COSMOS

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement Nº 609043

D2.3.2 Conceptual Model and Reference Architecture (Updated)

WP2: Requirements and Architecture

Version:	1.0
Due Date:	30/04/2015
Delivery Date:	30/04/2015
Nature:	Report
Dissemination Level:	Public
Lead partner:	ICCS/NTUA
Authors:	François Carrez (UNIS-editor), George Kousiouris, Achilleas Marinakis, Orfeas Voutyras, Panagiotis Bourelos (NTUA), Adnan Akbar (UNIS), Paula Ta-Shma (IBM), Bogdan Târnaucă, Leonard Piţu (SIEMENS), Juan Sancho (ATOS), Andrés Recio Martín (EMT)
Internal reviewers:	Bogdan Târnaucă (SIEMENS), Juan Rico Fernandez (ATOS)

www.iot-cosmos.eu

SEVENTH FRAMEWORK PROGRAMME

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	17/12/2014	G. Kousiouris	NTUA	Adaptation from Y1 Version
0.2	08/02/2015	F. Carrez	UNIS	Update on Domain Model section. Creation of Information and Instantiated Domain Model, update of Functional Decomposition (Functional View) according to Requirement Mapping
0.3	17/02/2015	G. Kousiouris	NTUA	Added comments on text
0.4	18/02/2015	J. Sancho	ATOS	Misc. changes
0.5	19/02/2015	A. Akbar	UNIS	Remove Interpolation/Extrapolation and update accordingly
0.6	06/03/2015	F. Carrez	UNIS	Updates on new topics from v02 + system use cases
0.7	12/03/2015	P. Ta-Shma	IBM	Review of section 8 / IoT Service Functional Group
0.8	18/03/2015	B. Târnaucă	SIEMENS	General description for ontology support in COSMOS
0.9	26/03/2015	P. Bourelos	ICCS/NTUA	Updates to sections 8 and 9 (System Use-case)
0.11	08/04/2015	A. Marinakis	ICCS/NTUA	Updates to sections 8 and 9 (System Use-case)
0.12	14/04/2015	P. Ta-Shma	IBM	Updates to several subsections in section 9
0.13	14/04/2015	B. Târnaucă	SIEMENS	Added description of the COSMOS Ontology, description of the Semantic Topic Management FC and description

				of the VE Resolution FC
0.14	15/04/2015	G. Kousiouris	ICCS/NTUA	Added updates in Section 8 and 9
0.15	17/04/2015	F. Carrez	UNIS	Draw and add most of System Use-case schemas (&review)
0.16	18/04/2015	L. Piţu & B. Târnaucă	SIEMENS	Update to Risk Analysis and Contributions to Information View
0.17	20/04/2015	J. Sancho	ATOS	Update CEP and SA Functional Components
0.18	21/04/2015	F. Carrez	UNIS	Revise Methodology section
0.19	22/04/2015	B. Târnaucă	SIEMENS	VE Registration FG, VE Retrieval FG, IoT Service Registration and Resolution
0.20	24/04/2015	A.R. Martín	EMT	Pictures in section 10 (Deployment View)
0.21	25/04/2015	F. Carrez	UNIS	Deployment View and document finalisation/review before formal internal review
1.0	29/04/2015	F. Carrez	UNIS	Finalisation of Deliverable after review and final check before submission

Annexes:

Nº	File Name	Title

Table of Contents

Table of Figures

List of Tables

Table of Acronyms

Acronym	Meaning
ACL	Access Control List
AE	Augmented Entity
AES	Advanced Encryption Standard
API	Application Programming Interface
(IoT) ARM	(IoT) Architectural Reference Model (from IoT-A)
AWS	Amazon Web Services
CB	Case (data)Base
CBR	Case-Based Reasoning
CDMI	Cloud Data Management Interface
CEP	Complex Event Processing
µCEP	micro (lightweight) CEP
CoAP	Constrained Application Protocol
CRUD	Create/Read/Update/Delete
DC	Design Choice
Dx.y.z	Deliverable (from WPx /Task x.y / iteration z)
DM	Domain Model
DNA	Dynamic Network Analysis
DV	Deployment View
e.g.	exempli gratia (for example)
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
H	High (as seen in the Risk Analysis)
HAN	Home Area Network
HMAC	Key-Hash Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HW or H/W	Hardware
I2C	Inter-Integrated Circuit
ID	IDentifier
i.e.	id est (that is)
IM	Information Model
IoT	Internet of Things
IoT-A	Internet of Things – Architecture (FP7 EU Project)
IT	Information Technology
Json	Java-Script Object Notation
KEM	Key Exchange and Management
KPI	Key Performance Indicator
L	Low (as seen in the Risk Analysis)
LDAP	Lightweight Directory Access Protocol
M	Medium (as seen in the Risk Analysis)
M2M	Machine-to-Machine

MAPE-K	Monitor/Analyse/Plan/Execute-Knowledge
MB	Message Bus
OWL	Web Ontology Language
P2P	Peer-to-Peer
PAA	Piecewise Aggregation Approximation
P-E	Physical-Entity
PKI	Public Key Infrastructure
QoS	Quality of Service
RA	Reference Architecture (part of the ARM)
RBR	Rule-Based Reasoning
RDF	Resource Description Framework
RDF-S	RDF Schema
REST	Representational State Transfer
RM	Reference Model (part of the ARM)
RPC	Remote Procedure Call
R/W	Read/Write
RSA	Rivest Shamir Adleman (Asymmetrical encryption)
SA	Social Analysis
SAw	Situation Awareness
SAX	Symbolic Aggregation approXimation
SD	Service Description
SHA	Secure Hash Algorithm
SIoT	Social Internet of Things
SNA	Social Network Analysis
SNIA	Storage Networking Industry Association

SPARQL	Simple Protocol And RDF Query Language
SPI	Serial Peripheral Interface
SQL	Structured Query Language
SSH	Secure Shell
STP	Security Trust and Privacy
UC	Use-case
UML	Unified Modelling Language
URI	Uniform Resource Identifier
VE	Virtual Entity
VPN	Virtual Private Network
vs.	versus
WP	Work-package
w.r.t.	With Respect To
XML	eXtensible Markup Language
XP	eXPerience

1. Executive Summary

This WP2 deliverable (D2.3.2) presents the updated version of the document describing the overall architecture of COSMOS. It follows the initial version D2.3.1 released in Year 1.

The main objective of COSMOS project is to provide the mechanisms needed to make Things in the IoT domain smarter while increase also the interoperability of various hardware and software solutions by generalizing the use of semantic and ontologies. Smarter things will behave more autonomously using cognition loop and situation awareness (not focussing only on current situation but also on expected future ones using prediction based on machine learning and Complex Event Processing techniques). They will be also empowered with social capabilities like experience acquisition and sharing based on strong trust and reputation mechanisms. COSMOS aim is therefore to provide solution developers the means (in the form of platform and functional components) 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 supporting autonomous and social behaviours as outlined above. The work described in the report was carried out in the framework of WP2 – Requirements and Architecture- and this second iteration of the deliverable provides the revised analysis, design and specification of the initial architecture of the COSMOS platform released in Year 1. This document is therefore the second 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* (ARM) from the IoT-A FP7 project [1].

Based upon the outcome of the updated Requirement Collection (see D2.2.2) and the result of the Requirement mapping (conducted at the end of the Requirement collection phase) on the one hand and on the experience gained from the first year of the project on the other hand, a complete COSMOS Functional View has been elaborated and agreed upon. This document also provides a completed COSMOS Domain Model with a fragment of it applied to a specific scenario from EMT. A full description of all Functional Components identified in the COSMOS Functional View is provided followed by a extensive list of System Use-cases which illustrate the usage of those Functional Components (with a large number of Figures showing explicit inter-component interactions) in different situations. Those use-cases then represent typical design patterns that can be reused when designing different scenario use-cases. In addition to those system use-cases, the COSMOS Information View also include a first version of the COSMOS Core Ontology (which complies to the ARM Information Model) and some hints about information storage within the COSMOS platform. Finally few additional views i.e. Physical-Entity, Context and Deployment Views have been added to the Functional and Information Views in order to cover all aspects of the COSMOS architecture like advised in the ARM. This second version of the deliverable is therefore fully compliant to the IoT Architectural Reference Model.

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. Enhancing information reliability

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 unreliable 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). Moreover, data itself is of limited value until it is processed intelligently in order to extract higher-level information or knowledge which can be used in order to make appropriate decisions; processing this unreliable and incomplete data is by no means a trivial task.

COSMOS will address this problem by providing mechanisms for inferring high-level knowledge from unreliable and incomplete data. In this regard, COSMOS will explore different data mining methods based on machine learning and probabilistic theory in order to extract meaningful information from this volatile and incomplete data.

2.2. Embedding intelligence into Things

According to CISCO[1] 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 need to cooperate altogether and interact in order 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 for Smart Cities is that by making objects smarter, cost savings and increased efficiencies are created, thus allowing for long-term economic growth, increased sustainability and energy savings.

COSMOS enhances 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. By enabling them to exploit dynamic social networks, COSMOS enables aspects of collaborative learning and information sharing in order to enhance Things reaction to specific problems in a decentralised way. Moreover, real-time analysis of data achieved, either locally on the Things or centrally, in order to exploit collective information enables situational knowledge acquisition, identification of complex events and conditions and propagation of this information back to the device in order to optimize their 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

[1] As seen @ http://www.cisco.com/web/solutions/trends/iot/portfolio.html (last accessed 27/4/2015)

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 delivers data and information management mechanisms in order to handle the exponentially increasing “born digital” data. COSMOS also develops 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 mechanisms are being developed, exploring the interplay between storage and analytics on networks of data objects. The combination of these data sources and the ability to process them either in real-time or in historical mode offers a powerful mean to applications working on top of these structures to exploit and derive knowledge and create enhanced service offerings. Furthermore, COSMOS provides the ability to dynamically annotate data coming into the platform, thus allowing adaptive metadata indexing, a key feature for a per case exploitation of these data.

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-like [22] 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 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 facilitates 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. Furthermore, privacy preserving mechanisms can be applied on the platform level data management functionalities, through relevant computational components handling division based on access rights, information abstraction and per case filtering of identified sensitive information. The provisioning of reliable and secure data to users and applications are being 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 serves other purposes as well since it is, for instance, a key element in the COSMOS social analysis, and therefore aids in creating a trusted ecosystem in which malicious users are identified and isolated.

2.5. Extension of Things semantics

COSMOS provides the semantic models and the mechanisms for describing the building blocks of its environment. Semantic description of Virtual Entities, IoT services, Data Bus topics or applications, 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 also provides 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 build upon and extend social media mechanisms in order to identify the aforementioned patterns and to incorporate this information into the description of Things. This allows Things to evolve (thus becoming smarter and more reliable) as well as to enhance the management of the networks of Things.

COSMOS provides a decoupled semantic model that is separated in terms of generic concepts, provided in the COSMOS ontology and applicable to all cases, the Social Ontology that covers the social attributes and characteristics and finally the Domain Specific Ontologies that capture the specificities of each use-case, (including specialized characteristics of devices, domain concepts and attributes). Through these capabilities extendibility of the semantic model is enhanced, decoupled from each domain, while retrievability based on specific features and criteria can be applied, for management or application definition aspects.

2.6. Differences from initial version (D2.3.1)

In this second version of the COSMOS Architecture document, structural changes and additions of new content have been brought as follows:

- Major revision of Methodology section;
- Structural change according aligned with the adopted methodology;
- The initial COSMOS Domain Model proposed in D2.3.1 has been fully extended and new instantiated Domain Model for the Madrid scenario is provided, as well as the instantiated Information Model;
- Updated Functional View according to the Requirement Mapping achieved in Task T2.2; new components have been identified,. It also includes component description and high-level interface descriptions;
- First iteration of the Information View. This view explains how information is handled in the system and in particular provides the used ontologies and also gives a comprehensive list of System use-cases, as well as collaboration of existing components in order to achieve an added value functionality;
- Extension of the Risk Analysis section, including a STRIDE-based categorization and DREAD analysis;
- First iteration of the Deployment View Section. A first draft of the Deployment View for the Madrid scenario is introduced. It will be completed along with the two other Camden and Taipei Scenarios in the final iteration of this document.

