COSMOS

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement Nº 609043

D2.3.1 Conceptual Model and Reference Architecture

WP2 Requirements and Architecture

Version: 1.0
Due Date: 30/04/2014
Delivery Date: 02/05/2014
Nature: Report
Dissemination Level: PU

Lead partner: ICCS/NTUA
Authors: Spyridon Gogouvitis, Achilleas Marinakis, Orfefs Voutyras, Vassilis Psaltopoulos (NTUA), Francois Carrez, Adnan Akbar (Surrey), Jozef Krempasky (ATOS), Paula Ta-Shma (IBM), Bogdan Tarnauca, Leonard Pitu (Siemens)
Internal reviewers: Mónica Aguirre, Jozef Krempasky (ATOS)
Bogdan Tarnauca (Siemens)
Francois Carrez (Surrey)

www.iot-cosmos.eu

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

Version Control:

Version	Date	Author	Author's Organization	Changes
0.1	26/03/2014	Spyridon Gogouvitis, Achilleas Marinakis, Francois Carrez,	NTUA, Surrey	Migrating Internal architecture document including Domain Model.
0.2	2/4/2014	Spyridon Gogouvitis	NTUA	Change in name and minor updates
0.3	09/04/2014	Spyridon Gogouvitis	NTUA	Minor Updates and Tasks allocations
0.4	16/04/2014	Spyridon Gogouvitis, Vassilis Psaltopoulos, Francois Carrez, Jozef Krempasky, Achilleas Marinakis	NTUA, Surrey, ATOS	NTUA: Executive Summary, Introduction, System Capabilities, Actors and Use cases, Information View, Conclusions, Data Mapping component Surrey: Methodology, COSMOS Domain Model ATOS: Message Bus, Event Detection and Situational Awareness
0.5	25/04/2014	Paula Ta-Shma, Bogdan Tarnauca, Leonard Pitu, Adnan Akbar, Orfeas Voutiras, Achilleas Marinakis	IBM, Siemens, Surrey, NTUA	IBM: Data store and metadata search Siemens: Security Management Module, H/W Board, Semantic Topic Management, VE Registry, Social Monitoring Surrey: Prediction NTUA: Planner, Discovery, Experience Sharing

0.6	26/04/2014	Spyridon Gogouvitis	NTUA	Updating Use Cases, Sequence Diagrams
0.7	28/04/2014	Jozef Krempasky, Mónica Aguirre	ATOS	Internal Review
0.8	30/04/2014	Spyridon Gogouvitis	NTUA	Updating based on ATOS review
0.9	02/05/2014	Spyridon Gogouvitis, Orfefs Voutiras, Bogdan Tarnauca, Francois Carrez	NTUA, SIEMENS, Surrey	Siemens review and updates based on the review.
0.10	02/05/2014	Spyridon Gogouvitis, Orfefs Voutyras, Bogdan Tarnauca, Francois Carrez, Leonard Pitu	NTUA, SIEMENS, Surrey	Pre-final version ready, pending only use cases description.
0.11	05/05/2014	Andrés Recio, Sergio Fernández	EMT	Inclusion of Madrid use case
0.12	09/05/2014	Spyridon Gogouvitis, Orfefs Voutyras, Paula Ta-Shma	NTUA, IBM	Final Version

Annexes:

Nº	File Name	Title

Table of Contents

List of Figures

List of Tables

Table of Acronyms

Acronym	Meaning
API	Application Programming Interface
ARM	Architectural Reference Model
AWS	Amazon Web Services
CBR	Case-Based Reasoning
CDMI	Cloud Data Management Interface
CEP	Complex Event Processing
CoAP	Constrained Application Protocol
CRUD	Create/Read/Update/Delete
D	Deliverable
DM	Domain Model
DNA	Dynamic Network Analysis
ETA	Estimated Time of Arrival
FC	Functional Component
FG	Functional Group
FM	Functional Model
FPGA	Field-Programmable Gate Array
FV	Functional View
GPS	Global Positioning System
GUI	Graphical User Interface
GVE	Group VE
HAN	Home Area Network
HMAC	Key-Hash Message Authentication Code
HTTP	Hypertext Transfer Protocol

HTTPS	Hypertext Transfer Protocol Secure
HW	Hardware
I2C	Inter-Integrated Circuit
ID	Identifier
IM	Information Model
IoT	Internet of Things
IoT-A	Internet of Things - Architecture
IT	Information Technology
JSON	Java-Script Object Notation
KPI	Key Performance Indicator
LDAP	Lightweight Directory Access Protocol
M2M	Machine-to-Machine
MAPE-K	Monitor/Analyse/Plan/Execute-Knowledge
OWL	Web Ontology Language
P2P	Peer-to-Peer
PE	Physical-Entity
PKI	Public Key Infrastructure
QoS	Quality of Service
RA	Reference Architecture
RBR	Rule-Based Reasoning
RDF	Resource Description Framework
RDF-S	RDF Schema
REST	Representational State Transfer
RM	Reference Model
RPC	Remote Procedure Call

SHA	Secure Hash Algorithm
SNA	Social Network Analysis
SNIA	Storage Networking Industry Association
SPARQL	SPARQL Protocol And RDF Query Language (recursive then)
SPI	Serial Peripheral Interface
SQL	Simple Query Language
SSH	Secure Shell
UML	Unified Modelling Language
URI	Uniform Resource Identifier
VE	Virtual Entity
WP	Work-package
XP	Experience

1. Executive Summary

This report (D2.3.1) presents the preliminary version of the document describing the overall architecture of COSMOS. The main objective of COSMOS project is to provide the mechanisms needed to make Things in the IoT domain smarter and increase the interoperability of various hardware and software solutions. Its aim is to provide solution developers the means of integrating multitudes of different data sources (in the form of Virtual Entities- digital counterparts to Things- and IoT-services), as well as using the data processing capabilities, storage, information retrieval, complex event processing and other COSMOS functionalities, which otherwise would have meant increased design and development costs. The work described in the report was carried out in the framework of WP2 - Requirements and Architecture- and provides the analysis, design and specification of the initial architecture of the COSMOS platform. This is the first out of the three reports, which will describe, in more detail as the project advances, the functionality of COSMOS, the functional and non-functional capabilities and the individual components as well as their interactions.

The output of WP2 is of high importance for the other WPs, but also for the project itself, since these reports include the guidelines for developers to implement their components, associating, on the same time, the user's requirements and the scenarios with specific functionalities and building blocks. Therefore, from the beginning of the project, WP2 decided to follow, at the level that this is possible, a well-established and successful methodology for the design of concrete IoT Architectures, namely the IoT Architectural Reference Model from the IoT-A FP7 project [1]. Moreover, by using UML models, the results of WP2 are more easily and effectively comprehensible and adopted.

Initially WP2 analysed from the architectural point of view the user requirements and key parameters included in the D2.1.1 [2] and D2.2.1 [3] reports, taking into account the technical and business prerequisites for the COSMOS platform. Based on this analysis, the COSMOS Domain Model was developed in compliance to the IoT Domain Model from IoT-A. In addition a Risk Analysis was conducted, highlighting the importance and cross-cutting nature of security for COSMOS. The analysis was followed by the identification of the platform capabilities and the design and presentation of conceptual models for the COSMOS platform, which describe the main processes in the scenarios such as Registration or Discovery. Part of the process was also aimed at confirming that the identified models and capabilities can be implemented with the technologies of the future trends/innovations acknowledged in the D2.2.1 report.

In Section 8 an overview of the platform design and the main subsystems are presented to the maximum level of detail feasible at this point in time, since there exist numerous open issues between the development WPs. Additionally, the expected role for each block and the interactions with the other blocks are explained. Partners provided inputs for this section, based on their expertise in the project and it is anticipated that in this report, especially regarding the sections of the architecture overview, the complete COSMOS project design will come together and be presented. This is extremely useful for the development WPs to start their work in a consistent and efficient way. During the project months to follow, WP2 will focus on the analysis of subsystems and their interactions and hence this section will be considerably updated in the next versions of the report, describing in details all the components, interfaces but also the specifications about the hardware and software tools that will be adopted and used throughout the project.

2. Introduction

In this section we elaborate the main innovations and ideas which serve as a basis for our initiative and are fundamental in realizing the project's vision. These include the extension of Things semantics to capture their social behaviour, the analysis and enhancement of the reliability of information Things provide, the embedding of intelligence into Things, so as to allow them to learn and adapt, the provision of scalable data and information management and the provision of security mechanisms across all the layers.

2.1. Enhancement of the reliability of information things provide

Volatility management within the IoT community is approached from the perspective of the management of unpredictable and unreliable data provided by heterogeneous sensors and devices. The unpredictable IoT data environment stems from the underlying network conditions and device resource-constraints, with data collected by sensors and devices becoming volatile with time (i.e. the quality of data can vary with different devices through time and could simply become unavailable or wrong). The volatility (e.g. incomplete time series of sensor readings due to faulty devices) can make the task of processing, integrating, and interpreting the real world data then a challenging task.

COSMOS will address this problem by providing mechanisms that enhance the reliability of information sources. Interpolation and extrapolation techniques will allow providing missing information in cases of failures or intermittent communication.

2.2. Embedding intelligence into Things

According to CISCO during 2008, the number of things connected to the Internet exceeded the number of people on earth and by 2020 there will be 50 billion, shaping a rich digital environment. Sensors, intelligent fixed and mobile platforms (e.g. smartphones, tablets and home gateways), massive scale cloud infrastructures and other network-enabled devices will all need to cooperate and interact to create value across many sectors in smart cities. This digital environment creates a treasure trove of information, which is the key enabler for embedding wisdom into objects. The added value in a city context is that by making objects smarter, cost savings and increased efficiencies will be created, thus allowing for long-term economic growth. Overcoming the management limitation for the exponentially increasing number of things requires for techniques that move intelligence to the edges.

COSMOS will enhance the management of a large number of things by embedding intelligence into them and allowing them to describe, exchange and learn based on others experiences. Moreover, situational knowledge acquisition and analysis techniques will make things aware of conditions and events affecting IoT-based systems behaviour.

2.3. Scalable data and information management

Networks of things generate and exchange a large volume of different data ranging from Raw data to Information and Knowledge. Things provided by different resource owners generate diverse data types of a large amount (e.g. sensing information, profiles or data from data bases, online information, etc.) which is often linked, aggregated and combined into new groupings or structures.

COSMOS will deliver data and information management mechanisms to handle the exponentially increasing "born digital" data. COSMOS will develop mechanisms for the data lifecycle management, from capturing to storing (as networks of stored objects), processing (through computation functions to be executed close to the storage) and delivering. Scalable, highly available, resilient to failures, and widely accessible storage will be developed, exploring the interplay between storage and analytics on networks of data objects.

2.4. Security across different layers

In spite of increased use of mobile devices for specific and well-defined usages, many remain sceptical about the broader deployment of Internet of Things, fearing Big Brother intrusion rather than seeing the opportunity of accessing and exploiting the content feeds in new and creative ways that benefit the whole society (including those same sceptical users). Consequently, developing sustainable Smart City applications requires for mechanisms to ensure security and trust and preserve privacy. Such approaches should address both the low levels of IoT environments (i.e. hardware-coded techniques) as well as the data management and application levels. Tamper-resistant smart devices, dynamic and evolutionary trust models, secure data stores, applications with build-in security and privacy are critical for sustainable smart city applications. Tussles of personal privacy and freedom of expression are expected to be a consequence.

COSMOS will facilitate IoT-based systems with end-to-end security and privacy, from hardware-coded approaches on the devices level, access control, encryption, multi-tenancy and cross-application mechanisms on the data level, to the IoT services level with the injection of privacy –preserving mechanisms within things themselves.

The provisioning of reliable and secure data to users and applications will be enhanced by developing the mechanisms to asses and publish the trustfulness and reputation of the various actors involved in the COSMOS environment (VEs, IoT services, applications, etc.). Such an assessment will serve other purposes as well since it is, for instance, a key element in the COSMOS social analysis.

2.5. Extension of Things semantics

COSMOS will provide the semantic models and the mechanisms for describing the building blocks of its environment. Semantic description of VEs, IoT services, Data Bus topics or applications will not only make the retrieval of particular elements easier and faster (since such descriptions are capturing also the links between these elements or other concepts which might be relevant for a retrieval request) but will provide the input data for complex analysis mechanisms.

Rich semantics are also needed to capture the social behaviour of Things. Given that objects follow the interaction and operation patterns of their contributors / owners (with respect to administrative rules, location, collaboration properties, information exchange, experience sharing, etc.) we will build upon and extend social media mechanisms to identify the aforementioned patterns and incorporate this information into the description of Things. This will allow Things to evolve (thus becoming smarter and more reliable) as well as enhance the management of the networks of Things.

3. Methodology

As a primary decision of the project, it was decided that the IoT-A Reference Architecture [1] will be used as a basis for the specification of the COSMOS Architecture. This has helped in having a common language from the early stages of the architectural process as well as a structured means of deriving the initial COSMOS architecture, focusing at developing a re-usable architecture that can fit different application scenarios.

3.1. IoT-A Methodology

The *Architectural Reference Model* (ARM) introduced by the IoT-A project [4], consists of three interconnected parts which are the *Reference Model* (RM), *Reference Architecture* (RA) and Guidance. The ARM also follows the best practices in software engineering as introduced by Rozanski & Woods [6] when designing the RA as we will see later. The constituents of those three parts are detailed below:

- The IoT Reference Model consists of a set of Models:
  - Domain Model
  - Information Model
  - Functional Model
  - Communication Model
  - Security/Trust/Privacy Model
- The IoT Reference Architecture consists of Views, View-Points and Perspectives as introduced in [6]:
  - Functional View
  - Information View
  - Deployment & Operation View
  - Security Perspective
  - Interoperability Perspective
  - Resilience Perspective,…
- A set of Guidances (also called best practices) that define the overall process that should be used when deriving a concrete IoT architecture out of the ARM. In particular the Guidance part proposes a large set of tactics and design choices that can be associated to the perspectives, i.e. to qualities of the system the designer wants to meet.

More precisely the RM provides a set of models that are used to define certain aspects of the architectural views. One of the most important models is IoT Domain Model [5]. It defines taxonomy of IoT concepts (e.g. Physical, Virtual and Augmented Entities, Devices, Resources and Services) and a set of relationships between those concepts.

It defines the IoT domain in general; a customization of this generic model w.r.t. a specific IoT application allows generating a common understanding of that domain (like identifying the entities of interest for that application, identifying the resources, e.g. sensors, actuators, etc.). We actually generated this customized version at the beginning of the COSMOS project that helped us to speak one voice as far as the COSMOS vocabulary is concerned. The COSMOS Domain Model is available in section 4.

As we listed above, the RM also provides:

1. an *Information Model* (IM) which is a meta-model used to describe information as handled within the system,

2. a Communication Model,
3. a *Functional Model* (FM) used as the foundation of the Functional View and, finally,
4. a model for Security, Trust and Privacy

All those models can be used and shared between various teams of people dealing with different aspects of the architecture.

Based on the RM models, the RA consists of a set of Views (used to represent certain structural aspects of the system) and Perspectives (that focus on quality of the system that spans different views, e.g. Security, Resilience, etc.). For general information about Views, View points and Perspective, please refer to Rozanski & Woods work [6].

The IoT *Functional View* (FV) (see Figure 1 below) is very important as it proposes a layered model of *Functional Groups* (FG), which maps to most of the concepts introduced in the DM, together with a set of essential *Functional Components* (FC) (and associated interfaces) that an IoT system should provide.

In Chapter 10 we try to fit the COSMOS Functional View within the general FV introduced by IoT-A. We foresee that in future refinements of the Architecture new Functional Groups and Components will be identified and introduced, as the IoT Functional View from IoT-A initially aimed at staying as generic as possible.

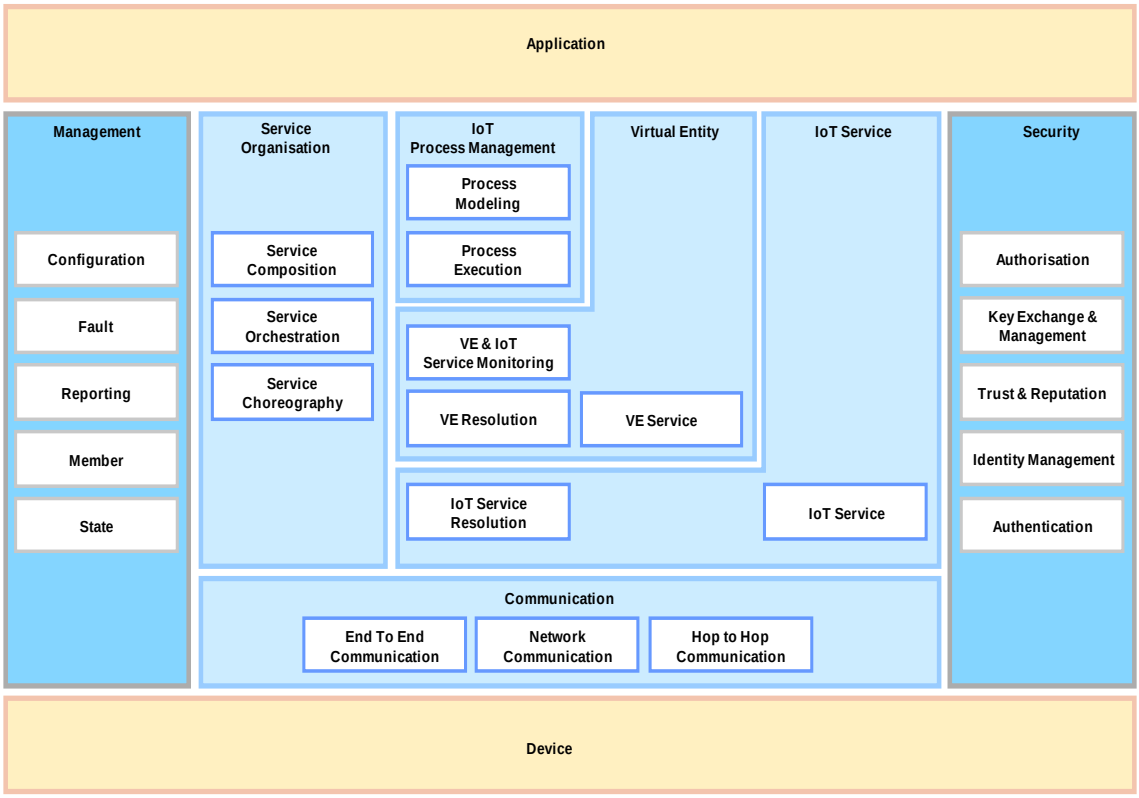

Figure 1. IoT-A ARM functional view

It is worth mentioning that the FV is not exhaustively developed. The Information View, based on the IM, complements the FV and provides a more detailed view about how information is to be handled in the system (including details about the components where the information is handled) and how it flows within the system. The perspectives are mainly derived from non-functional requirements and consist of activities and related tactics.

Last, but not the least there is the Guidance part. It defines the process that based on the RA and RM will lead to the generation of the concrete IoT architectures. In particular it defines the requirement process, introduces additional views (i.e. Physical View and Context View) that are not part of the ARM as they are application dependent and explain in general how and in which order the set of architectural views (which constitute a concrete architecture according to Rozanski & Woods) should be generated. It also gives a large (but not exhaustive) list of design choices that can be used as recommendations to achieve certain system qualities (see perspectives above).

The process followed to create a concrete architecture is as follows: firstly we defined the COSMOS Domain Model based on the IoT Domain Model introduced in the ARM. This task was achieved at an early phase of the project and helped to set-up a common grounding as far as getting a common and agreed terminology is concerned. Then we define the Physical-Entity (P-E) and Context views of the system, then define the requirements and finally derive the Functional View, the Information View, the Operational View and the Deployment View.

It is worth noting that the Physical-Entity View and Context View are not exhaustive in the sense they capture specificities of the Use-Cases coming from Madrid and Camden. This process was also followed in defining the initial architecture of COSMOS. The work on the Physical-Entity view is described in Section 5.1.1 while the context view is described in Section 5.1.2. The requirement gathering and analysis has been described in deliverable D2.2.1. Based on this input, in Section 7, use case diagrams have been used to describe the key functionalities of the platform that have been identified in the platform analysis process. Use case diagrams depict an overview of the usage requirements of the system. The actors can be easily identified as well as their association with any interactions described by the use cases. Use case diagrams do not provide in depth analysis of the use cases, but rather a visualization that can help in the preliminary phases of the requirements analysis process. Following, we derived the functional view of COSMOS which is an overview of the system functional blocks and how they interact. Every functional component is described in high level including its interaction with other components of COSMOS. Components may both provide and require interfaces. This section formed the basis in defining the interrelationships between different components that may be developed by different partners and in that way established a formal communication channel between them that is necessary during the development process. More detailed descriptions of the components can be found in the Design and Open Specification Reports of the work packages.

In Section 9 the information view of COSMOS is presented through the use of sequence diagrams. These diagrams model the flow of logic within a system in a visual manner, enabling both to document and validate the logic behind the system's development, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artefact for dynamic modelling, which focuses on identifying the behaviour within a system. Other dynamic modelling techniques include activity diagramming, communication diagramming, timing diagramming, and interaction overview diagramming. Sequence diagrams, along with class diagrams and physical data models are the most important design-level models for modern business application development.

Finally, in Section 10 we attempt to align our architecture to the *Functional Groups* (FGs) proposed in IoT-A. The Functional Groups of IoT-A are shortly presented and the COSMOS functional components are mapped to them by identifying similar functionality.

4. COSMOS Domain Model

4.1. Introduction

In COSMOS we try to strictly follow-up the IoT Architectural Reference Architecture introduced by IoT-A as a tool (made of Models, Views, Perspectives and Guidance) for deriving concrete architectures. One of the most important models proposed by the ARM is the IoT Domain Model. It can be considered to some extent as a formal definition of what the IoT domain is made of, in terms of IoT concepts and relationships between those concepts. In the following section we therefore introduce all concepts and terms considered in the COSMOS project and we put them in perspective with those referenced within the IoT Domain Model. As a result of this exercise we will get a COSMOS Domain Model that eventually fits into the framework introduced by IoT-A.

4.2. Definition of Terms and relations

Virtual Entities **(VEs)**: VE's are at the heart of an IoT system. They are a representation of physical objects in the Cyber-world. Properties of the VE's can be populated/instrumented using tags, sensors (incl. tag readers, positioning sensors, temperature sensors etc.) and actuators which are devices. Devices and hosted IoT resources are therefore used for bridging the Cyber and Physical worlds; Sensors do report information about the physical entity while actuators are used to modify the state or properties of the Physical Entity.

Typical examples of VE's in COSMOS are:

- Buses, bus stops and traffic lights in the Madrid use-case
- Flats/household in Hildebrand use-case

VE's may have their own goals and be equipped with an internal logic in order to achieve their goals. VE's get perception through accessing sensors readings via IoT Services (Resource-centric IoT Service) and can impact their environment or undertake physical actions using actuators (triggered via other IoT Services). Finally VE's may interact with other VE's for various purposes like:

- collaboration (sharing a common goal with other VE's),
- cooperation (getting help from other VE's in order to achieve their own objective),
- advertising about VE's properties/attribute (via a VE-centric IoT Service)
- offering actuation service (via VE-Centric IoT Service)

The persistence of VE's data and historical data is ensured by a Data Store in the form of Cloud Storage.

