatically using ML methods.

4.5 Use-Case Scenario

We have chosen Intelligent Transportation as a use-case scenario in order to demonstrate our approach due to its immense impact on Smart City. Traffic management represents one of the important aspects of Smart City with a strong impact on the potential development of the city and the quality of life. The city of Madrid has deployed thousands of sensors which publish their data in open data license. The city of Madrid has traffic speed sensors and traffic density sensors which are direct indicative of traffic condition.

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 an average traffic flow/traffic density which represents the average number of vehicles passing through a certain point. The highly dense traffic with high average speed may indicate free flow traffic while the less dense traffic with low average velocity might or might not be a bad traffic condition.

Real-time monitoring of traffic system is a very important aspect of Smart City applications. Traffic has to be managed in an optimized way by analysing parameters like traffic flow and traffic density in real-time. There are thousands of sensors deployed in Smart Cities generating gigabytes of data per day. The combination of different sensors forms complex patterns, and these patterns can be exploited in order to infer complex events. A pseudo code for an example of a rule can be:

- If *Current Velocity* $(V_{current})$ < *Threshold Velocity* $(V_{threshold})$ **and** *Current Traffic density*$(D_{current})$> *Threshold Density* $(D_{threshold})$ for **5 *minutes*** → *Traffic Congestion*

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.

4.6 Conclusion

In this chapter, we have described the functionality and the background knowledge of Event Detection Functional Component. We have proposed a novel technique based on CEP and Machine Learning for inferring complex event adaptively. The proposed architecture is able to update the rules automatically and ensures the system performance for dynamic scenarios where the statistical properties of the underlying data may change over time.

5 Situational Awareness Functional Component

5.1 Introduction

As it was stated in the updated version of State of the Art document [1], it is not an easy task to provide a closed definition of what *Situation Awareness (SA)* is. In spite of that, we are focusing on the definition that defines the entire SA process as "*the perception of the elements in the environment within a volume of time and space, comprehension of their meaning and the projection of their status in the near future" [29]*. Following this approach, worth going into details of each step:

- **Level 1 - Perception of the elements in the environment:** The first step in achieving SA involves perceiving the status, attributes, and dynamics of relevant elements in the environment. This will be commonly referenced as "acquiring context", "receiving data streams", "exploiting information from data sources", and the like;
- **Level 2 - Comprehension of the current situation:** Based upon knowledge of Level 1 elements, particularly when put together to form patterns with other elements, a holistic picture of the environment will be formed, including a comprehension of the significance of information and events;
- **Level 3 - SA Projection of future status:** It is the ability to project the future actions of the elements in the environment, at least in the near term, that forms the third and highest level of Situation Awareness. This is achieved through knowledge of the status and dynamics of the elements and a comprehension of the situation (both Level 1 and Level 2 SA).

In order to complement the above, the following summarizes the categorization of *Context Information* that best represents the Use Cases that are being developed in our work:

- **System Context:** Corresponds, for a given application, to the context information of the system (software and hardware) on which it runs as well as contextual information of the used communication system (for example, the wireless network type).
- **User Context:** It is any information that can characterize a user. This may be the age, location, medical history, biometric information, emotions, etc. It may also be user activities, social relationships, family, friends, colleagues, ...
- **Environment Context:** Represents information describing the physical environment that is not covered by the system and user contexts. Particularly the context information from external sources such as temperature, climate, lighting, ...
- **Temporal Context:** Defines all information related to time, especially: hour, day, month, year...

5.2 Functional Overview

The large volume of data which is made available in an IoT environment does not necessarily mean applications can take effective decisions directly or make correct interpretations. For example, a single vehicle reporting a low speed is not always an indication of a traffic jam. On the other hand, a large number of vehicles reporting slow speeds on a particular highway section can be interpreted as an indication of such a traffic condition.

COSMOS intends to support the transition from raw data to value added information by providing mechanisms which facilitate SA at two distinct levels: the *Virtual Entity* (*VE*) *centric SA* and the *platform level SA.* Both of them will be described in the following sections.

Despite their differences both levels are targeting the extraction of value added information from raw data in order to place VEs into the relevant context when required. Also targeted is the information sharing among VEs.

The sharing of information among VEs is supported through the use of semantic descriptors for the resulting SA information. This allows VEs to query for SA information either from the platform or from other VEs.

5.2.1. Data Sources

While analysing the prerequisites for the generation of SA information we have identified four distinct data sources which VEs should be able to access.

Figure 13 below depicts the data sources as well as the flows involved in the SA information generation and sharing.

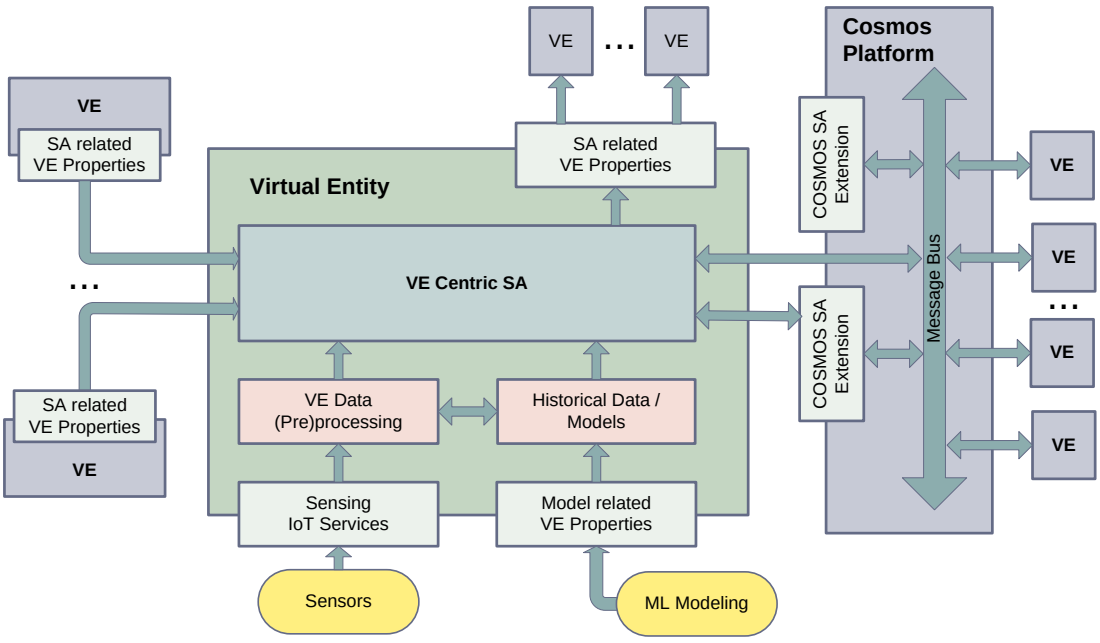


Figure 13: Situational Awareness data exchanges

Internal sources – raw data

The most basic and direct way VEs are able to obtain data for the generation SA information is by using direct sensing through the sensors which are associated to each VE. In other words, the code implementing the VE centric SA is accessing its own sensors, and applies to the cases where for the SA assessment there is no need for further data processing.

Internal sources – processed data

In this case the SA context is generated by mechanisms which could combine data from multiple sensors of the VE, an internal data processing mechanism, and even VE level historical data.

A simple example would be the computation of the expected battery lifetime for a device used in telemetry, for instance a remote weather station which could read the battery voltage and

drawn current, the measured charge up to the moment, the device and the ambient temperature and also use historical data about the battery charging /discharging cycles.

External sources – COSMOS platform SA extensions
This is mainly the case where the VEs are not able to extract directly information about their environment through their own sensing capabilities but can use publicly accessible services. These might be any service, provided that the VE is able to access it and has a compatible interface. Apart from external data sources, COSMOS itself will support the development of platform extensions. These are able to process raw data submitted by VEs and to generate reusable and shared SA information which any VE can use.

Such an extension could provide traffic related information, as presented in the example described at the beginning of the section, based on the raw location data submitted by vehicle associated VEs.

External sources – other VEs
This is the case where VEs require SA information about conditions which are beyond their own sensing range and which can be provided by other VEs.

This is another sharing mechanism COSMOS will provide and facilitates VE to VE direct communication without the data being relayed through the platform.