3. Methodology

As a primary decision of the project, it was decided that the IoT *Architectural Reference Model* (ARM) [1], proposed by the IoT-A project [4], will be used for deriving the Architecture of COSMOS. Using the IoT ARM, our expectations were then to agree on a common vocabulary and grounding, structure and methodology to rely on for specifying the COSMOS architecture. The two first iterations of the COSMOS architecture documents have shown that this objective was reached. All partners have been adopting the same language and having for instance all components described in the same framework utilizing the same viewpoints has helped identifying issues and reach common understanding. The remaining of this section reminds some basics about the IoT ARM (and methodology) and show how this methodology was applied to COSMOS (which gives also the structure of the document).

3.1. IoT-A Methodology

The IoT ARM, consists of three interconnected parts which are the *Reference Model* (RM), *Reference Architecture* (RA) and Guidance. The IoT 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 IoT 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. It also provides a complete process for handling the Requirement collection process (see below).

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* (DM) [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 (see Figure 2 in Section 4.3) 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 have actually generated this customized version at the beginning of the COSMOS project and updated it in this second iteration of the architecture deliverable. This Domain Model helped us to speak one voice as far as the COSMOS vocabulary and concepts are concerned. The COSMOS Domain Model is available in Section 4.

As we listed above, the RM also provides, in addition to the Domain Model:

1. an *Information Model* (IM) which is a meta-model used to describe information as handled within the system. This model is not expected to be updated. However we will provide instantiated (and therefore compliant) Information Model for each of the COSMOS Scenario in the third release of the Architecture document;
2. a Communication Model: The IoT Communication Model aims at defining the main communication paradigms for connecting elements, as defined in the IoT Domain Model. It provides a reference set of communication rules to build interoperable stacks, together with insights about the main interactions among the elements of the IoT Domain Model;
3. a *Functional Model* (FM) used as the foundation of the Functional View. This model shows the Functional Groups as proposed in the IoT-A ARM. The Functional Decomposition process, based on this Functional Model and the identified set of requirements (as found in D2.2.2 deliverable – Annex 1) leads to the COSMOS Functional View (which can be found along with all component descriptions in Section 8 with the view it-self being summarised in Figure 7);
4. a *Security, Trust and Privacy* (STP) Model: that defines rules and ways to mitigate them (e.g. anonymisation for privacy) for the three domains of Security, Trust and Privacy;

Please note: *In the rest of the document we don't refer explicitly to the Communication Domain and STP Model. However the risk analysis from IoT-A and Security Functional Component are referred to in the Functional View.*

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 (focusing on the quality of the system that spans different views, e.g. Security, Resilience, etc.). For general information about Views, Viewpoints and Perspective related concepts, please refer to Rozanski & Woods work [6] and to Carrez *et al.* [1].

The "generic" IoT *Functional View* (FV) proposed by IoT-A (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 IoT DM, together with a set of essential *Functional Components* (FC) (and associated interfaces) that an IoT system should provide (those are part of the "generic" Functional View as proposed by IoT-A). This list of "initial" FC is the result of the requirement analysis based on the IoT-A Unified requirements (abbreviated UNIs) and this FV is the IoT-A functional View). Of course, in the COSMOS project we had our own requirement collection process and therefore came up with our own requirements- some of them being totally new, some being existing (or adaptation of) UNIs from IoT-A-.

The Functional Decomposition we obtain when doing the requirement mapping to the Functional Model, generates a COSMOS Functional View (see Section 8.1) which is close to IoT-A one (but the renaming of some Functional Components). However few more functional components specific to COSMOS requirements have be identified and added within the relevant Functional Groups.

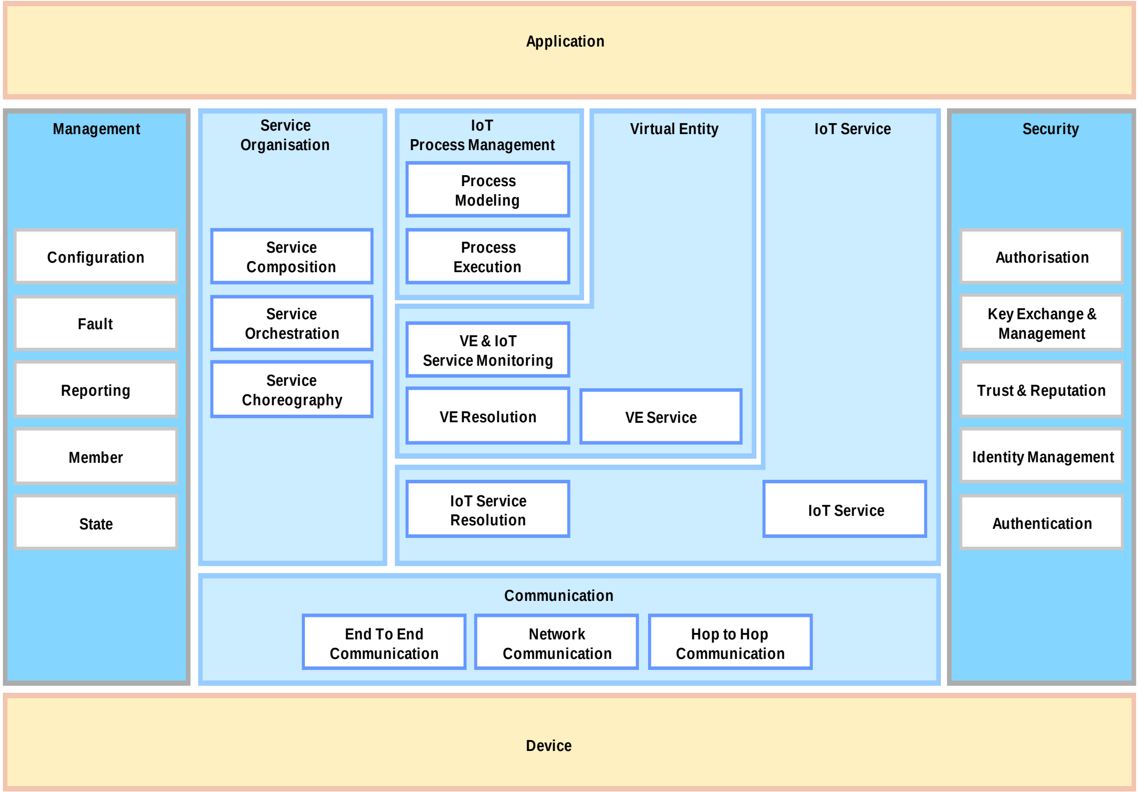


Figure 1: IoT ARM "native" Functional View (from IoT-A)

It is worth mentioning that the IoT FV proposed by IoT-A is not exhaustively developed and focuses on essential building blocks that should be considered for any IoT architecture development.

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, which is referred later on as System Use-cases. The perspectives are mainly derived from non-functional requirements and consist of activities and related tactics.

Last, but not least, the IoT ARM also provides a comprehensive Guidance Chapter. 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 IoT ARM as they are too much application dependent. It also explain how and in which order the set of architectural views (which constitute a concrete architecture according to Rozanski & Woods [6]) should be generated. It finally gives a large (but not exhaustive) list of design choices that can be used as recommendations to achieve certain system qualities (see perspectives above).

3.2. Applying the IoT ARM to COSMOS

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. A first version was discussed and made available 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. A second complete version of the Domain Model has been elaborated for this deliverable iteration in Section 4. We introduce the Information Model of IoT-A and will propose instantiated ones (one per scenario) in the last iteration of this document in Year 3. Then we define the Physical-Entity (P-E) and Context views of the system (Section 6).It is worth noting that the Physical-Entity View and Context View are not exhaustive in the sense they capture specifics of the Use-Cases coming from Madrid (in this second iteration). The work on the Physical-Entity view is described in Section 6.1.1 while the context view is described in Section 6.1.2.

Section 7 provide a major revision of the Risk Analysis taking into account the Generic Risk Analysis from the IoT ARM.

The second iteration of the requirement collection, analysis and mapping has been described in deliverable D2.2.2. Based on this input, we have completed the list of Functional Components already available in the previous document iteration and placed them within the diagram of the Functional Model, resulting in Figure 7. Section 8 also provides a summarised list of Functional Components in addition to a more complete description that also gives high-level interface definitions.

Section 9 is focussing on the Information View and provides different viewpoints: System Use-cases that describe typical usage (pattern) of the Functional Components with inter-component interactions, ontology description and information about data storage as handled in COSMOS.

Section 10 provides finally an introduction to Deployment View and a preliminary version of the DV for the Madrid Scenario followed by a Conclusion in Section 11.

4. COSMOS Domain Model

4.1. Introduction

In COSMOS we strictly follow 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

Physical Entities (P-Es): Physical Entities are the objects from the Real-World that are virtualized in the cyber-space using Virtual Entities. In COSMOS the P-E are smart objects in the sense that they have perceptions (provided by the attached Sensors), react to changes in their environment, take decisions and enjoy a certain level of autonomy and can be acted on using attached actuators. In addition they establish social relations (e.g. follower, followee etc.) based on Trust and Reputation and eventually exchange knowledge and experience. Of course all those features are embodied within the concept of Virtual Entities.

Virtual Entities (VEs): VEs are at the heart of an IoT system. They are a representation of physical objects in the Cyber-world as stated above. Properties of the VEs 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. The IoT Domain Model makes a distinction between VEs being active (called Active Digital Artefacts) in the sense they are equipped with some “service or business logics” and VEs being passive (called Passive Digital Artefacts) –typically a database record describing some P-E properties-. In COSMOS, VEs are generally modelled as Active Digital Artefacts.

Typical examples of VEs in COSMOS will represent and act on behalf of P-E:

- Bus-VE, Bus-stop-VE and Traffic light VEs represent respectively buses, bus stops and traffic lights P-Es in the Madrid use-case;
- Flat-VE represents flats/household P-E in Hildebrand use-case.

VEs are therefore the digital extension of the Physical-Entities that are used to manage, monitor, control their physical counterparts P-Es. They may have their own goals and be equipped with an internal logic in order to achieve their goals. VEs 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 VEs may interact with other VEs for various purposes like:

- collaboration (sharing a common goal with other VEs);
- cooperation (getting help from other VEs in order to achieve their own objective);
- giving access to VEs properties/attribute (via access to VE-centric IoT Service);
- giving access to raw data (via access to r-Centric IoT Service);
- offering actuation service (via VE-Centric or r-Centric IoT Service).

A P-E can be associated with more than one VE, where for instance each VE would focus on a specific facet of the P-E (for a given Bus-P-E we could have one associated VE$_{mngt}$ (Passive Digital Artefact – as mainly a database entry) focussing on management aspects while another associated VE$_{op}$ (Active Digital Artefact) would be more focused on real-time properties of the bus during operation. In any case, a VE cannot be associated with more than one P-E.

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

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

From the Domain Model point of view, VEs interact with IoT Services, which expose using standard protocol, the functionality provided by Resources (which often rely proprietary protocol stacks). 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.

VEs –as software components- do not have to "run" at the P-E side. Where the VE is deployed is a deployment issue that can be made explicit within the Deployment View. Pragmatically and depending on the Devices supporting the P-E, a VE could be part of a micro-controller that also manage Sensing and actuating, could be partially deployed between a gateway and a micro-controller (smarter functionalities being part of the gateway while the IoT Resources would be hosted within the micro-controller), or hosted in the cloud; many options are possible depending on the nature of the P-E, the devices attached to it and the complexity of services it offers through its VE(s).

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

Typical examples of GVEs in COSMOS are:

- Bus lines-GVEs 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;
- House-GVEs relying on Flat-VEs in the Smart Energy management (Hildebrand scenario).