VE's offer service interfaces to other VE's or higher-level services (like orchestration engine for instance). Example of service could be e.g. accessing VE's experience or initiating cooperation, exchanging knowledge etc.

From the Domain Model point of view, VE's interact with IoT Services, which expose the functionality provided by Resources. Often Resources are running on IoT Devices. A look-up service can be used for a VE to identify which IoT Service it may access. And as seen earlier, interfacing with a VE is possible through IoT Services.

***Group Virtual Entity* (GVE's):** COSMOS makes a distinction between individual VE's (representing single objects) and group VE's (VGE's) that aggregate/encompass a potentially large number of VE's. The IoT Domain Model does not make such difference as it is allowed for VE's to be aggregations of various VE's). As for VE's GVE's have their own properties (mainly based on properties of embedded individual VE's) and have their own objectives (often management and optimization functions). Like VE's they offer interfaces via IoT Services. Finally Group VE's can embed other Group VE's.

Typical examples of GVE's in COSMOS are:

- Bus lines in the Madrid use case: the objectives of a bus line could be 1/ to monitor the performance of all buses sharing the same itinerary across a city and 2/ to communicate with bus station/stops in order to notify about ETA.
- Flat VE's relying on Room VE's in the Smart Energy management (Hildebrand scenario).

IoT Devices: In COSMOS, IoT Devices are the hardware supporting the sensing and actuation functions. Micro-controllers, batteries, ROM memory etc. are Devices (without the IoT prefix).

IoT Resources: are the software embedded in IoT Devices that provides the raw readings (for sensors) and actuations. In the COSMOS project we should not consider accessing Resources directly; instead we should access the IoT Service exposing the IoT Resource. IoT Service could for instance expose with a standardised interface the raw reading provided via the IoT Resource enriched with meta-data. In this case, they are called Resource-centric IoT-Service

IoT Services: We can consider different kinds of IoT services depending on their level of abstraction:

- (Resource-centric) IoT services are exposing the IoT Resources using standardised interface and adding meta-data to the raw reading available at the resource level.
- (VE-centric) IoT Services are associated to the VE's and are used for accessing VE's attributes/status.
- (integrated) IoT Service are combinations of the two above when combining different readings from different sensors (e.g. "secured" room can depend on lock/unlock status, presence indicators and light status).

Few examples follow:

- In the Madrid use case, a Bus VE is accessing the Bus CanBUS via a resource-centric IoT Service in order to know its speed, diesel autonomy etc… and can access an embedded GPS chip in order to be aware of its location. The Bus VE may then expose those attributes (speed, autonomy etc…) through VE-centric IoT Services (maybe enriching the raw data with meta-data like {location, timestamp}) via a REST / CoAP interface. A line GVE could then access the characteristics of the Bus through the VE-Centric IoT Services exposed by each individual bus; Of course a bus may be associated with other services which do not relate directly to IoT Resources e.g. pushing a new itinerary, getting an ETA relating to a specific Bus Stop (see the Service section below); A GVE could be associated (or expose) with an (integrated) IoT Service when for instance one of its attribute is built up from aggregation of individual VE's attributes (indeed from the IoT point of view, a GVE represents a "composite" object made of individual objects and its characteristics are built up from readings coming from the individual objects)

IoT Services are associated with Service descriptions that can be used to discover particular Sensing/Actuation capabilities.

Service: Services (without IoT prefix) are associated to VE's and GVE's but do not relate to specific Resources as illustrated in the example above. Services are not part of the IoT Domain Model but could be added to the global picture for the sake of clarity.

COSMOS_User: A user of the COSMOS platform will mainly interact through IoT Services and Services. A user can be a human person or a Digital Artefact (VE, Application, and Service). This is not shown for simplicity. It is worth noting that the services offered by the platform itself are not represented in the Domain Model as they are not constrained to the IoT Domain, but are general enablers.

4.3. COSMOS Domain Model

The following picture shows a short and simplified fragment of the COSMOS Domain Model restricted to VE's (individual VE's and Group VE's), IoT Resources, Services (IoT Services and "classic" Services and Devices) and shows how they relate to each other.

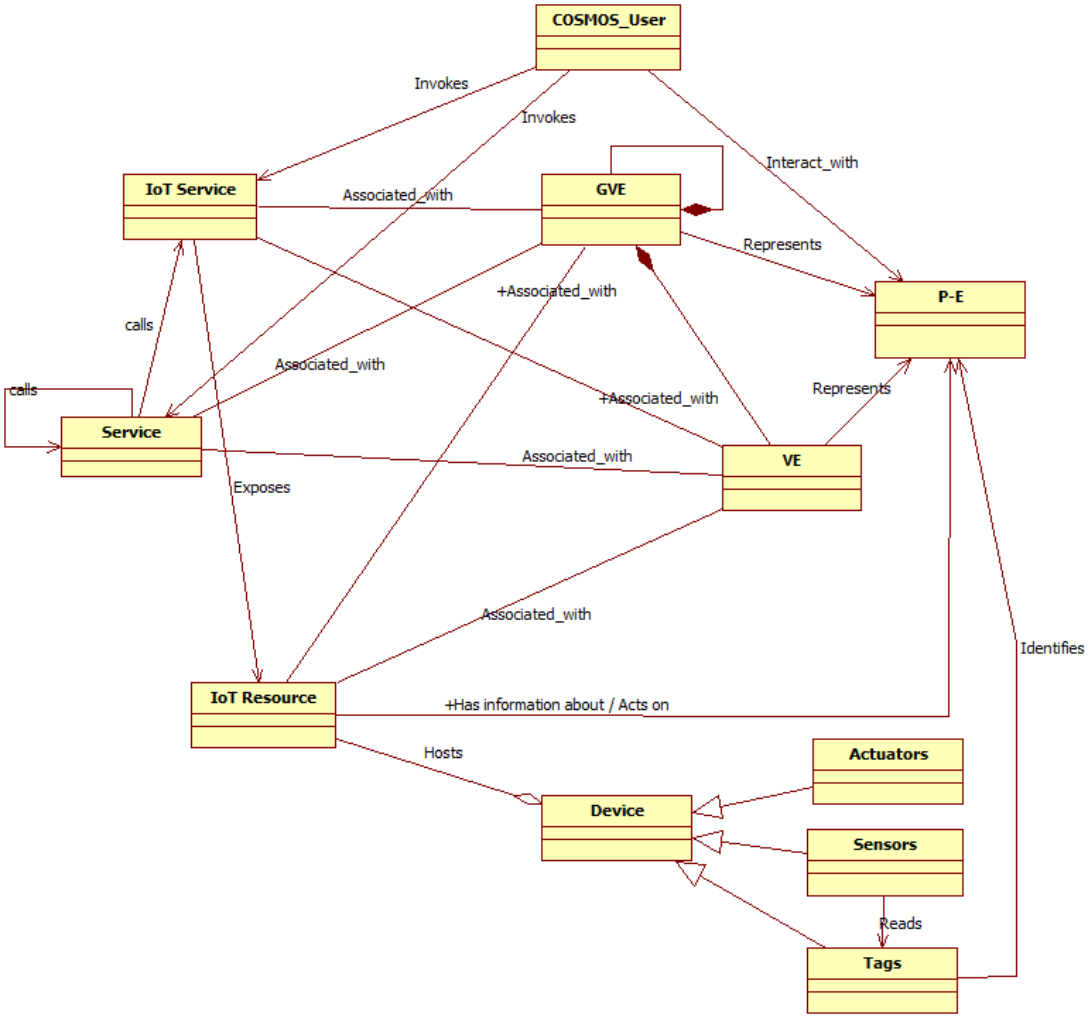

Figure 2. COSMOS Domain Model

5. Physical-Entity and Context views

5.1. Introduction to IoT Physical-Entity and IoT Context View

In this section we briefly describe the process needed to derive the Physical-Entity and Context Views. The Madrid Use Case is used as an example.

The ARM methodology introduced by IoT-A covers a number of Models, Views and Perspectives and gives details on how to build and use those. However there are two views, namely the IoT Physical-Entity View and IoT Context View, which are very important, considering the whole architecting process but which however are not part of the views as described in chapter 4 of the IoT ARM document. The reason is that those views relate very much to the intrinsic characteristic of the specific IoT case being designed and consequently it is very difficult to extract a generic way of handling them, via a generic Physical-Entity Model for instance (for the P-E View).

5.1.1. Physical-Entity View

The IoT P-E View relates to the physical entities (sometimes also called Entities of Interest) which are in the scope of the IoT system under design, and which will be virtualised in the form of Virtual Entities (as shown and explained in the Domain Model). This view will describe in detail the properties of the Physical Entities (and how they are captured in the VEs) and describe which devices are used and what are their relationships to the P-E (and associated properties). It also explains where those devices are situated compared to the P-E (touching, fixed, in proximity, etc.). The P-E also describes the nature of the information captured by devices w.r.t to the P-E / VE properties.

5.1.2. IoT Context View

According to the ARM, the IoT Context View consists of two different entities: the IoT Domain Model (see section 4 in this document) and the Context View as introduced by Rozanski & Woods in [6] (not prefixed with IoT then). The Context View describes “the relationships, dependencies and interactions between the system and its environment (the people, systems, external entities, with which it interacts)” [6].

Applied to one of the COSMOS use cases, e.g. the Madrid use case, the context view will have to identify in the details all actors involved and all external components (all VEs and GVEs for instance) with which the COSMOS platform is interacting. It will also describe the nature of their interactions. The concerns addressed by [6] cover the following aspects:

- System scope and responsibilities;
- Identity of external entities and services and data used;
- Nature and characteristics of external entities;
- Identity and responsibilities of external interfaces;
- Nature and characteristics of external interfaces;
- Other external interdependencies;
- Impact of the system on its environment.

5.2. Madrid Use Case

5.2.1. Introduction

In Madrid, EMT operates the city bus lines (in total 215 lines) through a fleet of 2095 vehicles, which have an average age of 6.04 years. In 2011, EMT operated a total of 7.11 million hours and 95.45 million kilometres, with an average operating speed of 13.43 Km/h. The buses are equipped with GPS devices providing information regarding their location and speed. Moreover, City of Madrid has deployed sensors in the streets associated to the traffic lights, which can be found throughout the city and allow remote management. In this scenario we propose to take into consideration the available information from mobile sensors deployed in the buses regarding the routes, vehicle information such as car speed, location information from GPS devices, as well as from static city sensors such as traffic lights location and changing intervals. Additional information may be provided by citizens from their mobile devices (such as smartphones and tablets). Given the big number of vehicles and their operation, the goal of the scenario is for COSMOS to provide the ability to manage the different things as well as the reliability of the information they provide. COSMOS will help to extract knowledge at real-time out of the data streams contributing to the situational awareness of buses.

Several different scenarios are under consideration for developing and utilizing the COSMOS offerings:

- CASE 1: Forwarding a bus to a bus stop outside of the usual trip or path. The purpose is to analyse the possibilities for optimized management of trips, through which a bus could not expand its route if no traveller needs to descend at a specific stop;
- CASE 2: Detecting presence of people at traffic lights for selective control of the opening of green phase only when required;
- CASE 3: Contextual leisure and commercial offers to users regulated depending on the waiting time for the arrival of the bus;
- CASE 4: Focusing on groups which require protection (such as handicapped people or children). Including the possibility of making a check-in once on board the bus and monitoring by a caregiver or responsible for ensuring the trip or the collection on arrival;
- CASE 5: Notification of a breakdown of a bus en route to a nearby assistance vehicle and calculation of the approach path considering the two moving vehicles;
- CASE 6: Planning a route for an emergency vehicle to a final destination, altering traffic light phases to enable, as far as possible, favouring the way of the vehicle to its final destination; and
- CASE 7: Foster mobility around the city using public services by deploying elements of “gaming”, turning buses into social meeting areas, libraries or any other model that enhances the recreational use of public transport.

We will use Case 4 as a basis for the rest of this section. A more detailed scenario illustrating Case 4 is defined as follows: A kid takes a bus to go to school, and his device connects with the bus in order to notify a third person (like family member) whenever he gets on and where and whenever he gets off the bus. Moreover hisfamily member is notified whether he has reached his final destination or not. Estimated time of arrival is calculated continuously based on the bus position and traffic and incidences information coming from other buses and traffic lights.

5.2.2. Physical-Entity View

In order to better understand the overall process of defining Madrid use case, it might be necessary to take into account the two fundamental information units which are involved to control the bus system of the city:

1. Bus: EMT vehicles offer an extensive range of available and useful data for the COSMOS system. Among its benefits, we should consider the following:
   - It is a mobile element of the city;
   - Its system contains an open Linux Suse 11 architecture, including Web services under API-REST;
   - It is predictable in terms of their position, and they are locatable in time and space;
   - They communicate with a highly sophisticated central control unit;
   - It allows the connection of new sensors and devices;
   - It is possible to inter-connect two, several or all of the units in operation;
   - Brings the information throughout the city of Madrid;
   - In rush hour there are 1800 buses running.

2. Traffic lights: traffic lights can be a cornerstone in managing the fixed elements within the city. Among other possibilities it offers:
   - They are found throughout the city, both in isolated areas and in areas of high traffic density and population;
   - They control pedestrian and vehicle crossings;
   - They are communicated by areas and allow remote management;
   - They have high-speed communications with the control centre;
   - They allow extending the possibilities of developing integrated bus-traffic light or citizen-traffic light systems.

These are the main physical entities that will be involved in all scenarios of the Madrid use case. More physical entities may be used in specific cases.

Before defining possible use cases, we would like to point out which could be the virtual entities that may interact, such as:

a. Transport:
   - Bus and integrated sensors (position, route, arrival times);
   - Bus lines (trip, all buses position and status);
   - Bus stops (status, position and bus arrival times).

b. City:
   - Traffic lights (signal phase and cycles);
   - Traffic density;
   - Incidences.

5.2.3. Context View and Use Case Architecture

Madrid has defined a working plan to merge its bus service architecture and traffic architecture into a single model and system under COSMOS framework, to use it as a unique semantic layer with capacity to make asynchronous calls, offering to COSMOS an events subscription environment from which to implement its VE model.

As indicated in the following diagram, this model supports two architecture models:

1. VE architecture in a format P2P, using communication, for instance, between a Smartphone app and the bus.
2. VE architecture on a centralized model, publishing states and VE at a server platform.

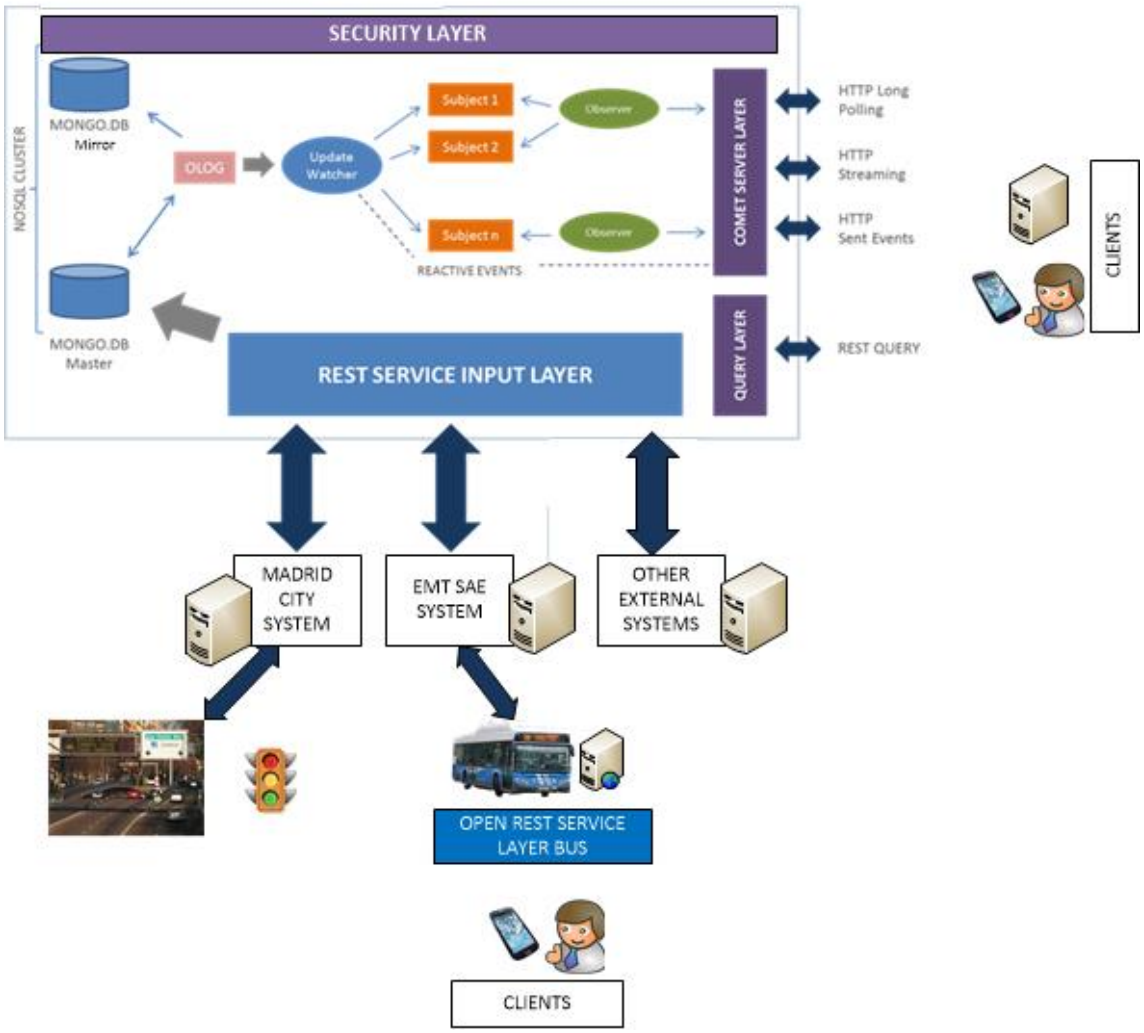

Figure 3. Madrid Use Case Architecture

The use case number 4 intends to act as a scenario for testing both aforementioned architecture models (bus local and server). In addition to this, we aim to create an inter-operable infrastructure to be used as a support to other COSMOS use cases in Madrid.

In the following paragraphs we identify the entities that make up the use case and how they may interact with each other and with COSMOS.

The different entities are:

- Special person: A human user that needs special care;
- Care-giver: A human user (e.g. family member as introduced earlier) who is responsible for the special person above;
- Monitor application: This is the application that allows the care-giver to monitor the status of the special person. The application is divided into a mobile application that runs on smartphones and a central server;

- VE-Handicap: This is the VE that is created by a mobile application that runs on the special person's mobile device and represents the special person;
- VE-Caregiver: This is the VE that is created by a mobile application that runs on the Caregiver's mobile device and represents the Caregiver;
- VE-Mobility: This is the VE that represents the EMT application presented in Figure 3;
- VE-Bus: This is the VE that represents a Bus;
- VE-Traffic light: This a VE that represents a traffic light;
- GVE-Bus Line: This is a GVE that contains all the VE-Buses of a specific line.

Based on these entities we can further define a scenario which can be divided in different phases. All depicted phases are not necessarilyto be considered in sequential order. For example, phases seven, eight and nine may be continuously running in parallel along the other phases. It also worth noting that the presented scenario is intended as a basis for identifying the different entities and their possible interactions. It is envisioned that this will be changed and enhanced as the project progresses.

5.2.3.1 First phase: User Registration

In the first phase the human actors, namely the special person and the caregiver sign up with the Monitor application. More specifically:

A user logs into a website introducing its profile:

- User, password
- Profile type:
    - Caregiver profile:
        - Watchman, school tutor
        - Parents or legal tutors
        - Medical personnel
    - User profile:
        - It requires tracking/observation
        - It requires medical care
        - It has reduced mobility capabilities (and type).
        - It has a mental disability
        - It requires to be picked up once at destination.
        - Caregivers list (and available hours to track the user)
        - Enabled tracking (true/false)
        - Photographing user on bus enabled (true/false)
- Contact details:
    - Phone number
    - Address
- Transport data:
    - For each day of the week:
        - Starting time of the whole journey
            - For each bus stretch or section of the journey:
                - Bus lines list to be used in each stretch or section (there could be more than one)
                - Estimated arrival time to the bus stop

- Origin and destination bus stops

Moreover, VE-Buses and VE-traffic lights are registered with COSMOS and publish their information under specific topics of the message bus. This information is also persisted in the cloud storage and can be used later to provide analytics. The VE-Mobility also subscribes to information coming from the VE-Buses and VE-traffic lights.

5.2.3.2 Second phase: VE Registration

The special person and the caregiver download the application on their mobile devices. The mobile application registers itself with COSMOS providing the semantic information it needs, thus creating the VE-HANDICAP and VE-CAREGIVER. The user is able to associate a VE-HANDICAP with a VE-CAREGIVER.

5.2.3.3 Third phase: Application activation

The special person selects the planning of the day in the application and activates it. The VE-HANDICAP contacts the VE-Mobility and receives the route for the plan.

The VE-HANDICAP registers with specific topics of COSMOS and receives notifications. Moreover, the VE-HANDICAP or the VE-Caregiver may inject situational awareness rules (to CEP-based Situation Awareness block) asking for aggregation and analysis of events coming from VE-Buses and VE-Traffic lights. The VEs can thereafter subscribe to new topics and receive notifications on events. The application remains active in the Smartphone until activation stops.

5.2.3.4 Fourth phase: Information coming from various sources

VE-buses and VE-traffic lights continuously publish information to COSMOS. These are persisted in the cloud storage. Examples of what information may be shared:

- Geographical location and status of each bus of the lines;
- Expected time of arrival to each bus stop up to its final destination;
- Distance from the trip start;
- Incidents that are happening;
- Traffic conditions;
- Traffic light phases;
- Bus conditions, for example how crowded is the bus.

This information can be used to create predictive models of the buses location, update model of traffic condition etc.

5.2.3.5 Fifth phase: Start of a Trip

The user begins the trip approaching to one of the planned bus lines. Since the time in which the wireless access point of one of the possible buses is detected, the VE-HANDICAP starts a dialogue with the VE-BUS. This dialog can be repeated several times as the user could finally lose that bus or ride another, so the analysis of the use of a particular vehicle must go through the time in which the VE-HANDICAP remains linked to a particular VE-BUS including displacement of meters. The VE-BUS, from that time, may incorporate VE-HANDICAP data and attributes forwarding them towards the VE-Mobility. The VE-Caregiver is notified of this event.

5.2.3.6 Sixth phase: Trip Monitoring

Based on the situational awareness rules that have been injected in the system, COSMOS accumulates and analyses events so that it can become reactive to the on-going situations, which could be:

- Appearance of the user on a bus from a different bus line than expected;
- Advance or delay forecast to reach the transhipment or meeting point;
- Not reaching the destination stop by descending elsewhere or continuing the route beyond the destination stop;
- Problems with the transhipment route;
- Relocation of the user;
- Forecast of incidents.

Based on this, the VE-CAREGIVER is able to obtain information from the different states offered by the COSMOS based on the events analysis and its predictive model. This information, will enable the VE- CAREGIVER:

- To keep track of the VE-HANDICAP;
- To get alarms based on the state of the Service, the anticipated time of arrival, delays of departure, advances, etc;
- To send alerts to the bus driver on the state of the VE-HANDICAP travelling aboard;
- To get alarms generated by lack of foresight on the route.
- To get alerts from the user (urgent help or panic button).