For example, in a logistics scenario, lorry VEs could be able to find other lorry VEs operating on the same route and ask about the current restrictions in place or about queues at border crossing points. The way these data sources are built and used is described in the following sections.

5.2.2. Data Flow and Processing Model

The key concept of COSMOS is the VE which adds a new level of abstraction and functionality on top of IoT services. As mentioned before, the SA approach in COSMOS relies not only on the VEs' capability to sense the operating environment but also on its social characteristics and the ability to share information either in raw or processed state.

The ability to produce and consume data using such sharing mechanisms has to be considered at design time and COSMOS facilitates this task by providing the necessary infrastructure and design patterns. The following paragraphs describe this from the perspective of the relevant COSMOS actors.

VE developers
The VE developer is the one which builds VEs around existing or new IoT services and adds new functionality with the support of COSMOS. While using only internal sources, SA computation requires no or limited interaction with the platform or other VEs. Still, as mentioned above, determining a VE's SA context can be achieved by also using information from other VEs or from the platform SA extensions. This is only possible if the conditions listed below are met.

The first condition is that VEs share completely or partially their own raw data or even their own SA information. This data can be made accessible directly, by the VEs using their own interfaces, or indirectly, by publishing data to a COSMOS platform instance. The task of the VE developer is to implement these COSMOS compatible interfaces based on the provided design patterns and to properly describe the VE's capabilities when publishing it to the VE registry.

The second condition is that VE developers design and develop VEs considering the VEs' ability to access information which is beyond their sensing capabilities by relying on external data sources. This means that once a VE instance is deployed, a binding between such external sources and its own state variables has to be created based on a set of requirements, as later described.

Both conditions are easy to be met since COSMOS supports these requirements by providing an uniform way of accessing data (by using interfaces and data formats compatible between VEs) and by providing the means to search and address this data through the use of the semantic descriptions published and stored into the VE registry.

Application developers
Application developers are using VEs and the platform in order to create VE enabled applications. VEs are deployed either by VE developers or by application developers depending on the scenario. With regards to the SA assessment based on external data sources, at the moment of the deployment, the mapping of the requirements for the bindings mentioned above have to be configured so that the VE is able to retrieve the relevant data sources.

These requirements specify the semantic type (e.g. a VE needs outside temperature information from a specific location) as well the data type (e.g. which kind of schema is required). This is needed in order to provide semantic data compatibility between the producer and the consumer, but also to build the VE registry query parameters for finding the appropriate data sources.

Thanks to COSMOS, the task of the developer is limited to the proper description of these requirements and to the use of the VE registry query mechanism.

Platform owners
Each COSMOS platform instance is operated by a platform owner (e.g. a municipality, a telecom company, an internet service provider, etc.). A platform owner can attract VE developers and application developers by providing extensions of the COSMOS instance they own. These extensions provide additional functionality such as publicly available SA data sources.

Platform extension developers
The above mentioned platform extensions are domain specific implementations which can be plugged into a COSMOS instance in order to provide additional functionality. This extension mechanism is required since the COSMOS platform core functionality is meant to be domain independent.

A COSMOS platform extension is nothing else than an application which is able to consume data published by VEs to the platform or data from other data sources, process it so that it adds value to that data and then share the results by publishing them back to the platform. These extensions can run as stand-alone components, independently of the platform but can also use, if required, platform level functionality such as the event processing or the ML models and prediction mechanisms exposed by the platform.

An SA extension follows the same pattern and is meant to provide value added SA information to VEs by processing high volumes of raw data published by other VEs.

SA in COSMOS is available, as mentioned before at two distinct levels: VE centric and platform level respectively. The main difference between the two lies in the data processing model.

At VE centric level the SA assessment is performed mainly based on the VE's internal data sources. Also, the data processing takes place internally, as part of the VE implementation. The VE developer bears the responsibility for the design, development and testing of this functionality as well as for the proper annotation of the results (to support further reuse based on information sharing) once the VE is published into the VE registry.

Platform level SA involves the processing of large quantities of mainly raw data submitted by VEs. It uses techniques which might require more computing power and are not suitable for VE centric deployment. Some of these processing components are also provided by the COSMOS platform such as those for complex event processing, prediction using machine learning models, advanced storage and processing using servlets.

In contrast with the VE centric approach, the platform level SA does not require any VE developer effort, except for the construction of the queries used to retrieve the information exposed by the platform extensions. This is because in this case, VEs are using SA information which is made publicly available by the platform. SA extensions are built for the platform owner so that it can extend the core functionality of the platform it operates. Depending on the processing requirements, these extensions can be deployed together with the platform or as stand-alone applications. It is the task of the platform extension developer to properly design, develop, test and annotate these services which facilitate information reuse.

The backbone for the platform level information sharing is a message bus which follows a publish/subscribe pattern. Choosing such a design pattern was natural given the requirements mentioned for information sharing. VEs act as publishers (when submitting their own data) as well as subscribers (when receiving reusable information such as SA assessments). The same applies for different components of the platform as well as for the platform extensions. A description of chosen message bus solution is found in section 5.3.3.

5.2.3. Information sharing and retrieval

As mentioned above, information sharing is a key concept in COSMOS. COSMOS provides the components which facilitate data processing and storage at both VE as well as platform level. Nevertheless, the ability to produce and store raw data and value added information does not suffice when it comes to information sharing, since such scenarios involve different data sources and sinks employed by different owners.

Such data needs to be retrievable and properly addressable. To support these requirements, COSMOS employs the use of semantic descriptions of the VEs and of the platform level data sources and sinks (implemented as message bus topics).

Based on the light-weight, domain independent COSMOS ontology, and a dedicated API and tools, VE or platform extension developers are able to describe each data source and sink. This description seeks not only interface compatibility with regards to the data types but also from the semantic type so that the matching between the producer and consumer is guaranteed. As a result, a VE will not only be able to request a data source returning a Double data type but a Double representing the temperature in °C from specific room of a office building.

The ability to describe the data from both the data type as well as the semantic type perspective, combined with the querying mechanism which includes such constraints, extends the ability to share and reuse data among VEs and among VEs and the platform.

These mechanisms are described in detail in the Revised Architecture Deliverable D2.3.2 [30].

5.3 Connection with other components

Apart from the semantic technologies used for the description of the COSMOS entities, such as the ones listed above, the platform makes use other technologies to extend its functionality and facilitate the use of IoT services.

In order to achieve the process of real-time situational awareness, the combined used of this technologies should comply with the following facts:

- Be very efficient to deal with huge amounts of events;
- Predict the occurrence of 'interesting' situations;
- Be tolerant to various types of noise;
- Deal with dynamically changing situations.

5.3.1. Complex Event Processing Engine

A CEP engine is designed as a component to be fed by an amount of data sources, whose ultimate goal is to generate value added information in the form of complex events, that are the output generated after processing many small, independent incoming input events, which can be understood as a given collection of parameters at a certain temporal point.

In relation to the three levels of Situation Awareness, µCEP engine can provide up to Level 1 and Level 2 SA working on its own. To do so, the µCEP instance features one or more Event Collectors modules to acquire information from heterogeneous Data Sources, as it was presented in section 5.2.1 Internal Source providing raw data (Level 1 SA). Together with a specific set of DOLCE Rules, this Context Information (system, user, environment or temporal) is processed at the time of arriving to the engine, where historical data is maintained by means of Time Sliding Windows and Tuple Windows. When a certain rule is triggered, relevant Complex Events are generated by aggregating input data streams along with post-processed information, becoming an Internal Source delivering processed data (Level 2 SA). Finally, the Complex Event Publisher module transfers the value added information to the VE or component which will take advantage of it. Furthermore, when the µCEP is used in combination with other techniques, such as ML, it is possible to get Level 3 SA, as will be explained later on this document.

5.3.2. Machine Learning

ML represents –as said in previous chapter already- any algorithm which learns from the data. Methods based on ML have the potential to extract high-level knowledge from raw IoT data and to contribute for novel applications. We have described several ML methods in section 2. In this part, we will explore how the high-level knowledge can be used for Situation Awareness at both VE centric level and platform level.