Augmented Entity (AE): Even of not directly used in COSMOS, it is worth mentioning that the IoT-A Domain Model introduces the concept of Augmented Entity (AE). The AU is the composition of a VE and the P-E it is representing in order to highlight the fact that the two entities belongs together and make a whole entity. Augmented Entities are the "things" in the term Internet of "Things" as an AE represents both the physical and digital aspect of the thing.

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 (r-IoT Service) are exposing the IoT Resources using standardised interfaces and possibly adding meta-data to the raw reading available at the resource level; They all connect to an sole resource (sensor or actuator);
- VE-centric IoT Services (ve-IoT Service) are associated to the VEs and are used for accessing VEs attributes/status or to access VE-level services not directly connected to VEs attribute or situation; In the Functional View the VE Service FC deals with such accesses;
- integrated IoT Service (i-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 r-IoT Service in order to know about 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-IoT Services (maybe enriching the raw data with meta-data like {location, timestamp}) via a REST / CoAP interface. A Bus Line-GVE could then access the characteristics of the Bus through the ve-IoT Services exposed by each individual Bus-VE; Of course a Bus-VE 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 i-IoT Service when for instance one of its attribute is built up from aggregation of individual VEs 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.

VE properties: All of the above mentioned service types are wrapped under the VE property concept. This provides an unified way of describing, retrieving and accessing these services. VE properties allow a richer and uniform semantic description despite the differences between these services. This is guaranteed by the mandatory attributes required to be filled in during the services’ endpoints description.

Services: Services (without IoT prefix) are associated to VEs and GVEs 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). 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 revised version of the COSMOS Domain Model as was shown in the previous version of this deliverable. It adds Digital Artefacts and Applications to the existing VEs (individual VEs and Group VEs), IoT Resources, Services (IoT Services and “classic” Services and Devices) and shows more detail about how those key concepts relate to each other.

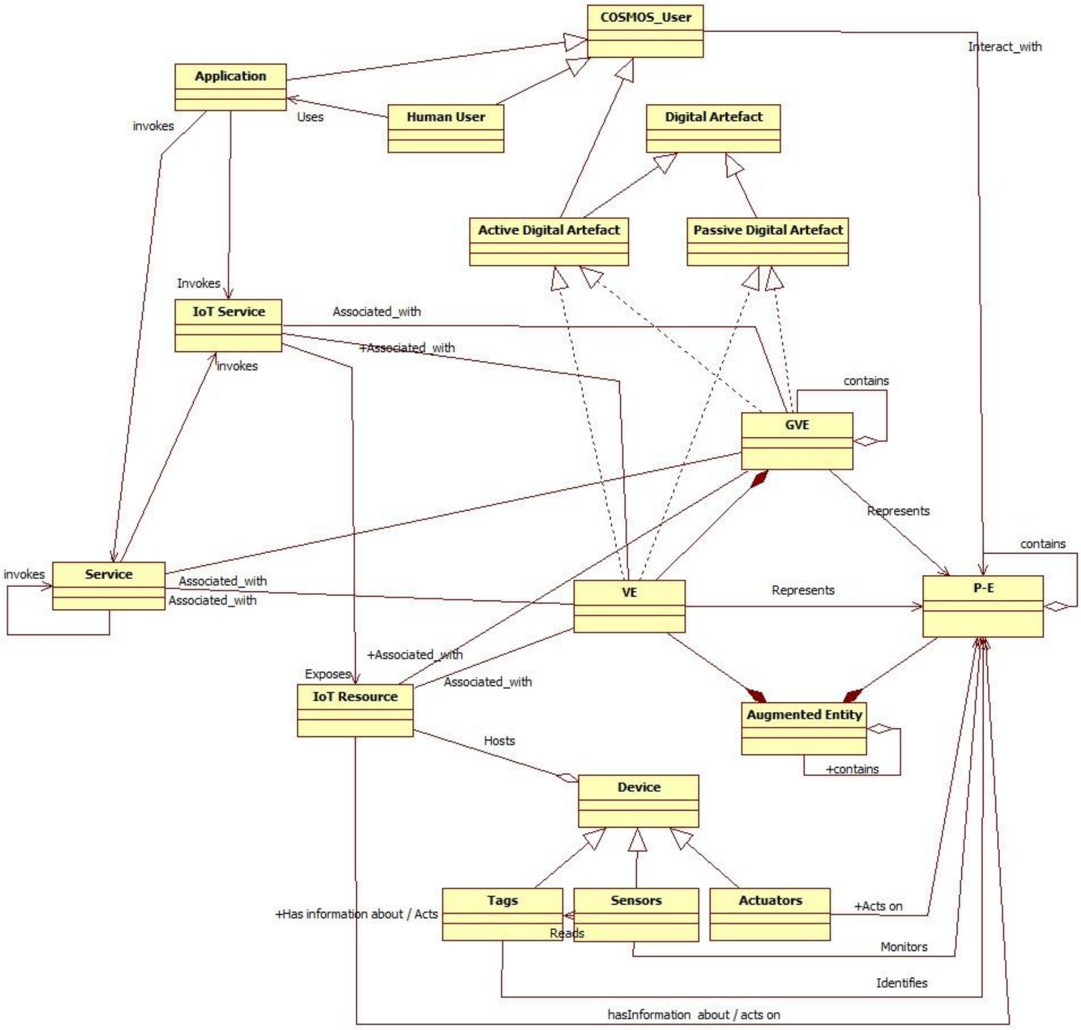


Figure 2: COSMOS Domain Model (revised)

4.4. MADRID Scenario instantiated Domain Model

This model shows not only classes but the different instances that are part of the EMT scenario.

1. COSMOS-user:
   - **CareGiver:** The care giver is monitoring the SpecialPerson along his/her journey via the CareGiverAPP. He is also responsible for configuring the journey via the SpecialPersonAPP;
2. Physical-Entities: The physical entities supervised by the COSMOS platform are

   - **Bus-P-E:** they are the physical vehicles that transports citizens in Madrid. They are equipped with various sensors (CANBUS, GPS,…) and can communicate with the COSMOS platform via cellular communication;
   - **BusLine-P-E:** A bus line is not a physical object as such but a set of bus sharing an itinerary;
   - **TrafficLight-P-E:** They control Pedestrian and Vehicle crossings and have high speed communication to the EMT center. They report status and can be actuated individually;
   - **BusStop-P-E:** they are scattered among the city and can be shared my more than one bus line;
   - **BusDriver-P-E:** The driver operates within the bus and apart driving the bus itself, he is also notified about incoming "specialPerson" through the BusDriverAPP hosted on his smartphone;
   - **SpecialPerson-P-E:** Person needing special attention along his/her journey when entering the Bus;
3. Virtual Entities: Associated VEs in the Madrid scenario are:
   - **BusVE:** The BusVE represents the Bus-P-E. It is associated with many attributes and properties that can be accessed using dedicated IoTServices;
   - **BusStopVE:** The BusStopVE represents the BusStop-P-E;
   - **SpecialPersonVE:** it represents the SpecialPerson-P-E;
   - **BusDriverVE:** it represents the BusDriver-P-E;
   - **TrafficLightVE:** Represents the TrafficLight-P-E;
4. Group Virtual Entities
   - **BusLineGVE:** The BusLineVGE comprises the BusVEs sharing the bus line itinerary. Assignement of BusID to a Bus line is very dynamic and can change on daily/hourly basis;
5. Devices
   - **GPS:** is attached to the BUS-P-E and provides its location;
   - **CANBUS:** is a set of sensors dedicated to the BUS as a vehicle. It is attached to the Bus-P-E;
   - **BusPMV** (bus Electronic Panel) displays information relating to the bus (which line it is part of e.g.);
   - **BusDriverPanel:** is used to realise soft actuation towards the driver. It allows to display messages that relates to the SpecialPerson;
6. Applications:
   - **CareGiverAPP:** application running on a smart device used to follow up on the Person needed special attention journey;
   - **MobilityApp:** is used to find a route (matching a Bus itinerary) that fit a journey plan;
   - **BusDriverAPP:** application used by the driver to get in touch with information related to the scenario;
   - **SpecialPersonAPP:** The SpecialPersonAPP application is in particular responsible for the handshake with the BusDriverAPP resulting in the driver being aware of a SpecialPerson entering the bus.

The following schema shows a fragment only of how the MADRID scenario looks like Domain Model-wise (what we call here MADRID scenario instantiated Domain Model). It shows in particular some of the relation existing between a high level CareTakerAPP and the underlying COSMOS Services associated with the MADRID scenario-specific SpecialPerson VE and the most general MADRID use-case VEs BusVEs, LineGVEs and BusStopVEs.

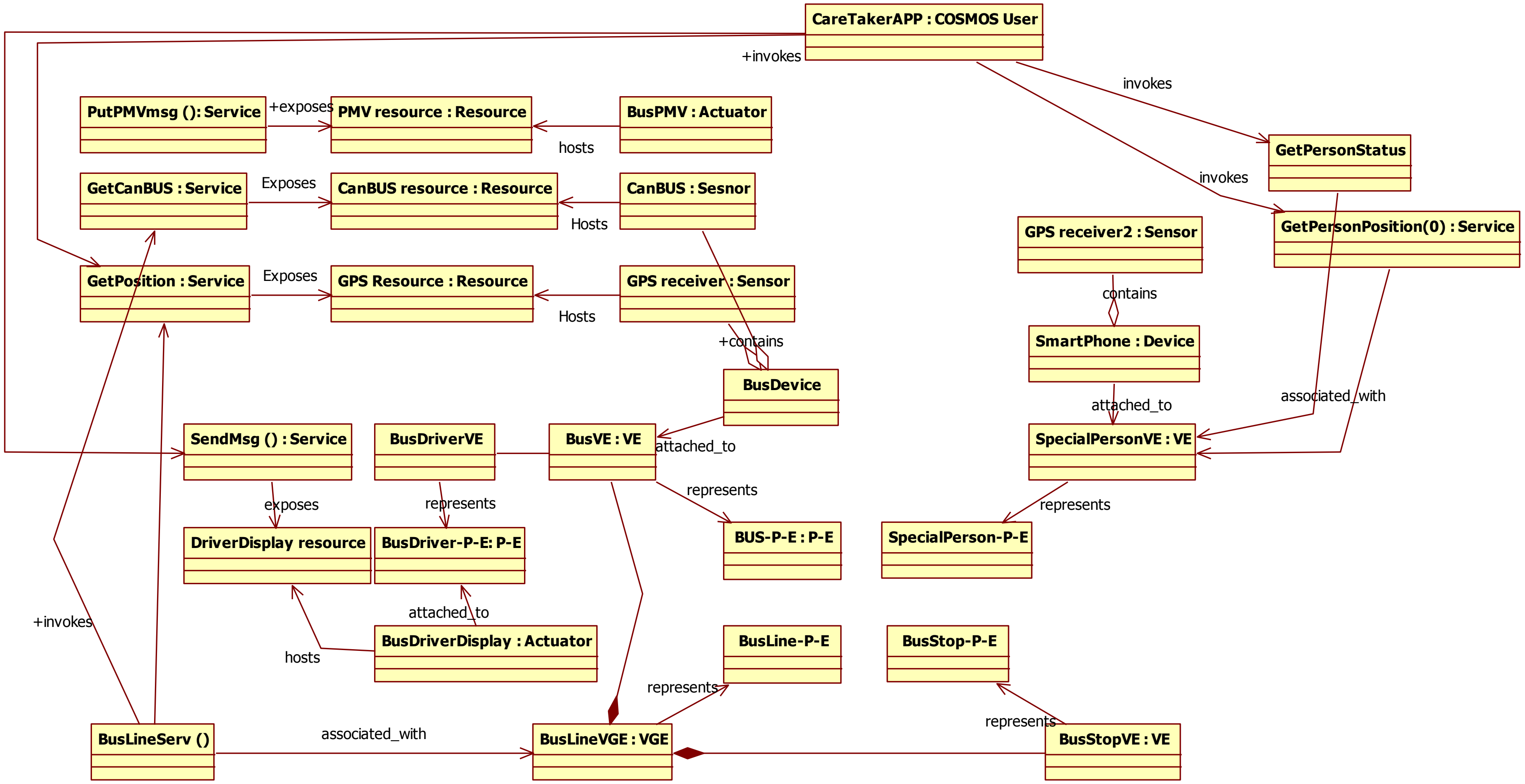


Figure 3: Instantiated Domain Model for EMT scenario (fragment)

5. Information Model

The second model introduced by the IoT ARM, as part of the Reference Model, is the Information Model. In this Chapter we describe and explain the IoT IM in general; this allows to understand later on (Section 9) how COSMOS Information View complies to it, as far as the different choices we make in term of ontology use and encoding are concerned. In the third version of the deliverable we will instantiate the IM into three declinations corresponding to each of the COSMOS Scenarios.

