Cosmos

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement Nº 609043

D6.1.1 Reliable and Smart Network of Things: Design and Open Specification (initial)

WP6: Reliable and Smart Network of Things

Version: 1.0
Due Date: April 2014
Delivery Date: 9th May 2014
Nature: Report
Dissemination Level: PUBLIC
Lead partner: 5 (University of Surrey)
Authors: F. Carrez (editor) & A. Akbar (Unis), A. Marinakis (NTUA), B. Tarnauca (Siemens), J. Krempasky (Atos)
Internal reviewers: EMT / Siemens

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	TOC + 6.1 content
V0.2		Previous + Achilleas Marinakis	UNIS, NTUA	6.1 and 6.3 content
V0.3		Jozef Krempasky	UNIS, NTUA, ATOS	6.3 + Intro/Concl + All sections update
V0.4		ALL	ALL	
V0.5	08/05/2014	Previous + Jozef. Krempasky + Adnan Akbar	ATOS, UNIS	Chap 3 & Chap2 updates
V0.6	08/05/2014	ALL	ALL	Fine-tuning
vFINAL	12/05/2014	F. Carrez	UNIS	Last check & Delivery to COSMOS management

Annexes:

Nº	File Name	Title

Table of Contents

Table of Figures

Table of Acronyms

Acronym	Meaning
API	Application Programming Interface
CEP	Complex Event Processing
CRUD	Create/Read/Update/Delete
EM	Expectation Maximization
GPS	Global Positioning System
HTTP	Hyper-Text Transfer Protocol
IGR	Information Gain Ratio
IoT	Internet of Things
IP	Internet Protocol
MaL	Maximum Likelihood
MAP	Maximum A Posteriori
MAPE-K	Monitor/Analyse/Plan/Execute - Knowledge
ML	Machine Learning
MLP	Multi-Layer Perceptron
OWL	Ontology Web Language
PA	Predictive Analysis
PMML	Predictive Model Markup Language
RDF	Resource Description Framework
SA	Situation Awareness
SPARQL	SPARQL Protocol and RDF Query Language (Recursive acronym)
SQL	Simple Query Language
URL	Unified Resource Locator
VE	Virtual Entity

1 Introduction

The main objective of this work package is to ensure the reliability of the COSMOS by providing continuous data and information access based on interpolation and extrapolation, to provide the means for situational awareness and understanding how things have behaved in comparable situations previously. Different methods for time series analysis of historical data will be explored to build a suitable model which can be used for interpolation and extrapolation purposes. 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 exploit Machine Learning techniques in conjunction with *Complex Event Processing* (CEP) to provide adaptive solutions for dynamic scenarios to optimize the performance of CEP for situational awareness. Finally different methods for experience sharing between virtual entities will be investigated so that things can be made smarter and able to take 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.

However it is worth mentioning that this iteration of the deliverable (corresponding to Period 1 of the project) is more focusing on interpolation, extrapolation, prediction and situation awareness. Experience definition and experience sharing is still “work in progress” (the task started later on) and will be further elaborated in D6.1.2 due in nearly 12 months.

This document is intended to provide a broad approach to meet the objectives as discussed above. In the first year, we have tried to explore different existing and possible techniques offering room for research and novelty. In the second year, we will narrow down our approach to the most optimized and novel techniques.

The document is organized as follows: Chapter 2 is linked to task T6.1 objectives about improving the reliability of things by ensuring via different means continuous raw data flow and data prediction, Chapter 3 (relating to task T6.2) is focusing on situation awareness based on CEP techniques (further methods based on Machine Learning will be explored and proposed in Period 2 and 3), Chapter 4 (relating to task T6.3) explains the concept of Experience and Experience sharing. Finally Chapter 5 provides a conclusion and introduces the next steps.

2 Reliability of information

2.1 Volatility of information and how to overcome it

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 connected using less reliable wireless links. In order to prolong their battery life, information provided by these devices may be sporadic and less reliable. In real world dynamics, the connection with device can be temporary disconnected and sensor readings at certain time can be missed. There are different components in COSMOS framework which need raw data information on the run to make decisions in real time. Temporary withheld of a system or missing sensor readings will certainly affect the overall performance of the system.

The main idea to overcome this problem is to build prediction models for the devices using predictive analytics including *Machine Learning* (ML) techniques such as time series analysis of historical data and predict the missing readings on the basis of it. The same model can be used to extrapolate the readings and can be used as alternate data source in case of temporary disconnection from the devices.

Another usage scenario for the models created relates to the data validation, where read data is compared to the generated data using the model which acts as a “reference” value. Differences between the two values exceeding a predefined threshold could signal a faulty sensor or an untrusted data source.

2.1.1. Interpolation/Extrapolation

In the field of numerical analysis, Interpolation is defined as the method for constructing new data points within the range of a discrete set of known data points. In the context of Internet of Things, Interpolation can be taken as filling of missing time series readings of various sensors using different time series analysis methods. Whereas, extrapolation refers to predicting the values of sensors in case of temporary withheld of a system by extending the time series in future. Both are closely related to each other and hence explained together. For complex scenarios where the entity of interest is dependent on multiple features, Interpolation requires accurate modelling of a system. And that same model can be used to extend the time series to predict the readings in case of malfunctioning of any sensor. The major task will be the modelling of a system using historical data. One possibility involves time series analysis on historical data, identifying the features, building a model and then optimizing the model. Different machine learning techniques will be used for making a model and verifying it. In the first year of project, we intend to implement different state of the art methods for building a model and then compare their performance after applying on our use case scenarios. There are different techniques available in literature but most of them employ offline methods. We will thus address one of the COSMOS framework requirements, which is to predict the missing value in real time.

Cosmos will explore the means of building and integrating into applications different interpolation/extrapolation mechanisms which are addressing different usage scenarios and would also provide users the component template for integrating interpolation models.

COSMOS will provide the mechanisms to facilitate the building of off-line models by processing the input data (before persisting it for later use by the ML algorithms), and by providing the

means to annotate and retrieve models to be re-used by other *Virtual Entities* (VE) or applications.

Another aspect which COSMOS will explore is related to the building of models "on the fly" by using on-line machine learning techniques. This will not only be used for creating models from scratch but also for updating models based on newly acquired data.

2.1.2. State Estimation/Prediction

Estimation methods are based on laws of probability and derived from Control Theory. Estimation is used to compute a process state vector from a measurement vector or a sequence of measurement vectors [1]. It is used to compute or estimate the state of a system at particular instant. In the context of Internet of Things, State Estimation Methods will be used when we want to estimate or predict the state of a system using current or previous inputs.

One of the most popular techniques used for state estimation in real time systems is based on Kalman Filter. The Kalman filter operates recursively on streams of noisy input data to produce a statistically optimal estimate of the underlying system state. In Internet of Things domain, Kalman filters can be used for instance in source localization and tracking applications [2, 3] . The algorithm works in two steps, prediction step and updating step. In a former step, it produces the estimates of the current state variables with uncertainties associated with the estimated values. In the later step, these estimates are updated using weighted average, with more weight given to estimates with high certainty.

One technique used extensively in literature for state estimation is based on *Maximum Likelihood* (MaL). According to Brown et al. 1992 [4], state estimation methods based on likelihood are more suitable when the system is not the outcome of a random variable. In other words, we know the likelihood or probability density functions of different possible states.

Few examples of the distributed maximum likelihood estimators used are Decentralized *Expectation Maximization* (EM) algorithm [5] and the Local Maximum Likelihood Estimator [6] that relax the requirement of sharing all data as nodes computes local unbiased estimates that converges towards the global maximum likelihood situation.