At platform level, different type of data sources -as discussed previously- are generating huge amounts of data which is heterogeneous, mobile and ever changing along time. Data from different data sources form complex patterns and analysing, inferring and correlating these complex patterns in order to access the current situation is by no means a trivial task. Consider a simple example of Situation Awareness at platform level as shown in Figure 14, where an instance of different events happening at the same time is shown. The task of SA component is to combine this information from the external data sources to extract high-level knowledge that represents the current situation. Pattern recognition techniques such as classification and clustering can be explored in order to detect particular situation.

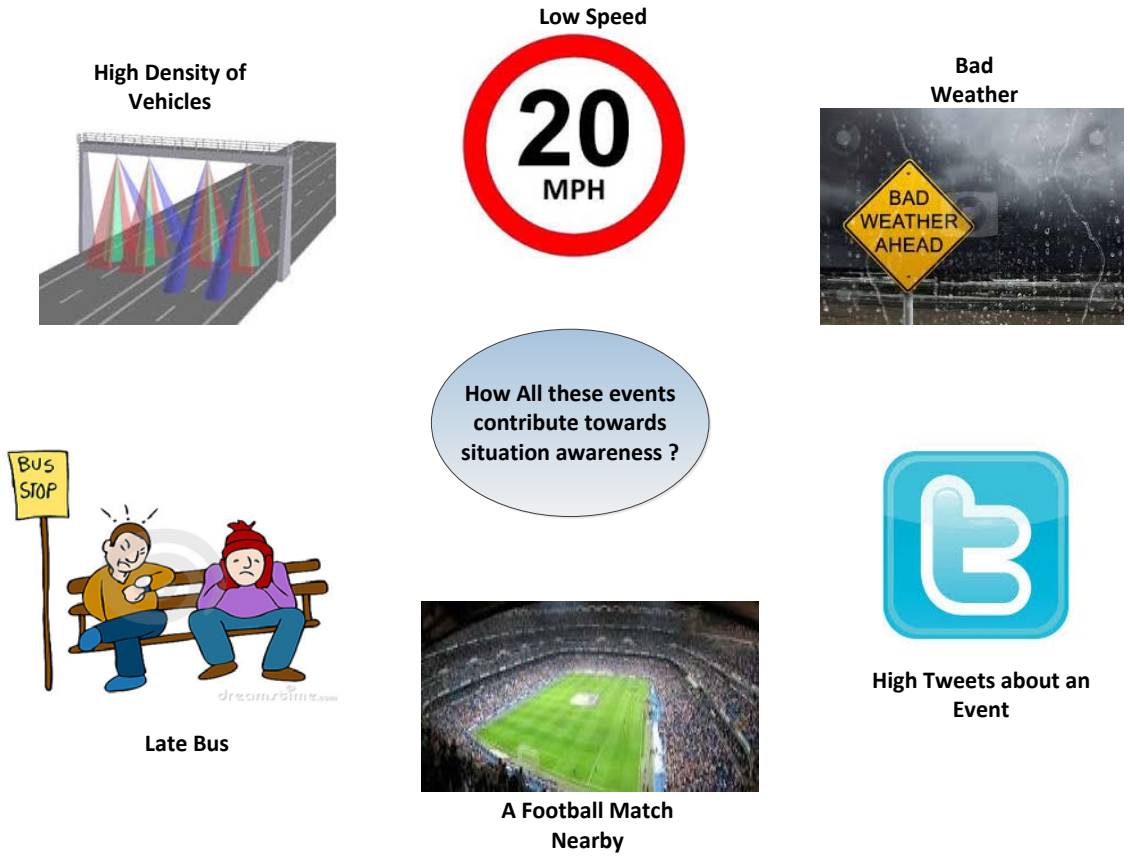


Figure 14: An Example of Platform Centric Situation Awareness

At VE-centric level, the amount of data generated will be less, but at the same time the resource constrained nature of VEs limits the use of complex ML tasks. One proposed solution is to exploit the historical data of VEs in COSMOS platform in order to train ML models and then extract high-level knowledge. The work done in Year 1, in which occupancy state of a user is inferred from his electricity usage pattern is an example of VE-centric SA.

The discussion until now falls into perception and comprehension of current situation but the vision of SA in COSMOS is beyond it. We intend to explore the projection of Situation Awareness for potential future events. The value of predicting future events is of immense importance as it enables the administrators of the platform to take pro-active approach to potential situations. We intend to explore prediction techniques from ML domain in order to provide projection of SA. One possible approach for the projection of events is shown in Figure 15 below.

5.3.2.1. Projection of SA using ML

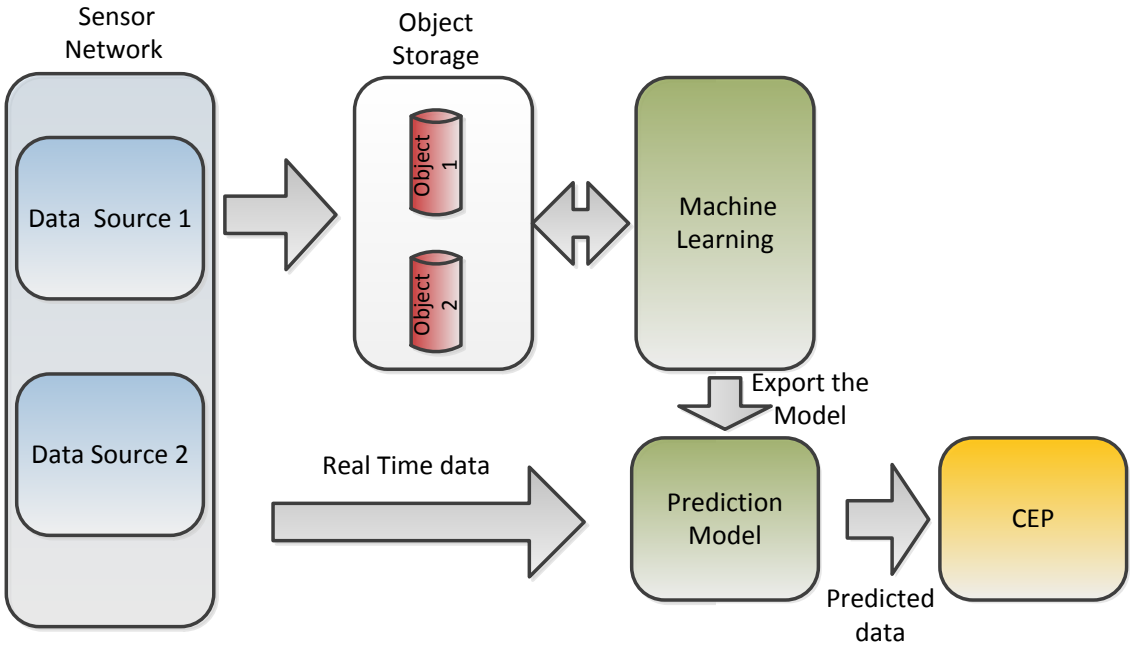


Figure 15: Prediction of Events for SA Projection

We propose to explore ML techniques for the projection of SA for potential future events. It enables the system administrators to adopt a pro-active approach and take measures before the actual event happens.

A high-level architecture for the proposed approach is shown in the Figure 15. The intuition behind our proposed approach is that if the input data streams to CEP are predicted data (in future), then CEP can be used for predicting the potential future event. The model for predicting the data can be trained and validated offline using historical data sets. The accuracy and complexity of prediction models will be the main research challenges in this work. Prediction Models trained on historical data may become inaccurate with the passage of time due to phenomena of concept drift. The statistical properties of the underlying data may change with the time. Prediction Models should be able to cope with the changes and adapt accordingly.

5.3.2.2. Automatic generation of Rules for CEP using ML

As discussed earlier, manual setting of rules and patterns limits the use of CEP only for expert's domain and poses a weak point. In this regards, we intend to apply predictive analytics principles to improve decision-making and performance of existing CEP solutions by exploring adaptive and automatic solutions. Both fields have many overlapping concepts but yet there exist only few solutions where PA concepts have been applied to CEP. One possible research area to explore is automatic generation of rules for CEP in order to cope with above-mentioned limitations. CEP language uses number of operations for describing pattern of events. In [31], authors applied the ML techniques for generating automatically the following five most commonly used operations:

1. determine the relevant time frame to consider, i.e. the window size;
2. identify the relevant event types and the relevant attributes;

3. identify the predicates that select, among the event types identified above, only those notifications that are really relevant, i.e., determine the predicates for the selection operator;
4. determine if and how relevant events are ordered within the time window, i.e., the sequences;
5. identify which event should not appear for the composite event to happen; the negated event notification

The authors applied (Information Gain Ratio) IGR principle and implemented each operation in different module. This is just one example where ML can be used in conjunction with CEP. We intend to explore more predictive analysis methods in conjunction with CEP to provide more adaptive, automatic and optimized solutions that can lead to more accurate, faster and consistent performance.