The IM focuses on the description of the structure of Virtual Entities as a representation in the cyber-space of Physical Entities. The representation of the information (either it is encoded in XML, RDF, binary or any other) is kept away from the Information Model and left to the architect's choice or to the scope of some Design Choices when dealing with Perspectives.

The central part of the IM consists of the structure of the VE which is modelled using a set of attributes and which are associated (via the association relationship) to Service Description. These associations make the link between an attribute and a corresponding get<attribute> function (r-IoT Service) for instance (in this case a readable attribute) (resp. set<attribute> for an attribute relating to actuation).

The Attribute is the aggregation of one to many Value containers. Each of those containers contains one single value and one to many metadata (e.g. time stamp, location, accuracy, …).

VE are described through Service Description where each Service would be characterised e.g. by its interface or any useful information that a lookup service can exploit (e.g. the COSMOS VE Resolution or IoT Service Resolution).

As IoT Service are exposing Resources which are themselves hosted by Devices, the IM authorises Service Description to contain 0 to many Resource Description(s) and Resource Description to contain 0 to many device Description(s). The structure of Descriptions is not constrained by the IM and therefore left to the Architect's own choice.

The IoT Information Model is shown in Figure 4 below.

The information as it is handled in COSMOS follows strictly this meta-model (which again does not make any assumption on how the meta-schemas are encoded). For the various level of description, COSMOS bases the work on Ontologies and RDF triples for the serialisation. The VE Resolution FC manages the associations between VEs and IoT-Services.

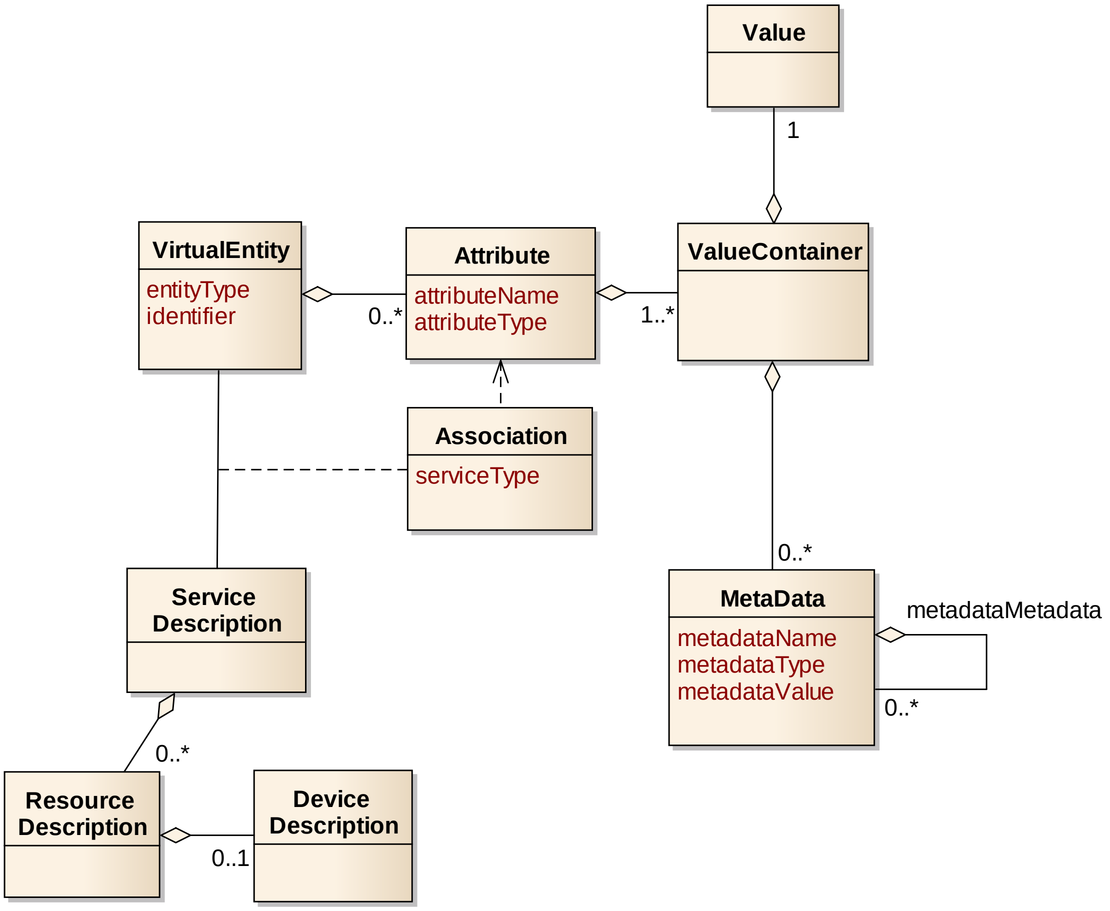


Figure 4: IoT Information Model as defined in the IoT ARM

6. COSMOS Physical-Entity and Context views

6.1. 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 Scenario 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 (see below), 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).

6.1.1. 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 P-Es (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.

6.1.2. IoT Context View

According to the ARM, the IoT Context View consists of two different parts: 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 (e.g. all VEs and GVEs) 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.

6.2. Context View for Madrid Scenario

6.2.1. Introduction

In Madrid, EMT operates the city bus lines (in total 203 lines) through a fleet of 1900 vehicles, which have an average age of 6.04 years. In 2011, EMT operated a total of 7.11 million hours and 91 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 the green phase of the traffic light 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 his family 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.

6.2.2. Physical-Entity View for Madrid Scenario

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;
   - It communicates 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:
   - 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 accessed 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);

6.2.3. Context View and Use-case Architecture for Madrid Scenario

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.

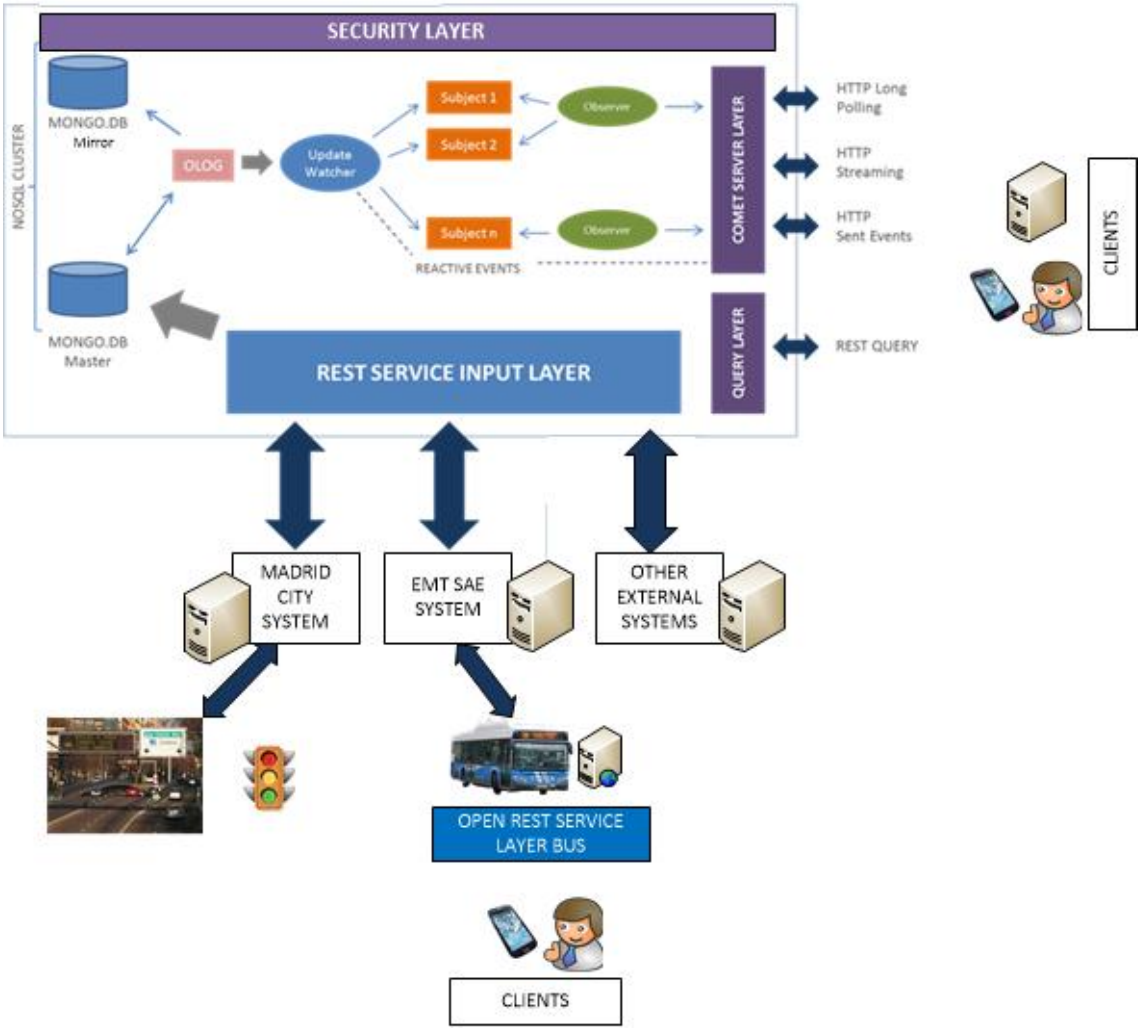


Figure 5: Madrid Use Case Architecture

The use case number 4 "Special Person needing care" 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.

Based on the Virtual Entities defined in Section 4.4 we can further define a scenario which can be divided in different phases. All depicted phases are not necessarily to 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.

6.2.3.1 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, BusVEs and TrafficLightVEs 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 MobilityApp also subscribes to information coming from the BusVEs and TrafficLightVEs.

6.2.3.2 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 SpecialPersonVE and CareGiverVE. The user is able to associate a SpecialPersonVE with a CareGiverVE.

6.2.3.3 Third phase: Application activation

The special person selects the planning of the day in the application and activates it. The SpecialPersonVE contacts the MobilityApp and receives the route for the plan.

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

6.2.3.4 Fourth phase: Information coming from various sources

BusVEs and TrafficLightVEs 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.

6.2.3.5 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 SpecialPersonVE starts a dialogue with the BusVE. 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 SpecialPersonVE remains linked to a particular VE-BUS including displacement of meters. The BusVE, from that time, may incorporate SpecialPersonVE data and attributes forwarding them towards the VE-Mobility. The CareGiverVE is notified of this event.

6.2.3.6 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[2] 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 CareGiverVE 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 CareGiverVE:

[2] It is worth mentioning here that by using this service the couple (SpecialPerson, CareGiver) accepts the principle of being tracked by the COSMOS system. There is therefore in this scenario no breach into user privacy whatsoever.

- To keep track of the SpecialPersonVE;
- 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 SpecialPersonVE travelling aboard;
- To get alarms generated by lack of foresight on the route.
- To get alerts from the user (urgent help or panic button).

6.2.3.7 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.

6.2.3.8 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 predict the position of a bus.

6.2.3.9 Ninth phase: Autonomous behaviour and experience sharing

Moreover, the CareGiverVE 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 CareGiverVE. 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 SpecialPersonVE gets off at an unknown location a solution needs to be found. When this occurs, the CareGiverVE 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.