5.2.3.7 Seventh phase: Data Analytics

COSMOS records data regarding the history of the times and locations of user travel as well as other information regarding buses and bus rides. This data, residing at the COSMOS Data Store side, can be used for analysing:

- What are the locations that a special person or a caregiver is familiar with? When new locations are encountered the application can give additional instructions and verify that the users find their way;
- Has the special person got off at the wrong bus stop in the past? In this case special care could be taken to avoid this;
- How crowded do we expect the bus to be? This could depend on the particular bus stop, the direction of travel (to/from the city centre) and the day and time of travel. It could be predicted by looking at the history of bus rides and how crowded they were.

5.2.3.8 Eight phase: Prediction of values

Historical data persisted on the cloud storage can also be used to make predictions on the reported values. For example, it is possible that the bus the special person is riding fails to report its position for a period of time. In this case it is possible to use the prediction functionalities of COSMOS to estimate the position of the bus. This is based on retrieving historical data that have been persisted in the Cloud Storage and creating prediction models that are able to extrapolate the position of a bus.

5.2.3.9 Ninth phase: Autonomous behaviour and experience sharing

Moreover, the VE-CAREGIVER can obtain experience, while the Caregiver can define how problems (undesired situations) should be faced or ask for a solution to new problems from COSMOS or other VEs. Consequently:

- The Caregiver may define situations and actions that should be taken when these situations occur by using a library provided by COSMOS and running on the VE-Caregiver. This allows the VE to react to events that are published by COSMOS. That way, a VE can be autonomous, in the sense that it will be able to execute plans without a demand from the user;
- The Caregiver may also define a situation with an unknown solution. For example a Caregiver may define that if the VE-Handicap gets off at an unknown location a solution needs to be found. When this occurs, the VE-Caregiver will find other similar VEs through COSMOS and request that they share their experience on similar cases.

The above mentioned scenarios are indicative of how COSMOS can be used to model the Madrid use case and how the different actors may interact between themselves and COSMOS in the scope of a smart application.

6. Risk Analysis

As with every new system, there are a number of security vulnerabilities that can be associated to threats. These concerns can have different root causes and affect one or more aspects of the COSMOS environment. As COSMOS tries to answer a few of these concerns, a detailed risk analysis has been performed which covers both operational and functional security. Each risk is categorized and based on an assigned grade it is mitigated if is valid for the COSMOS environment.

Derived from this risk analysis are thus the security requirements which form the basis for work-package 3 (End-to-End Security and Privacy) and which can be found also in the Requirement list of D2.2.1. Using the risk analysis as a starting point, enabled the overall system security to reach further into the system and increase the confidence level.

As depicted in Figure 4, the risk analysis is based on the risk analysis recommendations of the National Institute of Standards and Technology [7].

For this reason the first step is to define the scope of the effort. In this case the scope or goal of COSMOS is to ensure the availability of the COSMOS platform to all involved parties while maintaining proper privacy and security for the exchanged data. As COSMOS relies on IT infrastructure there is a threat concerning the availability of the IT infrastructure itself. Still, as COSMOS is a “user” of the IT equipment, this risk, regardless of its rating, is considered a risk for the IT infrastructure provider and not for COSMOS itself thus no mitigation plan will be provided for it.

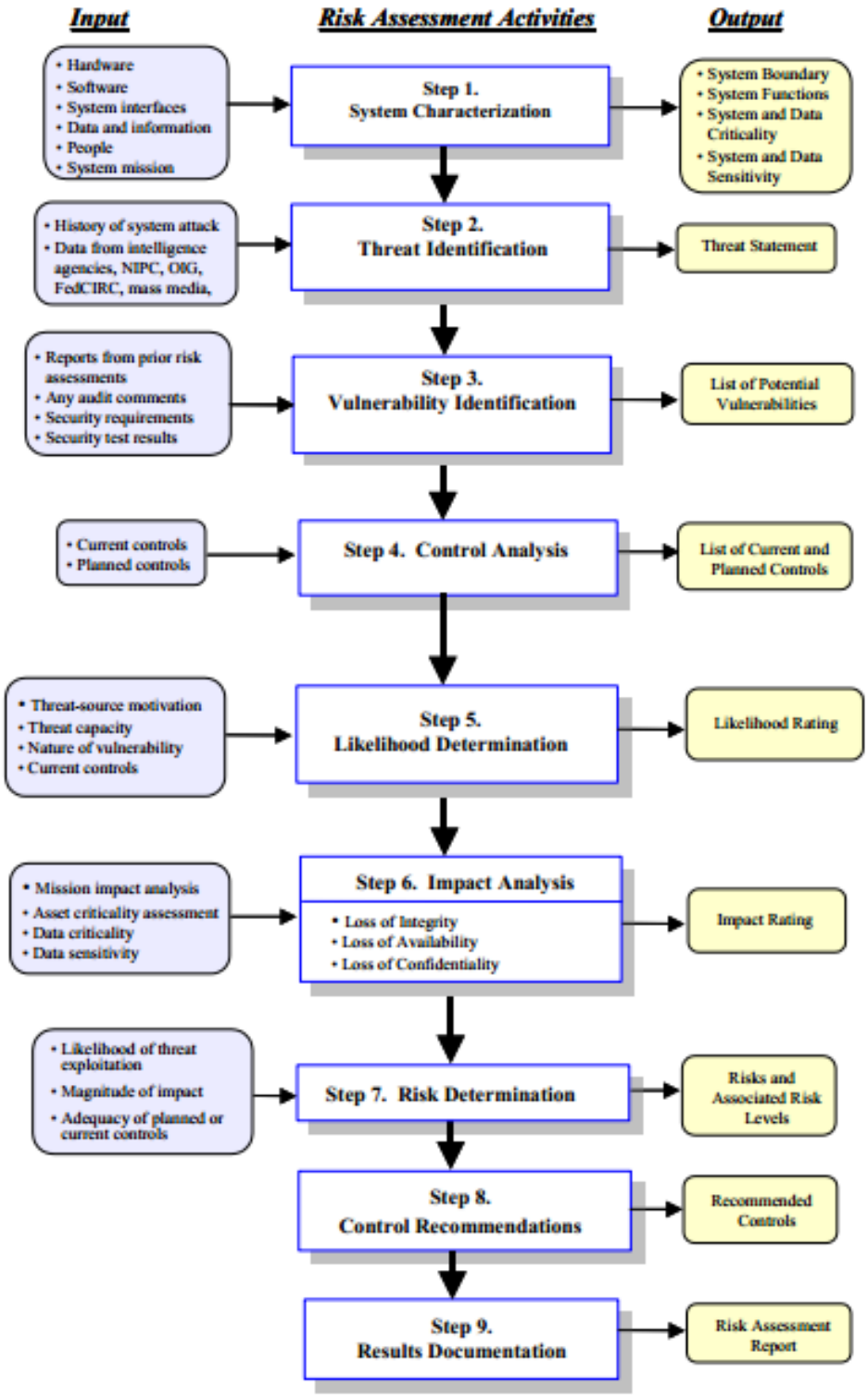

Figure 4. Risk Analysis Flowchart

Table 1. Identified Risks

No.	Risk	Risk Rating
1	Unavailability of IT infrastructure	HIGH
2	Loss of data due to unattended access to COSMOS	HIGH
3	Loss of data due to improper data handling	LOW
4	Loss of confidentiality due to improper user management	HIGH
5	Loss of confidentiality due to improper data handling	LOW
6	Loss of data integrity	MEDIUM
7	Unattended access to COSMOS which might compromise the system availability	MEDIUM
8	Un-trusted VEs might get access to COSMOS	MEDIUM

Table 2. Security Risk Heat Map

		Potential Impact			
		Low	Medium	High	Critical
Probability	Critical				
	High			R2, R6, R7	
	Medium	R5	R3	R4	R8
	Low			R1	

R1 - Unavailability of IT infrastructure

The risk of failing IT infrastructure in case of unlikely events or improper handling (e.g. non-working fire extinguishers) is a real threat which however is not hanfled in COSMOS. This risk is associated with the IT infrastructure provider and is only considered for safety concerns that are proper data backup. As IT infrastructure is not a research topic within COSMOS it is not only considered as a threat and no direct mitigation techniques are applied.

R2 - Loss of data due to unattended access to COSMOS

Unattended access to an IT system is always a probable threat. COSMOS handles sensitive data and acts as a link between different parties thus proper data manipulation is a must. Unattended access might come either from an attacker, using any means possible (e.g. common hacks, readily available on the market) or from normal users which find security

breaches. In order to mitigate this risk, strong security, in form of unique key based authentication, is enforced throughout COSMOS.

R3 - Loss of data due to improper data handling

As data forms the main payload of COSMOS the risk of having data loss events is a reality. In order to mitigate this risk, proper data management (e.g. the Data Bus) mechanisms are in place, which prevent these sorts of events. These mechanisms assure that data is checked for integrity and that the non-repudiation concept is enforced throughout the system, even if no cryptographic primitives are in place.

R4 - Loss of confidentiality due to improper user management

Clients form the main input and output of COSMOS – COSMOS exists because and for clients. Therefore, user management is the basic building block that COSMOS relies on. Clients feed in data but also consume it. In case of an IoT platform, like COSMOS, proper user management assures the availability and safety of both the platform itself and the handled data stream. In order to mitigate the risk of improper user management, COSMOS enforced strong security using key based user login and management schemes. Similar to the User Rights Management system in an operating system, COSMOS allows, based on access lists, only partial access to data for each logged client.

R5 - Loss of confidentiality due to improper data handling

Improper data handling is a trait of poorly developed IT systems. Mitigating this risk is a long discussed issue for which COSMOS will not provide direct answers. Instead the COSMOS project will rely on state-of-the-art mechanisms and standards in developing the COSMOS platform and assuring proper data handling. This risk was considered from the very first architectural drafts and will be part of the main “drivers” in further developing the COSMOS platform.

R6 – Loss of data integrity

Data integrity is a critical issue in any IT system. In order to mitigate this risk, the system architecture enforces data integrity checks at module, sub-system and system level.

R7 - Unattended access to COSMOS which might compromise the system availability

Unattended access in this case might compromise the system availability (e.g. a security attack which prevents COSMOS from handling client requests) if successfully carried out. In order to mitigate this risk the overall COSMSOS environment relies on strong cryptography. All involved parties must authenticate for any operation which is performed within COSMOS. Even more, clients are only allowed to access data to which they have rights to (e.g. are the rightful owner or have access to it) .

R8 - Un-trusted VEs might get access to COSMOS

As stated before, unattended access to the system might pose a risk to the COSMOS platform. The same is valid also for VEs which might be improperly enrolled and can provide ”fake” data streams. This risk is mitigated by using a unique API over all VEs and secure authentication mechanisms. As each VE has a unique, traceable security key, the risk of un-trusted VEs being enrolled is reduced drastically. Having a unique API is on the one hand side a constraint but on the other hand side it enhances security and tractability which are must-have traits for any COSMOS partner.

Based on this risk analysis, the WP3 requirements were derived. They form the basis for the security architecture which COSMOS relies on.

7. System Capabilities, Actors and Use Cases

The functional and non-functional requirements of the platform have been derived from the user requirements set out by the end users of the platform and are presented in detail in D2.2.1. Based on these requirements the current chapter gives an overview of the basic capabilities of the COSMOS platform. Moreover, in this chapter we present the main actors as identified in D2.1.1 and the interactions they have with the COSMOS system in the form of UML use cases. In each case a table is used to describe the use case and provide information such as the actors involved, assumptions that may be made, the sequence of actions that need to be performed as well as any other non-functional considerations.

This non-exhaustive set of interactions is intended to form the basis for the initial specification of the system architecture.

7.1. High Level Capabilities of COSMOS

Based on the requirements collected as part of D2.2.1 the following capabilities have been identified:

1. ***Security & Privacy: Integrity*;** COSMOS shall provide the ability to check data for functional correctness (e.g. identify defect and/or disconnected sensors and/or devices);

2. ***Security & Privacy: Data protection*;** Data must be "secured" in order to allow a high enough security level:

    a. eavesdropping -> encryption
    b. data modification -> Encryption + integrity checks
    c. replay attacks -> integrity checks
    d. identity theft -> encryption + authentication
    e. non-repudiation -> digital signature + encryption + authentication

    Moreover, the use of standard communication interfaces and operating systems minimizes the risk;

3. ***Security & Privacy: Key Management*;** COSMOS shall provide the means to create, manage and store security tokens. Each VE with a hardware security board should be able to use the hardware security features as a root of trust - software should only handle the high level security operations;

4. ***Security & Privacy: Authentication*;** COSMOS should provide authentication mechanisms for the functionalities provided, including such aspects as cloud storage and meta-data search;

5. ***Data Storage*:** The COSMOS platform should provide the means to collect raw data and persist it in a format that is suitable for subsequent analysis. Metadata should automatically be added to the data and can be later used for searching;

6. ***Data Analysis:*** COSMOS should be able to provide the ability for application developers to specify application specific analysis that can be run close to the data;

7. ***Raw data processing:*** COSMOS should be able to provide processing capabilities on the raw data. It should therefore allow VEs to publish data to COSMOS and ask for specific processing to be executed on them. This can include functionality such as event identification and complex event processing that can ultimately be used for situational knowledge acquisition;

8. ***Semantic Discovery:*** The COSMOS platform should allow VEs to find other VEs based on a set of criteria across different administrative domains. By annotating VEs with semantic information, describing their functional and non-functional attributes, applications are able to have enhanced knowledge of the services that they can employ without needing to be configured a-priory, but by being able to find service at run-time. The description of VE can also include geographical information that can be used, as spatial relations are of prime importance in the physical world. While VE discovery is a key aspect for the COSMOS environment, the semantic approach will not be limited to the VE description only. All the relevant building blocks of the COSMOS environment will be semantically described (applications, data bus topics, IoT services etc.) thus increasing the discovery as well as the analytics options of the platform;

9. ***Social Monitoring*****:** The COSMOS platform needs to be able to track the evolvement of VE's relations. This should be done with the least overhead. Therefore a mechanism needs to be developed that will allow monitoring the status of VEs and their interactions with other VEs or other COSMOS entities;

10. ***Self-management Coordination****:* The management of VEs in COSMOS needs to be done in a decentralized and autonomous way;

11. ***Prediction:*** Data produced by VEs might be volatile, and therefore incomplete. COSMOS should provide the ability to VEs to publish their data and ask for predictions mechanisms to fill these gaps either through interpolation or extrapolation;

12. ***Experience exchange:*** VEs should be able to exchange their experiences. This could take many forms such as exchange of security information, reputation scores or action plans;

13. ***Social aspect of VEs:*** VEs in COSMOS collaborate and form communities (which for example share experience and solutions to similar problems or goals). COSMOS should be able to analyse these structures, derive characteristics, such as the importance of a VE and take decisions based on them. Part of the social characteristics of VEs is its reputation. Therefore, the COSMOS system should offer mechanisms to build and maintain objects' reputations (according to various criteria). In addition the system must be able to quarantine an object, the reputation of which, does not meet a given criterion.

7.2. COSMOS Actors

As is defined in D2.2.1, a set of actors have been identified. These include the following:

VE/GVE: As described in our domain model, VEs are a representation of physical objects in the Cyber-world. VE's may have their own goals and be equipped with an internal logic in order to achieve their goals. VE's get perception using sensors and can impact their environment or undertake physical actions using actuators. The sensors and actuators are exposed as IoT services. Finally VE's may interact with other VE's for various purposes like collaboration (sharing a common goal with other VE's), cooperation (getting help from other VE's in order to achieve their own objective). GVEs are aggregations of VEs and function like VEs in the way they interact with the system.

VE Developer: The VE developer is responsible for building COSMOS compliant VEs. He/she is responsible for respecting the APIs needed to register a VE with the platform and make it accessible to other VEs though well-known interfaces.

COSMOS-enabled Application: A COSMOS-enabled Application is an application that utilizes the offerings of COSMOS. An application may be able to access physical entities (exposed as VEs) that belong to different administrative domains than the application itself. It is able to discover such VEs through COSMOS and engaged them through COSMOS defined APIs. Moreover, applications, as VEs, are able to use the services of COSMOS, such as situational awareness or prediction.

Application Developer**:** The Application developer provides the applications utilizing the COSMOS offerings or mixing it with input from the VEs. The application developer can create applications based on abstract VEs which will be later mapped to the real ones, once the application is configured, thus providing the means of more generic applications.

COSMOS Platform Provider**:** COSMOS platform provides services through a standard interface to applications and VEs. The provider may provide the platform on a pay per use basis, based on the different services applications use.

End-User: The end-user is the entity that uses the COSMOS-enabled applications. The end-user runs the application which may use preconfigured VEs or VEs discovered at run-time.

Other roles, such as Equipment provider or network provider may exist. These are considered out of scope for the project

7.3. System Use Cases

Following the capabilities and actors presented earlier, a high level view of the way the COSMOS system interacts with external actors is presented in the following paragraphs.

7.3.1. Registration

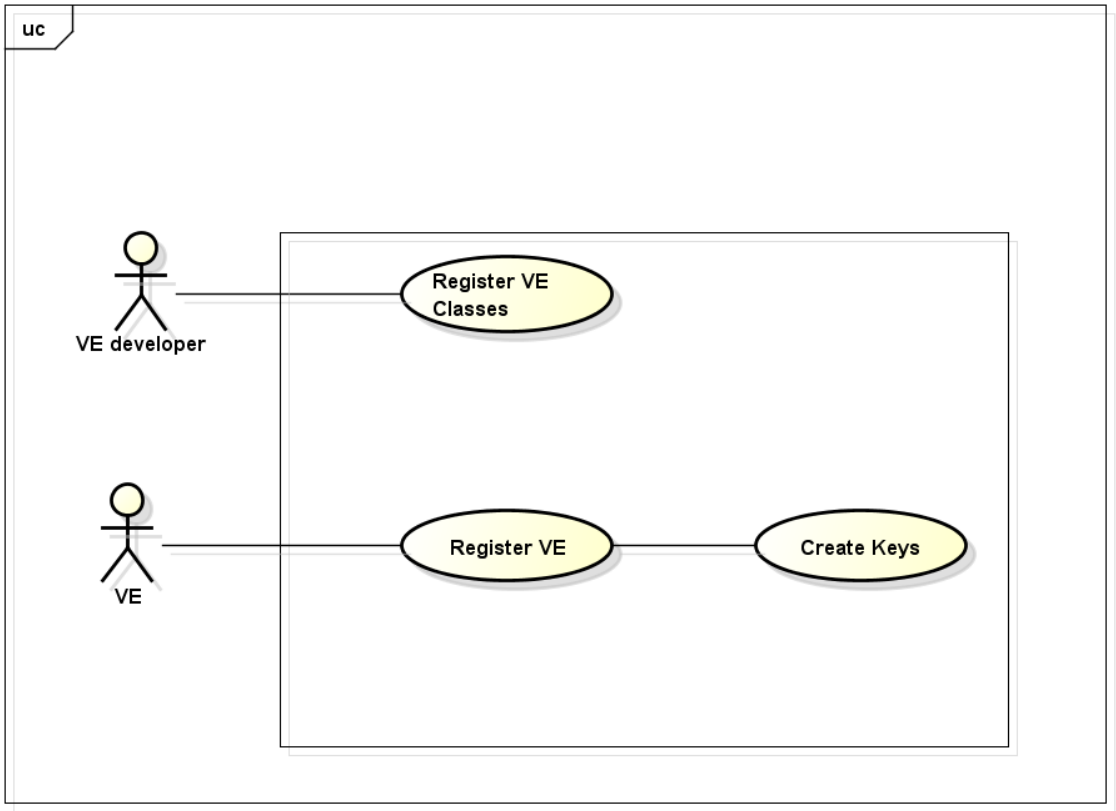

Figure 5. Registration Use Case

Table 3. Registration Use Case Details

Use Case	VE Registration
Description	Every VE needs to be registered with COSMOS. During registration the VE (or VE developer) provides to COSMOS a semantic description of itself. This description is stored in a semantic registry. The description will have information such as services and IoT-services exposed, capabilities, etc. Also during registration a unique identifier is created that can be used identify the VE and public/private keys are generated that can be used to authenticate the VE. Also a VE developer may be possible to register VE classes. These can be considered as “VE-templates” that can be later used to instantiate specific VEs.
Actors	VE, VE developer
Assumptions	-

Steps	1) A VE developer registers to COSMOS a VE class that can be used as a template. This step is optional. 2) The VE registers itself to COSMOS providing a semantic description. 3) COSMOS returns to the VE a unique identifier and a security token.
Non-functional	-

7.3.2. Publish Data

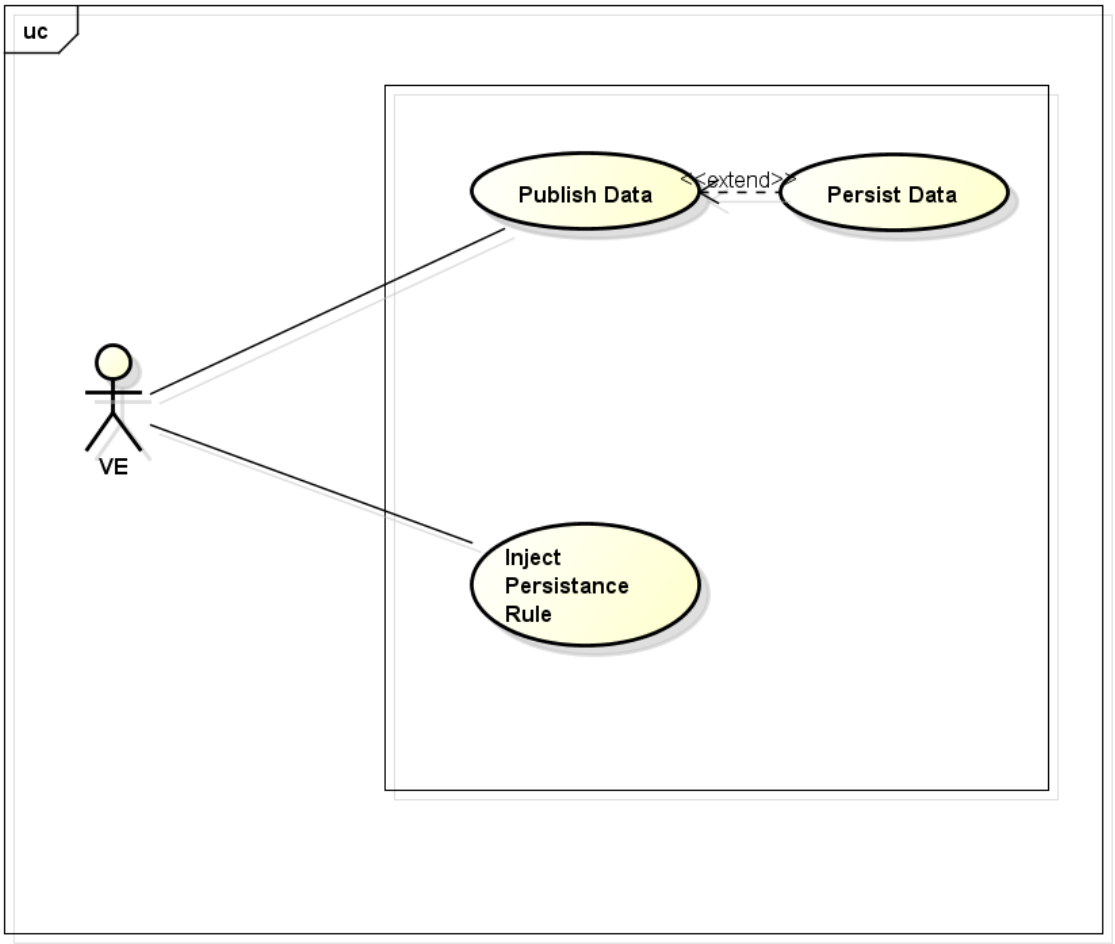

Figure 6. Publish Data Use Case

Table 4. Publish Data Use Case Details

Use Case	Publish Data
Description	A VE should be able to publish data to COSMOS. Some of these data may also be persisted in the Cloud Storage. These data can be consumed by other VEs or COSMOS components.