Maximum Likelihood Estimator is not viable option if the state of a system to be measured is the outcome of a Random Variable. In such cases, *Maximum A Posteriori* (MAP) Estimator which is based on Bayesian theory is better option. The difference between MaL and MAP is that MaL assumes that state *x* is a fixed through unknown point of the parameter space, while MAP takes *x* as the outcome of a random variable with prior PDF known. In [7] MAP estimator is used to find the positions of mobile robots in a known environment and track the positions of autonomously moving objects.

In systems, where the states are fixed and are represented by deterministic expressions, MAP and MaL cannot be used. In such scenarios, one suitable technique is called Least Square Estimation Method. Least Square Method is a mathematical optimization technique that minimizes the difference between observed and predicted values of a state. The least square method searches for the value of *x* that minimizes the sum of the squared errors between actual and predicted observations.

2.2 Use cases for Interpolation and Extrapolation

2.2.1. Madrid Use Case:

In Madrid use case, buses are the main source of data and they are publishing their GPS data after regular intervals to the server. The application of Interpolation/extrapolation block will be to predict the location and velocity of bus at any given time instant.

The first step is to identify different features on which bus location and velocity will be depending. The main features are

1) Initial position and velocity
2) Road conditions
3) Traffic conditions
4) Weather conditions
5) Traffic Signals
6) Bus Stops Info

These features cannot be limited to the above mentioned features because in real world dynamics, there will be many more features effecting the bus route. In order to elaborate it, consider a simple example below.

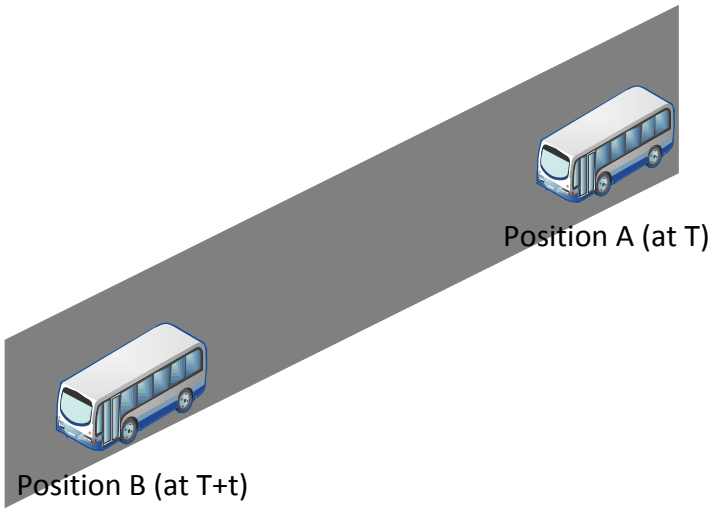

Figure 1 Madrid Bus Scenario on a straight road

Let us consider a simple scenario, where initial position of bus is A at time instant T. Now at any time instant (T+t), we miss the GPS reading from the bus. Different components within COSMOS which need this information will call Interpolation/extrapolation block in order to predict the location and velocity of bus A at time (T+t). In this scenario, missing value will depend on the initial position and velocity, traffic conditions on road and geographical features of road (curves, bending etc).

But the previous example is rather simple example of GPS data interpolation/extrapolation. Let us consider another scenario as shown in figure 2 below.

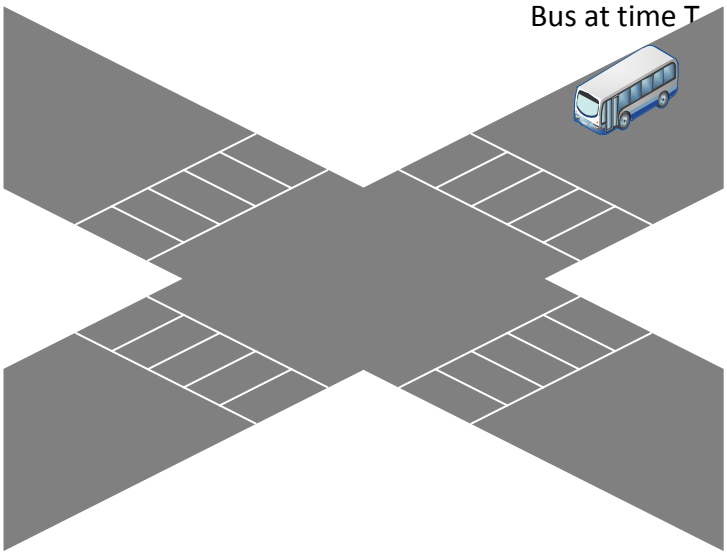

Figure 2 Madrid Bus Scenario approaching a junction

In this example, bus is approaching a junction at time instant T. Now if we miss the reading at certain time instant (T+t), the prediction model should incorporate several new features which were not present in first example. Traffic lights state and number of people crossing zebra crossing are few examples.

2.2.1.1. Proposed Approach: "Historical Data Analysis"

In historical data analysis method, a model is built using past values of a bus. It is quite common that traffic jams or slow traffic appears on a regular basis on certain parts of the route and at certain times of the day. After following bus routes for several days, you will observe certain patterns repeating every day. For example, you will observe much dense traffic density in the morning due to rush hour. In the same way, during summer vacations traffic density is comparatively less as compared to other days or time of the year. Hence such correlation of road conditions with respect to historical data is exploited in Historical Data Analysis.

- **Pre-processing of data:**

First step involves linearization of bus route. The route of a bus is divided into small segments assuming that route is linear for that small segment. And then the GPS position of a bus along with its velocity is mapped on one road segment using map matching algorithms. In this way, Data is gathered over several weeks to have an average value of buses velocities along every small road segment. Different steps involved are

1) ***Digital Maps:***

   For good and efficient use of GPS data, an accurate digital map is very important. GPS data from the vehicle does not match perfectly to the GPS coordinates of roads due to noise and imperfections in hardware. The route is divided into small segments of

roads represented by lines in digital maps. Such lines are joined by nodes with known GPS coordinates known. Such types of maps are called digital maps and are required for our application.

2) Coordinate transformation:

The GPS data from the vehicle cannot be directly used in digital maps. The digital map uses 2-D plane coordinate system whereas the GPS data is in geodetic coordinate system which is a 3-D system. In order to perform map matching, we need to transform the GPS coordinates into 2-D coordinate system according to requirements of digital map used.

3) Map matching algorithms:

The matching of observed GPS coordinates on digital map itself lies in research domain. It poses different problems depending on the orientation of road and the value observed. In literature, different map matching algorithms are discussed. We intend to explore it and use the best possible option in order to increase our system robustness.

- **Modelling:**

Average velocity within a particular road segment can be used to predict the position of a bus at any time. Three possible approaches for keeping average velocity values are

1) Aggregating all records with equal weights
2) Sliding window sampling Method
3) Weighted aggregating Method

We intend to explore all three possible approaches and then compare their performance to reach the most optimized method.

The instant velocity of a bus or the road conditions can differ from its previous average values in real time scenarios. In this regard, a novel approach based on Kalman filter along with historical data analysis is proposed for updating the model on run.

- **Kalman Filter:**

Kalman filter works in two steps. In the first step, it will predict the position of a bus on the basis of our model which was built using historical data analysis. In the second step, it observes the current position and updates its model accordingly by giving more weight to current observation. It works in a recursive way which opens the possibility for real-time solutions. One novel approach can be to build a prediction model using any state of the art technique and to implement Kalman filter on it for real time updating of model. The validation of such models in real-time environment will be explored. The general approach is shown in the diagram below.