7. Risk Analysis

As every IT system, COSMOS is subject to security vulnerabilities which can be associated to threats. These concerns can have different root causes and affect one or more aspects of the COSMOS platform. A detailed risk analysis has been carried out in the IoT-A [reference] with the aim at identifying and classifying risk pertaining the IoT. Regardless of the risk analysis method used there are 2 major steps to be performed: identification of critical components which are subject to threats and the actual threats that may be affecting the identified components. Based on the definition of both use-cases and threats, risks are identified and categorized based on their probability of occurrence and impact on the system. There are 2 major approaches to risk analysis where the first one is based on human experience and a set of assumptions while the second one is based on architectural models. The latter one is the method of choice used in the IoT-A – a method more suited for IT systems with large-scale communication networks. Also the current risk analysis has been performed according to the recommendations of the National Institute of Standards and Technology [7] , as depicted in Figure 6.

The first risk analysis resulted in the high-level identification of risks as depicted below.

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	🟧

As proposed by the IoT-A a more fine-grained risk analysis has been performed. Firstly a list of elements to be protected is defined. COSMOS is aiming at enhancing smart cities applications thus acts as a "middleware" between people and a great number of technologies. Thus the main asset to be protected is represented by the human actor. Security attacks may not physically harm the human actor but may provide losses or cause discomfort. The human's susceptibility to security attacks is highly dependent on the end application thus the analysis should be extended for every given scenario in order to cover all possibilities and therefore mitigate the identified risks. As with every human factor privacy is of greatest importance and lies, right next to security, at the basis of every IT system which targets humans. The term privacy reflects information which a human considers personal and does not agree to share. As COSMOS is a cloud platform various communication channels and data carries can be used to push or pull data from it. A major asset to be protected is therefore the communication channels. Threats targeting communication infrastructure are wide spread, as it can be seen on http://www.digitalattackmap.com/. An additional element to be protected are the so called devices such as sensors, actuators, gateways, etc. which form the IoT-A device mesh. These devices, ranging from very low to very high computing power, are subject to a great number of security threats as depicted in [17]. Backend services, Infrastructure Devices and Global Systems form the COSMOS IT infrastructure (e.g. servers, gateways, mainframes, DNS, etc.) represent the backbone of the platform. If compromising a "device" leads to localized data loss and/or device malfunctions, successfully executed attacks on these services and devices can cause the entire platform to crash. Threats targeting backend services are on the rise as [18] shows with the major targets set on industrial espionage and data theft. Still COSMOS relies on IT infrastructure thus merely being a "user" of it therefore IT threats and risks targeting the IT infrastructure itself are not in the focus of the COSMOS project thus no detailed analysis nor mitigation plan are provided.

Identified risks are categorized following the IoT-A model of using STRIDE [19] (**S**poofing Identity, **T**ampering with Data, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, and **E**levation of Privilege). Risks can be categorized as non-human (e.g. natural occurring disasters) and human (e.g. identity theft, hack attacks or human errors).

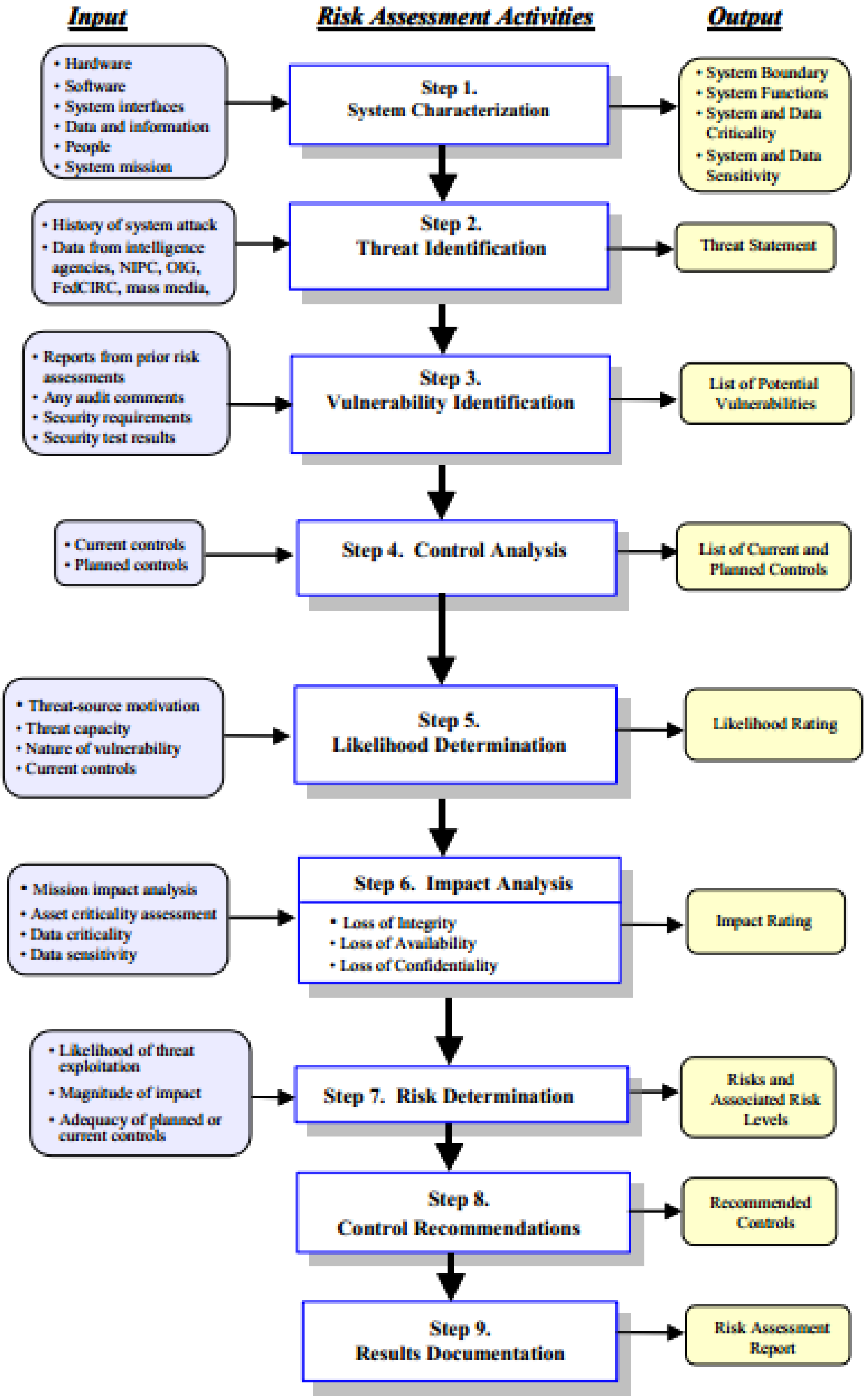


Figure 6: Risk Analysis Flowchart

Table 3: Attacks and RIsks

	Human Actor	Privacy	Communication Channels	Devices
Spoofing Identity	Attacks may generate data on behalf of somebody else.	Identity theft. Human actor interacts with a malicious peer.	Access to restricted services.	Loss of device. Loss of correspondence between VE and physical device.
Tampering with Data	Attacks may lead to wrong/modified data being provided.	-	Wrong service calls. Wrong data push or pull requests.	Loss of control over a device (e.g. actuator). Loss of control over generated data and/or transmitted data.
Repudiation	No proof of integrity and originality can be made.	-	Local DDoS attacks cannot be traced back to their source.	Data is not correctly routed. Data is changed.
Information Disclosure	Theft of information and/or identity can occur.	Attacker gains knowledge of otherwise private information.	Access to restricted data (linked to spoofing identity).	Disclosure of device secrets. Possible changes in data path and/or data content.
Denial of Service	Critical services may fail.	-	Service disruptions.	Device is disabled. DoS to an actuator.
Elevation of Privilege	-	-	Wrong authorization. Wrong information propagation.	-

Using the identified and assessed risk (see Table 3 above) a simplified DREAD analysis is performed. DREAD stands for **D**amage, **R**eproducibility, **E**xploitability, **A**ffected users and **D**iscoverability. As in the IoT-A, the simplified analysis method uses 2 level (L – low, M –

medium, H – high) to characterize each criteria. For each risk a mitigation plan is provided which is further discussed in D3.1.2 where the security and privacy architecture of COSMOS is described (see Table 4).

Table 4: DREAD Analysis

Element to Protect	Risk	DREAD rating	Example of Causes	Mitigation Plan
Human Actor	Attacks may generate data on behalf of somebody else.	H/L/M/L/L	Spoofing attacks, theft of identity	Enforce strong security Enforce ACL Cryptographic protocols
	Attacks may lead to wrong/modified data being provided.	H/L/M/L/L		Enforce strong security Cryptographic primitives & protocols
	No proof of integrity and originality can be made.	L/L/M/L/L	Replay attacks	Enforce strong security Communication protocols Cryptographic primitives
	Theft of information and/or identity can occur.	D/L/H/L/L		Enforce strong security
	Critical services may fail.	H/M/M/L/L		Enforce strong security ACL Restricted access Self-healing capabilities
Privacy	Identity theft.	H/L/H/L/M	Credential theft Spoofing attacks Brute-force attacks Man-in-the-	Enforce strong security Communication protocols Cryptographic primitives

			middle attacks	
	Human actor interacts with a malicious peer.	L/H/H/M/L	Redirection attacks	Enforce strong security Authentication Authorization ACL Message non-repudiation
	Attacker gains knowledge of otherwise private information.	M/M/M/L/H	Unprotected forms Wrong authentication policies	Enforce medium security Weak encryption Communication protocols Anonymity
Communication Channels	Access to restricted services.	H/L/M/L/L		ACL Security protocol
	Wrong service calls. Wrong data push or pull requests.	H/L/M/L/L		Enforce weak/medium security Data integrity checks
	Local DDoS attacks cannot be traced back to their source.	M/H/L/H/L		Enforce medium security Attack identification and localization schemes Attack source isolation
	Access to restricted data (linked to spoofing identity).	M/L/M/L/L		Enforce medium security Security policies Self-healing capabilities
	Service disruptions.	M/H/L/H/L		Enforce medium/high security

				MAC authentication Security schemes Communication protocols One time pad schemes
	Wrong authorization.	M/L/L/H/M		Enforce medium security Enforce cryptographic based protocols Enforce security policies
	Wrong information propagation.	M/L/L/H/M		Pattern identification Attack isolation
Devices	Loss of device.	L/L/H/L/L		Enforce weak security Enforce authentication schemes Use cryptographic credentials based on HW or SW cryptographic primitives
	Loss of correspondence between VE and physical device.	M/L/M/H/L		Enforce strong security Use ACL Use tamper detection mechanisms Communication protocols Authentication & Authorization schemes

	Loss of control over a device (e.g. actuator).	M/M/M/L/M		Enforce strong security ACL Strong cryptographic primitives (HW) Authentication & Authorization schemes
	Loss of control over generated data and/or transmitted data.	H/M/H/M/L		Enforce strong security Authentication & Authorization schemes Communications protocol
	Data is not correctly routed.	M/M/H/M/L		Enforce strong security Authentication & Authorization schemes Strong cryptography Communications protocol
	Data is changed.	H/M/H/M/L		Enforce strong security Authentication & Authorization schemes Strong cryptography Integrity checks
	Disclosure of device secrets.	L/L/L/L/H		Enforce medium security Use identity management

	Possible changes in data path and/or data content.	M/H/M/M/M		Enforce medium security Communication protocols Security policies

Derived from this risk analysis are thus the security requirements which form the basis for WP3 (End-to-End Security and Privacy). Using the risk analysis as a starting point enabled the overall system security to reach further into the system and increase the confidence level.