Actors	VE
Assumptions	The VE is registered and authenticated.
Steps	1) The VE publish data to COSMOS 2) Data are routed to subscribers 3) Data may be persisted based on persistence rules injected by the VE along with metadata generated by the system
Non-functional	-

7.3.3. Topic Management

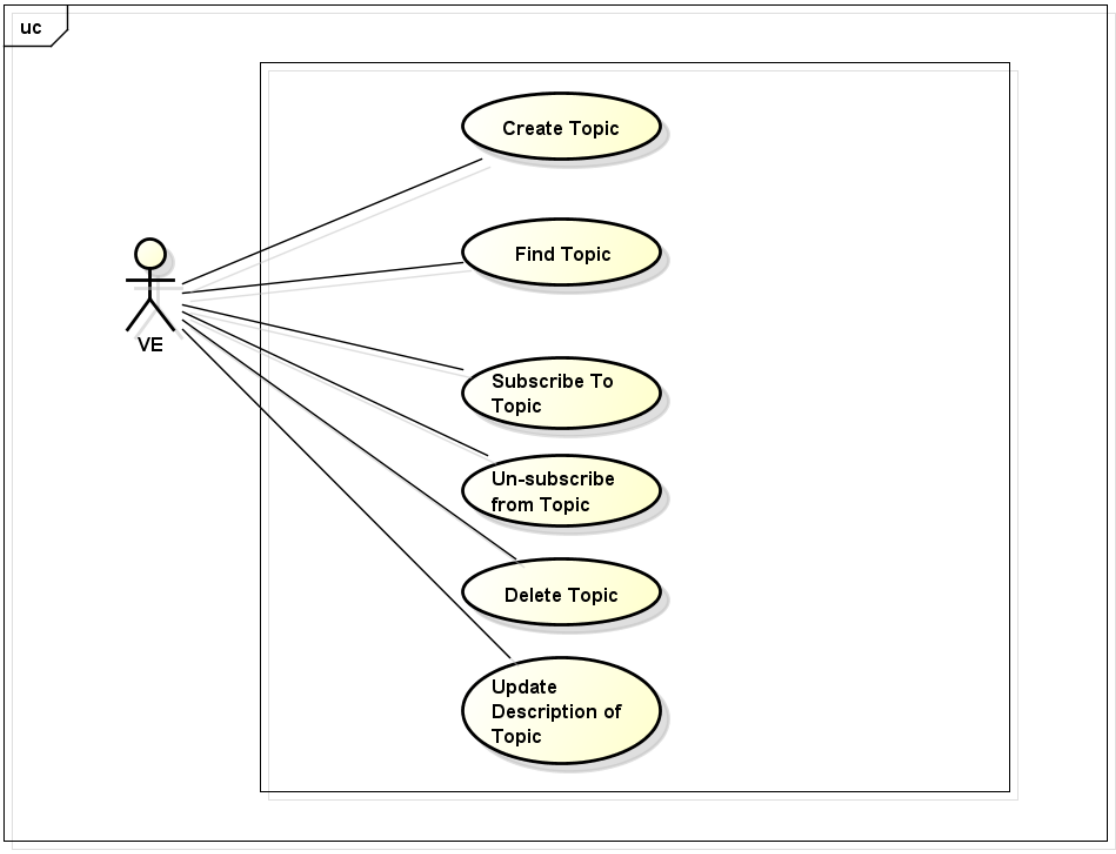

Figure 7. Topic Management Use Case

Table 5. Topic Management Use Case Details

Use Case	CRUD on Message Topics
Description	The COSMOS Platform will provide the ability to entities to use a message routing component. Therefore a VE should be able to: • Create Topics on a message bus and semantically annotate them • Find Topics based on their semantic descriptions

	• Subscribe to Topics and receive events • Un-subscribe from Topics • Delete Topics • Update Semantic description of a Topic A Topic can be hosted either within the COSMOS platform or externally, in which case only the semantic description of the topic along with the endpoint is hosted within COSMOS.
Actors	VE
Assumptions	The VE is registered and authenticated.
Steps	-
Non-functional	-

7.3.4. Find VEs

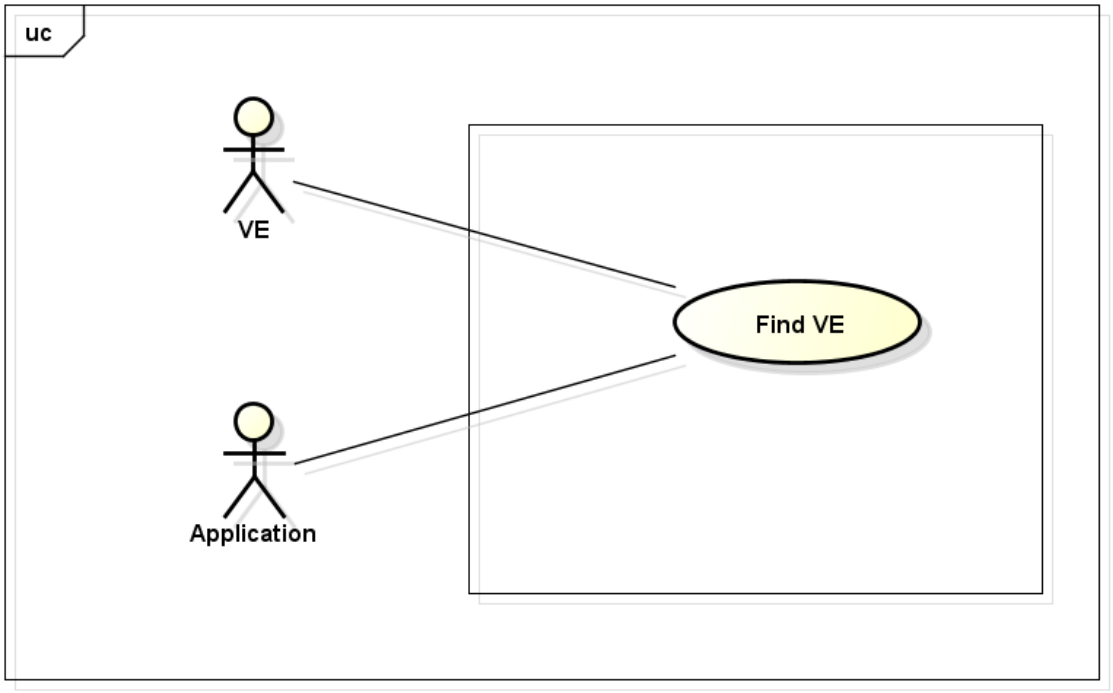

Figure 8. Find VEs Use Case

Table 6. Find VEs Use Case Details

Use Case	Find VEs
Description	A VE or Application will be able to query the COSMOS platform and find other VEs based on their semantic description, IoT-services they expose, experience they can share etc.

Actors	VE, Application
Assumptions	The VE is registered, authenticated and authorized. VEs are semantically represented within COSMOS.
Steps	-
Non-functional	-

7.3.5. VE to VE communication.

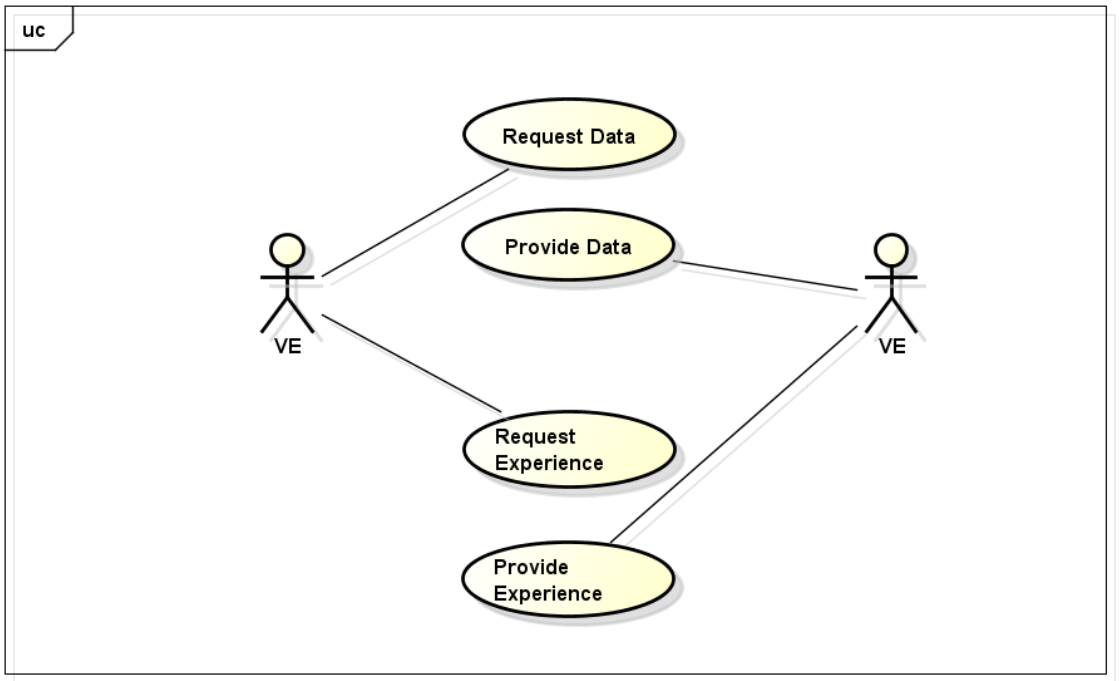

Figure 9. VE to VE Communication Use Case

Table 7. VE to VE Communication Use Case Details

Use Case	VE to VE communication
Description	VEs are expected to communicate directly with one another. They should therefore expose specific services that are well known to all VEs. Such service include requesting collaboration, by asking another VE to provide its data either directly or by publishing to the COSMOS message routing service, and requesting another VE's experience.
Actors	VE
Assumptions	The VE is registered and authenticated. VEs expose well known services.

Steps	-
Non-functional	-

7.3.6. Prediction (Interpolation / Extrapolation)

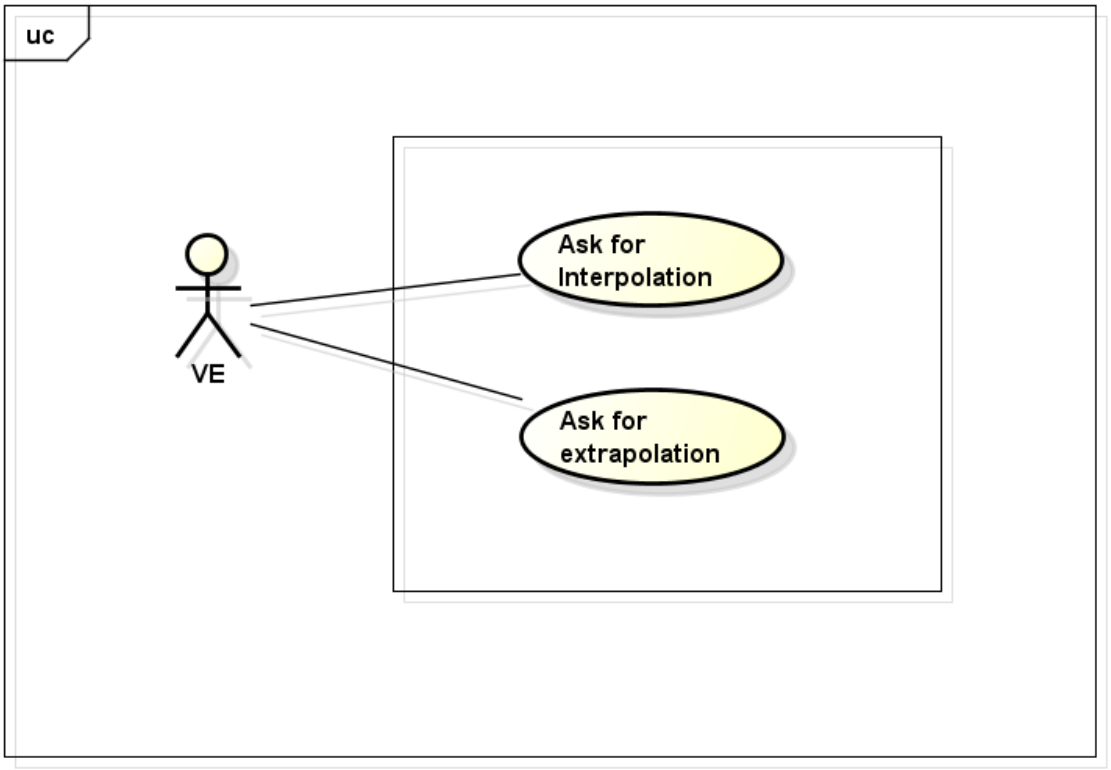

Figure 10. Prediction Use Case

Table 8. Prediction Use Case Details

Use Case	Prediction
Description	In some cases, the COSMOS platform will provide the ability to VEs to ask for predictions on the observed measurements. This can be in the form of predicting future or predicting/interpolating missing values. A simple example can be one VE might be interested in to find his time of Arrival at destination using particular route according to current traffic scenario. It can register itself to COSMOS platform and get the required information by following standard query format.
Actors	VE
Assumptions	The VE is registered, authenticated and authorized.
Steps	1) The registered VE will search for relevant prediction model which is stored semantically 2) It will pass the query in a standard way defined which will include

	relevant model and VE information using the interface provided
Non-functional	-

7.3.7. Situational Awareness

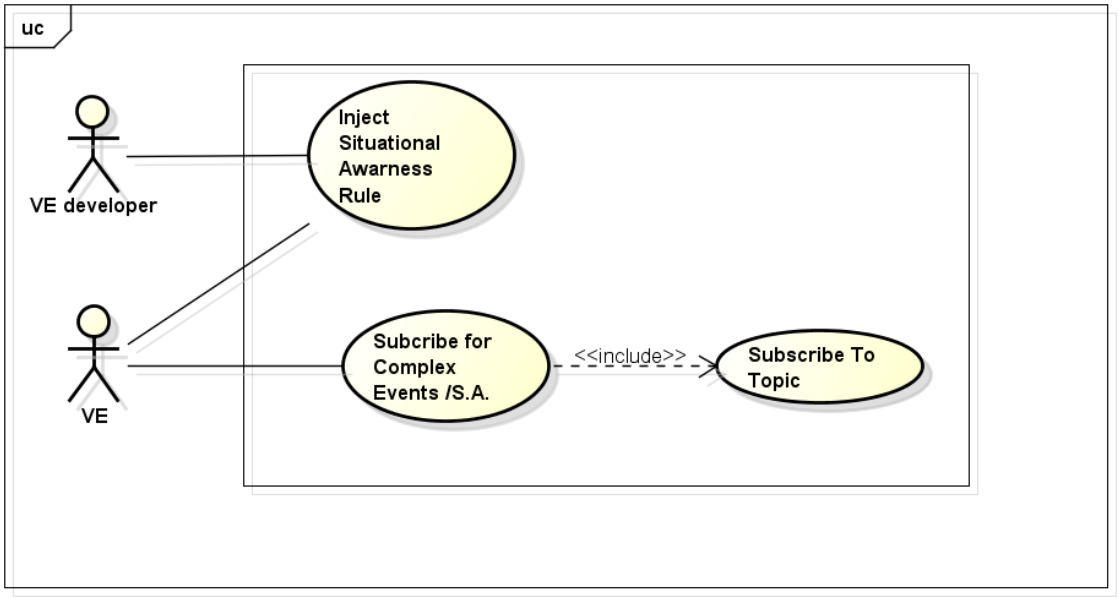

Figure 11. Situational Awareness Use Case

Table 9. Situational Awareness Use Case Details

Use Case	Situational Awareness
Description	The COSMOS platform will allow VEs to acquire situational awareness based on Complex Event Processing Techniques
Actors	VE
Assumptions	The VE is registered, authenticated and authorized.
Steps	1) A VE or VE developer injects rules to the system specifying how complex events should be detected 2) The VE subscribes in the topic related to the complex events 3) The complex events are published in the message bus
Non-functional	-

7.3.8. Data Analytics

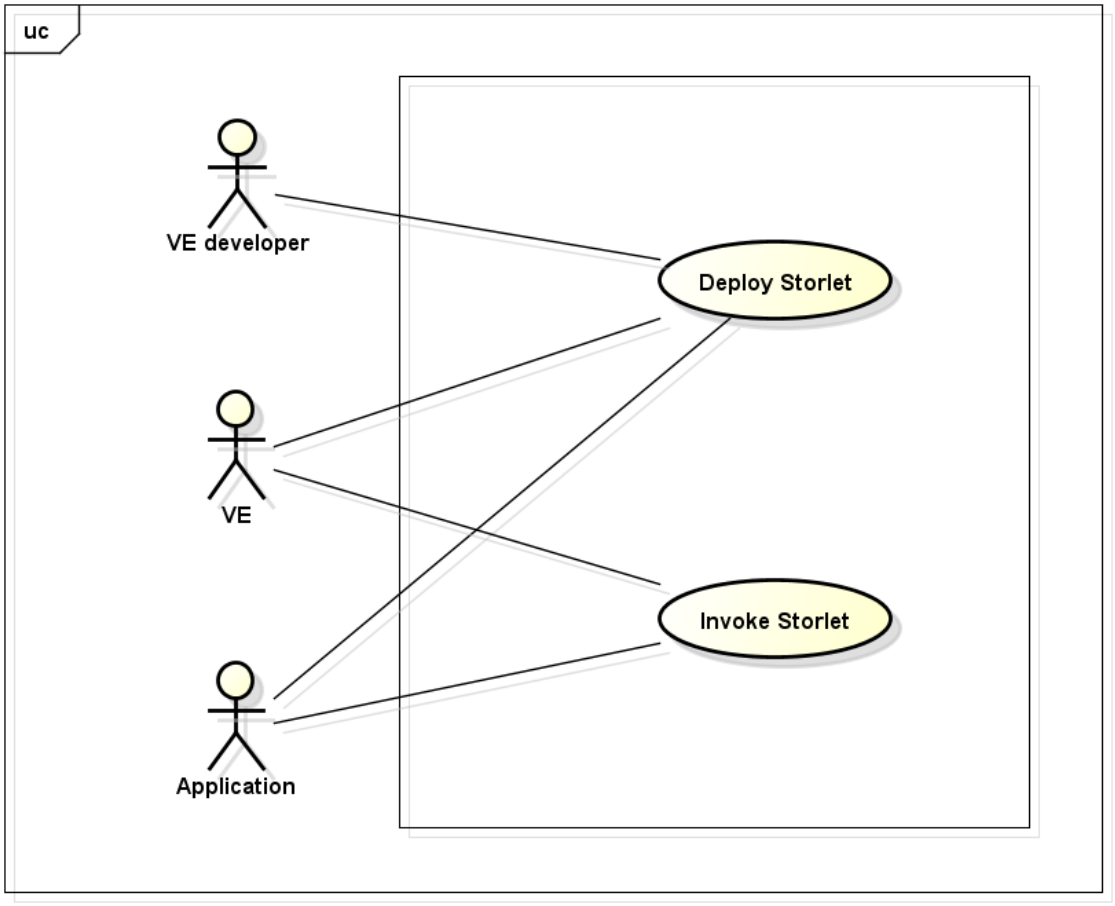

Figure 12. Data Analytics Use Case

Table 10. Data Analytics Use Case Details

Use Case	Data Analytics
Description	The COSMOS platform will allow the definition and execution of data analytics on stored objects. This will be achieved through the notion of storlets, which are computational objects that run inside the object store system.
Actors	VE developer, VE, Application
Assumptions	The VE is registered and authenticated.
Steps	1) A storlet is deployed at the Cloud Storage side. This can be done by any actor that has permission to access the storage using the implemented API. 2) A storlet is triggered. The storlet should run on the specified stored objects and return its output.
Non-functional	-

7.3.9. Cases and Problems Management

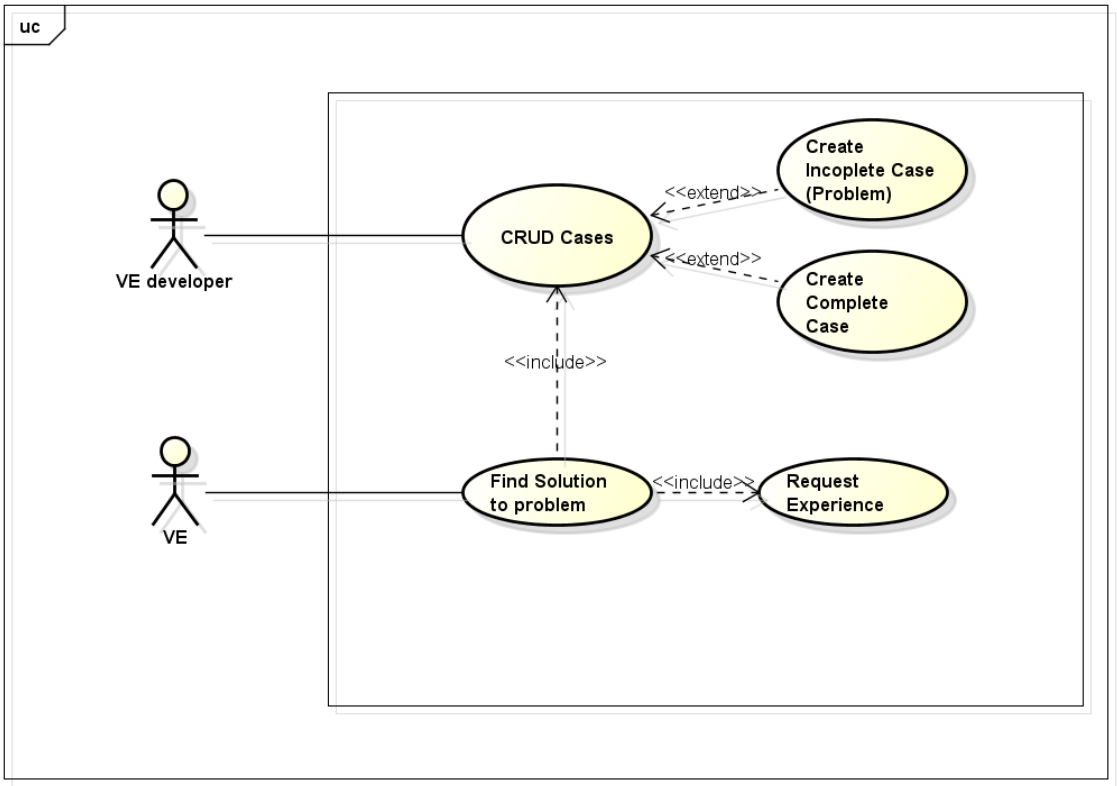

Figure 13. Cases and Problems Management Use Case.