5.3.3. Message Bus

The Message Bus FC entails a key integration point for the cooperation of several entities in a common scenario. Based on a publish/subscribe messaging mechanism, it allows for sharing information from heterogeneous data providers by abstracting the inherent complexity that implies dealing with different protocols and data formats. To do so, each COSMOS component provides the necessary modules to adapt to this horizontal communication bus. Furthermore, the new Message Bus implementation provided by COSMOS during Year 2 improves the way data queues are handled, thus providing new mechanisms for fault tolerance and data persistence.

5.3.4. Forging CEP and ML

CEP and ML have many overlapping concepts but yet belong to two different research fields. CEP acts on real-time data from multiple data sources in order to analyse, correlate and infer a more complex event whereas ML methods are based on exploiting historical data to learn models which can be deployed for prediction. CEP engines require rules or patterns to detect an event from data streams which have to be given manually by the administrators of the system. Based on this, there is an assumption that administrators have the required background knowledge which sometimes is not available or not precise enough. So manual setting of rules and patterns are in a sense weak point of CEP.

We proposed to explore ML methods by exploiting historical data for finding rules for CEP. As we have discussed previously, that VE-centric SA is performed mainly locally as a part of VE implementation. A light version of CEP provides a good solution for combining data from different local sensors of VE to extract high-level knowledge. The resource constrained nature and limited memory of VE are the limiting factors of exploiting historical data and using complex machine learning models. Our proposed solution fixes this problem as part of complex computations for finding optimized parameters for CEP rules can be done in the central platform and the rules for the running CEP at VE level can be updated periodically.

5.4 Interfaces

For both the VE-centric approach as well as for the platform level SA, the interfaces provided for accessing SA information follow the same pattern as those of the "standard" VE properties.

In fact, there is no difference between accessing an IoT-related VE property and an SA related one.

SA information access over both REST as well as message bus endpoints is supported at VE as well as platform level. In addition to the interface type, we need to emphasize the fact that in either case the endpoints are also semantically described as any other endpoint of the VE allowing SA information retrieval and sharing.

5.5 Use-case Scenario

Existing Madrid EMT transportation infrastructure provides possibility to build contextual models based on data streams related to various sources, such as the context of a particular VE Bus –position, velocity, schedule, CO_2 emission–, as well as the status of the surrounding environment of the VE –weather condition, scheduled city events, local traffic congestion–. Taking advantage of these data sources, combining them with other external related ones, and even making up relevant historical data, it is possible to foresee certain scenario evolutions like traffic jams or bus delays. Figure 16 highlights the suitability of a SA mechanism in the *Decision Making* process that is being followed.

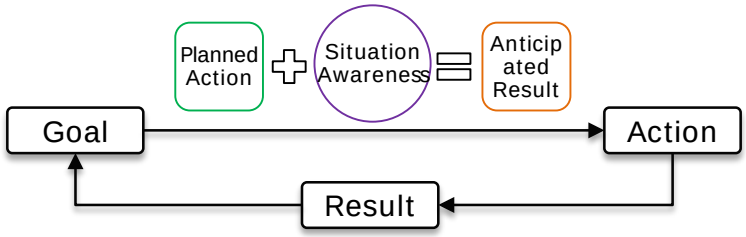


Figure 16: SA in Decision-making process

Attending to this diagram, it is crucial to have a clear identification and description of the goal to achieve. In the urban mobility scenario we are going to monitor the journey of a *person with special needs* on his/her way home, while the correspondent *caregiver* is sent notifications if any abnormal situation may occur. In this sense, the planned action starts taking the right bus and ends up stopping at the correct bus stop, what defines the temporal boundaries of the SA process. Next step consists in selecting the data sources that will fed the system components involved in the use case, such as the location of the person with special needs, the route of those buses arriving nearby user's home and the traffic state condition in the city, to name only the most relevant ones. By acquiring, processing and maintaining this information during the required amount of time we are able to predict if the problem is going to be solved appropriately. The following Figure 17 depicts this process.

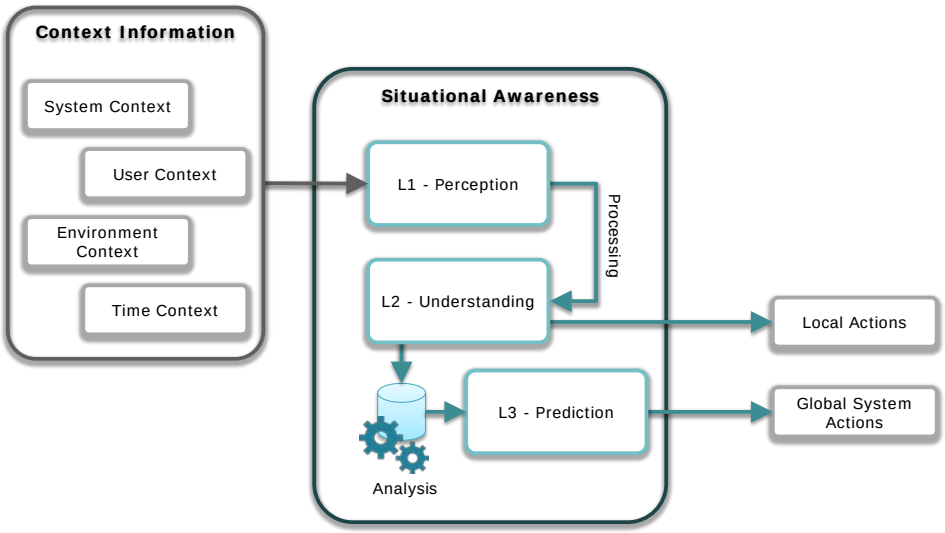


Figure 17: Situation Awareness process

Ingesting context information into the Situation Awareness functional component means that data sources are connected to the message bus publishing to topics, the ones that the µCEP components are subscribed to. Each time a rule is triggered, the generated information may flow to one or more components depending on what level of SA is needed in each particular case. As an example, in the urban mobility scenario the user's location is continuously compared with the location of the bus so to understand if the user keeps being inside the bus or not. Moreover, an additional source of data may provide valuable information into the equation, such as the WiFi networks the user mobile is detecting and the SSID the bus is broadcasting.

5.6 Conclusion

The nature of the IoT based applications (highly dynamic environment involving many times the mobility of user or even resources, the large number of data sources, their volatility, the number of involved actors etc.) brings new challenges for application developers but also opens up new opportunities. One of these opportunities is the ability to integrate situation awareness into the business logic of the application and access data about things beyond the coverage range of the things addressed by the application.

COSMOS is aware of this opportunity and therefore supports the SA through a series of data processing and sharing mechanisms which are exposed both as VE level as well at platform level. Depending on the type of the data sources and expected SA information, SA processing tasks can be executed as part of the VE functionality or in a centralized and high capacity manner when run by the platform.

These tasks can be light-weight processing jobs (especially in the case of the VEs with limited processing resources), can include advanced processing techniques relying on complex event processing or ML-based prediction models. In both cases, the output of the processing tasks (which is the SA information) is semantically described using the endpoint description model provided by the COSMOS ontology.

In Year 2 and 3 of the project we intend to demonstrate the feasibility of all these processing tasks and the ability to include them at VE and platform level, also supported by the semantic model.

6 Experience Sharing Functional Component

6.1 Introduction

During the Year 1 of COSMOS, we identified two classifications of Experience:

- The first was Cases as those are described in the context of the *Case-based Reasoning* (CBR) technique and correspond to the Planning component of the MAPE – K loop approach thoroughly introduced in WP5 deliverables;
- The second was Models with an emphasis to Prediction Models as those where described during Year 1.