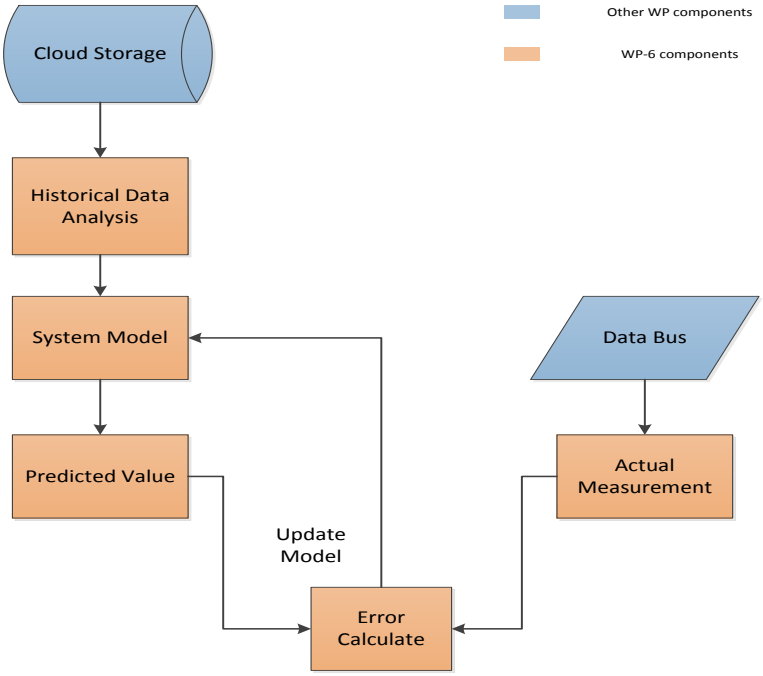

Figure 3: Proposed Approach for Model Updating using Kalman Filter

2.2.1.2. Alternate Approach using Artificial Neural Network

Another possible approach is to use *Multi-Layer Perceptron* (MLP) which is a feed forward artificial neural network. A MLP consists of multiple layers of nodes in a directed graph with each layer fully connected to the next one. It is an example of supervised learning technique called back propagation for training the network. Every node has a non-linear activation function which enables it to model non-linear relations. In our use case scenario, day of week, date, time, weather conditions, Bus number, route number, instant speed and location can all act as input to the system for training. Data can be divided into three parts named as training, cross validation and testing.

Training period for such a large data set can be significantly high. Validating the model on the run can also be a major issue in this approach. It does not consider underlying probability density function or probabilistic information about the variables under consideration that tend to decrease its accuracy in dynamic and uncertain situations. Due to these reasons, it is not the first approach to explore.

2.2.2. London Use Case

We can apply interpolation/extrapolation block in London use case scenario in different ways. One simple scenario can be if we are getting data from a temperature sensor located in particular flat and suddenly data becomes unavailable due to some fault. Different techniques related to Predictive analytics can be applied to predict the temperature at that instant. We can use regression analysis using previous data to predict the missing value. In the first step, we will identify temperature dependent variables. Time, day of week, month will all be considered in model to make it accurate. Another concept related to spatial interpolation may also be applied in this context. For example we know the temperature reading of the nearby

sensors located in the same building or surrounding buildings. We can use the change in temperature between neighbors to predict the temperature reading of missing sensor reading.

2.3. Connections with other Components

The prediction block will be connected with the following three main components in order to create prediction models and later update them iteratively.

1) Cloud Storage
2) Data bus
3) Semantic store

It is connected with Cloud Storage in order to build prediction models using time series analysis of historical data. And as explained above, once a model is built, it will keep updating iteratively using real time measurements to update the model accordingly. For this purpose, it is connected with the Data Bus in order to access real-time data published by Virtual Entities. The interface with the Data Bus can also be used by other Virtual Entities which are interested in future values of other Virtual Entities by passing queries to the Prediction block. Overall information flow diagram is below.

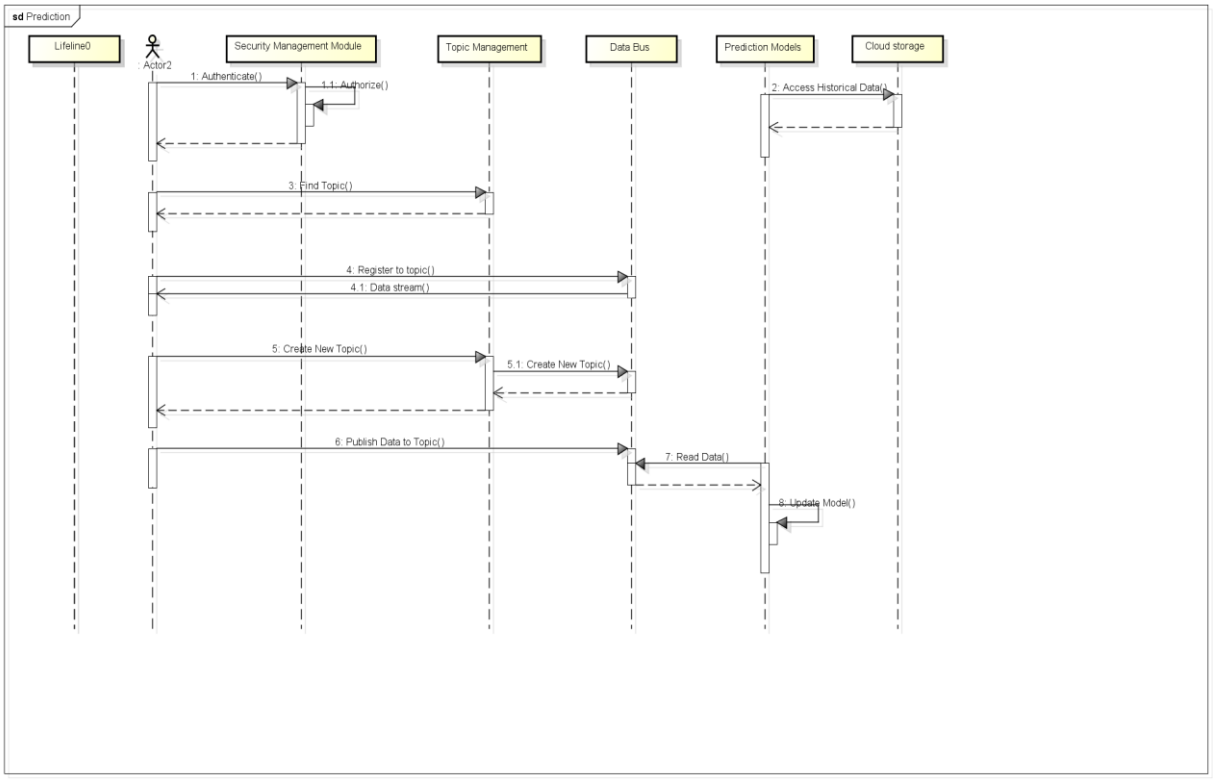

Figure 4: Information flow involving Prediction block

Once a model is created, either off-line or on-line, it has to be described and persisted in order to be later used by other VEs.

We propose to describe the models semantically so that they can be later retrieved or updated using new datasets. The description of the model should include the description of the attributes used to build the model, the description of the class, the application domain as well as the description of the algorithm and the parameters used in the model building process.