Table 11. Cases and Problems Management Use Case Details

Use Case	Cases and Problems Management
Description	The COSMOS platform will allow VEs to acquire situational awareness based on Complex Event Processing Techniques
Actors	VE developer, VE
Assumptions	The VE is registered, authenticated and authorized.
Steps	1) A VE developer Creates Cases a)a Case may be complete (containing a Problem and a Solution) b)a Case may be incomplete (Containing only a Problem) 2) A VE asks to find a Solution to a Problem. 3) COSMOS may engage other VEs for their experience in the form of Solutions to a Problem. 4) If a Solution is found the Case may be updated.
Non-functional	-

8. Functional View of COSMOS

8.1. High Level View

Based on the previous analysis, the COSMOS platform functionalities can be grouped into the following groups:

1. Security, providing functionality such as key management and authentication;
2. Data Management, providing functionality such as persisting data on the cloud storage;
3. VE Management, providing functionality such as discovery of VEs based on semantic queries;
4. Analysis, providing functionality such as prediction and situational awareness.

The high level view of COSMOS is depicted in Figure 14. A colour code is used to group the components. Worth noting that, from a deployment point of view, the COSMOS framework will include services both at a platform level as well as within VEs. Therefore, COSMOS will also require VEs to provide a set of common services and interfaces, such as Experience Sharing, called COSMOS-services. These services will form the basis for the decentralized and autonomic properties envisioned by the project.

A VE may optionally also host its own Message Bus, apart from the one offered by COSMOS. In this case the VE is responsible for maintaining the routing functionality of the Message Bus, but the VE is able to use the COSMOS functionality for Semantic Topic Management. This way other entities can use the COSMOS functionalities to discover the topics offered by the VE. Moreover, pre-processing functionalities may also be included in the VE. We have chosen to depict this in the high level view of the architecture as pre-processing is an essential part of the VE lifecycle.

One of the key design choices that have been made is that the different components of COSMOS will offer a REST-full HTTP interface. The choice of HTTP REST has been made as it is widely adopted in the IoT and Cloud domains, enables interoperability and allows for easy development. For the same reason JSON has been chosen as the data format for Year 1, even though other options might be explored future iterations of the architecture.

It should also be noted that there are multiple actors relating to COSMOS, such as VEs, Applications and human end-users. All of these actors, may interact with some of the functionality offered by COSMOS. For example, a human actor may access the Data Storage, retrieving for instance a data object, the same way that a VE may access it. Therefore, in the following paragraphs the word *user* may denote either a human or machine user. Only where the distinction is crucial will this be specified.

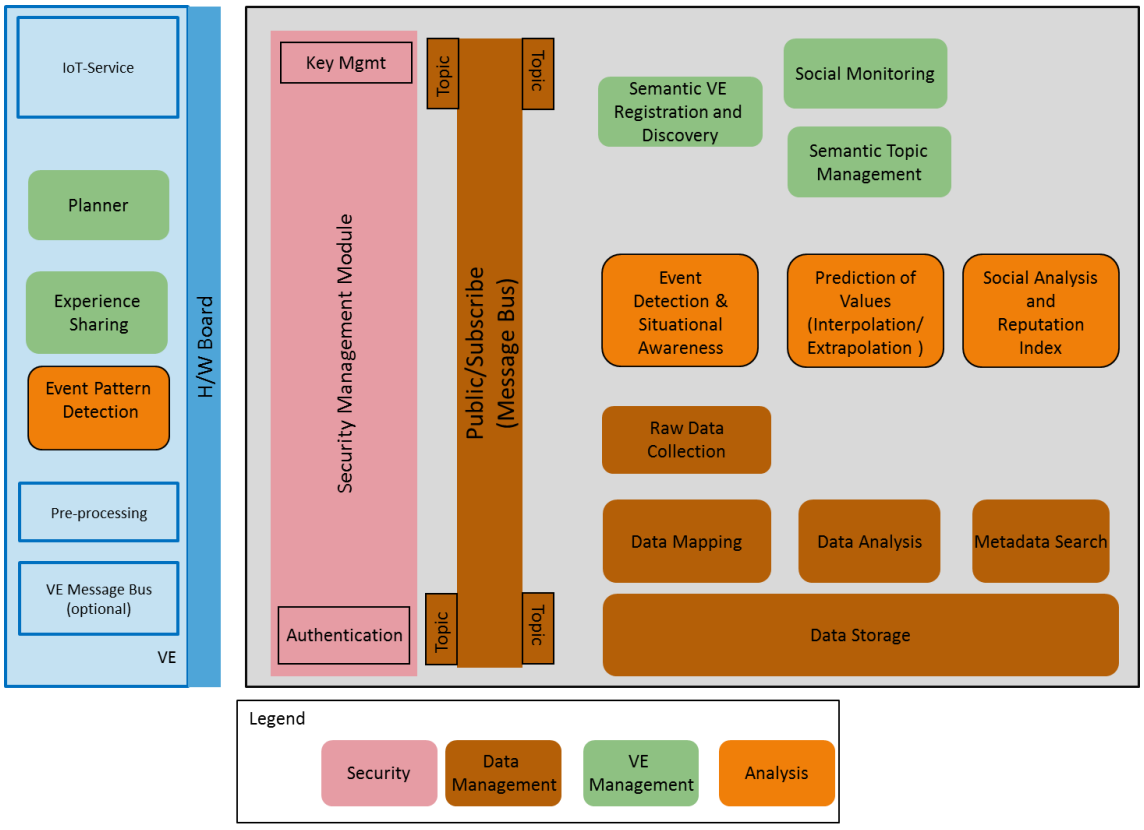

Figure 14. High Level Architecture

8.2. Functional Components

In this section we give an overview of the Functional Components that make up the COSMOS eco-system. A description of each component is given, followed by the functionality the component is expected to provide along with a description of the interactions the component may have.

8.2.1. Security Management Module

8.2.1.1 Description of the component

The COSMOS environment can be viewed as a black box which provides various services to VEs. These services handle three basic data types:

- Security critical: information is both secret and privacy critical;
- Security aware: information which can be secret but is not privacy critical;
- Non-secure: public information which contains no secret and is not privacy aware.

As a black box, COSMOS needs to handle the three basic data types, thus it needs to provide following services:

- Authentication: while VEs need to be authenticated into COSMOS, data has to be genuine. In this context, both communication parties need to validate each other in a consistent manner;
- Integrity: authenticated data has to be accurate and consistent over its life cycle, from source to destination;

- Non-repudiation: none of the parties should be able to deny its actions within COSMOS;
- Availability: the information needs to be accessible when required and with minimal delay.

Therefore, the security management module serves as a generic security gateway for the COSMOS environment. Following an iterative design approach, the Security Management Module provides basic security mechanisms which strengthen the COSMOS environment.

The goal of the Security Management Module is to allow only authenticated clients access to the COSMOS environment thus enabling the COSMOS services, running within the COSMOS environment, to trust the information.

8.2.1.2 *Functionality/Interfaces*

The Security Management Module which will run in within the COSMOS platform consists of:

- A key management and generation sub-module: each client has a unique key which is used for authentication purposes. The key is used to both authenticate the client as well as to encrypt the information flow between the two communication parties;
- User management sub-module: similar to user level permission management, this service will allow authenticated clients seamless access to the data storage object while "blending out" information which they are not the rightful owner of.

These two sub-modules form the foundation of the Security Management Module. Using a REST-full interface, the Security Management Module is configured by an administrator which is already authenticated within COSMOS. VEs and human users can use services exposed by the Security Management Module such as:

- Key generation and exchange service (e.g. Diffie-Hellman key exchange mechanism);
- Authentication service (e.g. using SSH or a REST interface over HTTPS).

VEs which are going to be used within COSMOS need to meet certain security criteria in order to be able to provide or consume information. Thus highly secure VEs need to be equipped with either a Hardware Security Board or a Software Security Module. Each VE will use the same interface for configuration and data exchange, but will use a read-only tag to signal its security level and thus trustworthiness.

8.2.2. H/W Board

8.2.2.1 Description of the component

The Hardware Security Board consists of a physical hardware device which provides the link between sensors (data generators) and the COSMOS environment/platform. The Hardware Security Board can be either attached to one sensor or can be a hub for an entire collection of sensors (e.g. temperature, pressure, humidity, surveillance cameras, etc.).

In order to provide high trustworthiness a hardware coded security forms the foundation of the HW Security Board. This layer provides basic security primitives which are used by software drivers and applications as the so-called "root of trust".

The Hardware Security Board consists therefore of a FPGA platform device (e.g. Xilinx Zynq [8]). This provides the necessary means for developing the hardware coded security components while making use of standard, state-of-the-art computing processors. The operating system of choice is Linux which provides not only the necessary platform for

developing the high-level software applications but also enables the usage of the security hardware modules.

The hardware components within the Hardware Security Board provide:

- Secure Boot:using encrypted flash memories and device-unique keys, enables only trusted software applications to be executed;
- Secure Storage: allows for on-chip key storage while protecting against common security attacks which target key recovery;
- Secure execution: using hardware partitioning schemes, unsecure software applications are sandboxed, thus protecting the rest of the computing platform from malicious software or malware;
- Cryptographic hardware accelerators: allow for fast, on the fly encryptions and decryptions to be performed, without performance loss.

8.2.2.2 *Functionality/Interfaces*

The connection between the Hardware Security Board and the outside world can be realized using standard interfaces such as:

- Ethernet
- WiFi
- ZigBee
- I^2C
- SPI
- Analog front-end (e.g. analog-to-digital converters).

The security functionality is provided as a service and can be routed over any of these physical interfaces. Data packets can be transported using any software protocol available given that the necessary glue-logic is implemented. Therefore common transport layers are applied such as HTTP, HTTPS, SSH, etc.

The Hardware Security Board is enrolled, as any other VE, using the Diffie-Hellman key exchange algorithm. With the help of SSH or HTTPS as transport layer and a RESTful interface, the HW Security Board is configured.

8.2.3. Privelets

8.2.3.1 Description of the component

Within the context of COSMOS virtual entities will be able to contact each other as well as with the platform. This information exchange has a high possibility of violating the user's privacy in a way that he is not aware of.

Our goal is to ensure that every VE (and therefore its user) shares only the intended information and leaves out all other that is considered as private and is believed to affect its privacy.

The design and implementation of privelets follow an iterative approach. As a first step, messages about to be sent from a VE are analysed and specific fields are filtered based on specific configuration options.

8.2.3.2 Functionality/Interfaces

A privelet is a mechanism embedded (or external) to the VE which acts as a filter.

At first, let us see what happens before the data exchange. A VE, which participates in the COSMOS platform, generates information that sends to other VEs and to COSMOS. Every message sent follows a specific format that is consisted of different tags. A configuration file is used to specify the level of privacy for each tag.

So according to each tag's configuration, a privelet lets all the public tags to be exchanged and it keeps safe or discards all the private ones by generating a new message which looks like the one received by the VE before the filtering process, without the private tags.

In other words, data generated from the VE must pass through that filter and then be exported to another VE or the COSMOS platform. The VE generates data with a certain interval update and wants to share it through our platform or between other VEs. It generates a JSON file where all those data pass through this mechanism called privelet. Moreover, when the physical entity also has a UI, such as a mobile device, it will be used to ask the user to provide configuration options. When a new message is received, it is analysed and the information contained are presented to the user, who is able to define which parts are private and which are not. After that any data exchanged from the VE, will have the same configuration regarding the privacy level of each tag in order to decide what data to send to the platform and what to keep. The privelet will not change each tag's privacy level unless the user calls it again and re-configures it. After this configuration the data exchanged will be filtered and exported with the privelet configuration.

By filtering all the considered as private tags we could eliminate the chances of de-identification caused by quasi-identifiers. A quasi-identifier is a piece of information that is not by itself unique identifier but when combined with other quasi-identifiers they can become information of a personal identification exposing the user's private life's information.

So if someone would like to choose all his tags to be sent on the platform, it is highly possible that anyone could use databases that store other information regarding his personal life and are linked to the data exchanged though COSMOS platform and build a map about him being able to track or predict the further moves on his life. By using the privelets, we give the user the ability to avoid all this and in this way we expect to make data transferring and exchanging anonymous and ensure privacy for our users. So the privacy level and protection stays on the user layer.

8.2.4. Message Bus

8.2.4.1 Description of the component

COSMOS will integrate many distributed clients (Virtual Entities). An important aspect of Virtual Entities is that they are independent and should be easily capable of being integrated so that they work together and share experience and are easily discoverable.

To enable each Virtual Entity and smart applications build on top of COSMOS platform to focus on particular comprehensive set of functionality and yet delegate partial functionality to other components or Virtual Entities, a message bus solution is used which aims to:

- Provide convenient infrastructure to integrate a variety of distributed Virtual Entities and internal COSMOS components in a simple way;
- Decouple the message publishers from those which are interested in the messages (subscribers);

- Support further COSMOS requirements such as orchestration, “intelligent” message routing, provisioning and maintain integrity of messages as well as reliable transport of messages.

8.2.4.2 Functionality/Interfaces

Reliable Communication

The message bus enables different and/or distributed clients to communicate and transfer information in a reliable way. Reliable Message Bus communication requires data to be immutable single atomic units and serializable so that they can be converted into a simple byte stream and transferred over the network. The message serialization plays a key role in reliable transfer based on two concepts:

- **Send and Forget**: publisher sends a message to the message bus and can be confident that at some point in time, receivers will receive this message. The message is initially stored in publisher’s computer;
- **Store and Forward**: the message bus transmits message to receiver’s computer and stores it there. In case of error, or intermediaries, transmit can be repeated as many times as needed.

Platform & language integration

The universal connectivity is the heart of the message bus pattern. Message bus provided by COSMOS will offer support for wide range of different languages, technologies and platforms as well as extension mechanism for simple integration of future technologies.

Asynchronous Communication Model

As mentioned in previous chapters, publishers and subscribers are decoupled in a way that the publisher does not know who (if any) subscriber will receive a message. Same applies to subscribers which who may not be interested in the source of particular messages. Therefore neither publishers nor subscribers have to wait for underlying message bus to deliver actual message. This results in several important implications:

- Message Stack: the message transfer rate does not depend on message consumption rate, therefore if transfer rate is higher, messages are serialized and stacked on subscriber’s computer;
- Flow Control: the message bus can automatically decrease message transfer rate by exerting backpressure on connections that are publishing too fast. The adverse effect on publishers is minimal if high rate is only temporal i.e. in form of spikes;
- Disconnected Operation: this mechanism enables publishers to publish messages in offline mode. The messages are serialized on publisher side waiting until subscriber connection is available;
- Threading model: notification mechanism and immutable property of messages open the possibility for more efficient threading management compared to traditional RPC pattern.

Mediation

Both publishers and subscribers communicate to message bus only. Depending on a message exchange mechanism, subscribers receive shared messages from message bus without a need to connect to specific publishers.

8.2.5. Semantic Topic Management

8.2.5.1 Description of the component

COSMOS is intended to provide developers extended interoperability mechanisms and, as a result, the key building block required for building a new application will be semantically annotated. This applies IoT services, VEs, topics, COSMOS applications and other building blocks used either by the application developer or by the internal COSMOS components. The ability to describe of these blocks not only through simple metadata attributes but with rich semantic annotations using ontologies shared by different parties will facilitate the retrieval of relevant blocks and the use of inference mechanisms to support recommendation or adaptation mechanisms.

VEs, COSMOS applications, as well as the COSMOS platform itself can produce as well as consume raw and processed data. Depending on the application and the scenario, data will be transferred through peer-to-peer mechanisms as well as using a message bus approach. The latter mechanism is suitable mostly to those scenarios where data with a high potential of reusability (either directly or after a processing step) is made accessible to different parties. A simple scenario would be that where vehicles are reporting anonymously their location and driving conditions in order to be used for performing traffic condition monitoring. The resulting traffic information could then be published in near-real-time.

Nonetheless, the same raw traffic data could be used to compute various statistics about the traffic or analysis based on other factors such as construction works in the city, events, weather data, etc.

In such scenarios, the message bus communication pattern would be used, with data being published to different topics (e.g. raw traffic data could be published to a topic, while the resulting traffic condition data to another topic). These topics will be semantically annotated so that the nature of the published data, its producers, the update frequency, the underlying data types are described. The Semantic Topic Management component will expose the necessary functionality to allow the annotation of topics which are created into the message bus and the synchronization with the message bus so that topics on the message bus are always semantically annotated and the description of topic is always backed by a topic defined on the bus.

8.2.5.2 Functionality/Interfaces

COSMOS will provide the semantic stores which will be used to persist the description of the topics. Such stores are providing a SPARQL [9] querying interface for basic CRUD operations but also for complex requests specific to the semantic domain.

The Semantic Topic Management will provide wrappers over the SPARQL queries so that the description of the topics can be created, read, updated and with the use of a simple yet flexible API, thus hiding the complexity of the queries.

These operations will be used in conjunction Message Bus so that the topics being published are consistent with the semantic description and vice versa. The Semantic Topic Management will also provide the API required for topics retrieval operations based on different search criteria. This will be used either for direct topic retrieval on in conjunction with the Discovery or the Recommendation Components.

8.2.6. Raw Data Collector

8.2.6.1 Description of the component

The Raw Data Collector component is responsible for subscribing to topics in the Message Bus marked as persistent and storing them in the Data Store. It will aggregate data and periodically dump it in its raw format to the Data Store while collecting various statistics. This data will later be transformed by the Data Mapping component.

In Year 1, this component will be combined with the Data Mapping component, because the mapping used there will be straight forward.

8.2.6.2 Functionality/Interfaces

Message Bus topics can be marked as persistent in the Topic Management component in order to denote that they should be stored persistently in the Data Store. The Raw Data Collector component needs to be notified of all topics marked as persistent in order for it to subscribe to all such topics.

The Raw Data Collector component will receive data from the Message Bus in formats supported by COSMOS. For the first year, a JSON format will be supported. This component will write data to the Data Store component using the OpenStack Swift REST API [10].

8.2.7. Data Mapping

8.2.7.1 Description of the component

This component is responsible for periodically processing the raw data which are stored in the Data Store, in order to transform them into a format and data layout suitable for subsequent analysis and also to extract static metadata such as VE's IDs or timestamps. In addition, it annotates the data objects with enriching metadata coming from the social description of the VEs, which is stored in a semantic registry. The mapped data are stored persistently in the Data Store.

As mentioned above, in Year 1, the Data Mapping component will be combined with the Raw Data Collector component.

8.2.7.2 Functionality/Interfaces

The OpenStack Swift REST API will be used so as to create the data objects with their associated metadata, get the object details, get its metadata and update them. This is very important, because metadata like location, trust & reputation index and so on, can be dynamic.

Query languages, like SPARQL, can be used for extracting VE's social attributes, which are stored in the triple stores, during the registration phase in the COSMOS platform. This information is used to enhance the stored data object with proper metadata, allowing for better search functionalities and analytics on the persisted information.

8.2.8. Data Store and Metadata Search

8.2.8.1 Description of the component

The purpose of the COSMOS Data Store component is to persistently store COSMOS data and make it available for search and analysis. The open source OpenStack Swift object storage

software will be used in order to implement the COSMOS Data Store. Data will be organized into containers and stored as objects. In Year 2 and/or 3 of the project, the question of whether additional cloud storage frameworks are needed, in addition to object storage, will be examined.

In order to make metadata useful for applications one needs the ability to search for objects (or containers, accounts) based on their metadata key-value pairs. This functionality is not supported by Swift today. Therefore we extend Swift to index metadata and to support searching for objects (containers, accounts) according to their metadata keys and values.

8.2.8.2 Functionality/Interfaces

The OpenStack Swift REST API can be used for *Create, Read, Update and Delete* (CRUD) operations on containers and objects, and also supports annotating containers and objects with metadata. We extend this REST API to allow metadata search – a search request is a Swift GET request with a specific header denoting it as a metadata search and with certain parameters.

8.2.9. Data Analysis close to the storage

8.2.9.1 Description of the component

Data analysis on the persistent storage will be done using a storlet mechanism. Storlets are computational objects that run inside the object store system. Conceptually, they can be thought of the object store equivalent of database stored procedures. The basic idea behind storlets of performing the computation near the storage is saving on the network bandwidth required to bring the data to the computation.

Computation near storage is mostly appealing in the following cases:

- When operating on a single huge object, as with e.g. healthcare imaging;
- When operating on a large number of objects in parallel, as e.g. with a lot of time series archived data.

The storlet functionality in COSMOS is developed in the context of the Openstack Swift object store.

8.2.9.2 Functionality/Interfaces

There are three APIs of interest in the context of storlets:

- **The API one needs to implement when writing a storlet.** Currently storlets can be written in Java according to a specific storlet interface;
- **The API for deploying a storlet.** This allows the code implementing a storlet and other code it depends on, such as external software libraries, to be uploaded to the object store;
- **The API for invoking a storlet.** For year 1, we support invoking a storlet as part of a Swift GET request using the Swift REST API. Storlet parameters are provided using certain headers. In future there will be additional mechanisms for invoking storlets.

8.2.10. VE Registry

8.2.10.1 Description of the component

VEs are key building blocks in the COSMOS environment. They are consumers as well as producers and processors of data and are exposing IoT resources and services. Such IoT resources have features and operations which the VE are exposing so that they can be integrated into the COSMOS environment. Since interoperability as well as openness in the COSMOS environment are essential requirements, VEs will be semantically described so that they can be easily retrieved and that their capabilities and constrains are accessible, and understandable by other actors or components of the environment.

The VE Registry will make use of the semantic model elaborated by the IoT.est [11] project for the description of the IoT services and will be extended so that VEs are extensively described.

The Registry is going to provide discovery mechanisms. The query engine ARQ that is distributed with Jena [12], supports standard SPARQL and SPARQL/Update (SPARQL 1.1) as query language. Furthermore it supports distributed SPARQL queries and different extensions, like aggregations. The use of SPARQL queries allows the storing, querying and inference for RDF data that exist in the system registries.

The semantic stores considered for use in the COSMOS project are the open-source Sesame [13] and Jena. Other options are evaluated as well. Both of them are supporting query interfaces based on SPARQL which is a powerful language and is widely used in both the research and production level projects. The discovery component would also include security features since access to the triple store information might be subject to role based restrictions.

8.2.10.2 Functionality/Interfaces

As in the case of the Semantic Topic Management component, the VE Registry will be constructed around semantic stores providing SPARQL querying interfaces. The provided API of the VE Registry will expose the CRUD basic operations over VEs as well as complex VE retrieval operations. This will be used either for direct VE retrieval on in conjunction with the Discovery or the Recommendation Components.

The API of the Discovery component would be reach enough to allow the search of different building blocks based on various criteria. Besides the method parameter based search criteria, we will investigate the means of expressing more complex search criteria expressed directly using SPARQL snippets while maintain the semantic store integrity and security. Direct and unrestricted access to the triple stores should be avoided.

8.2.11. Social Monitoring

8.2.11.1 Description of the component

The COSMOS platform will include an advanced monitoring component which will provide not only basic logging functionality but also extended support for analytics. This will include: platform status monitoring, where various components of the platform are reporting their activity and status; VE/Application level monitoring, where VE and application relevant KPIs are determined; VE interaction monitoring where the focus will be put on KPIs relevant for the social analysis.