8. COSMOS Functional View

In this chapter we provide a Functional Decomposition of the COSMOS architecture according to the generic *Functional View* (FV) provided by the IoT-A ARM. The Functional View consists of *Functional Group* (FG) (at the coarse-grain level) and *Functional Components* (FC) within every FG. According to [1] the IoT-A ARM proposes the following list of 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 Resource-centric IoT-services. Through this FG can be accessed also the VE-centric IoT Service like for instance the ones related to experience sharing.
- ***IoT Service FG:*** The IoT Service FG contains functions relating to r-IoT Services. Those services expose the resources like sensors and actuator and provide a mean for reading sensor value or actuating. 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 this section we provide a Functional Decomposition of the COSMOS architecture as a set of functional components implemented within the "generic" IoT-A Functional Groups introduced just above.

- Application FG:
  - **Application Client FC** side components are necessary in some cases to implement the client side logic. These components are either consuming information from the COSMOS platform (and performing other actions such as application specific visualization) but can also publish information towards the platform, provided that they follow the specification to be registered as VEs. In the application process, they may cooperate with an application server side logic, that is hosted by the COSMOS platform.
- IoT Management FG:
  - **Pattern Reusability FC:** this component is responsible for storing template flows (in terms of service workflows) that can be used for multiple purposes and by many roles. It can include for example flows to update and configure a specific service (e.g. update a rules file for the CEP engine), initialization flows

(e.g. topic creation) or application templates that have pre-defined common service combinations. The templates may or may not be configured, based on the availability or flexibility of information that we need to achieve (e.g. the IPs of the COSMOS platform may be set if the templates are shared with other platforms, the topic to which to register may not be set etc.). The purpose of this component is to automate and abstract a large part of the system use cases in order to be used for easier platform management or application design.

- Service Organisation FG:
  - **Message Bus FC** and **Semantic Topic Management FC** belong to the Service FG. The Message Bus FC provides functionality similar to the Service Choreography FC which offers a broker that handles Publish/Subscribe communication between services. The Semantic Topic Management FC, which is used to create delete etc. "semantic" topics associated with published messages is tightly coupled with the Message Bus and thus is considered part of the same FG, instead of being part of the Management FG;
  - **Social Analysis FC:** The Social Analysis component is a Platform positioned component which uses individual VE ranking information to extract on demand recommendations for VEs demanding updates on their Followees lists. This information stems from Social Monitoring component rankings. It is therefore reasonable to locate it within the Service Functionality Group;
  - **Service Orchestration FC:** The Service Orchestration component is a borderline component between Application FG and Service Organization FG. Its aim is twofold. Initially it is to automate a set of processes of the platform that are exposed as abstractions (to either the platform entities or external roles such as application developers). Secondly it is to act like a workflow definition tool through which application developers may create application server side logic that may be necessary for the application to operate (e.g. create respective topics, coordinate application clients etc.) and use platform based services.
- Virtual Entity FG:
  - **Planner FC:** The Planner is a VE side component which uses CBR as a Reasoning method for Case recognition and retrieval. The Planner has two modes of remote Case acquisition when needed. It subscribes to events via the Message Bus (which is realised within the Service Choreography FC and the chosen Solution to the Problem can be an actuation or a message. The second method of Case acquisition is through the Experience Sharing component which provides Cases from remote VEs (also part of the Virtual Entity FG). The Planner may perform rudimentary Solution evaluation at which point with or without End User input, the Social Monitoring component receives the result of said feedback;
  - **Social Monitoring FC:** After a Solution from an incoming Case provided by a remote VE has been evaluated, in the way described in the Planner FC, the Social Monitoring component begins to calculate the ranking updates for the Case provider. Social Indexes that may be updated, include Reputation, Trust and Reliability. The metrics which influence said Indexes, include Shares, Applauses, Mentions and Assists. The Indexes are used in eventually creating the aggregated Dependability Index;
  - **VE Resolution FC:** This component allows to discover VEs, VE-centric services this include all services for accessing VE properties) and associations between

a VE and Resource-centric IoT Services. It also allows to create such VE-r-IoT Services associations;

- **VE-data Pre-Processing FC**: Pre-processing techniques are of fundamental importance for large-scale data applications. In this context, COSMOS will explore several pre-processing techniques at different levels. It explore pre-processing techniques on VE historical data by moving computations close to the storage using storlets at one hand and outside the storage using distributed machine learning platform (Apache Spark) on the other hand;
- **Privelet FC:** The Privelet FC component is involved in both VE2VE and VE2COSMOS communication. Its purpose is to enhance the privacy of the VEs as well as to enable the establishment of trust relationships between them.
  Regarding VE2COSMOS communication, Privelets provide the VE developer with the capability to filter the data that are pushed to the message bus, meaning that the information considered private will not be published.
  VEs will communicate with each other within a P2P COSMOS VPN. using virtual IP addresses in order to retain privacy. Privelets are used to ensure authentication during VE2VE communication so as to avoid impersonation. Private data are again inaccessible and in addition repetitive requests are ignored by the VEs;
- **VE-level Stream Analysis FC (μCEP):** Having a light-weighted Complex Event Processing engine acting at the VE level for the sake of pre-processing is being considered. This engine would interact directly with the IoT Services located at the VE in order to pre-process raw data;
- **Situational Awareness FC:** This component manages the generation of value added information following a SA process, so the acquisition of context of a VE (SA Level 1), its understanding (SA Level 2) and predicted evolution (SA Level 3). It involves the usage of other FCs such as the Message Bus FC, the VE-level Stream Analysis FC (leading in Level 1 & Level 2 SA) and the Event (Pattern) Detection FC (SA Level 3);
- **Event (Pattern) Detection FC:** The combination of different data sources in IoT form patterns; which are complex and dynamic in nature. Different patterns formed by these raw data sources are indicative of different hidden events. In COSMOS, we will develop methods based on machine learning and complex event processing techniques for pattern recognition in order to infer complex events from these data sources;
- **Experience Sharing FC:** This component is responsible for managing the VEs' Experience, answering Experience Sharing requests and possibly propagating such request to other VEs. It also uses Social Indexes, in order to better manage incoming requests during periods of increased incoming traffic and is also responsible for proactively Sharing occurrences that may arise.

- IoT Service FG:
  - **IoT Services FC** are not part of the COSMOS platform but must be implemented by the VE developers as a way to expose the underlying resources. Sensing and Actuation are accessed via IoT Services. IoT Service are dynamically bound to the VE via an association created via the VE Registry component;
  - **IoT Service Resolution:** The IoT service resolution FC provides functionalities needed for discovering and contacting resource-centric IoT services. It also allows to manage the service descriptions. The functionalities include

discovery (based on criteria which are as rich as the descriptions are), retrieval of service locator (REST endpoint), look-up of description based on service ID;
  - **Raw-Data Pre-processing FC:** COSMOS also provides the means for running pre-processing on raw data before storing it. One option is to use CEP for filtering and aggregating raw data streams before storing it. Pre-processing methods running at different levels provide flexibility to the application developer using COSMOS platform;
  - **Inference & Prediction FC:** This component relates to values accessible through IoT-services using machine learning and other related methods. Deployment-wise this module is operating at the cloud storage but could instantiated at the VE level. In this component, we will exploit historical data in order to get more insight from the data and understand it in a better way. The historical data will be used to train Machine Learning models which can be used for prediction with incomplete data;
  - **Analytics FC:** The Analytics FC enables analytics to be performed on COSMOS data, with a current focus on the historical data, although real time could also be included. . It is the place where some data-analytics on historical data can be implemented. Consumers of the results of analytics computations are typically VEs or from the Application FG;
  - **Meta-Data Search FC:** the Meta-Data Search FC retrieves objects from the Cloud Storage FC based on search criteria on the object metadata and through a REST GET API;
  - **Data Mapper FC:** The Data Mapper is responsible for collecting data from the Message Bus FC and forming an object before storing it in the cloud together with metadata. The Data Mapper FC could be realised as an IoT Service in combination with the Member FC that provides functionalities for accessing the VE Registry;
  - **Cloud Storage FC:** The Cloud Storage FC offers a RESTful API for storing/accessing objects (originating from the Data Mapper FC). This FC also provide the ability to inject storlets that can be used to perform pre-processing/aggregation tasks close to the storage.
- Security FG**:**
  - **Authentication FC:** authentication is a functionality offered within the Security FG of the IoT ARM. It relies also on *Key Exchange and Management* (KEM) FC;
  - **Authorisation FC:** Authorisation is natively present in the Security FG as the Authorisation FC and relies on Member FC (see below) for managing actors of the COSMOS platform and allocation of access rights (e.g. who is authorised to access a particular VE or a particular IoT Service offered by a VE);
  - **Key Exchange and Management FC:** this one is also part of the IoT ARM. It covers the entire lifespan of cryptographic keys from generation and distribution to blacklisting or phase-out. Key distribution is based on Diffie-Hellman key exchange protocol and is enforced on a VE level. Key management (as in key storage and right management associated with each key) are also covered by this FC;
  - **Cryptographic Non-repudiation FC:** this FC enforce non-repudiation check based on the H/W Board Communication Accountability FC;
  - **H/W Board Communication Accountability FC:** this FC focuses on the security aspect of Accountability as it is generally understood. It tracks access to the crypto module / security primitives for non-repudiation purpose and computes the so-called reputation index of the VE;

  - **Checksum FC:** this FC focuses on integrity aspects of data packets. It ensures the functional integrity of encrypted data packets by detecting and correcting bit-wise errors.
- Communication FG:
  - **VE2VE Communication Channel FC:** When a VE-a to VE-b communication is involved, VE-a needs to fetch VE-b public key to encode the communication (e.g. an ve-IoT-Service or r-IoT Service invocation) before sending the message to VE-b. Then VE-b uses its own private key to decrypt the message. The protocols repeats itself when VE-b answers to VE-a back. This ensure secure peer to peer communication and peer-to-peer authentication between VEs;
  - **VE2COSMOS Communication Channel FC:** when the H/W Board plugs to COSMOS, it triggers the generation of a RSA key pair by the KEM FC (either implemented as pure software or provided by a H/W Board under the authority of COSMOS). DH is used between COSMOS domain and the newly enrolled H/W Board (preconfigured with a master key) for the purpose of the RSA key-pair distribution and for the symmetric key distribution (in a second round of DH).
- Management FG:
  - **Member FC**: This FC is managing COSMOS users (in the broad sense, meaning Physical users, applications and (G)VEs) and the way they are authorised to access services, (Group) Virtual Entities and Applications. Consequently the ACLs are created and managed through this FC and in addition the enforcement of Authorisation will be heavily based on this FC. This FC covers the management (creation, deletion, updates) of the actors and the management of the ACLs themselves (including revoking). Additionally actors might be associated with accounts that contains some additional information that can be used by other FC outside the Security scope; configuration and handling of password (for physical person) is also part of this FG in cooperation with the Security FG (in particular the authentication FC);
  - **Configuration FC**: Security management as it is described above at the level of FCs is part of the Configuration FC (in particular GUI templates,…), meaning the “configuration” part is handled in the Management FG while the supporting security functionalities are provided by the Security FG;
  - **Fault FC:** in case the system is breached (e.g. a key stolen or recurring/multiple authentication or access failures) the fault management FC is informed by the some of the security FCs.
- IoT Process Management FG:
  - **Pattern Reusability FC**: this component serves as a repository of template flows that may be reused for a number of purposes, as identified in D7.6.2. These flows may be for enhancing application logic, automating platform or VE management and using platform or VE services.