For the model persistence there a different formats which will be used. Whenever possible, the PMML format will be used since it provides a standard, platform independent representation. The advantage of this approach is that models build with one machine learning tool or library can later be used by other tool, or application using a different library which supports the PMML format.

Nevertheless, some models cannot be represented using the PMML format, thus an alternative solution will be to use the serialization of the models. For instance, models built with Java based, open-source machine learning software suites such as Weka of Rapid Miner, can be serialized using standard Java serialization and the serialized object persisted for later use. Whenever these models are needed, their serialized representation will be retrieved from the storage location and deserialized into the application.

Therefore, in order to use already existing models, persisted into the Cloud Storage or into other location, these models will have to be semantically described and the semantic description would have to include a reference to the location of the model.

Developers, VEs or applications will be able to query the semantic store for models based on multiple criteria, depending on their scenario requirements. Once a model meeting those requirements is found it can be retrieved from the URI included into its semantic description and used accordingly.

2.4. Interfaces

The application developer either develops a VE or uses one already provided in order to retrieve temperature data from a particular location. The data will be retrieved either directly from the VE (via an advertised IoT Service) or from a topic should the VE decide to publish it directly to the DataBus.

Since there are chances that this data source will not be always available the development of a prediction model has been decided. In order to achieve this, the application developer will use the COSMOS storage services in order to record historical data to be later used for the construction off-line of the 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). Once a model is created, it can be then

associated with a data source so that it can be used by VEs or applications if the data source fails to provide the necessary data. In this case, the application developer will rely on the Prediction block either to use the provided prediction model or to update it.

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.

Client/VE-Prediction Block API:

An API is required to connect the Client or VE to Prediction block for the following purposes

1) A Clinet or a VE will send a request using the API to register its interest in particular VE or topic stating the interested characteristics (temperature of a building, location of a bus etc.).
2) A Client or VE can also use API to get required prediction value. An example can be a client sending a query to prediction block to find the arrival time of the bus at particular location.

COSMOS components-Prediction Block API:

Different components might need to interact with the prediction block for predicting the missing instances of sensor readings. This API will be vital in ensuring reliability of a system.

2.5. Predictive Analysis and CEP

Predictive Analysis (PA) applies diverse disciplines as discussed above such as probability and statistics, machine learning and artificial intelligence for prediction in dynamic and uncertain environment. It employs a variety of methods and techniques from data mining and statistics that explore current and historical data to make predictions about future event. Whereas Complex Event Processing engines are optimized for large volumes of streaming events. It does not store the data, instead queries are stored and they are run on the data streams in real time. Complex Event Processing techniques detect an event after it has occurred. In many applications, prior prediction of an event is more useful then detecting an event when it occurs. And several situations require exact knowledge about the event. In these cases, PA cannot be used due to its uncertainty. Both fields have many overlapping concepts but yet there exist only few solutions where PA concepts have been applied to CEP. We intend to extend the Prediction Models build in this section to use in conjunction with CEP techniques discussed in next section to improve the performance of CEP and to extend current state of the art techniques for prediction of events.

2.6. Conclusion

In Year 1, the main focus was on exploring different techniques which will contribute to the main functionality of the architecture blocks. We started from literature review and reading about state of the art methods used in our context. The different techniques along with the possibility of novelties are being explored. We intend to narrow down the techniques from practical perspective in future. The handling of such large data and maintaining and updating

the models in near real time are the main Research issues which will be dealt in Year 2 and Year 3.

3 Situational knowledge acquisition and analysis

3.1 Decision making in complex environments

A COSMOS platform aims at providing support for the decision making for highly automated applications. Making a decision in complex dynamic environment is a very difficult task. The decision making processes are only as good as their information on a situation.

Situational Awareness (SA) (Figure 5) is being aware of what is happening in given context (complex, dynamic environment) in terms of the time and space. Besides various business use cases, SA also helps to identify potential critical threats such as threats to health and safety. A typical SA use cases for COSMOS are described in chapter 3.5.

There are many situations especially in IoT world where it is critical to make a correct decision based on proper evaluation of the current situation. Considering the fact that many different devices and sensors are deployed in remote places, this often means that enormous number of varying parameters and/or time critical conditions have to be considered. In these situations it is common that the situation evaluation is supported by the Complex Event Processor attached to multiple sources of various events carrying important information.

COSMOS aims at improving situational awareness beyond the current state of the art by enhancing CEP based situational assessment processes with adaptive feedback loop at runtime. This approach opens new possibilities for more precise and reliable situational awareness.

In addition to CEP techniques, machine learning techniques also play their specific role in contribution to situation awareness process.

Situational awareness is important to COSMOS components, COSMOS enabled applications as well as Virtual Entities need to be aware of their surroundings, opportunities and the potential hazards they face.

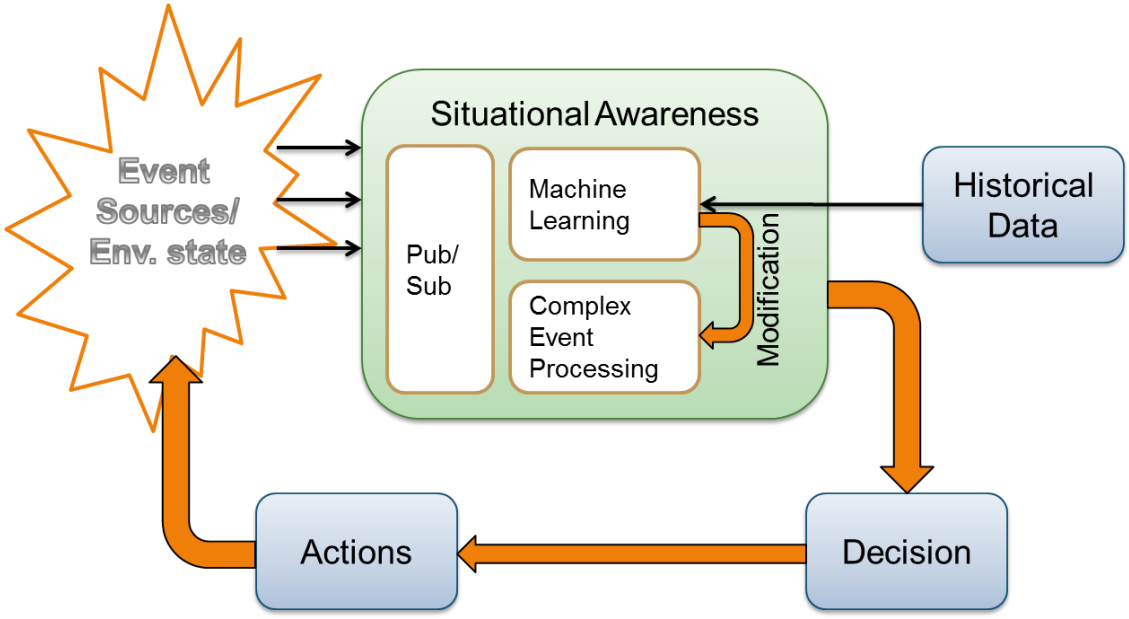

Figure 5: Situational Awareness

A high level functional composition of situational awareness is illustrated on figure 5. The reason that the machine learning block modifies CEP situation assessment at run-time is that it enables correction and fine-tuning of initial parameters based on analysis of historical data.

3.2 Context Modelling and Reasoning Techniques

A context is useful for representing and reasoning about a pre-defined/restricted state space within which a problem can be solved, and use it as the basis for information analysis and filtering.