The Social Monitoring consists of a subset of the monitoring functions and is focused on determining and measuring interactions between VEs or between VEs and the COSMOS platform. Therefore, a set of KPIs relevant for the social analysis will be defined.

While the interaction between VEs and the COSMOS platform is guaranteed since the platform can easily track VE requests, the interactions between VEs performed using a peer-to-peer mechanisms could be performed without any monitoring, since the platform is not directly involved. Nevertheless, the platform will provide the interfaces which will allow VEs to report KPIs about the interaction with other VEs even if the COSMOS platform in not directly involved into the communication. Using this mechanism, the Social Analysis component will be fed with data even when VEs are communicating directly, provided that VEs report relevant KPIs to the platform.

8.2.11.2 Functionality/Interfaces

The Social Monitoring component provides, as mentioned a subset of monitoring functions. Monitoring is performed using different data collecting mechanisms, depending on the scenario. These are reflected in the interfaces which the component will expose.

When VEs are interacting with the COSMOS platform (e.g. when subscribing to a topic, when querying for topics or other VEs), the components involved in such interactions will report these operations using the dedicated API.

Dedicated API will also be provided so that consumers of the monitored data (e.g. Social Analysis component, dedicated analytics modules, etc.) are able to retrieve relevant data based on different data filtering criteria.

8.2.12. Social Analysis

8.2.12.1 Description of the component

The VEs and GVEs have to communicate with each other, to share their cases, to use IoT-services of other VEs etc. In other words, they have to interact with each other and thus to operate as social actors and have a set of dyadic ties between them. Consequently, the COSMOS social structure can be characterized as a social network.

The events that are generated at the Social Monitoring level can be evaluated at different platform levels (node level, group level or system wide) against a set of rules. The rules, which can be added, deleted or updated at runtime, may be specified by the consumers of information to set and control the flow of events and the aggregation output. The evaluation results can be used by the Planner or other COSMOS-components and can be used as a form of service for the users.

The social network perspective provides a set of methods for analysing the structures of whole social entities and their networks. For the study of these structures, COSMOS can use *Social Network Analysis* (SNA) to identify local and global patterns, locate influential entities and examine network dynamics.

Social network analysis is the analysis of social networks viewing social relationships in terms of network theory. These relationships are represented by nodes (representing individual actors within the network) and ties (which represent relationships between the individuals such as friendship, similarity etc.). These networks are often depicted in a social network diagram, where nodes are represented as points and ties are represented as lines.

In general, COSMOS social networks will be self-organizing, emergent and complex, such that globally coherent patterns will appear from the local interaction of the elements that make up the system. These patterns will become more apparent and rich as the size of network increases. However, a global network analysis of all the relationships between millions or billions of VEs is not feasible and is likely to contain so much information as to be uninformative. The nuances of a local system may be lost in a large network analysis, hence the quality of information may be more important than its scale for understanding network properties. Thus, social networks should be analysed at the proper scale, depending on the application or the needs of a user or a functional component of COSMOS. Although levels of analysis are not necessarily mutually exclusive, there are three general levels into which networks may fall: micro-level, meso-level and macro-level.

There is a great variety of metrics that can be used under the functionalities of the Social Analysis, offering more detail and information about the networks being analysed. Indicatively, some of the main metrics are:

- **Homophily/Assortativity:** The extent to which VEs and GVEs form ties with similar versus dissimilar others. Similarity can be defined by social characteristics or attributes that are domain-dependent (e.g. domain). This is one of the main characteristics that will be taken under consideration when COSMOS recommends new friends for a VE;
- **Mutuality/Reciprocity:** The extent to which two VEs reciprocate each other's friendship or other interactions. For example, VE_1 may use the IoT-services of VE_2 in its case base, but on other hand, VE_2 may not do the same for VE_1;
- **Propinquity:** The tendency for actors to have more ties with geographically close others;
- **Structural holes:** The absence of ties between two parts of a network. Finding and exploiting a structural hole can give an entrepreneur a competitive advantage. This concept was developed by sociologist Ronald Burt and is sometimes referred to as an alternate conception of social capital;
- **Centrality:** Centrality refers to a group of metrics that aim to quantify the "importance" or "influence" (in a variety of senses) of a particular VE or group of VEs within the network. Examples of common methods of measuring "centrality" include between-ness centrality, closeness centrality, eigenvector centrality, alpha centrality and degree centrality.

As mentioned before, the services and functionalities of the Social Analysis component will be used by both the users ("External" use) and other functional components (Internal use) such as the Planner. From the plethora of the metrics available and the social interactions that can be monitored, it is quite evident that the Social Analysis component can provide a great number of functionalities, depending on the needs of COSMOS system and the projects goals. Some of the main functionalities that have been studied are the following:

- **Extraction of higher-level goals of VEs:** A feature that we could have in Social Analysis is the comparison of the same targets/goals of the VEs (maybe expressed by their Case Base) and the extraction of more abstract goals that will characterize certain groups;
- **Modelling and Visualization of networks:** Visual representation of social networks is important to understand the network data and convey the result of the analysis. Exploration of the data is done through displaying nodes and ties in various layouts and attributing colours, size and other advanced properties to nodes. Visual

representations of networks may be a powerful method for conveying complex information;

- **Recommendation of VEs:** By finding the similarities between VEs or identifying the needs of a VE, it is possible to produce many recommendation services. One representative example is the Friends Recommendation of COSMOS. This feature that could be one of the first functionalities developed in Social Analysis, by taking under consideration vital social characteristics of a VE such as its domain and its location, during the registration of a VE to the COSMOS platform, could come up with a list of VEs that would be presented as recommended friends: a set of VEs that could be useful to the VE and would provide the first steps for XP-sharing;
- **Extraction of structural characteristics of the networks:** There are many properties of the networks that could be analysed without direct modelling and could be of great use for recommendation services. Questions that could be addressed are whether there is any "leak of knowledge" from one team/cluster to another, if so, how fast does the information flow, whether a team has any organizational weak points that can be structurally overcome etc. A representative example is the discovery of **structural holes** (see above).

8.2.12.2 Functionality/Interfaces

There are many tools that have to be used from the Social Analysis component and a need for the development of new tools may exist. Social network and dynamic network metrics, trail metrics, procedures for grouping nodes, identifying local patterns, distance based, algorithmic and statistical procedures for comparing and contrasting networks, groups and individuals from a dynamic meta-network perspective, geo-spatial network metrics, identification of key players, groups and vulnerabilities, are but a few issues that have to be addressed.

Some of the main tools that have been studied and could be exploited by COSMOS are:

- **Dynamic Network Analysis and Social Network Analysis:** Tools for reasoning under varying levels of uncertainty about dynamic networked and cellular organizations, their vulnerabilities and their ability to reconstitute themselves, choosing *Dynamic Network Analysis* (DNA) metrics and then using one or more of the available optimizers to find a design that more closely meets an ideal as well as exploring network graphs. Some tools that have been studied are DyNet, EgoNet, Optimizer, NetMiner, SIENA, SNA, Socilyzer and Sociometrica;
- **Network Document Analysis and Data Entry:** Tools that enable the extraction of information from texts using Network Text Analysis methods and other techniques. Such tools that have been studied are SocIoS, AutoMap, ORA and iNet;
- **Network Visualization:** Tools for graph visualization and representation like Zoomgraph, ORA Flash Network Visualizer, daVinci and GraphViz;
- **Representation Formats, mark-up languages and ontologies:** DyNetML and GraphML could be used as a reference model.

8.2.13. Planner

8.2.13.1 Description of the component

We are going to develop an **ontology-based *Case-Based Reasoning* (CBR) planner** and adopt, initially, the **Flat Memory model**. The case base will be part of the social ontology of the VE provided by COSMOS.

Our planner acts as part of MAPE-K loop [14] (hence in an autonomous way) as well as "manually", in other words, on demand of the developer. When the planner senses a situation or accepts a query, it means that there is a new problem to solve. This problem is a case without solution part. The planner creates a target case and compares it with the cases available in the case base. If the planner does not find a case similar enough to the target case, the VE can ask other VEs (through experience sharing functionalities described in D6.1.1) for assistance. In other words, the VE can query the case bases of other VEs too, thus experience sharing and communication between autonomic managers takes place.

The developers can create a case (domain ontology) which has a problem and a solution:

i. **a problem** is going to be a series of events that have to be identified to trigger the solution. This description of events has to be linked with the corresponding topics on the COSMOS message bus. The problem can be simple (event) or complex (series of events);
ii. **a solution** (in its simplest form) can be the URI of an IoT-service. A solution can be primitive (1 task- IoT-service) or complex (series of IoT-services).

A case structure includes simple and compound attributes that describe the cases together with their types, weights etc. Following the example of most of the industrial CBR applications, we are going to propose forms to the developer to fill the case base. In our case, we can offer the developers the opportunity to create case bases for their VEs. Each VE may have its own Planner and Case Base. Each VE will have its own knowledge base with its own repository in order to facilitate M2M communication. COSMOS could provide a GUI template consisting of fields to fill in, in order to describe the problem and give the input for the similarity calculation. For example, the developer could define data for the simple attributes of a case and weights for its complex attributes. The entered information is then retrieved to build a case, while integrating their semantics. This phase is about formalizing the description of the problem to be solved by assigning values to the descriptors in the target case. The case is then stored in the case base.

Consequently, the planner has two retrieve modes:

i. The planner uses complete cases (problem and solution is defined). That means that it follows the changes on the topics that correspond to specific problems. When the planner gets notified by the Analysis or the Social Analysis component, the corresponding solution is forwarded to the Executor;
ii. The planner can accept as input a target case (a case with the description of the problem only) and create a new complete case. That means that the planner has to find similar cases in its case base or the case bases of other VEs, choose the most appropriate solution(s) and, in latter stages, create new solutions (e.g. services composition).

The choice of the solution could be made at a first step from social criteria, such as the reputation of the VEs that offer the IoT-service that corresponds to the solution of a case. For example, if a VE-bus has two cases with the same problem (e.g. fire detection) but different

solution (e.g. “call fire-truck 1” vs “call fire-truck 2”), if the IoT-service “call fire-truck 1” belongs to a VE with higher reputation than that of the IoT-service “call fire-truck 2”, then the planner would choose the first case.

The Case Base provided by COSMOS could also contain a classification of the cases. One such classification could be based on the different self-management attributes that the different cases implement (e.g. self-configuration), thus providing the autonomy of a VE.

At a next level, we could extend the Experience by CaseBase-type in order to include various types of case bases in the system. For example, we could use the CBR planner for building a service-recommendation component or even for modelling dynamic web-service composition. In this recommendation system, the Planner could work as a recommendation service which will take under consideration social characteristics of the VEs used in the cases. For this to happen, feature selection has to take place. Like other feature-based systems, in case-based reasoning, one area of research focus has been on how to select important features among all the features of the problem specification and weighting them to make the cases or to facilitate the retrieval.

In the future, the planner may follow a hybrid reasoning model e.g. CBR followed by Rule-Based Reasoning. The RBR could represent the general policies of the VEs, their goals etc. Moreover, rules could be extracted from the analysis of different Case Bases.

COSMOS will offer various services for the developer giving him the possibility to manage the case base. He could store, retrieve all or some cases, delete or share cases. Also, he could specify how a case is going to be stored, e.g. in memory or in persistence store.

8.2.14. Event detection & Situational Awareness

8.2.14.1 Description of the component

The evaluation process itself must be able to correctly evaluate situation taking into account very large number of varying parameters and/or within a very tight time conditions. In these situations it is common that the situation evaluator is supported by Complex Event Processor attached to multiple sources of various events carrying important information.

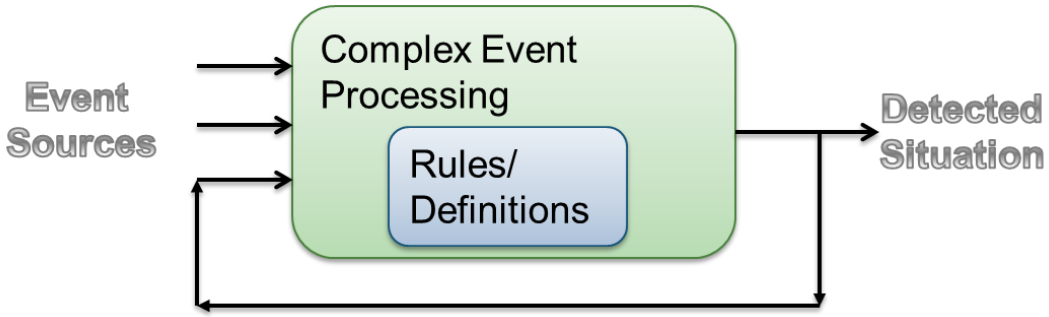

Figure 15. Situational Assessment

The Situational Awareness component within COSMOS aims at collecting and monitoring information coming from interconnected heterogeneous IoT platforms and things about their states and behaviour as well as environmental conditions around them to build situational awareness that will be further utilized by other components, things or smart city applications for:

- Recognition of potential hazards;

- Building experience, hence gain elevated situational awareness;
- Making decisions;
- Detecting sudden situations;
- Etc…

The perception of current situation has to take into account static as well as volatile properties with respect to space and/or time.

8.2.14.2 Functionality/Interfaces

Assessment of State and Behavioural Situation

The COSMOS will extract knowledge out of data streams provided by connected Virtual Entities. For example in Madrid use-case, COSMOS may collect information about busses and other vehicles like position, speed, line schedule and actual state of traffic lights and various operational conditions. The state and behavioural assessment will provide a foundation for understanding how information, events and decisions will impact goals and objectives both immediate as well as in the near future.

Assessment of Environmental Situation

Besides assessment of the state and behaviour of various things, COSMOS will also assess environmental situation in things vicinity. Information about temperature, humidity, time, etc. is also important for building overall situational assessment.

Real-time Situation Monitoring

The ability to process near real-time data will be achieved through the use of Complex Event Processing techniques. These techniques are suitable for fast data analysis based on temporal and structural relations between underlying events.

A near real time situation assessment will also serve following functionality based on near real-time data analysis:

- ***Notification/Anomaly Detection*:** Evaluation non-standard behaviour like an emergency alert, sudden or unexpected situations and failures based on available knowledge base;

- ***Support for Adaptation:*** Adjustment of resource utilization based on various conditions monitored through the real-time analysis of streaming data for situational awareness. For example adjustment of the energy consumption based on corresponding heat and electricity sensors as well as the environmental sensors;

- ***Disaster prevention and Increased Safety*:** For example in the Madrid scenario, safety may be further elevated by utilising information regarding the location and speed of the buses, the road lights, slippery and potholes, the weather conditions (e.g. ice) and the speed of other vehicles. Situational knowledge leads to situational awareness.

Overall Situation Assessment

A Thing connected to COSMOS have a sub-goal pertinent to its specific role that feeds into the overall goal required by smart city applications.

The overall awareness is especially important in domains where the information flow can be high hence there is an increased risk of human error which can potentially lead to serious consequences. The Assessment of overall awareness is also necessary for:

- ***Context association***: where same data in different context could have different meaning;
- ***Conflicts***:- Some things may influence/interfere with correct behaviour of other things. This includes e.g. conflicting integration of new device into network, missing support for versioning etc.

COSMOS will further utilize behavioural and environmental awareness assessment to create foundation for creating the overall awareness regarding the current situation. This gives a smart city application the ability to respond to changes within its situational environment with little or no direct instruction from users.

8.2.15. Event Pattern Detection

8.2.15.1 Description of the component

Some of the complex event processing capabilities provided by COSMOS for situation awareness can be also utilized internally by Virtual Entities for internal detection of temporal and structural patterns among events produced by associated things.

8.2.15.2 Functionality/Interfaces

For COSMOS platform, there are three different deployment scenarios under consideration:

1.) Event detection as service: The COSMOS platform will offer analysis and information extraction capabilities to external components. This deployment aims at providing services to components that lack hardware resources or are incompatible for hosting event detection solution. Depending on information exchange between VEs, other VEs can also subscribe to and consume results provided by this event detection service;

2.) Internal event detection: A use case where COSMOS client i.e. Virtual Entity offers execution environment that enables hosting of (or at least some artefacts of distributed) CEP. In this case, private instance for custom analysis of event streams;

3.) Hybrid event detection: A deployment of some artefacts of distributed CEP on VE execution environment. For example, for performance reasons, VE might host only Event Collector component for fast receives of data from internal sensors using internal machine-to-machine protocol and actual event analysis will be done by platform or another VE.

Context awareness

A prerequisite for context awareness is assessment of relevant situation. In order to utilize event detection mechanism for situational assessment, it is necessary to define and model context that will be analysed and evaluated based on input data stream(s). The COSMOS will provide a dedicated Dolce domain language specification [15] for definition of application specific context.

8.2.16. Prediction (Interpolation/Extrapolation)

8.2.16.1 Description of the component

In the world of Internet of Things, devices and sensors are deployed or used in varying conditions and different situations. Mostly they form a dynamic network connected using less reliable wireless links and in remote places. 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 a model for the devices using 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**.**

8.2.16.2 Functionality

Interpolation/extrapolation

Interpolation/extrapolation involves predicting the sensor values in case of discontinuity or unavailability of the sensor readings at certain time instant. This component will make sure that the performance and reliability of a system does not deteriorate even in the absence of disconnection with certain sensors. Some components of COSMOS need near real time information for critical decision making and this component will provide the value in case of temporary disconnection from devices using model based on the historical data. This component will act as an alternate data source.

Prediction Models

A system model will be build using time series analysis and other algorithms fed with historical data and will be able to provide prediction service to VEs. For example, a certain VE might be interested in the expected arrival time of another VE or to find the location of another VE at certain time. In such scenarios, high level applications can use the model formed in this section.

8.2.16.3 Interfaces

This module will be connected with cloud storage in order to access historical data. Prediction models will be build using time series analysis of historical data. In order to update the model in real time, it will be connected with the central data bus. Whenever VE publish its data, it will incorporate its reading and will update the model accordingly. Any application which will need prediction value of VEs can access it by passing query through data bus.

8.2.17. Experience Sharing

8.2.17.1 Description of the component

In the context of COSMOS, experience can be considered as models or cases which both derive from the MAPE-K loop approach and in particular from the analysing and planning phase respectively. In Year 1 we are going to focus on the cases, whereas models are going to be analysed in Year 2 and/or 3. A Case consists of a problem and its solution and furthermore the problem corresponds with a topic written in the message bus and a solution corresponds with

IoT-services exposed by the VEs. This component enables VEs to exchange their experiences with their friends VEs and thus to act in a more autonomous way.

8.2.17.2 Functionality/Interfaces

Experience Sharing component is related with the following functionalities, which are analytically described in the subchapter 4.2 of the deliverable 6.1.1:

- **Storing experience**: Apache Jena API can be used so as the VEs to store their experiences (cases) by adding instances of problems and solutions in their own case base;
- **Finding experience**: SPARQL queries can be used for requesting and receiving solutions from other VEs that have similar problems in their case base;
- **Choosing experience**: Planner component is called in case a VE has to choose between two or more offered solutions. The planner makes the decision according social characteristics of the VEs, like trust & reputation index. For Year 1, this index depends on how often a VE shares its cases or its IoT-services, but in Year 2 and/or 3 VEs are going to assess the usefulness of the experience they have received and provide their feedback. The latter should affect the value of the index.

9. Information View

This chapter presents the flow of operations for COSMOS through sequence diagrams. These operations are a subset of the final set of operations that COSMOS will be able to support as more will be added as the project evolves.

9.1. Registration and Key Distribution

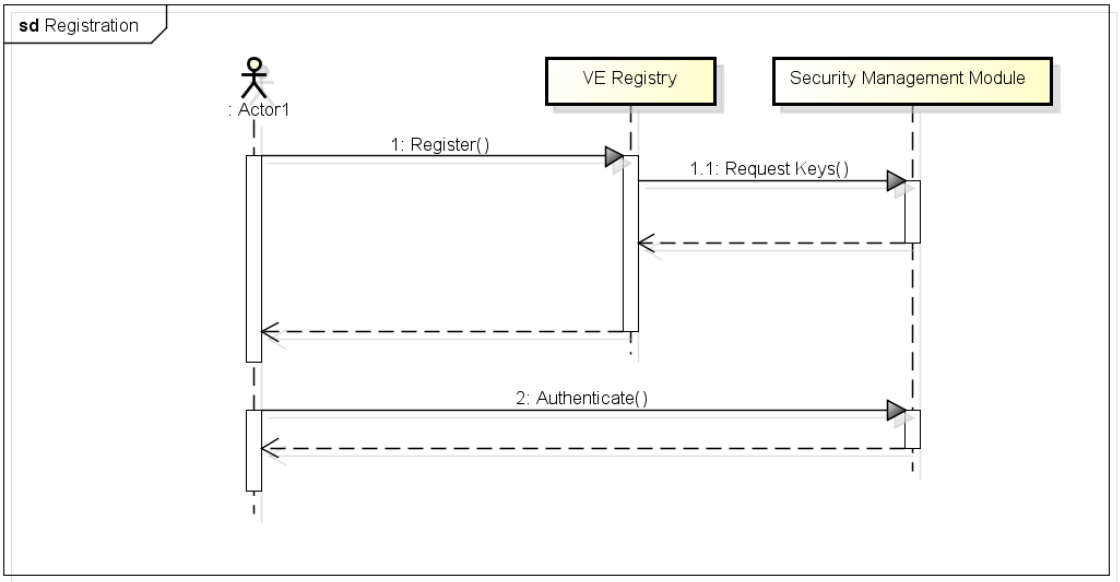

Figure 16. Registration and Key Distribution Sequence Diagram

For the COSMOS environment to be secure it is assumed that each client has a unique key. In order to have a unique key for each actor a key generation mechanism is used. As key generation is usually both time-consuming as well as a standardized operation, Openstack Keystone is used within COSMOS. OpenStack Identity (Keystone) provides a central directory of users mapped to the OpenStack services they can access. It acts as a common authentication system across the cloud operating system and can integrate with existing backend directory services like LDAP. It supports multiple forms of authentication including standard username and password credentials, token-based systems and AWS-style (i.e. Amazon Web Services) logins. Additionally, the catalog provides a query-able list of all of the services deployed in an OpenStack cloud in a single registry. Users and third-party tools can programmatically determine which resources they can access [16].

Key generation, distribution and management thus rely on Keystone.

VEs which want to be enrolled into COSMOS first need to register into the VE registry and into the Security Management Module. If the VE meets the necessary security criteria (e.g. implement the required API, has the security flag set and implements the required security primitives which are reflected by the security flag) the enrolment process begins. Keystone is triggered and a new key is generated for the enrolling VE. The key is distributed to the VE using Diffie-Hellman key exchange mechanism, using any standard communication interface (the “carrier” does not make the object of the process, only the information payload is relevant, regardless of the communication interface or protocol used).