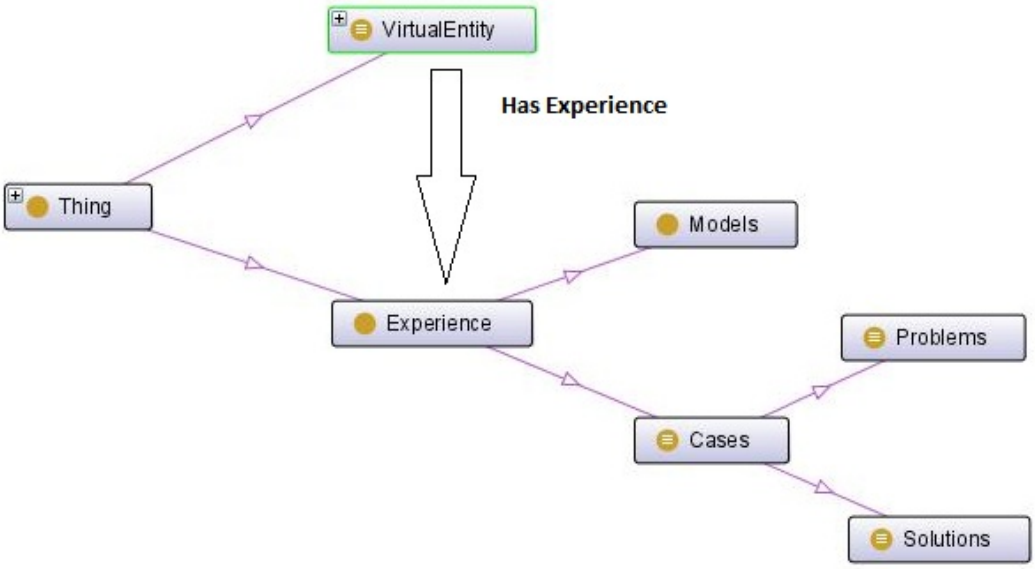


Figure 18: Experience in COSMOS

COSMOS focus was directed at implementing a method for enhancing the decentralized approach of VE operations, as was the impetus behind the use of CBR. Through Experience Sharing it was demonstrated that individual VEs could act in unison after being notified of a specific Experience need and generate a propagating recursive communication, to locate suitable Solutions to specified Problems. By using a list of known VEs (referred as "Followees"), the Experience Sharing component filters them in ways suitable to the current needs, as well as through Social characteristics.

Figure 19: Example VEs Followee connections

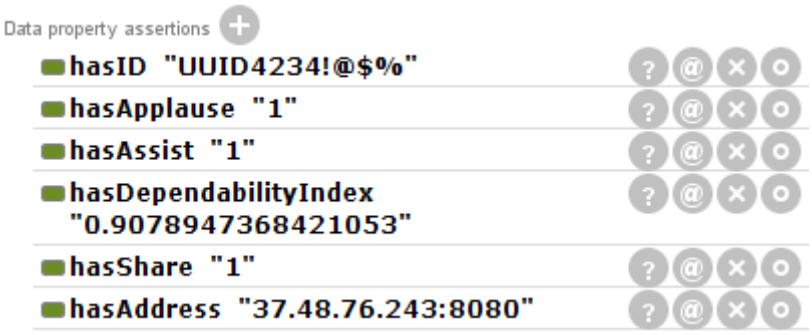


Figure 20: Sample data from a Followee

This leads to the contacting of Followees in a ranked fashion, based on their Social Grades. Additionally, by using the capabilities of the Planner, the process leads to increased chances of Solution recall and thus End-user or System satisfaction.

Experience Sharing also maintained data on the VEs providing the returned Experience and thus facilitates the work done on the Social Monitoring component, as it pertains to VE ranking and feedback.

Finally the concept of Experience Sharing has provided the framework for designing a variety of Decentralized Discovery Mechanisms in the context of WP5 and implementing a similar principle and functionality with Decentralized Followee Discovery.

6.2 Functional Overview

6.2.1.1. Resource Starvation Prevention

The primary approach to an eventual renewal and upgrade of the Experience Sharing component is the implementation of a load monitoring capability, so as to prevent the component in monopolizing resources on the VE executing environment.

During Year 1 the Experience Sharing component was implemented as a VE level service offered through a REST interface. Programmatically this was achieved using implementation of an embedded server and the accompanying servlet. The characteristic that promoted this approach is that since the Experience Sharing process does not involve modifying any local system resources (i.e. Case Base content) and simply accesses the persistent local store for the process of information retrieval, VEs can use the multithreaded parallel executing nature of servlets. Therefore in a decentralized environment, in which VEs demand swift access to information, specifically Cases during Year 1 but also extended into a Modelling approach for future Years, a target VE can accommodate a multitude of requests.

This creates the problem of resource starvation, in the sense of computational resources, because of limited hardware availability, which is the expected condition for most VEs that are to be deployed in the COSMOS Project. Also even if services remain accessible, the amount of time needed for the completion of a query still adversely affects user experience, or possible network response times, in the case of autonomous calls to the affected service. Therefore during Year 2 we will focus on a two-sided approach in resolving the expected problems.

The first part is connected to the improvement of the embedded server configuration. As mentioned in the initial and former version of this document D6.1.1 [32], VEs use the Jetty

server with minimal configuration changes. Therefore during the server instantiation, the server uses a configuration which is not optimized. The second part which is to be considered in tandem with the first, is the implementation of a request balancing mechanism on the service itself, which will act as a monitor of simultaneous connections to the Experience Sharing component, as those are translated into parallel running Threads. In order to achieve this, the servlet needs to implement a method by which a Thread safe approach to monitoring can be achieved.

Since implementation is being done in Java, COSMOS is focusing on the use of the Atomic variable classes that were added to JDK1.5. Called by IBM as the "hidden gem" [33] of JDK 1.5, they offer a Thread safe approach to any measurements VEs might need in a multithreaded programming environment.

Specifically by using Atomic Integers which can be increased on servlet start and decreased on servlet end the Experience Sharing component has access to information that can help it decide next steps, during times of great incoming request volume. In the event of too many incoming requests, potential client VEs can have their Experience Sharing requests dropped.

Dropping requests *en masse* from a specific VE is a heavy-handed method that does not differentiate on the actual VEs initiating communication and requesting knowledge.

An alternate approach is for the VE to possess multiple levels of load monitoring. Namely:

- If a VE has a low number of active incoming connections, then it will service all requests normally;
- The first threshold level of simultaneous connections will initiate a set of conditions, by which a VE will begin to demonstrate a more "selective" behaviour of request handling;
- The second threshold, which acts as a point of maximum load before QoS degradation, will initiate a strictly droppable approach on request handling.

In public good experiments, behavioural economists have demonstrated that the potential for reciprocal actions by players increases the rate of contribution to the public good, providing evidence for the importance of reciprocity in social situations. [34]

This is the principle that Experience Sharing is going to adhere to, meaning that if a VE has provided good quality Experience in the past, then the Social Monitoring Functional Component will have provided a good ranking grade for it. Therefore the selective behaviour previously mentioned will be directly dependent on previous subjective grading of the request initiator VE.

Testing will be integral in the refinement of the previously mentioned approach and will include various heavy use scenarios, targeted towards a multitude of server connections and incoming request thresholds. The tool used for the measurements and actual tests, is Apache JMeter [35]. These will include, among others, latency and throughput of the tested Services.