In order to help ensuring that situational awareness acquisition subsystem meets expected application requirements, it will be possible to model information about behavior of virtual entities as well as information about surrounding environment. Using models (see Figure 6 below), it is also possible to describe patterns and relations between entities that are used through the model description.

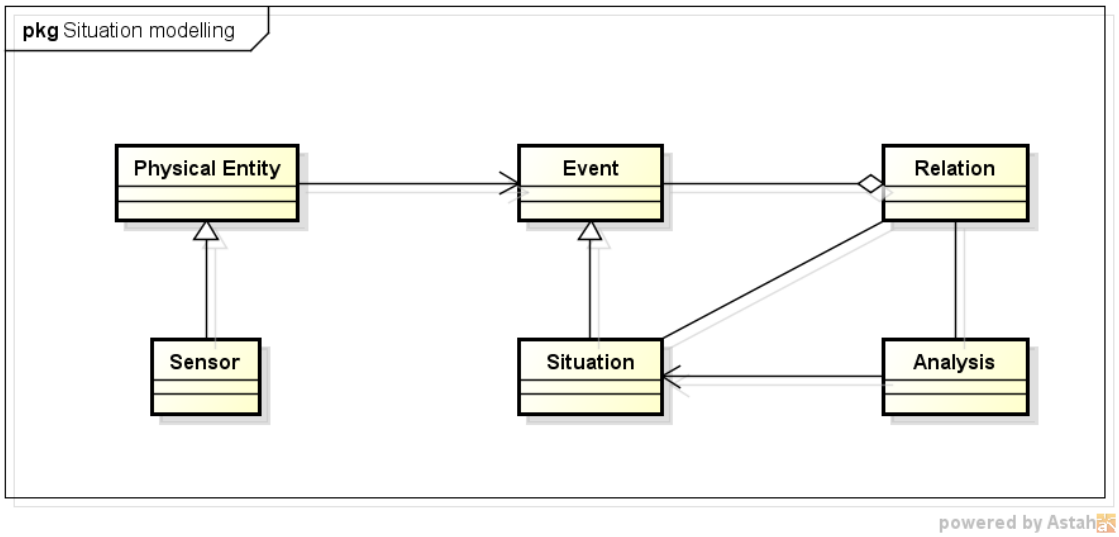

Figure 6: Context Modelling

The key elements in context modelling are entities, events and analysis. A Physical Entity represents an object in a situation which provides partial information about the current environment. A typical example can be a physical device i.e. sensor providing information about temperature, position, CO_2 production, velocity, energy consumption etc… An event is a data structure encapsulating all the relevant information about a particular situation. An event creation triggers the system to evaluate and analyze situations based on the relation between different events. This analysis may eventually lead to detection of more complex events and situations that can be utilized by decision processes.

3.2.1. Context Definition

For the year one, COSMOS will focus on defining situations, relations and analysis using the dedicated Domain specific language called “Dolce [8]” which is especially suitable for CEP technology. Therefore, a processing model provided by COSMOS is continuous. For next years,

an extended version of language which will also support machine learning technology will be introduced.

The Dolce [8] language aims at providing simple means to define events, event relations and event analysis in human readable form.

A typical event may be defined using following lexical grammar. The character @ indicates the use of custom keywords as identifiers, which is useful when matching events with defined rules.

Event Definition	**Rule Definition**

event @Event
{
use /*properties definition*/
{
[type] @Property
}
accept { @Property == [value] };
}
	complex @Situation
	{
	payload {[params]}
	detect @Event
	where [aggregate](@Event)
	in [window]
	}

3.2.2. Source Filtering

Business applications are only interested to only acquire a high level of intelligence from the available data with effective reasoning. Therefore, an effective analysis is in need to provide derived information through filtering and other techniques.

The filtering is one of most basic operations on event streams. Filtering evaluates a predefined logical or arithmetic condition based on event name or properties and/or constant values.

Supported filtering conditions consist of “equals”, “greater/less than”, etc., and logical conditions like “and”, “or”, “not” etc.

3.2.3. In-Memory Caching

In order to be able to perform analysis on subset of events within an event stream, received events have to be temporary cached in the memory. The actual in-memory caching is realized through moving windows technique. Windows with different properties produce different results and have radically different performance behaviors.

Cosmos will support following mechanisms for event selection:

3.2.3.1. Time Windows

A Time Windows are specified by time intervals. This is useful in many different situations where particular events or aggregated event values from same or possibly many different event streams fall within or exceed specified time period.

There are two different approaches for time window definition. The selection of proper approach depends on particular situation context.

- **Fixed Time Window** – buffers events for analysis every specified fixed time interval;

- **Sliding Time Window** – buffers events for analysis for specified time interval into the past based on the system time. For example, sliding window can be used to collect all readings during last hour.

3.2.3.2. Tuple Windows

A Tuple Windows are used to collect events from available sources based on number of occurrences. A typical example of tuple window is the collection of last twenty sensor readings for further evaluation.

The tuple window has a fixed size. Therefore its endpoints move together and events continuously expire with new events entering the tuple window.

New event carrying derived information is produced only when window closes i.e. reaches its defined capacity and when a new event arrives.

3.2.4. Aggregation

The aggregation facility continuously calculates various metrics across time and tuple windows and is used for assessment of trends in situations. Aggregations perform different computation over subset of data.

The supported aggregation operations over windows are as follows:

- Avg – average over a numeric event payload;
- Count – Count of events;
- Sum – Summation over a numeric event payload;
- Diff – Difference between two subsequent events;
- Min/Max – Minimum/Maximum over a numeric event payload.

3.2.5. Correlation

The event correlation functionality enables identification of the situation where different conditions occur or do not occur in a specific order. Therefore, the detection of situation depends on the occurrence of a series of events rather than on the occurrence of a single event. The events from multiple streams are correlated over period of time that may be seconds, hours or days.

3.2.6. Event Period

By default, events are transient. This means that events are valid for an infinite short time period. For assessment of some real world situations, it is useful to associate some events with fixed time period to define for how long the situation associated with event is valid.

The supported expressions of event period are as follows:

- Lasts – event is valid for specified period of time;
- Until - event is valid until specified event arrives.

3.2.7. Joins

The purpose of event joining is to create a new stream of events from several other streams based on some conditional matching.

A join between two data streams necessarily involves at least one window. To perform a join, it is usually necessary to wait for events on the corresponding stream or to perform aggregation on selected events.

3.3 Situational Assessment

COSMOS platform will continually evaluate given contextual models according to information received from the publish/subscribe data bus and notify listeners about identified situations.

A general functionality provided by SA functional block is:

- **Continuous assessment of actual situation at real-time –** detection of emergency situations, sudden or unexpected situations, failures and other anomalies, disasters, etc…
- **State and Behavioral situation assessment –** assessment of local situation related to Virtual Entities.
- **Environmental situation assessment –** assessment of environmental conditions in near vicinity of Virtual Entities.
- **Overall situation assessment -** assessment of group or overall situation based on all available information.

3.3.1. Protection Against Over-Aggressive Contextual Models

The Over-Aggressive Contextual Models are those that cannot possibly be evaluated nor evaluated using given resources.

An example of Over-Aggressive model is aggregation performed on all events received during the last day while rate of events is very high compared to available physical resources needed for caching.

There are two types of protection:

- Static analysis of contextual model – analysis is performed before actually evaluating models. This analysis performs checks such as required timing, storage and throughput checking.
- Runtime detection – analysis performed during evaluation phase. Contains several mechanisms for monitoring and ensuring that models are evaluated as expected using given resources.