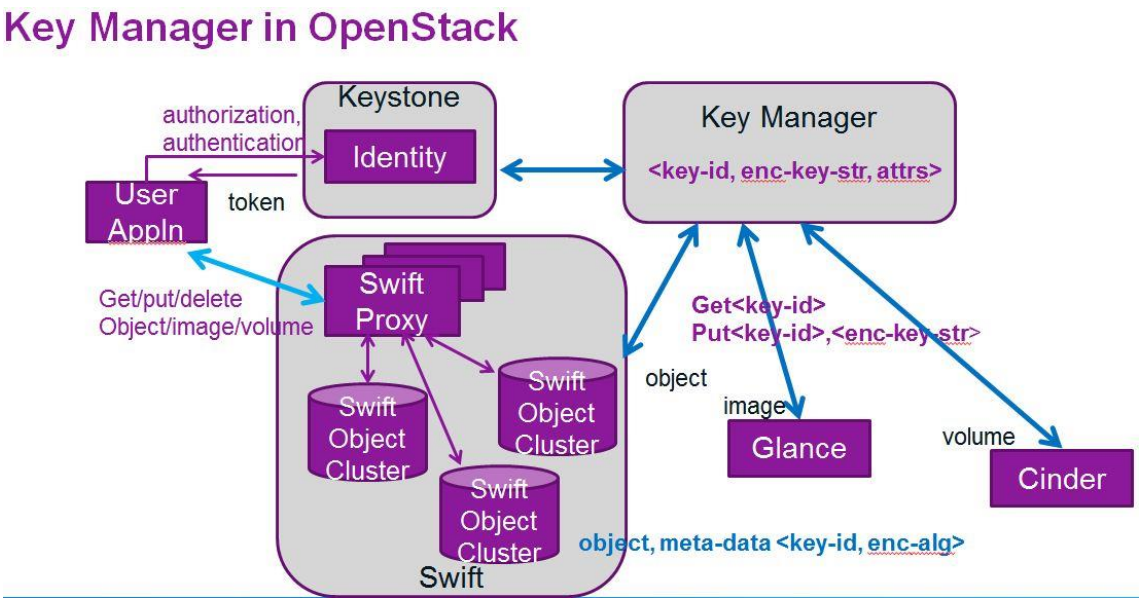

Figure 17. Key Manager in Openstack

The Keystone Key Manager Module keeps track of the key which are valid and in use as well as the revoked ones.

The key generated with Keystone is used for client authentication as well as for data encryption and decryption. On the COSMOS side both encryption and decryption are handled by Keystone and are implemented in software. On the VE side, depending on the VE type, the cryptographic operations can be executed in software, hardware or part in software and part in hardware.

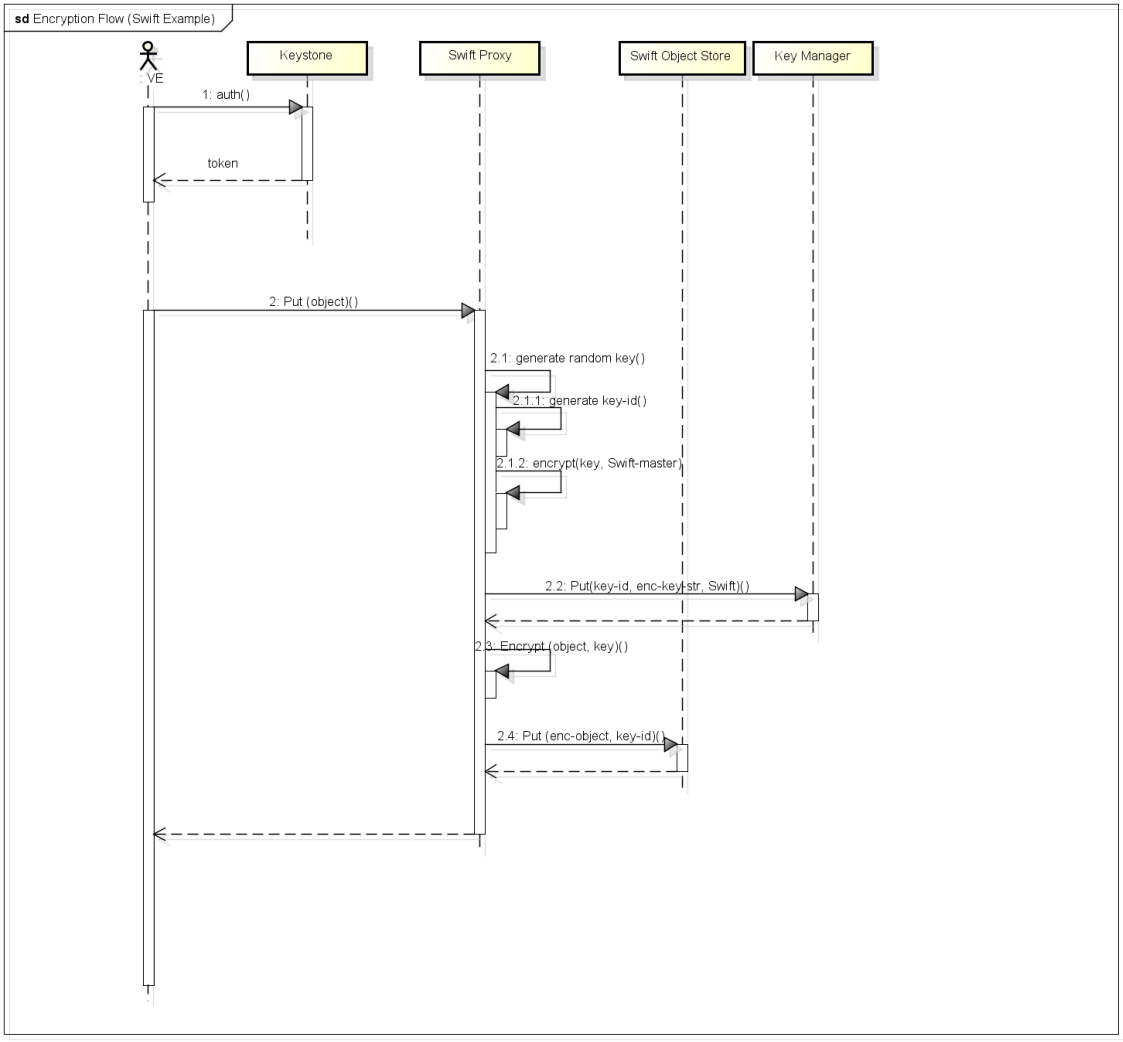

Figure 18. Encryption Flow Sequence Diagram

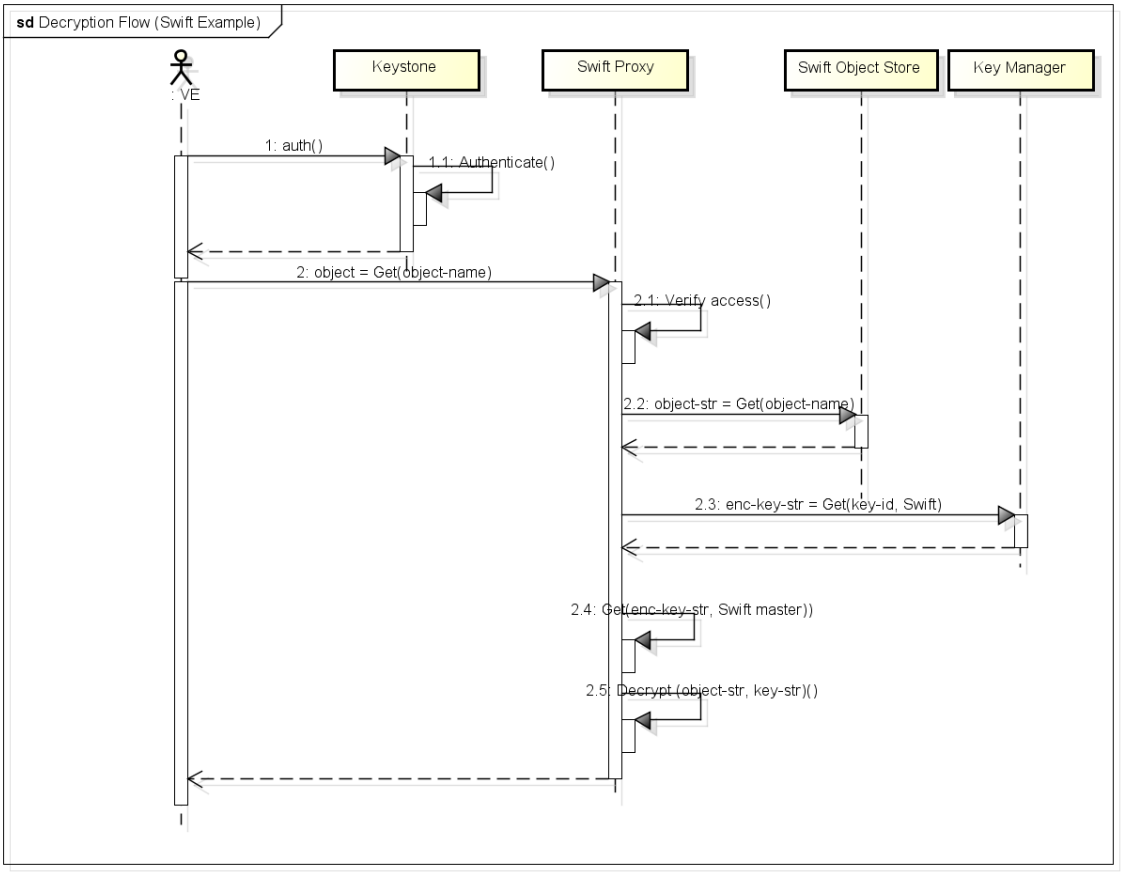

Figure 19. Decryption Flow Sequence Diagram

9.2. Data Management Operations

Figure 20 shows basic data flows through the system. A VE can authenticate itself in the system via the Security Management component. An authenticated VE can find a topic via the Topic Management component. Once a topic has been found, a VE can register to a topic, create a new topic or publish data to a topic. Data which is published to a persistent topic is collected by the Raw Data Collector and stored in the Data Store after extracting and annotating with metadata. The metadata is indexed for subsequent metadata search. The Data Mapping component may read the raw data in the data store and transform it into a format suitable for analytics.

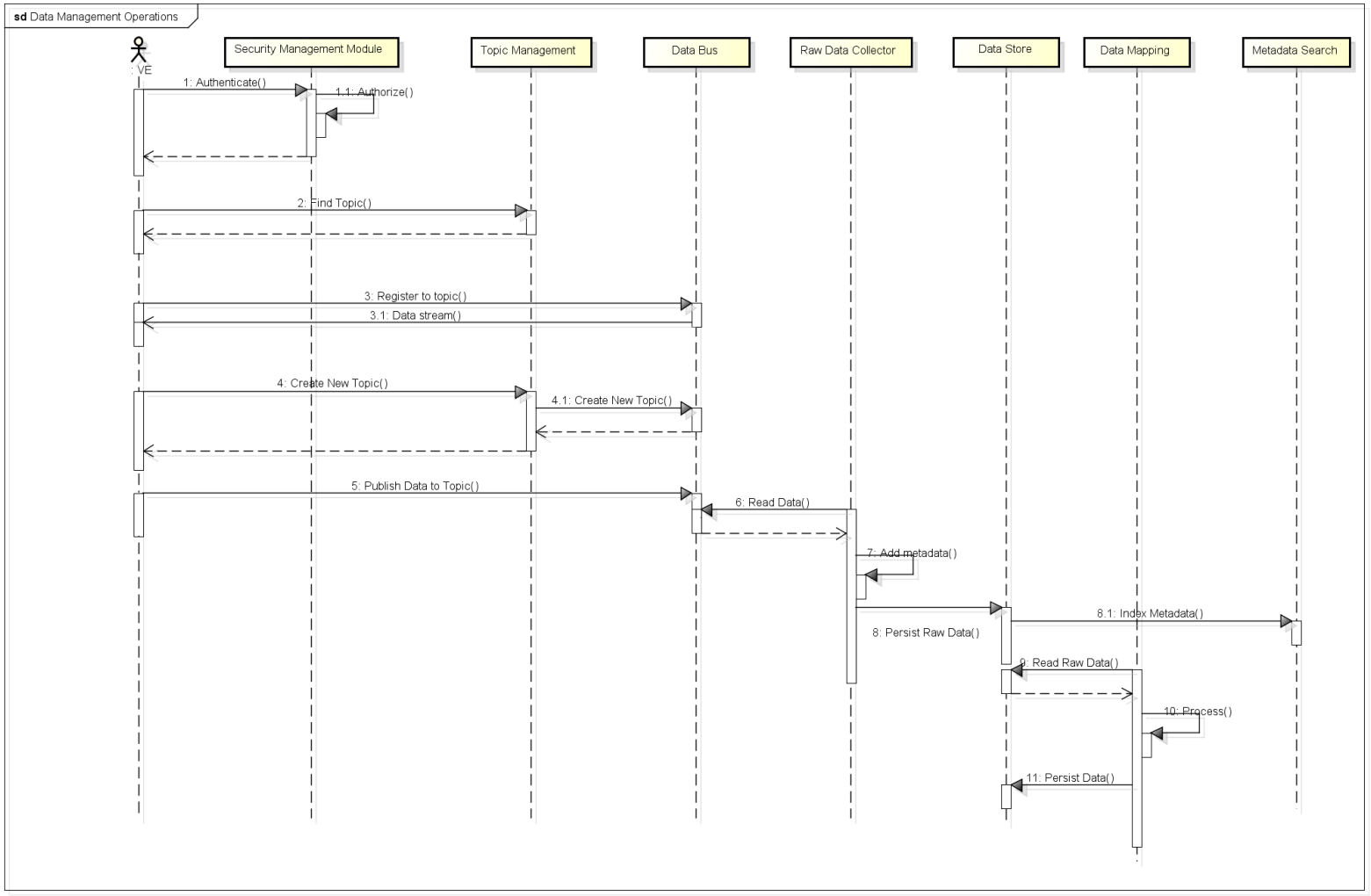

Figure 20. Data Management Operations Sequence Diagram

9.3. Metadata Search

Figure 21 shows the basic flow for metadata search. A human user or application submits a metadata search request to the Data Store, which sends a metadata search request to the Metadata Search component.

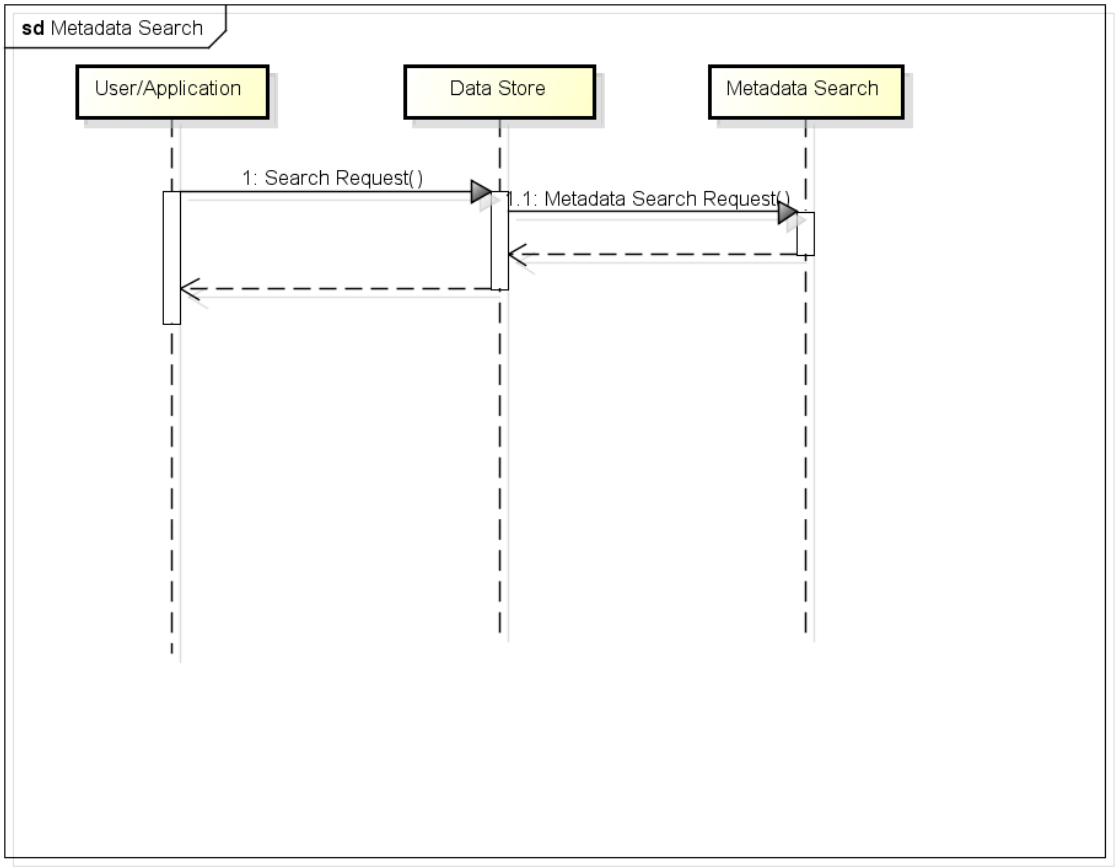

Figure 21. Metadata Search Sequence Diagram

9.4. Storlets

Figure 22 shows basic flows for storlets. A developer develops a storlet according to the storlet APIs and then can deploy it. Subsequently, storlets can be invoked.

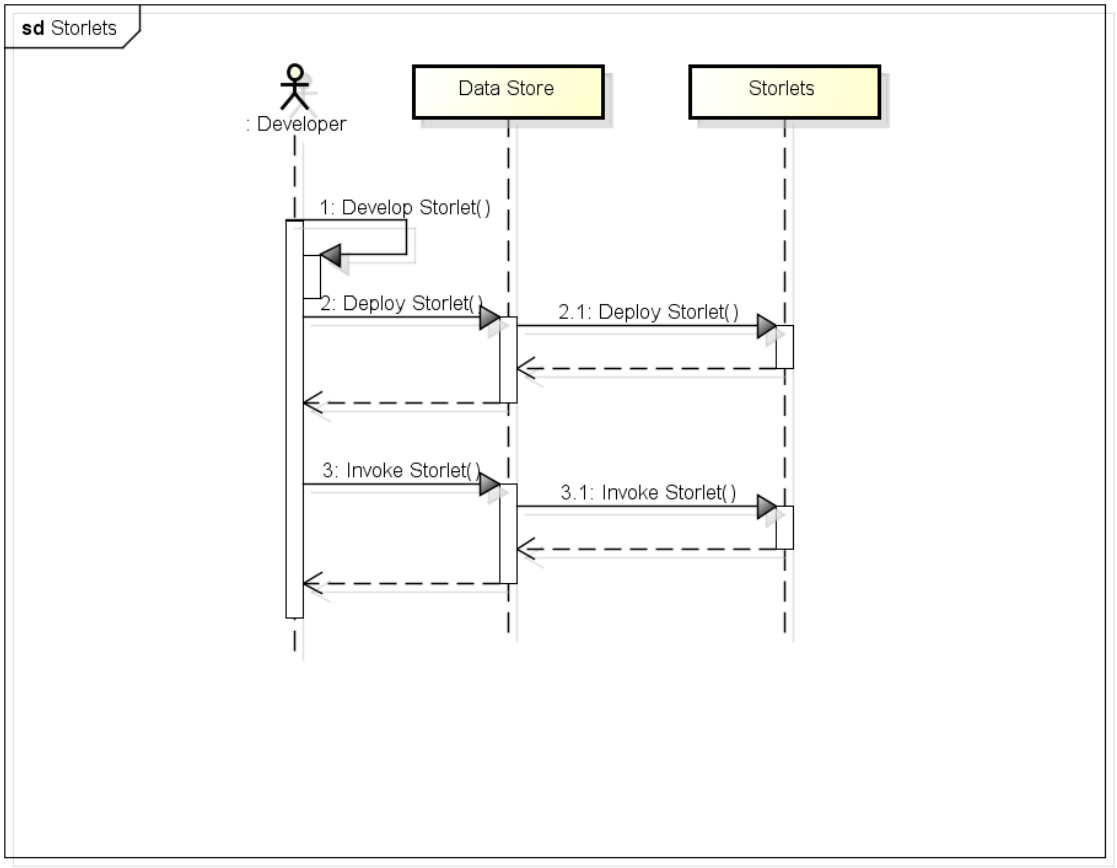

Figure 22. Storlets Sequence Diagram

9.5. Situational Awareness

As depicted in Figure 23, the first step is to define one or more specific situational context that will be assessed by situational awareness component. After defined context is uploaded to Situational Awareness using dedicated API, new topic is registered at Topic Management. A client has to register for this new topic in order to receive information about detected situation. Depending on monitoring and evaluation of available information, Situational Awareness will publish relevant information to registered topic. This information will in turn be passed to all subscribers.

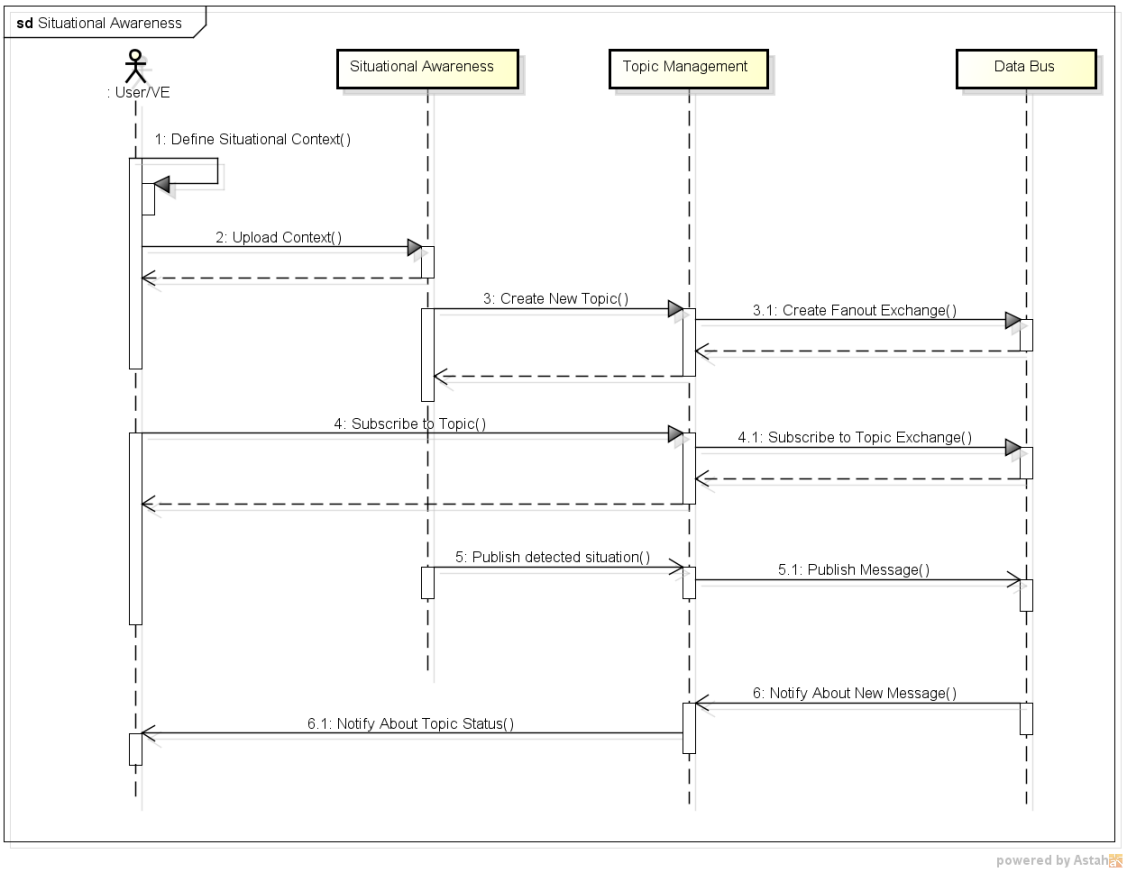

Figure 23. Situational Awareness Sequence Diagram

9.6. Prediction

In the context of Information flow, Prediction module can be divided into two parts. a) Updating Prediction Model b) Using Prediction Model. The virtual entities like sensors will publish their values regularly on the data bus and the prediction model will be updated accordingly in real time. Different Applications and Virtual entities registered with particular topic might be interested in finding real time or future values of particular VE. They can call prediction models in such scenarios.