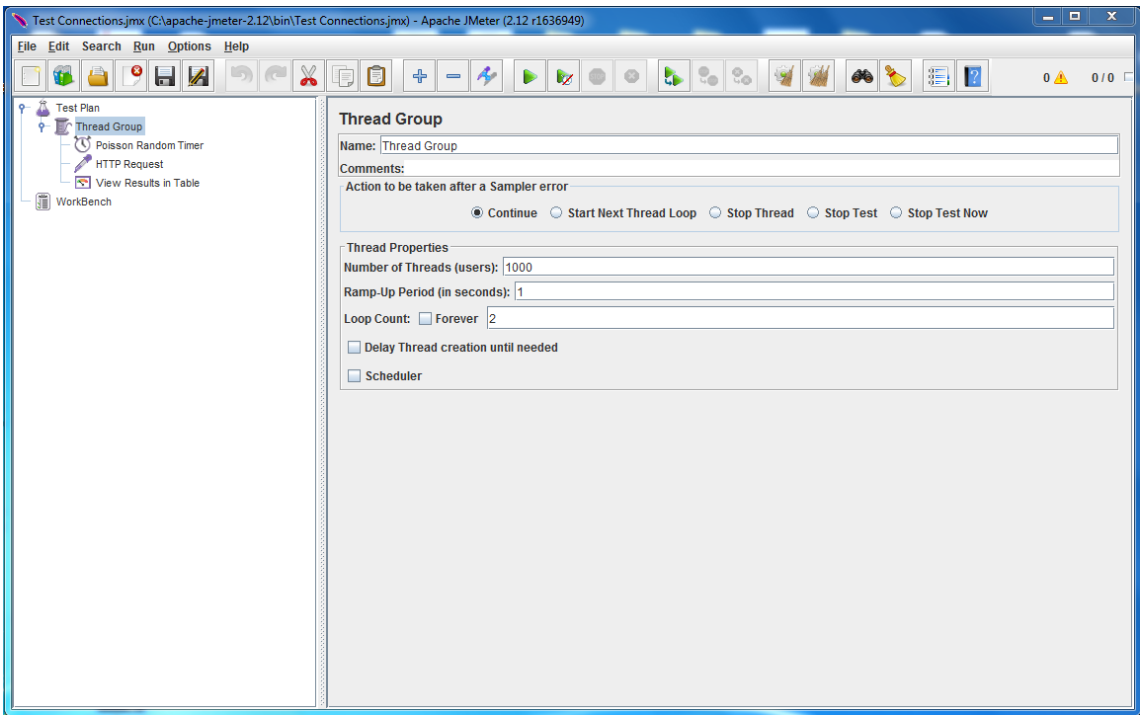


Figure 21: Apache JMeter tool for measuring Service performance.

Further enhancement of the Experience Sharing component for Year 2 includes the use of Models for promoting Knowledge acquisition.

6.2.1.2. Models and Experience Sharing

In a COSMOS ecosystem, many *Model Developers* (MD) could create models in order to solve a specific problem/prediction need. For homogeneity, these models should follow a specific API definition. This would enable generic client compatibility. However different problems have different model formulations, different input argument needs etc.

For a given problem (derived from the UC scenarios) COSMOS should define a proper method call format like the one of the following example, which implements REST communication:

```
AmountOfInfluence getAccidentImpact (AccidentLocation, PointOfInterest, TimeAfterTheAccident)
```

`PointOfInterest` is the location in which the client wants to measure the influence from the original `AccidentLocation`. The `AmountOfInfluence` response could be numerically the difference in the average speed. `TimeAfterTheAccident` is needed probably also, to calculate the varying effect over time.

A MD could implement the aforementioned interface and offer the model as a Service. Alternatively COSMOS could offer a Model repository with a REST interface as a front-end. The MD would also have to implement the backend Model to be used. At this point, COSMOS does not investigate possible details on backend Model development, but is merely interested in offering a well-defined process, with which a MD has to comply.

Of specific note is whether this Model can be treated as a VE for architectural and communicative purposes. If so communication to the remote Model Service could be accomplished through the Experience Sharing component method call instead of a specialized one, mainly because of the way the input parameters have been structured. For example, the previously mentioned input variables and the expected output could be mapped to the

variables used as input for the method call currently implemented as the starting point for the Experience Sharing sequence.

The primary requirement for Models to be shareable is defining their semantic description. By allowing Model description to be structured, the Experience Sharing components will be able to treat their storing and forwarding in the same manner as Cases. A local storage of Model descriptions is thus possible and can lead to similar retrieval methods as those of Cases. The connection of Models with VEs is plausible in the way their descriptions will be registered in the COSMOS ecosystem, though no strict registration process like the actual one of VEs is foreseen.

By maintaining a close connection between Experience Sharing and Modelling, there is also the possibility for reusing the functionalities of the Social Monitoring component. Therefore and provided Model Services can be considered as VEs, it is made possible for them to be stored and graded in the same sense as Friends inside the Friend list.

6.2.1.3. Proactive Experience Sharing

The third approach in magnifying the use of Experience Sharing, is the proactive use of the component in conjunction with the Situational Awareness component. In any use-case, there is potential need for a more active use of the knowledge diffusion mechanism, which does not lay squarely on receiving extra VE input for being activated. Knowledge in this case is a detected state change, which affects nearby VEs or which requires external VE actions to be corrected and thus is not part of the Runtime Adaptability.

A possible example on creating such a service asset is taken from a theoretical event in the Madrid Bus scenario. The Bus VE will use its SA component to detect a fault event from the engine. Considering that an engine fault leads to immediate immobilization, the Experience Sharing component will act in a proactive manner and propagate this event to the nearest Bus with the same next stop. The suitable target VE may be extracted from the SA.

It is expected that such a use of the Experience Sharing component will be closely connected to a more Application based approach of Proactive Experience Sharing.

6.2.1.4. Privelets and Virtual IPs

Following work being developed on the Year 2 edition of Privelets, Experience Sharing will have to adapt especially in the use of Privelet filtering code on top of the Experience Sharing component. Additionally the possible use of Virtual IPs in the context of Privelets will also have to be taken into account.

6.3 Communication with other Components

Experience Sharing Functional Component collaborates with the following components of the COSMOS project:

- **Planner FC (WP5):** the Planner of the VE passes the problem to its Experience Sharing Component, whereas the Planner of the friend VE passes the solution to its Experience Sharing Component;
- **Social Monitoring FC (WP5):** when the Experience Sharing Component of the friend VE sends back a solution, a relevant notification is given to its Social Monitoring Component.

All these interactions are shown in the following sequence diagram (Figure 22):

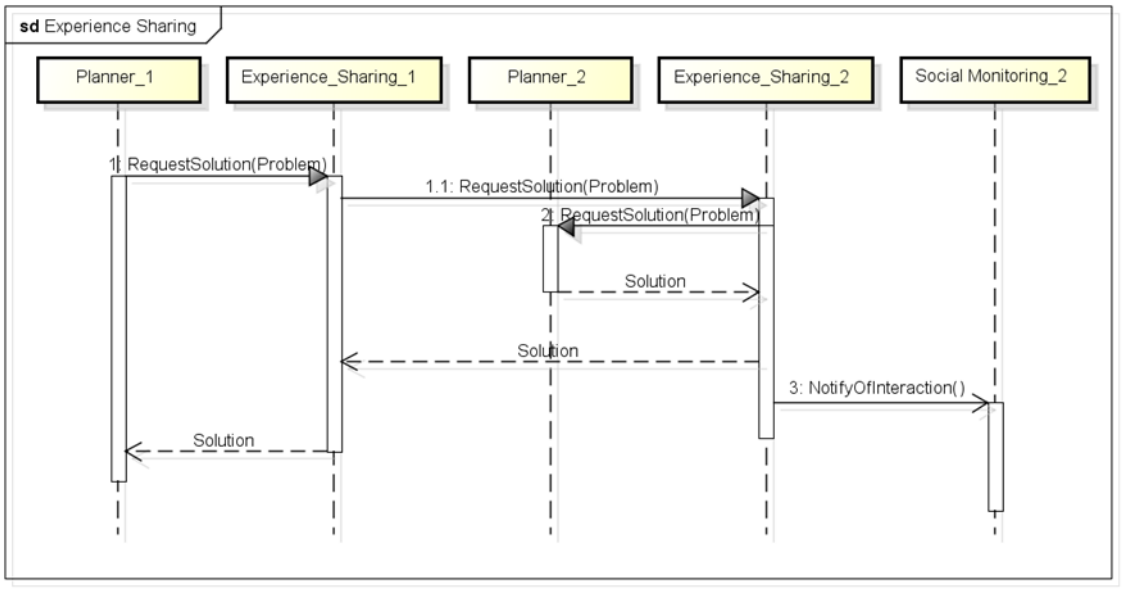


Figure 22: Experience Sharing Sequence Diagram

Additionally the Experience Sharing component:

- will run alongside the Privelet FC (WP3) and when a connection is established, if a great incoming load is detected then the component;
- contacts the Social Monitoring FC (WP5) for retrieving the ranking of the VE making a request

If the flow involves an incoming event to be propagated:

- the Planner FC will send the event to the Experience Sharing component as needed
- the Experience Sharing FC will propagate the event to the Message Bus FC (WP4) or initiate VE2VE communication