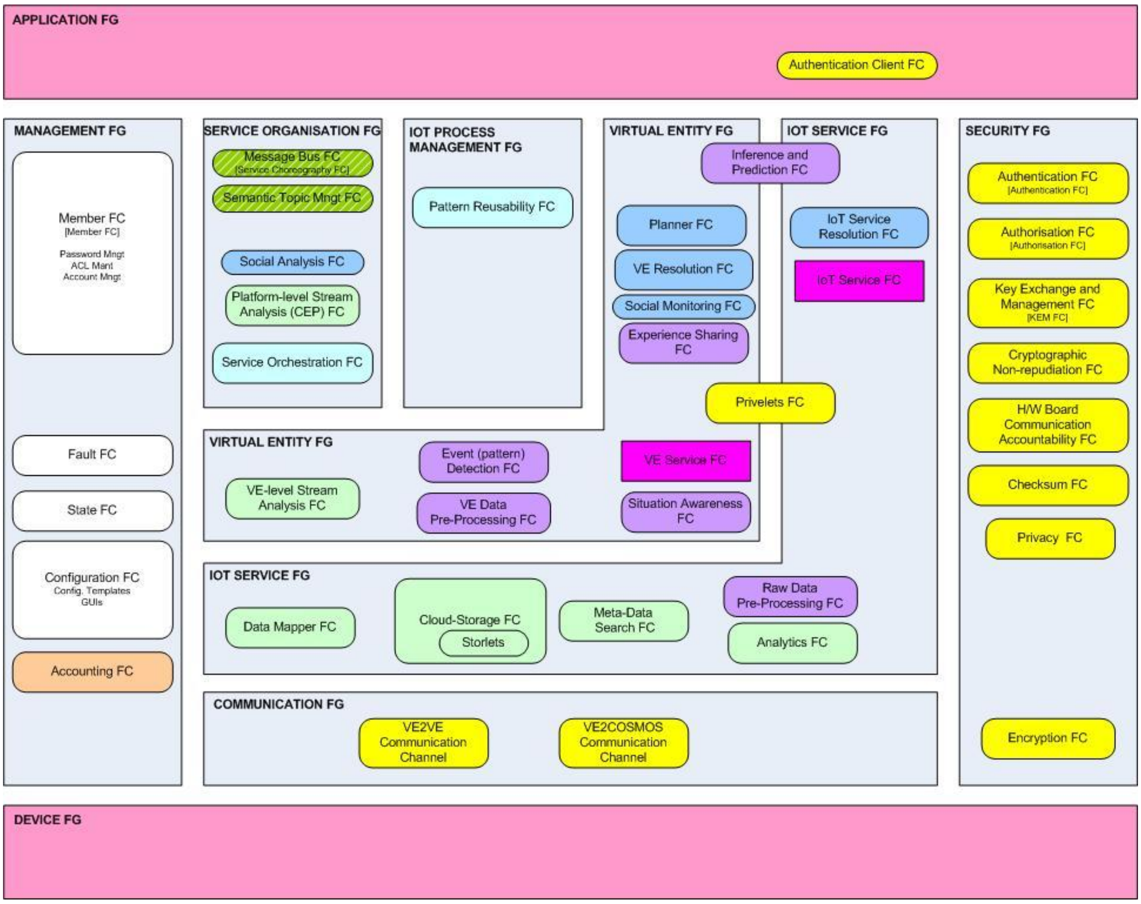


Figure 7: COSMOS Functional View

8.1. Component descriptions

We describe here the Functional Components (structured according to their related Functional Group) which have been under specification and development in Year 1 and 2 only. The list in this subsection is therefore a subset of the list shown at the beginning of section 8.

Please note*: some Functional Components are work in progress and part of the Year 3 objectives; they are therefore not mentioned in this iteration of the Architecture deliverable. The following list of Function Component description is therefore not exhaustive compared to the Functional View shown in* Figure 7 *above, and not all Functional Groups are represented.*

8.1.1. IoT Process Management FG

8.1.1.1 Pattern Reusability FC

Description of the component

As mentioned in the Service Orchestration FC, usage patterns of COSMOS services may be created in arbitrary combinations and integrated in a variety of ways in order to provide added value. The ability to have a structure that is able to handle such flows is key to enabling faster design, configuration and usage of the services.

Furthermore, these patterns may group relevant actions together, in order to abstract internal workings from the average developer and thus aid in the uptake of the platform. In the same rational, ecosystems in which such shareable flows are available to developers would enhance cooperation and building on top of existing applications. This is the purpose of the Pattern Reusability FC.

Functionalities/Interfaces

The needed functionalities of this FC are the abilities:

- To store , retrieve, update and delete configurable flows;
- To merge flows into abstracted functionalities;
- To share and cooperate on a given flow.

Given that this functionality is tightly coupled with the Service Orchestration FC, we will use the built-in features of Nodered[3] that support the aforementioned functionality, as well as incorporating a large user base and existing flow base. The Pattern Reusability FC is thus included mainly for completeness of the conceptual architecture.

8.1.2. Service Organisation FG

8.1.2.1 Message Bus FC

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.

Functionalities/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;

[3] http://flows.nodered.org/

- **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.

8.1.2.2 Semantic Topic Management FC

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.

Functionalities/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.

Figure 8 depicts the block diagram of the component and its main modules and interactions.

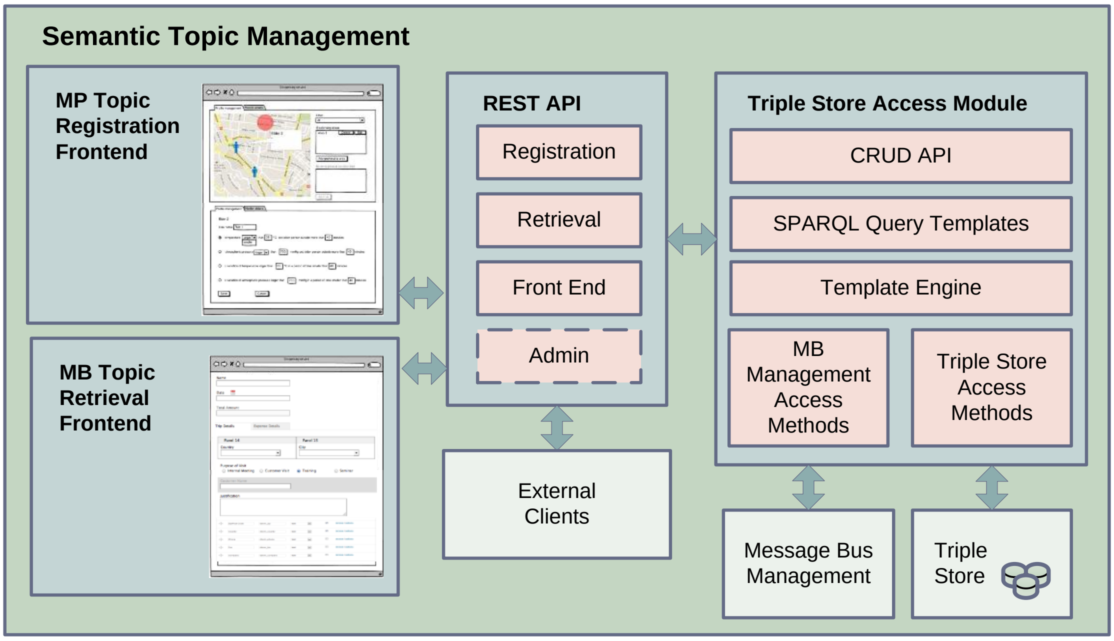

Semantic Topic Management
MP Topic Registration Frontend
MB Topic Retrieval Frontend
REST API
Registration
Retrieval
Front End
Admin
External Clients
Triple Store Access Module
CRUD API
SPARQL Query Templates
Template Engine
MB Management Access Methods
Triple Store Access Methods
Message Bus Management
Triple Store

Figure 8: Semantic Topic Management

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. A front end to support this functionality will also be provided.

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.

8.1.2.3 Service Orchestration FC

Description of the component

In order to aid application developers in combining COSMOS services, data sources and VE capabilities a suitable component/environment is needed for creating the data flows, linking the various services and configuring them appropriately. Thus the added value envisioned by a specific application through arbitrary combinations may be obtained. This framework should be modular, extendable and able to offer testing facilities for the developer, including a set of ready-made operations that would minimize integration effort.

Functionalities/Interfaces

The main functionality that is necessary is:

- Ability to invoke services of the COSMOS ecosystem (VE and platform) based on REST clients;
- Ability to integrate different services, potentially process their output and embed application specific logic;
- Ability to cooperate with a FC (Pattern Reusability FC) that would be responsible for storing reusable flows;
- A GUI for the developers to perform the links.

Again, the tool that has been selected for this case is Nodered as it covers the aforementioned requirements. More details in the usage of Nodered in COSMOS are included in D7.6.2, since it is mainly considered in our case as an integration tool. Its incorporation in this document is mainly to complete the functional view of COSMOS.

8.1.2.4 Social Analysis FC

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).

Functionalities/Interfaces

Based on the results of the Social Monitoring component and taking advantage of SNA, the SA component is used for the extraction of complex social characteristics of the VEs (e.g. centrality), as well as models and patterns regarding the behaviour of the VEs and the relations between them. The services and functionalities of the Social Analysis component will be used by both the users (External use) and other functional components (Internal use). 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 the system. Briefly, the functionalities that have already been presented in the WP5 deliverable are: computation of the Dependability Index of VEs, recommendation of VEs, extraction of structural characteristics of the networks, extraction of relational-models and finally, modelling and visualization of networks.

8.1.3. Virtual Entity FG

8.1.3.1 Virtual Entity Resolution FC

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 Resolution FC reuses some of the characteristic 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 VE Resolution 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.

Functionalities/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.

Figure 9 depicts the block diagram of the component and its main modules and interactions.

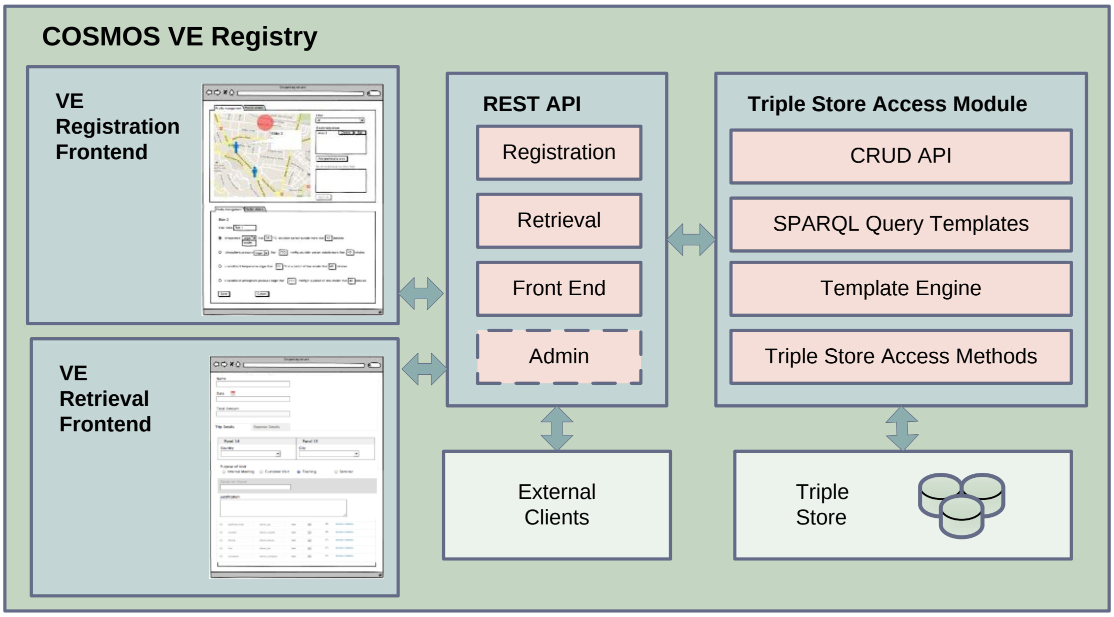


Figure 9: COSMOS VE Registry

The API of the Discovery component would be rich 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.

8.1.3.2 Social Monitoring FC

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.

The relation between a VE and its Followees is trust-based and non-mutual. This means that VE_1 may use the experience of VE_2, but on the other hand, VE_2 may not do the same for VE_1. A VE (*trustor*) trusts blindly its Followees (*trustees*) and requires access to their services.