9.6.1. Updating Prediction Model

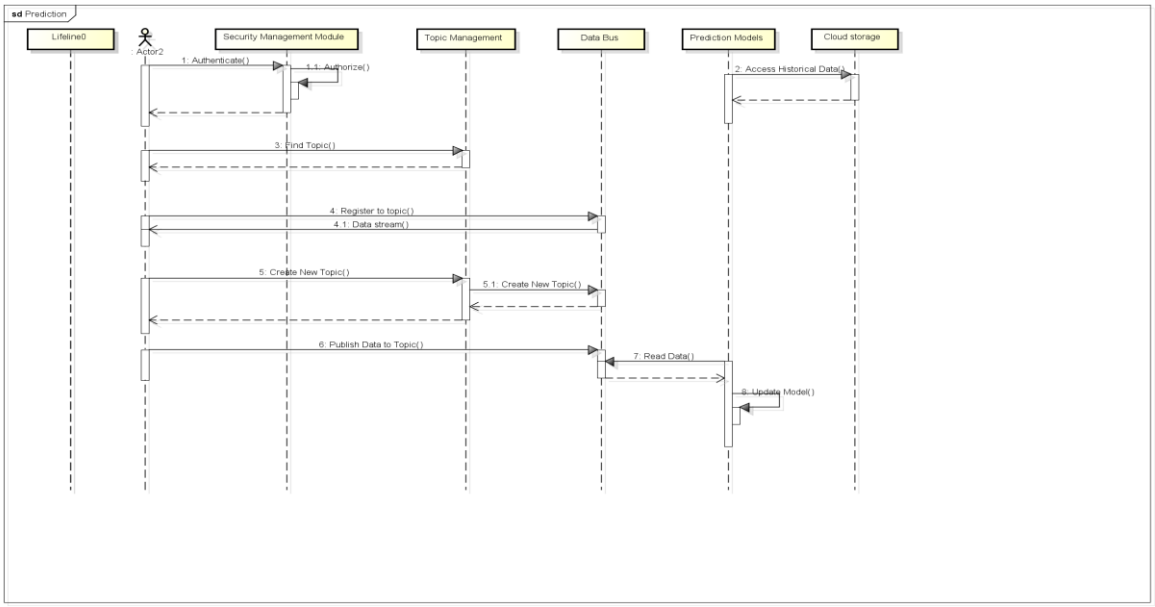

Figure 24. Updating Prediction Model Sequence Diagram

Prediction model was built initially using time series analysis of historical data stored in the cloud in offline mode. VEs or GVEs publishe their data regularly depending on their sampling time. Whenever a particular VE which is registered to particular topic, publishes its data such as position or velocity, the prediction model will be updated accordingly in real time by measuring the difference between standard deviation of measured and predicted value

9.6.2. Predicting using Model

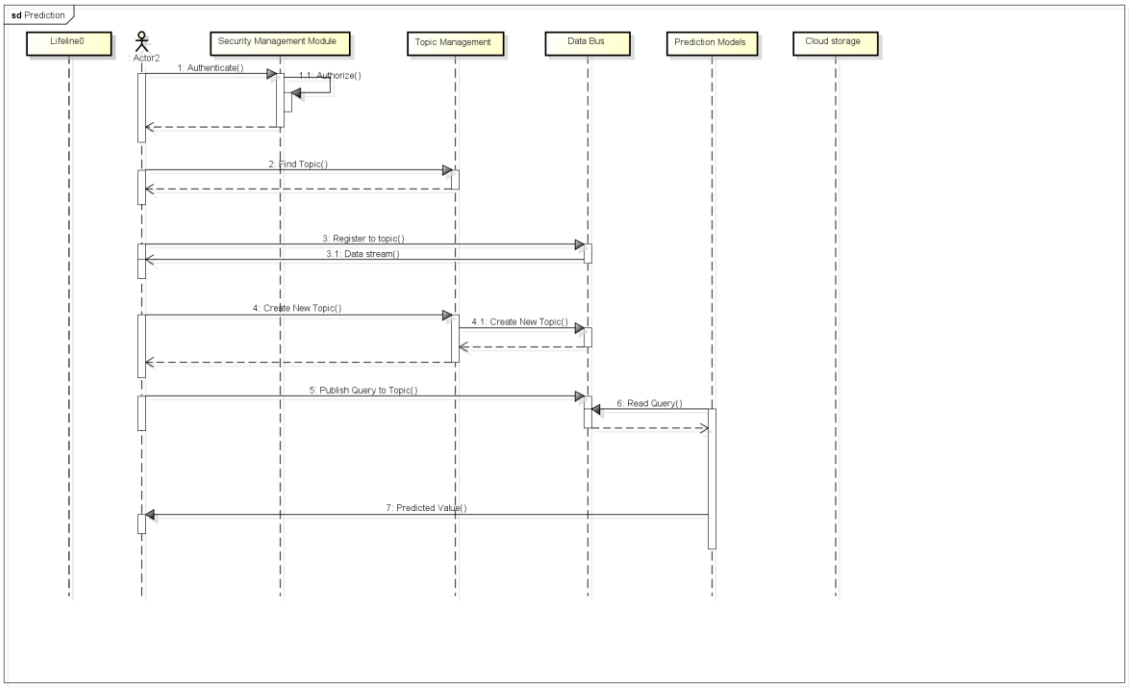

Figure 25. Predicting using Model Sequence Diagram

Whenever a component or VE in the framework is interested in finding the predicted value or position of other VEs, it sends the query to the data bus after registering to a dedicated topic. Prediction models will be used to find the future state or value of VEs. The above diagram shows the information flow when some outside component is interested in finding a predicted value. But it should be kept in mind, this is one of many possibilities. Internal components within framework like CEP might also be interested in the future values to optimize its performance.

9.7. Cases Management

Figure 26 shows the basic interactions needed for the autonomic functions of VEs based on the MAPE loop and Cases Management as discussed in D5.1.1.

More specifically, the Planner of a VE registers to a topic at the COSMOS Message Bus, which monitors a problem. When this problem is detected, the Planner gets notified and accepts the description of the problem as input. After this step, there are three scenarios:

- The Planner of the VE searches the Case Base of the VE for cases with similar problems. If such a case is found, then its solution is executed;
- If not, the Planner sends a request to the Experience Sharing component. This initiates the experience sharing sequence described in more detail in Paragraph 9.8. The Planner asks from the VE Registry for the Trust Indexes of the friends of the VE and chooses the best case taking under consideration the similarity to the problem and the trust index. Then, the solution is executed;
- If even the friends of the VE do not have any appropriate cases to offer, then the Planner sends a request to the Discovery component of the VE Registry, asking for cases with similar problems to the initial one. The Discovery component takes as input the problem that has to be found and social characteristics of the VE that could accelerate the discovery (e.g. “domain”) and gives as output to the Planner the appropriate cases and the trust indexes of the VEs which contained them. Then, the Planner chooses the best case taking under consideration the similarity to the problem and the trust index. Finally, the solution of this case is executed.

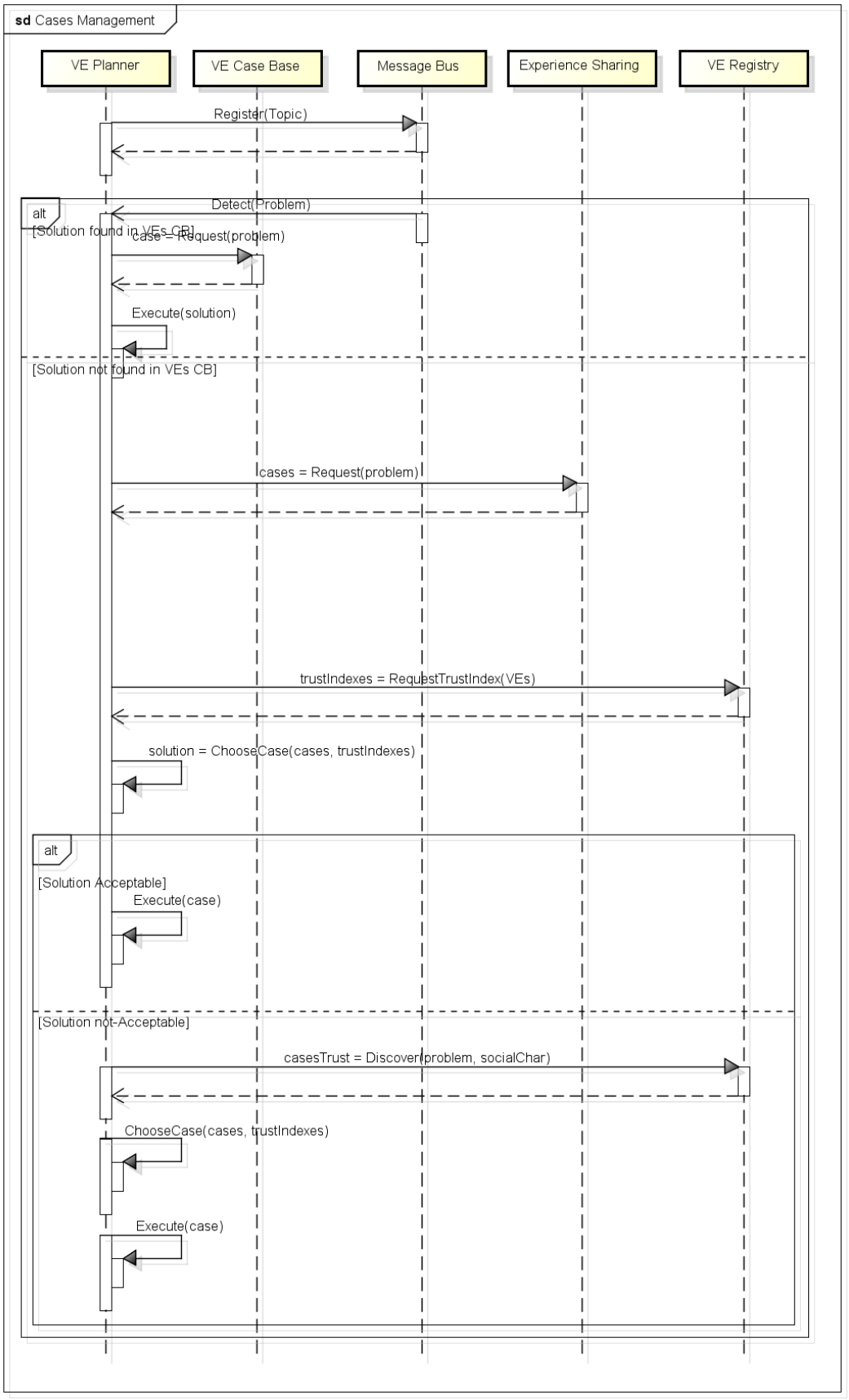

Figure 26. Cases Management Sequence Diagram

9.8. Experience Sharing

Figure 27 shows all the interactions that take place during the experience sharing process and those which are relevant to it:

- The planner of the VE1 asks the experience sharing component for a solution to the problem that occurred;
- The experience sharing component communicates with the corresponding component of the VE_2 and then the request is transferred to the planner 2;
- Once the solution is found, it follows the reverse path until it reaches the planner of the VE_1.

It has to be mentioned that when the Experience Sharing component of the VE_2 sends back a solution to the VE_1, the Social Monitoring component monitors this interaction and forwards this information to the Social Analysis component. The Social Analysis component analyses this information and forwards it to the VE Registry.

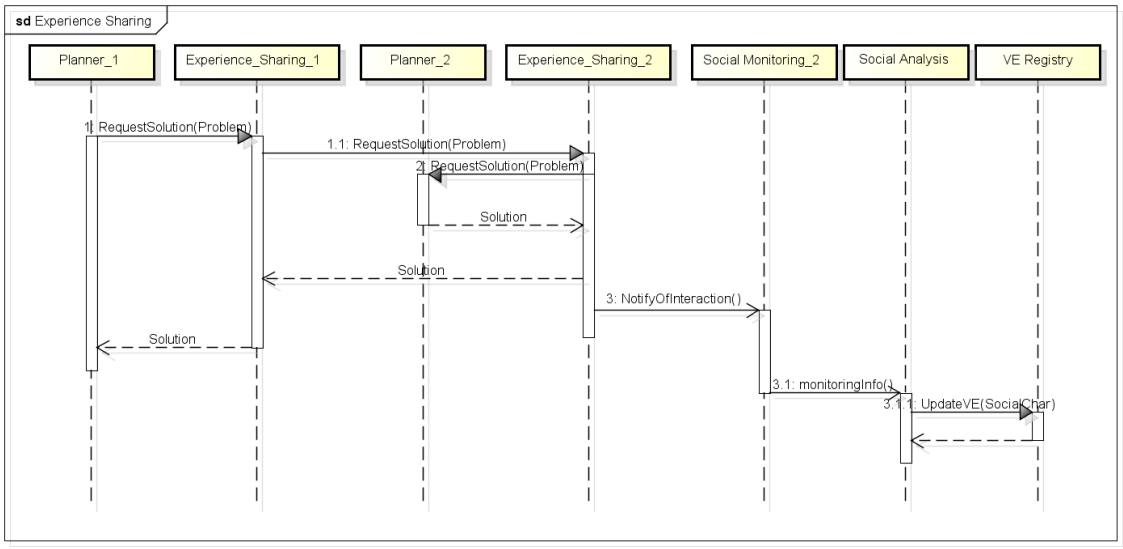

Figure 27. Experience Sharing Sequence Diagram

10. Mapping COSMOS to the IoT-A Functional View

In this chapter we attempt to map our architecture to the Functional Groups proposed in IoT-A. According to [1] the IoT-A ARM proposes the following FGs:

- ***IoT Process Management FG:*** The purpose of the FG is to allow the integration of process management systems with the IoT platform;
- ***Service Organisation FG:*** This FG is responsible for composing and orchestrating services, acting as a communication hub between other FGs. It contains the Service Choreography Functional Component which "offers a broker that handles Publish/Subscribe communication between services";
- ***Virtual Entity FG:*** This FG relates to VEs, containing functions such as discovering VEs and their associations with IoT-services, as well as monitoring functions.;
- ***IoT Service FG:*** The IoT Service FG contains functions relating to IoT Services. It also contains storage capabilities functionality. More specifically the ARM states that "A particular type of IoT Service can be the Resource history storage that provides storage capabilities for the measurements generated by resources";
- ***Communication FG:*** The Communication FG is used to abstract the communication mechanisms used by the Devices. Communication technologies used between Applications and other FGs is out of scope for this FG as these are considered to be typical Internet technologies;
- ***Security FG:*** The Security FG "is responsible for ensuring the security and privacy of IoT-A-compliant systems";
- ***Management FG:*** The Management FG contains components dealing with Configuration, Faults, Reporting, Membership and State. It should be mentioned here that this FG works in tight cooperation with the Security FG.

In Figure 28 we try to align the COSMOS functional components with the IoT-A Functional Groups.

The Planner, Message Bus and Semantic Topic Management belong to the Service FG. The Planner is responsible for finding suitable solutions and asking for their execution. Therefore if provides functionality similar to the Service Composition FC which resolves services that are composed of IoT Services and other services in order to create services with extended functionality. The Message Bus provides functionality similar to the Service Choreography FC which offers a broker that handles Publish/Subscribe communication between services. The Semantic Topic Management component is tightly couple with the Message Bus and thus is considered part of the same FG.

The Monitoring, Social Analysis and Reputation Index and Experience Sharing functionalities belong to the Virtual Entity FG, as these are all functions that relate and operate on the level of VEs.

The Cloud Storage component fits within the IoT Service FG as it provides functionality similar (although greatly enhanced) to exposing historical measurements of resources. For the same reason, the advanced functionalities of cloud storage, such as Data Analysis and Meta-data search are also part of this FG. The Prediction of Values component works on the IoT-service level as it relates to values accessible through IoT-services. The Event Detection and Situational Awareness component also fit with the IoT Service FG as its functionality is based on simple events provided through IoT-Services.

The Semantic VE Registration Component can been seen as touching three FGs, namely the Management, Virtual Entity and IoT Service FGs. This is because it relates to the Member FC of the Management FG, providing functionalities such as registering VEs with COSMOS, as well as providing functionalities for discovering and retrieving information both for VEs and for IoT-services

Lastly, the Security Management Module naturally fits within the Security FG.

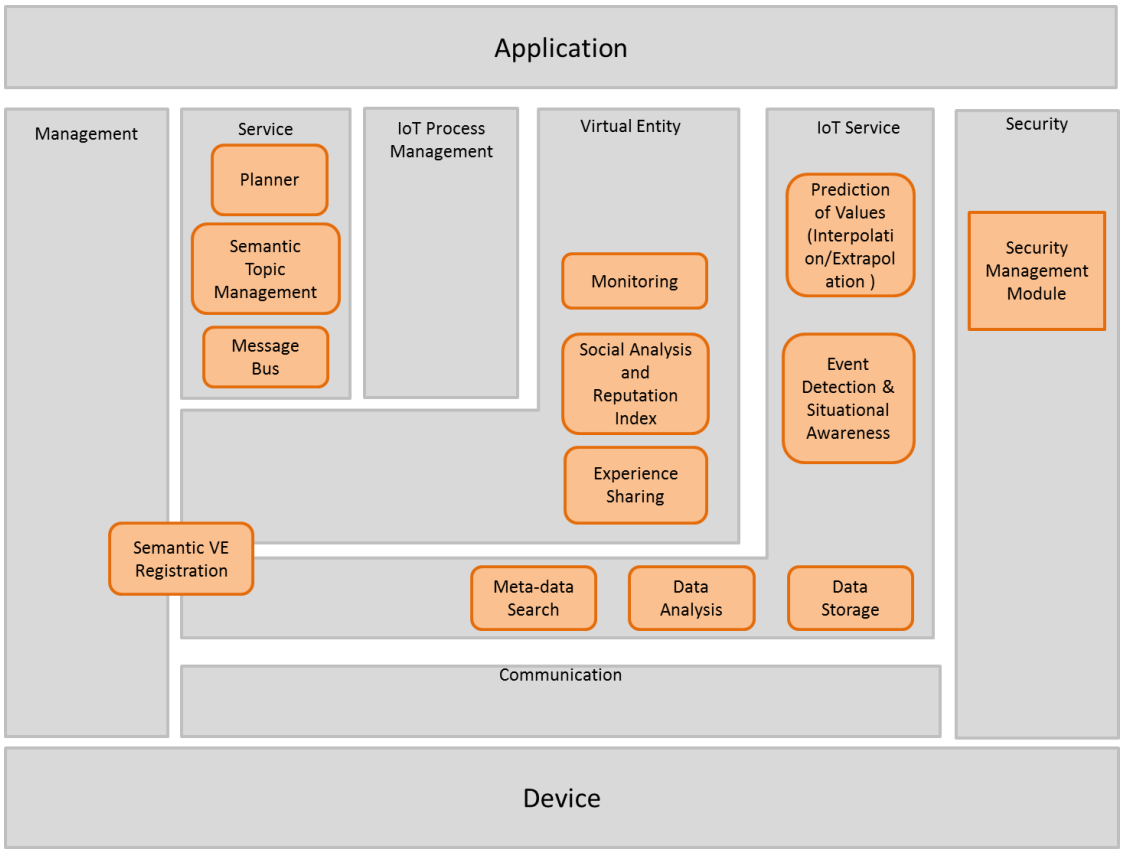

Figure 28. Alignment of COSMOS with IoT-A FGs

11. Conclusions

This report “Conceptual Model and Reference Architecture” presents the results of the work carried out in the scope of WP2 “Requirements and Architecture” of the COSMOS project during the first 8 project months.

Initially, this preliminary version of the COSMOS architecture highlighted the key architectural principles for the design of the platform. In addition, the methodology (IoT Reference Architecture from IoT-A and UML modelling) that has been followed for the analysis and design, in order to improve the efficiency of the WP2 work and quality of the results, was briefly described.

Furthermore an analysis of the input from Tasks 2.1 Market Analysis and 2.2 Requirements and state of the art analysis, from the architectural point of view, took place. Specifically the User Requirements and Use Cases were examined and the result was their prioritisation and association with specific parts of the platform.

The required capabilities for the platform and main processes -as part of the conceptual model- have been identified. Also an overview of the architecture and the main subsystems were presented, having as the centre of interest the main objectives of COSMOS.

It is expected that this report will be useful input for the whole project and especially for the development WPs (WP3, WP4, WP5 and WP6) that are the main consumers of the work carried out in WP2. In the initial description of the subsystems/components and their interactions, the role of each one in the overall architecture was clarified providing guidelines for their implementation and how these are going to be integrated from WP7.

As the project evolves, we anticipate in providing more insight for the overall platform and the individual components in the updated versions of the report. Specifically, all the processes of the infrastructure and the interfaces between the components will be described in depth while detailed specifications for each one will be produced. WP2 will further examine the application scenarios upon which will be based to create guidelines for the platform.

WP2 is in continuous collaboration with the other WPs of the project in order to acquire more requirements and to “fine-grain” the architecture design. WP2 will get feedback from the development work packages and more technical details about their components and their implementations in order to avoid any inconsistencies between the subsystems developed by different partners.

12. References

[1] F. Carrez *et al.*, "IoT-A Deliverable D1.5 – Final Architectural Reference Model for the IoT v3.0", www.iot-a.eu/public/public-documents/

[2] COSMOS Project D2.1.1 Market Analysis and Potential Report.

[3] COSMOS Project D2.2.1 State of the Art Analysis and Requirements Definition.

[4] IoT-A web site: http://www.iot-a.eu

[5] S. Haller *et al.*, "A Domain Model for the Internet of Things", in iTHINGS'2013 proceeding, Beijing, China (see also IEEE eXplore)

[6] N. Rozanski and E. Woods, "Applying Viewpoints and Views to Software Architecture" , Addison Wesley, 2011

[7] NIST Guide for Conducting Risk Assessments, http://csrc.nist.gov/publications/nistpubs/800-30-rev1/sp800_30_r1.pdf

[8] Xiliniz Zynq. http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/

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

[10] OpenStack Object Storage API v1 Reference http://docs.openstack.org/api/openstack-object-storage/1.0/content/index.html

[11] Internet of Things Environment for Service Creation and Testing (IoT.est) http://ict-iotest.eu/iotest/

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

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

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

[15] WonderWeb Deliverable D17 http://www.loa.istc.cnr.it/old/Papers/DOLCE2.1-FOL.pdf

[16] Keystone http://en.wikipedia.org/wiki/OpenStack#Identity_Service_.28Keystone.29