3.3.2. Runtime Model Updating

The COSMOS project will provide enhanced situational awareness by combining predictive and/or cognitive analysis with runtime situation analysis (CEP). The main idea to improve perception of cosmos environment is to tweak runtime analysis by analysis of historical or predicted situations.

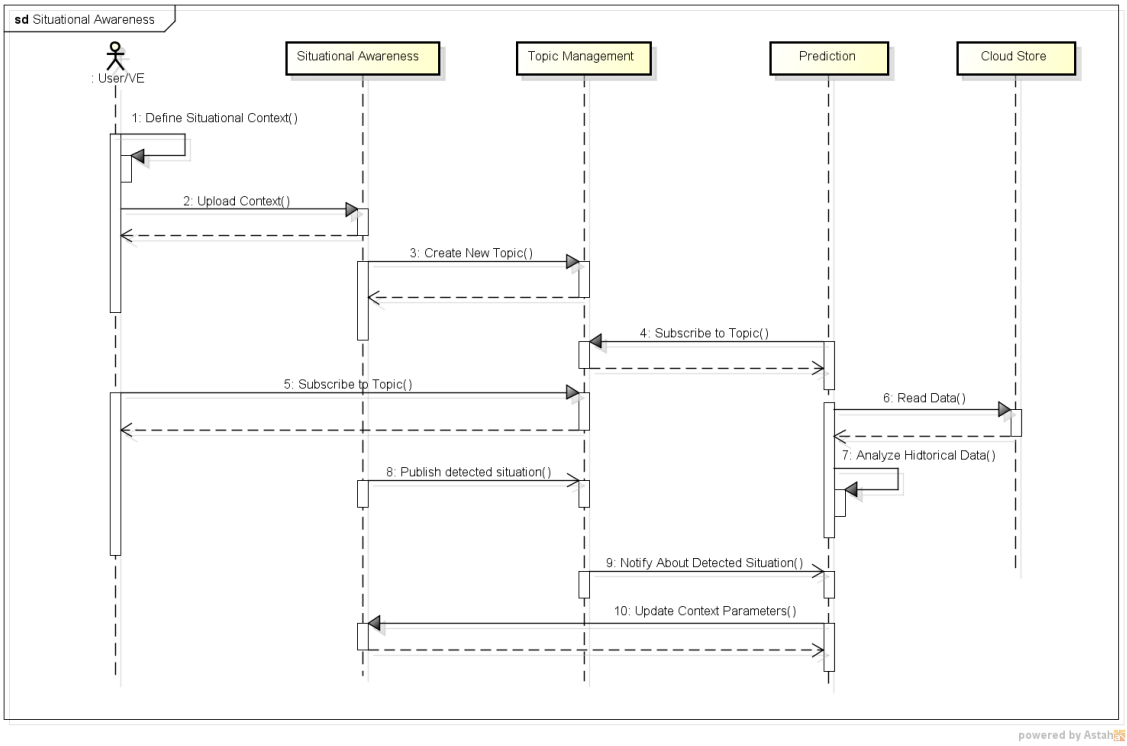

Figure 7: Runtime Model Updating Strategy

As depicted in Figure 5, a predictive analysis can be used in order to update or fine tune initial definition of situation analysis by analyzing historical data related to the same or similar situations.

3.4 Context Propagation

Depending on configuration, each type of detected situation can be published as an individual topic within semantic topic management functional block. Listeners can subscribe to particular topics and receive information about actual situation at real-time (as soon as the engine evaluates event for that topic), according to the contextual model analysis.

3.5 Use cases for Situational assessment

3.5.1. Madrid EMT Use Case

Existing Madrid EMT infrastructure provides possibility to build contextual models based on data streams such as bus position, velocity, schedule, CO_2 emission, as well as environmental information such as traffic light status, temperature, local traffic congestion etc. Typical examples of detected situation are: traffic jam, bus failures, traffic accidents, bus delays, etc.

3.5.1.1. Buses in queue detection scenario

In this scenario, the situational awareness feature will detect bus queues at Bus stops. This situational knowledge can be used by Cosmos for enhanced cruise control (for example eco-saving) as there will be no need to increase speed and thus to consume gas when there is a queue at a bus stop.

One feasible approach to detect bus queue is based on utilization of GPS position provided by busses.

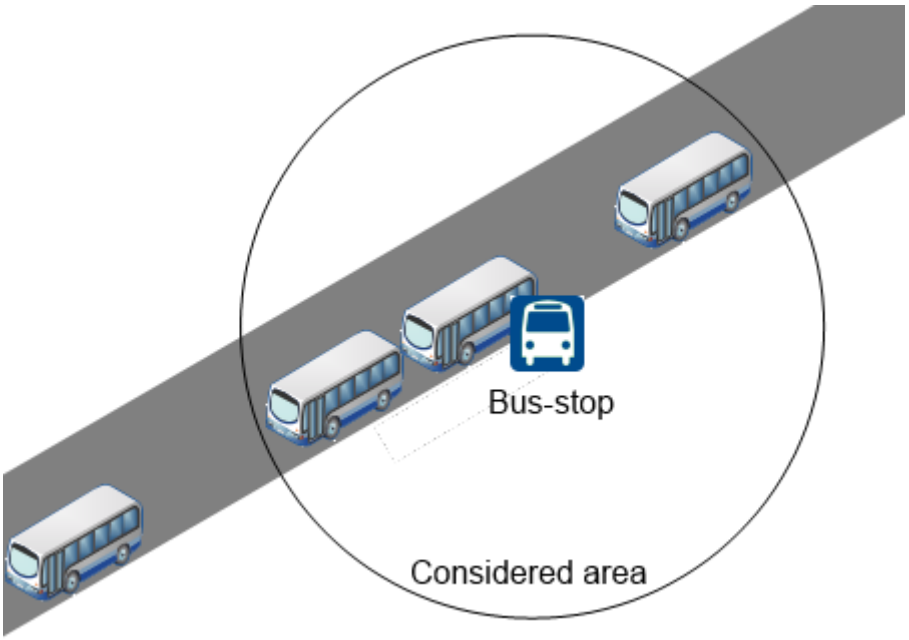

Figure 8: Queue detection

Definition and modeling of busses in queue situation:

As depicted in the Figure 8 above, we can model given situation by counting bus occurrences within specified area surrounding bus stop and time. Let's assume that for a particular bus stop, a bus queue can be identified when at least five different busses are present within a one hundred meters radius for more than one minute.

First step consists of definition of input events and corresponding attributes. An event representing bus within specified area can be defined using following Dolce [8] event construct:

Bus event definition

```
event BusNearStop
{
      use
      {
            int BusIdentity,
            pos Location,
            float Acceleration
      }
      accept( diff(Location, BUS_STOP_LOC) < 100 //meter )
}
```

A purpose of an accept statement is to select only those events for which a bus location is within specified proximity distance.

Finally, to define actual buses in queue situation, we need to specify counting of buses entering a bus stop area within specified time. This can be achieved by utilizing aggregate function "count" and a time window as shown in the following table:

Busses in Queue situation definition

```
complex BussesInQueue
{
    detect BusNearStop
    where count(BusNearStop) > 5
    in[1 minute]
}
```

3.5.2. Energy distribution use case

Heat and electricity consumption sensors as well as different environmental sensors are main sources of information that can be used for the situational assessment. For example, it is possible to build a contextual model that can be evaluated to detect at run-time when energy consumption of building can be reduced based on different continuous sensor readings within building itself and outside such as temperature, heating and lighting.