The following Figure 23 demonstrates the interactions between the components involving the Experience Sharing process:

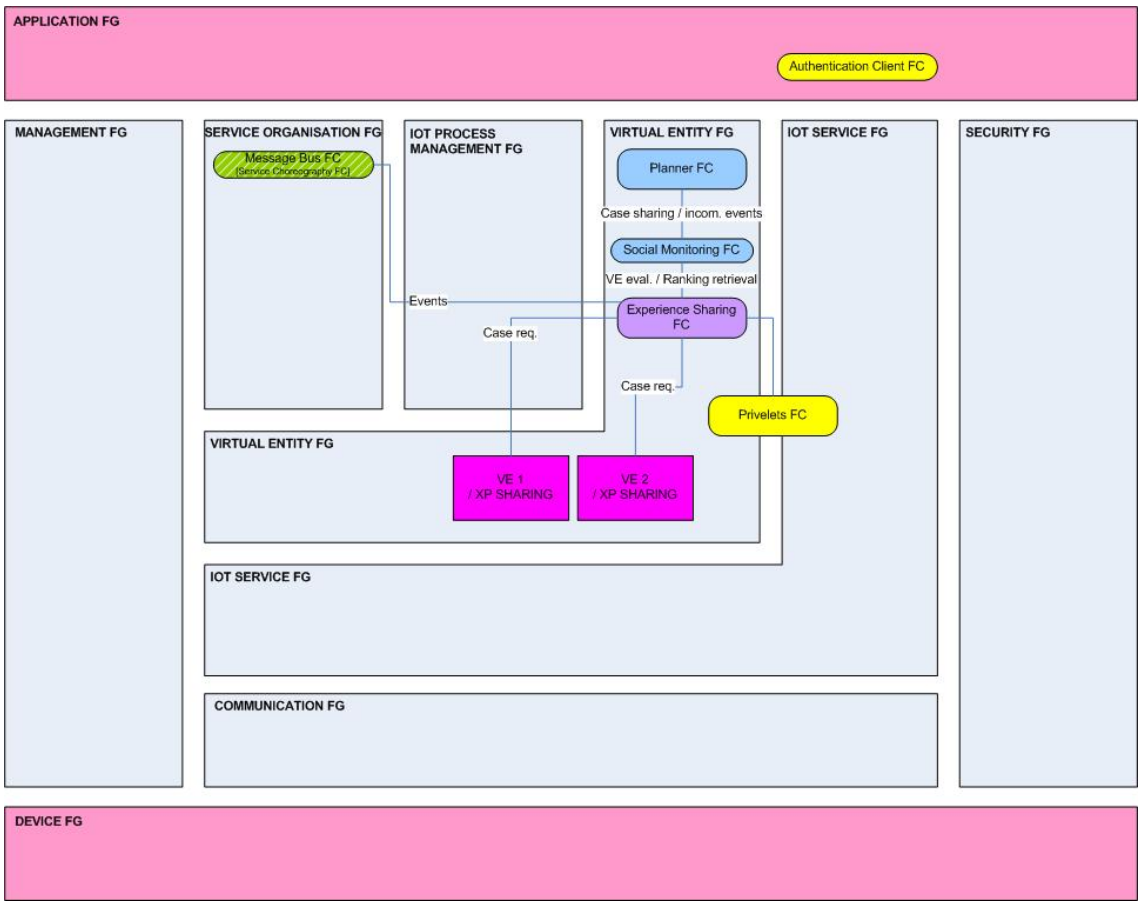


Figure 23: Functional Component interactions with Experience Sharing

6.4 API / Interfaces for Experience Sharing

The main goal of this Functional Component is to enable VEs to act in a more autonomous way, by sharing their Experiences which are related to the specific events that are detected. Experience Sharing, as functionality, is implemented in three steps, as it is shown below:

- **Storing Experience:**
  Through the use of the semantic store API, VEs can modify the contents of their semantic description. Regarding Experiences (Cases), it is very important that they can establish a persistent storage of their personal Case base, which contains all their Experience. This happens by creating new instances of Problems and/or Solutions or by adding connections to already existing ones;
  VEs are exposed as web services and are semantically described into the semantic store of COSMOS. The semantic description of the VE, contains not only the description of the underlying IoT-Services, of the VE capabilities and constraints but includes also references to any prediction models or experiences which it uses or has created;
- **Finding Experience**
  The next important step in the implementation of Experience Sharing is the actual access of the semantic storage in order to find and return a certain solution that

satisfies a problem. Such forays into the semantic store are made through the use of SPARQL and the API's Query Functions.
The API of the semantic store, dedicated to the Experience retrieval, will include specialized methods providing as input parameters the Experience search criteria.
A VE could easily create a query string, which would also be fully configurable based on information stored locally in VE variables. Therefore, any access of the VE in the semantic store can be accomplished and the actual Experience Sharing is made easier, due to the fact that Experience instances can be accessed and most importantly assessed by any VE.
Jetty server is used to enable VE2VE communication through HTTP, including finding and sharing Experience, by creating a servlet for each exposed IoT-Service that is accessible by a URL;

- **Choosing Experience**
  Having already described the process by which an Experience can be accessed, it is also important to mention that any retrieved information (including Experience), can also be assessed, ranked, etc., giving VEs the capability to choose one experience of those offered by other VEs. By defining a variety of social properties such as Trust & Reputation index, we enable VEs to check whether a Solution answers not only their queries but also satisfies some quality criteria. This process is executed by the Planner of the VE based on how many times a VE has shared its cases or how many times its IoT-services have been used;
- **Privelet modifications**:
  The API of the Experience Sharing Functional Component will not be modified greatly to accommodate changes and enhancements made by other components such as Privelets. One modification is the adding of new fields in the HTTP communication to accommodate for the inclusion of private keys between VE2VE communications participants. These keys are utilized by the VEs in order to authenticate and secure their transmissions;
- **Model Sharing**:
  The possible future inclusion of Models as Shareable Experience will also necessitate a further development of the API, taking into consideration the semantic description of a Model Developer uploaded Model. Such a description will be used as input to the component along with target VE address/port combination as is the case with current Experience. After that the remote VE will receive the request for Experience and initiate similar steps as those of the Case sharing mechanism, which are currently in effect;
- **Proactive Sharing**:
  Such an enhancement, will necessitate the further modification of the initiator method, in that it might be possible to differentiate between VE2VE and VE2MessageBus connection. Again the structure of events though which is going to emulate that of Cases, because of the CBR versatility, will keep the main forms of HTTP communication consistent with existing implementations. Possible publishing of events on the MB will be a new addition which will require description of the topic publishing to, or creating.

As it is explained above, the Experience Sharing FC is activated when an event (simple or complex) is detected and if the VE has no Solution corresponding with this Problem.

The VE asks its friends VEs for a Solution to the Problem, using SPARQL queries over HTTP. The interface of the Experience Sharing FC should require the following input parameters:

- The problem that occurred (e.g. “fire in the bus”);
- The IP address of the friend VE that is going to provide its solution;
- The port on which the server of the friend VE is running;
- The mapping of the servlet (e.g. “request/Experience”).

The component returns the solution of the friend VE, in case it exists.
Additionally depending on the type of further use of the component, the input that might be needed includes:

- Semantic description of Models in the case of Model sharing;
- Keys for the Privelet integration;
- Semantic description of topics and events for propagation of events.

6.5 Use Cases

6.5.1. Implementation of Experience Sharing during Year 1

Experience Sharing was used together with the Planner, in order to find Solutions to Problems that are not available locally, by initiating communication with the remote Experience Sharing components. After the retrieval of a Case the Experience Sharing component would call on the Social Monitoring to receive the outcome of the returned Case feedback and update Social Indexes accordingly. The Use-case with which development took place was that of Camden.

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

- the desired temperature;
- the time needed before (s)he arrives to the flat.

This is a problem regarding energy management where the desired temperature must be achieved right before the owner arrives to the flat.

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

This scenario was presented as the joint WP5 (Planner/Social Monitoring)-WP6 (Experience Sharing) demo at the Brussels Review.

6.5.2. Use Case for Year 2 implementation

The approach which is to be taken in developing the Y2 Experience Sharing component will include the scenario 4.6 as was described in D7.1.2 [36].