3.5. Automatic generation of Rules for CEP using ML

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 preliminary knowledge which sometimes is not available or not so precise. So 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. In real time dynamic scenarios, parameters of a system may change and performance of CEP may deteriorate in such dynamic scenarios.

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 [9], authors applied the Machine Learning 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 IGR principle and implemented each operation in different module. This is just one example where Machine Learning 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.

3.6. Interfaces

The situational awareness component will provide information in asynchronous manner. Therefore, a clients need to subscribe to detected situations based on their custom preference.

Virtual Entities may perform CRUD operations on context model at runtime. These operations are available via specialized REST API published by the situational awareness functional block.

A several different interfaces are mandatory for SA assessment:

1) A user needs to define different situational contexts so that machine can automatically evaluate and detect relevant situations. A context definition may come from VEs, Cosmos enabled applications or administrators. An enhanced declarative Dolce [8] language as well as new corresponding knowledge base can be provided.
2) New semantic API for registration of event sources over semantic topic management as well as direct P2P connection will be provided.
3) User may opt to also specify the customized source of detected events and register to high level events he is interested in.

3.7. Conclusion

Situational knowledge must change and update in timely manner with new input to the system. During the year 1, different event processing techniques and low level situation definition solution have been explored. These techniques provide tracking, monitoring, sensing and responding basis for the situational awareness.

Rule-based event processing techniques presented in this chapter are mandatory as they provide a core foundation for the situation awareness but are not enough for obtaining SA in dynamically changing conditions. Situation detection rules need to be continuously updated to handle dynamic environments. Therefore machine learning techniques combined with CEP will be explored in following years.

In addition to machine learning, in Year 2 and 3, semantic event processing will be explored. The goal is to achieve better understanding and automated detection of situations and relationships between incoming events by machine i.e. (semantic CEP extensions). This will

further enable real-time declarative processing of events, leveraged reaction to situations as well as enhanced declarative rule definition.

4 Experience and Experience Sharing

4.1 Experience in COSMOS

In the context of COSMOS, different meanings of experience can be defined, arising from the correlated phases of MAPE – K loop [10] approach, which is adopted for the implementation of the project regarding VEs' management:

- **Models building:** this type of experience is related to the **Analysis (A)** component, whose main functionalities are information process and events detection. Such kind of **models** are described in chapter 2 (prediction models), which could be built inside a VE in Year 2 and 3.
- **Cases:** this type of experience corresponds to the **Planning (P)** component, which is used in order to select the actions that need to be applied when a system reaches an undesired state. More precisely, a **case** can be considered as a combination between a problem and its solution, whereas a problem consists of one or more events. In other words, it is a kind of rule for an actuation plan, which is triggered when specific events are identified:

  Case = problem (event1, event2, event…) + solution

For Year 1, **cases** are examined, whereas **models** are going to be analyzed in Year 2 and 3.

The ontological graph that corresponds to the section of experience contains the class of **"Experience"** which is further divided into the subclasses of **"Models"** and **"Cases".** The **"Cases"** class is also further divided into the subclasses of **"Problems"** and **"Solutions"**.

A COSMOS specific ontology which will be created in order to facilitate the description of the key COSMOS building blocks. It will also include the necessary structures for describing the experience. This will also include the means to describe the cases.

The following image demonstrates that the instances (individuals) of the **"Virtual Entity"** class are connected to the inferred instances of the **"Experience"** class through the abstract property **"Has Experience"**. Specifically, in the COSMOS Ontology, described in subchapter 4.6 of D5.1.1, **Experience (case)** is linked with a **Virtual Entity** through the object-type property "**isIncludedBy**".

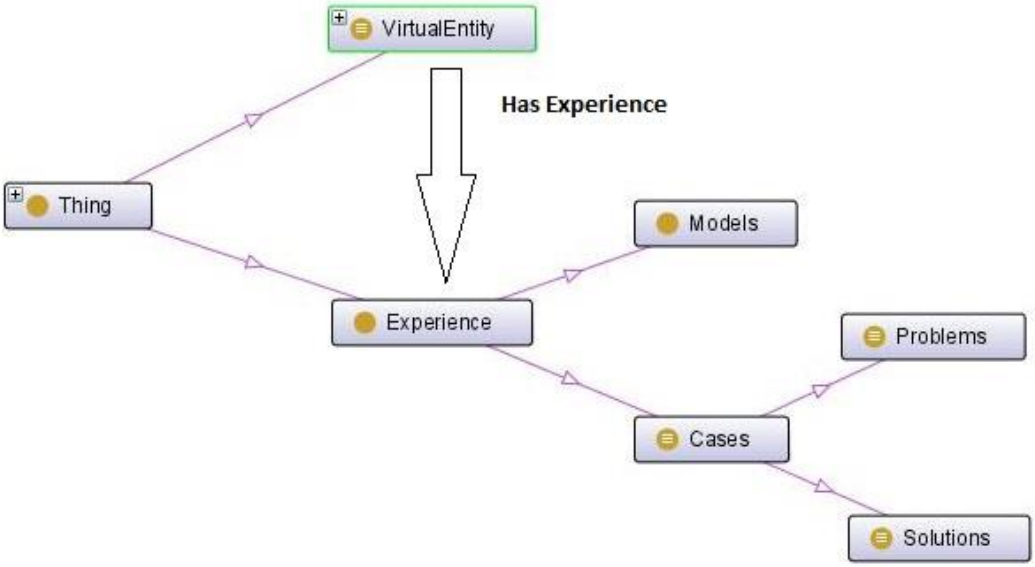

Figure 9: Experience in COSMOS

For the semantic description, an open-source triple store will be used such as OpenRDF Sesame [11] or Apache Jena [12] but other solutions will also be considered.

SPARQL [13] is a syntactically SQL-like language that can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middle-ware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. It also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs.

4.2 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 of the type cases, which is described in subchapter 4.1. 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. As described above, 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. Experience retrieval could also be performed using a generic query API where users would pass their own query string.
  While the former approach has the advantage of hiding the querying mechanism and providing a uniform access to the semantic store, the latter would provide much more flexibility, but would require user experience with SPARQL.
  In any case, 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 which 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.

4.3 Use cases for Experience Sharing

4.3.1. Madrid Use Case

Using the Madrid use case as an example, let us suppose that in a bus VE, the CEP engine detects the start of a fire, thereby issuing a warning for the **problem** (event and subclass of a **case)**, in the topic, which is called “fire in the bus”, in the message bus. The latter is referred in the subchapter 4.6 of the Deliverable 4.1.1. The VE has subscribed to the aforementioned topic and therefore receives the relevant notification. Then the following steps take place:

1. The bus VE looks at its own case base in order to find a **solution**, corresponding to this problem.

2. If there is no such solution, then the bus sends its problem to its friends, requesting their experience (solution). (finding experience)
3. If a friend bus has a similar problem in its case base then it sends back the solution which is related to the problem. (finding experience)
4. If a friend bus sends back a solution, then this interaction is tracked by the Social Monitoring Component (described in subchapter 4.3 of the Deliverable 5.1.1) and therefore the trust & reputation index of the friend bus is being increased.
5. If the bus receives two or more different solutions ("call fire-truck 1", "call fire-truck 2" and so on), then its Planner is activated in order to choose the appropriate one, based on some social characteristics, like trust & reputation index, which are stored in the COSMOS platform. The Planner is analytically described in the subchapter 4.1 of the Deliverable 5.1.1. (choosing experience)
6. If the bus has no friends or if none of its friends has a solution for the problem "fire in the bus", then the bus requests from the Registry & Discovery component (subchapter 4.5 of the Deliverable 5.1.1) to take recommendation for new friends. After that, the flow goes back to the step "2", whereas the recommendation phase is repeated until the VE gets a solution for its problem. (finding experience)
7. The VE stores the solution, in a way that is mentioned in the subchapter 4.2, in order to reuse it if fire happens again, or to share it with its friends. (storing experience)

In Year 1, the social characteristic, which is used from the Planner for "choosing experience", is mainly related to the number of times a VE has shared a case that is stored in its own case base. In year 3 a VE could be able to evaluate the usefulness of the experience that has received. In this case, the trust & reputation index could indicate not only how "popular" the VE is but also how efficient are the solutions that it shares.