The End-user plans a program, stating which is the desired temperature value for his/her flat for specific time intervals of the day/week etc.

Because of COSMOS, the flat has a knowledge base of past programs that can be used and thus it is able to estimate

- how the actuator should be used to achieve the best possible consumption (may not be optimal) and
- the needed budget.

Even if the flat does not have a good program in its knowledge base, the flat will be able to ask other similar flats for help.

Thus through socializing the VEs, the End-user does not have to act in an optimal way but just states his/her desires. Moreover, a prediction regarding the total budget needed for a schedule is provided by COSMOS from the very beginning.

The required End-user inputs are the desired temperature, the time periods for which the schedule must be in effect (these can be discontinuous) and the desired budget the End-user is willing to commit.

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

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

After the chronological series of Problems creation the Application begins making use of the CBR technique by aiming at the retrieval of similar Cases from the local Case Base, by using the Planner component capabilities of the VE. Aiding this process is the use of the **Experience Sharing** component, which is activated on the condition that the local Case Base (CB) has no suitable Cases for Solution retrieval.

The Experience Sharing component will make use of load monitoring as described previously and will make use of Privelets in the VE2VE communication.

A Solution has as properties:

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

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

6.6 Conclusions and Future Work

Building on the success of the Experience Sharing component during Year 1, we approach the Year 2 Experience Sharing development in the sense of enhancing the robustness and cover of the mechanism. By better refining its process and flow with load monitoring and controls and using possible work implemented in the Privelets component, we aim to build a more readily integrated mechanism. Such a direction will aid the eventual development of additional COSMOS Applications which may take advantage of future year semantic description of Models for Model/Experience Sharing as previously described or merely use the Proactive Experience Sharing which we aim to accomplish in future along with the Situational Awareness Functional Component.

7 Conclusions

As the Work Package name suggests, the aim of our work is to make “things” more reliable and smarter. The reliability of the system is induced by utilizing historical data in order to make prediction models which can be used for decision making in real world dynamic scenarios with incomplete data. Different Machine Learning and statistical methods were explored for inferring high-level knowledge from raw IoT in order to contribute towards more situation aware and autonomous systems.

Over the years, as the technology evolves, the amount of data at our disposal has also increased rapidly. And the availability of such diverse type of data enables us to induce intelligence in the different real world applications. The use case scenarios of London and Madrid provide us vast amount of Data which will be exploited in the COSMOS. Such a large amount of raw data has to be processed, correlated and synthesized in order to extract high level information in real time, in order to help in decision making. In year two, we proposed different solutions based on machine learning and CEP to infer knowledge and Situational Awareness from this data and demonstrated our approach using different use case scenarios. Finally, we have explored how experience of Virtual Entities can be used to make things more autonomous and to take decisions in new situations. Additionally, an initial mechanism of experience sharing is introduced. This version of the deliverable goes further in all topics extending the state of the art and features next steps in individual conclusion sections of each Functional Component described in the previous section.

8 References

[1] C. COSMOS, "COSMOS Project D2.2.2 State of the Art Analysis and Requirements Definition," .

[2] G. Marfia, M. Roccetti and A. Amoroso. A new traffic congestion prediction model for advanced traveler information and management systems. *Wireless Communications and Mobile Computing 13(3),* pp. 266-276. 2013.

[3] T. Salsbury, P. Mhaskar and S. J. Qin. Predictive control methods to improve energy efficiency and reduce demand in buildings. *Comput. Chem. Eng. 51*pp. 77-85. 2013.

[4] M. Limayem and C. M. Cheung. Understanding information systems continuance: The case of internet-based learning technologies. *Information & Management 45(4),* pp. 227-232. 2008.

[5] J. Provost. Naıve-bayes vs. rule-learning in classification of email. *University of Texas at Austin* 1999.

[6] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu and S. Y. Philip. Top 10 algorithms in data mining. *Knowledge and Information Systems 14(1),* pp. 1-37. 2008.

[7] A. Ben-Hur and J. Weston. "A user's guide to support vector machines," in *Data Mining Techniques for the Life Sciences*Anonymous 2010, .

[8] D. Tian, X. Zhao and Z. Shi. "Support vector machine with mixture of kernels for image classification," in *Intelligent Information Processing VI*Anonymous 2012, .

[9] G. Shakhnarovich, P. Indyk and T. Darrell. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice* 2006.

[10] H. Yang, L. Chan and I. King. "Support vector machine regression for volatile stock market prediction," in *Intelligent Data Engineering and Automated Learning—IDEAL 2002*Anonymous 2002, .

[11] D. C. Sansom, T. Downs and T. K. Saha. Evaluation of support vector machine based forecasting tool in electricity price forecasting for australian national electricity market participants. *J. Electr. Electron. Eng. Aust. 22(3),* pp. 227-234. 2003.

[12] A. Ding, X. Zhao and L. Jiao. Traffic flow time series prediction based on statistics learning theory. Presented at Intelligent Transportation Systems, 2002. Proceedings. the IEEE 5th International Conference On. 2002, .

[13] C. Wu, J. Ho and D. Lee. Travel-time prediction with support vector regression. *Intelligent Transportation Systems, IEEE Transactions On 5(4),* pp. 276-281. 2004.

[14] P. Ni, C. Zhang and Y. Ji. A hybrid method for short-term sensor data forecasting in internet of things. Presented at Fuzzy Systems and Knowledge Discovery (FSKD), 2014 11th International Conference On. 2014, .

[15] F. Ganz, P. Barnaghi and F. Carrez. Automated semantic knowledge acquisition from sensor data.

[16] S. Kamijo, Y. Matsushita, K. Ikeuchi and M. Sakauchi. Traffic monitoring and accident detection at intersections. *Intelligent Transportation Systems, IEEE Transactions On 1(2),* pp. 108-118. 2000.

[17] Y. Kallberg, U. Oppermann and B. Persson. Classification of the short-chain dehydrogenase/reductase superfamily using hidden markov models. *FEBS Journal 277(10),* pp. 2375-2386. 2010.

[18] G. Yu, J. Hu, C. Zhang, L. Zhuang and J. Song. Short-term traffic flow forecasting based on markov chain model. Presented at Intelligent Vehicles Symposium, 2003. Proceedings. IEEE. 2003, .

[19] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun ACM 51(1),* pp. 107-113. 2008.

[20] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica. Spark: Cluster computing with working sets. Presented at Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing. 2010, .

[21] S. Rabinovici-Cohen, E. Henis, J. Marberg and K. Nagin. *Storlet Engine: Performing Computations in Cloud Storage* .

[22] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun ACM 51(1),* pp. 107-113. 2008.

[23] 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.

[24] D. M. Tax and R. P. Duin. *Feature Scaling in Support Vector Data Descriptions* 2000.

[25] I. Jolliffe. *Principal Component Analysis* 2005.

[26] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A k-means clustering algorithm. *Applied Statistics* pp. 100-108. 1979.

[27] N. Shental, A. Bar-Hillel, T. Hertz and D. Weinshall. Computing gaussian mixture models with EM using equivalence constraints. *Advances in Neural Information Processing Systems 16(8),* pp. 465-472. 2004.

[28] C. COSMOS, "D4.1.2 - Information and Data Lifecycle Management ," .

[29] M. R. Endsley, T. C. Farley, W. M. Jones, A. H. Midkiff and R. J. Hansman. *Situation Awareness Information Requirements for Commercial Airline Pilots* 1998.

[30] C. COSMOS, "COSMOS D 2.3.2: Conceptual Model and Reference Architecture," .

[31] A. Margarra, G. Cugola and G. Tamburrelli, "Learning from the past: Automated rule generation for complex event processing ," in *DEBS`2014,* Mumbai, India, 2014, .

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

[33] http://www.ibm.com/developerworks/library/j-jtp11234/

[34] Fehr, Ernst; Simon Gächter (Summer 2000). "Fairness and Retaliation: The Economics of Reciprocity". Journal of Economic Perspectives 14 (3): 159–181. doi:10.1257/jep.14.3.159. ISSN 0895-3309.

[35] http://jmeter.apache.org/

[36] C. COSMOS, "COSMOS Project D7.1.2: Use Cases Scenarios Definition and Design," .