4.3.2. London Use Case

In this use case, a building can be considered as a VE and let us assume that it contains in its own local case base, a problem called "overheating". Each building is supplied with temperature sensors which could measure, for example, the indoor temperature of all the rooms and flats, continuously. By processing all these measurements, the CEP engine could detect the event "overheating" that was mentioned above. Similarly with the Madrid use case, the VE should firstly search in its base to find the appropriate solution to the problem and if it is empty, then it could request its friends VEs for help. In this case a solution could be "deactivate the heating system for three hours".

4.4 Communication with other Components

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

- Planner (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 (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:

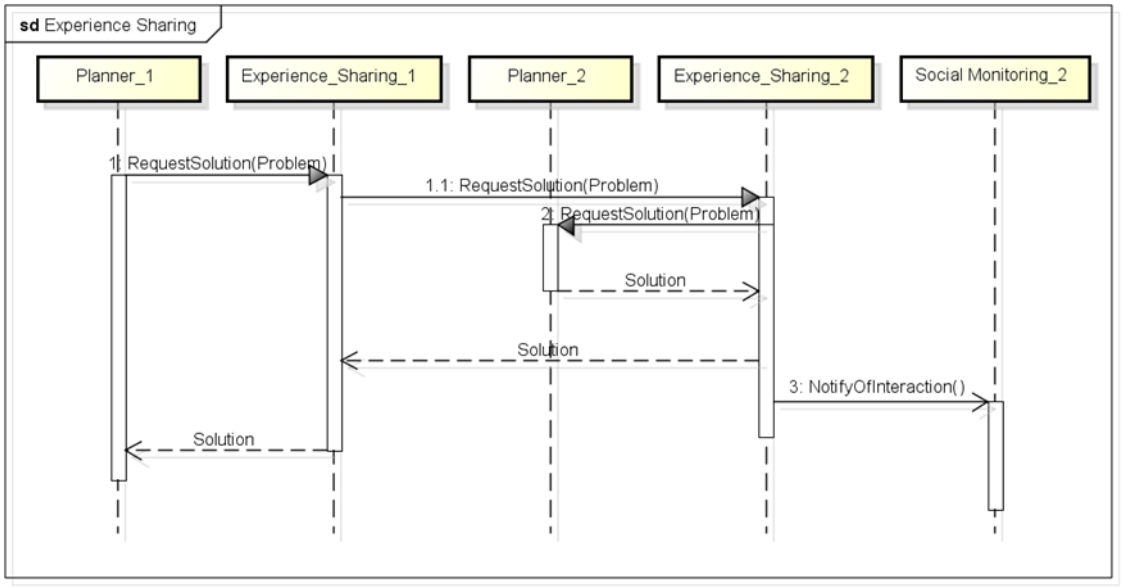

Figure 10: Experience Sharing Sequence Diagram

4.5 Interfaces for Experience Sharing

As it is explained above, the Experience Sharing block 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. However, due to security reasons and because of the fact that the mechanism should be friendly to the VE developer, who may not be familiar with SPARQL, the interface of the Experience Sharing Component should require the following input parameters:

- The problem that occurred (e.g. “fire in the bus” as it is mentioned in the subchapter 4.3.1)
- 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 this exists.

4.6 Conclusion

In Year 1, Experience, in the context of COSMOS, is defined as a case which consists of a problem and its corresponding solution. In Year 2 and 3, different meanings of Experience, like models, will be explored. Regarding the way a VE chooses the suitable Experience, from those offered from its friends, a simple Trust & Reputation index, which is related to the cooperativeness of a VE, is adopted. In Year 2 and/or 3 this index can be expanded, taking into account various interactions between VEs, like those that are analytically described in subchapter 4.3 of the Deliverable 5.1.1.

5 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 serve as an alternate data source in case of a discontinuity of data from the sensors. Different Machine Learning and time series analysis techniques were explored and later we narrowed down our approach to the most practical and adoptable methods for practical scenarios.

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 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 the year one, we focused on CEP solutions to infer knowledge and situational awareness from this data but in the later years, we intend to explore different Machine Learning algorithms as well. Finally in the fourth chapter, it is discussed how experience of Virtual Entities can be explored in order to make things more autonomous and to take decisions in new situations. Additionally, an initial mechanism of experience sharing is introduced.

Next iteration of this document will explore addition techniques for Interpolation/ Extrapolation and Prediction, and will propose a technology agnostic framework for Situation Awareness, meaning that other techniques will be explored (especially machine learning) in order to populate an abstract context that can be accessed by VE’s. As we said earlier the main focus for this topic in Year 1 was on CEP related techniques. Finally Year 2 version will also come with extended notion of experience and experience sharing in which we aim to give VEs the capability to assess the effectiveness and usefulness of the experience they have reused.

6 References

[1] B. R. Bracio, W. Horn and D. P. F. Moller. Sensor fusion in biomedical systems. *Proceedings of the 19th Annual International Conference of the Ieee Engineering in Medicine and Biology Society, Vol 19, Pts 1-6: Magnificent Milestones and Emerging Opportunities in Medical Engineering 19*pp. 1387-1390. 1997.

[2] T. Li, A. Ekpenyong and Y. Huang. Source localization and tracking using distributed a synchronous sensors. *IEEE Trans. Signal Process. 54(10),* pp. 3991-4003. 2006. DOI: 10.1109/TSP.2006.880213.

[3] H. Chen, P. Deng, Y. Xu and X. Li. *A Robust Location Algorithm with Biased Extended Kalman Filtering of TDOA Data for Wireless Sensor Networks* 2005.

[4] C. Brown, H. Durrant-Whyte, J. Leonard, B. Rao and B. Steer, "Distributed data fusion using kalman filtering: A robotics application in data fusion in robotics and machine intelligence," in San Diego, 1992.

[5] R. Nowak. Distributed EM algorithms for density estimation and clustering in sensor networks. *IEEE Trans. Signal Process. 51(8),* pp. 2245-2253. 2003. DOI: 10.1109/TSP.2003.814623.

[6] D. Blatt and A. Hero. *Distributed Maximum Likelihood Estimation for Sensor Networks* 2004.

[7] T. Schmitt, R. Hanek, M. Beetz, S. Buck and B. Radig. Cooperative probabilistic state estimation for vision-based autonomous mobile robots. *IEEE Trans. Rob. Autom. 18(5),* pp. 670-684. 2002. DOI: 10.1109/TRA.2002.804499.

[8] Dolce language specification https://forge.fi-ware.org/docman/view.php/11/1187/Dolce-Language-Spec-Release-v1.pdf

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

[10] “An architectural blueprint for autonomic computing." IBM, Autonomic Computing White Paper, June 2005, Third Edition

[11] Sesame http://www.openrdf.org/

[12] Apache Jena http://jena.apache.org/

[13] SPARQL Protocol And RDF Query Language http://www.w3.org/TR/rdf-sparql-query/