On the same spirit, examples of **interaction metrics** that are used are:

- **Shares:** How many times a VE has shared its services with other VEs. This value is used as an indicator of the popularity of the VE. However, for a more valuable evaluation, the number of followers, the amount of the shareable resources and the number of the received requests should be taken under consideration;
- **Assists:** How many times a VE has acted as a broker. This value is used as an indicator of the efficient social connectivity of the VE. The concept of Assists is only applicable to requests for XP-sharing and decentralized discovery;
- **Applauses (for Shares and Assists):** How many times the social shares or assists have been regarded as useful from the receivers. This value could be used as an indicator of the trustworthiness and the reputation of the VE. This is a quite important property, but rather difficult to monitor compared to other elements, as feedback regarding the quality of the provided shares/assists is needed;
- **Mentions:** How many times an IoT-service of a specific VE is mentioned in the Case Base of other VEs. This is another indicator of the popularity and reputation of a VE. The concept of Mentions is applicable only to requests for IoT-services.

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 is 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.

Functionalities/Interfaces

This component contains all the main tools and techniques that are used for the monitoring of the social interactions of the VEs (e.g. Shares, Applauses) and of the social properties of the VEs (e.g. Trust and Reputation). Its main objective is to collect, aggregate and distribute monitoring data (events) across the decision making components of the collaborating groups. The events are generated by interactions in response to - directly or indirectly - user actions (e.g. registering a new VE) or VEs' actions (XP-sharing). Social Monitoring forwards its results to the Friend Lists of the VE and can "feed" the Registry with these data on demand or periodically.

8.1.3.3 Planner FC

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 (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 D6.1.2) 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:

1. **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);
2. **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.

The planner has two retrieve modes:

1. 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;
2. 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.

Based on IBM's new autonomic deployment model, there are defined five levels of increasingly sophisticated self-governance of systems: Basic, Managed, Predictive, Adaptive, Full Autonomic. The Planner can reach all three last levels depending on the definition and the usage of the Problems and the Solutions. That way, a VE using a Planner can be:

- **Adaptive:** The VE can not only provide advice on actions, but can automatically take the right actions based on the information that is available to it on what is happening in its surroundings;
- **Full-autonomic:** The operation of the VE is governed by business policies and objectives (expressed by Goals);
- **Predictive:** This characteristic has not been presented yet and is the outcome of a "peculiar" usage of the Cases (using "inverse" logic). The VE itself can begin to recognize patterns, predict the optimal configuration and provide advice on what course of action the administrator should take, as well as predict the outcome of certain actions. For example, if the Problems of some Cases include both Events and Goals, then the VE could predict that by taking certain actions (described by Solutions) after an initial state has been recognized (Events) then the result will be the Goals. In this case, the Planner would create an incomplete Case consisting of an incomplete Problem (Events only) and a Solution and would search for similar Cases to find the Goals (which in this situation would be the predicted future state).

Self-optimisation of the Planner could be achieved by using a more sophisticated classification of the Cases based on the different self-management attributes that the different Cases implement. The classification that could be adopted is the one used for control loop functionalities in self-managing autonomic systems:

1. Self-Configuring Cases;
2. Self-Optimizing Cases;
3. Self-Healing Cases;
4. Self-Protecting Cases.

Thus, the System Case presented in the previous example can be characterized as a Self-Configuration/Optimization Case.

Finally because of the way we designed the Planner, it can be used generally as an **Ontologies Comparator**. That means that the Planner can be used for other services beyond the comparison of Cases that has been extensively described in the previous subsections. In other words, the Planner can reason on other parts of the Knowledge Base too, as long as ontologies are used, using parts of the very same retrieval procedure.

Functionalities/interface

COSMOS will offer various services to the Application developers giving them the opportunity to define CBs and CBR cognition loops in order to build their own COSMOS-enabled Applications. COSMOS could provide a GUI template consisting of fields to fill in, in order, for example, to describe the Problem and give the input for the similarity calculation. The developer could define data for the simple attributes of a Case and weights for its complex attributes. The entered information would then be retrieved to build Cases, while integrating their semantics. All interfaces and functionalities of the Planner pertaining to Case management will be used along some Application logic.

8.1.3.4 VE-level Stream Analysis FC (μCEP)

Description of the component

The improvements made in the CEP engine considered in COSMOS allow the VEs to manage event detection capabilities on their own. Being designed as a component to be fed by an amount of data sources, the µCEP engine ultimate goal is to generate value added information in the form of complex events, that are the output generated after processing many small, independent incoming input events, which can be understood as a given collection of parameters at a certain temporal point. This processing capability can be applied internally at the VEs side for internal detection of temporal and structural patterns among events produced by associated things.

Functionalities

Three different scenarios are conceived for the deployment of a CEP engine:

1) **Internal event detection**: The Virtual Entity enhances its execution environment hosting a lightweight CEP. Additionally to the inherent raw data provided by the local VE, a number of external data sources could feed the engine. Besides that, the output of the engine (complex events) can be locally consumed by the VE or even shared to other VEs or external components;
2) **Event detection as service**: Aims at providing services to components that lack hardware resources or are incompatible for hosting an event processing solution. Depending on information exchange between VEs, other VEs can also subscribe to and consume results provided by this event detection service;
3) **Hybrid event detection**: Under certain circumstances, for instance performance and processing capabilities, it will be valuable to take advantage of the modularization of the µCEP engine. The most common situation will be that in which a VE hosts locally the *Event Collector* module, while the *Complex Event Detector* and the *Complex Event Publisher* module may be hosted in a remote location, where more hardware resources are made available.

Interfaces

VE - µCEP :

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

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

Application Developer - µCEP :

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

- Editing a `*.dolce` file on their own, using their preferred Text Editor. This approach provides full control over the definition of the rules;
- Using a specific WebGUI developed by COSMOS project, what facilitates the creation of a DOLCE Rules file in an easier manner, following a guided wizard.

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

8.1.3.5 Event-Pattern Detection FC

Description of the component

There are many applications in IoT which requires real-time processing of data such as intelligent transportation systems and smart buildings. CEP engines can be deployed for processing, analyzing and correlating event streams from different data sources to infer more complex events in real-time. CEP engines require rules or patterns to detect an event from data streams which have to be given manually by the administrators of the system. Based on this, there is an assumption that administrators have the required background knowledge which sometimes is neither available nor so precise. The manual setting of rules and patterns limits the use of CEP only for expert's domain and poses a weak point.

In this regard, this component exploits historical data and use machine learning techniques for finding rules and provide automatic and adaptive solutions for detecting an event.Functionality

This component provides the following functionalities.

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

Functionalities/Interfaces

This component uses VE-level stream analysis using µCEP and inference/prediction block for machine learning. The interfaces for this component are same as described in the 8.1.3.4 with an additional interface for Inference/Prediction component which is described below.

Event Detection-Inference/Prediction

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

8.1.3.6 Privelets FC

Description of the component

Within the context of COSMOS, VEs are able to communicate with each other as well as with the COSMOS platform. This information exchange has a high possibility of violating the VEs privacy in a way that it is not aware of.

Privelets is a component that runs at the VE side and its role is, acting as a filter, 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.

Moreover, during Year 2 and Year 3, we plan to enhance the privacy of VEs by enabling them to use virtual identities while sharing their data that are not filtered as private. Privelets is also responsible for the authentication process on top of any kind of VE2VE communication, in order to avoid possible VE impersonation.

Functionalities/Interfaces

VEs can publish their (public) data either through the COSMOS Message Bus (push approach) or through direct VE2VE communication (pull approach).

- **VE2COSMOS:** The VE Developer is responsible for filling in the Privelets configuration file, in order to tag the VEs data either public or private. The component filters the private data and therefore the JSON messages, that are published to the Message Bus, contain only the public fields alongside with the mandatory ones (please see section 9.2.1);
- **VE2VE:** When a VE receives an information request (REST GET or POST) from another VE, then Privelets component:
    - Authenticate the VE which sent the request;
    - Check whether the request is considered as repetitive according to the configuration file (if yes, then it is ignored);
    - Check whether the specific information is tagged as private (if yes, then it is not shared)

Similarly, when the response comes back, Privelets that runs at the VE user side, is used to authenticate the VE which gave the response.

8.1.3.7 Situation Awareness (SAw) FC

Description of the component

The large volume of data which is made available in an IoT environment does not necessarily mean applications can take effective decisions directly or make correct interpretations. COSMOS intends to support the transition from raw data to value added information by providing mechanisms which facilitate SAw at two distinct levels: the VE centric SA and the platform level SAw. In order to do so, the following three levels of SAw are provided: Level 1, acquisition; Level 2, understanding; Level 3, prediction of evolution. With respect to the difficult task of dealing with such a variety of data streams, several types of context information are envisioned: system, user context, temporal and environment context. In turn, different kinds of context are mapped to different scenarios or use cases.

Functionalities/Interfaces

In brief, the functionality of a SAw process consists in generating knowledge out of incoming data streams. In this sense, it is essential to provide means to data acquisition from many, distributed, heterogeneous data sources. Then, the understanding of this information will be assessed applying rules and condition checking, what involves a data stream analysis tool such a CEP engine, and even more advanced application-specific algorithms such as Machine Learning techniques. These mechanisms are suitable for fast data analysis based on temporal and structural relations between underlying events, and have been selected to deal with the following situations:

- **Notification/Anomaly Detection:** Evaluation of 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. For example, adjustment of the energy consumption based on correlations of heat and electricity consumption with environmental measurements;
- **Disaster prevention and Increased Safety**: 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.

8.1.3.8 Experience Sharing FC

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. 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.

Additionally we are exploring ways to provide a more adaptable behaviour of request handling by implementing ways in which the Experience Sharing FC can handle the monitoring of the incoming amount of requests and based on performance thresholds, determine on consulting Social Ranking criteria whether to process the request or not.

Also, in order to enhance the use of Experience Sharing, we are considering the proactive use of the component in conjunction with the Situation Awareness component. In any use-case, there is possible need for a more active use of the knowledge diffusion mechanism, which does not lay squarely on receiving extra VE input for being activated. Knowledge in this case is a detected state change which affects nearby VEs or which requires external VE actions to be corrected or simply must be shared for the continued uninterrupted operation of the Network of Things.

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

Finally for Year 3 we expect work on the field of the semantic description of topics, leading to the implementation of Model Developers uploading custom Models on the platform, which will be able to be shared as Cases, in the sense of informing remote VEs of their existence and functionality.

Functionalities/Interfaces

Experience Sharing component is related with the following functionalities, which are analytically described in the subchapter 6.4 of the deliverable 6.1.2:

- **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 Year 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;

Privelet modifications

The API of the Experience Sharing component will not be modified greatly to accommodate changes and enhancements made by other components such as Privelets.

One modification is the adding of new fields in the HTTP communication to accommodate for the inclusion of private keys between VE2VE communications participants. These keys are utilized by the VEs in order to authenticate and secure their transmissions;

Model Sharing

The possible future inclusion of Models as Shareable Experience will also necessitate a further development of the API, taking into consideration the semantic description of a Model Developer uploaded Model. Such a description will be used as input to the component along with target VE address/port combination as is the case with current Experience. After that the remote VE will receive the request for Experience and initiate similar steps as those of the Case sharing mechanism, which are currently in effect;

Proactive Sharing

Such an enhancement, will necessitate the further modification of the initiator method, in that it might be possible to differentiate between VE2VE and VE2MB connection. Again the structure of events though which is going to emulate that of Cases, because of the CBR versatility, will keep the main forms of HTTP communication consistent with existing implementations. Possible publishing of events on the MB will be a new addition which will require description of the topic publishing to, or creating.

8.1.3.9 VE-data Pre-processing FC

Description of component

Pre-processing is an important step in IoT for many reasons. The amount of data is increasing exponentially in IoT and the processing of such large data with minimum time latency is an important factor which can be optimized by the use of proper pre-processing methods. Several aggregation techniques are commonly used in IoT for reducing the total amount of data traveling through the network. In this context, *Piecewise Aggregation Approximation* (PAA) and *Symbolic Aggregation Approximation* (SAX) are the most common techniques.

Most of the devices in IoT are connected with wireless links in a dynamic environment and resource constraint nature of these devices affects the communication link and their performance. The deployment of cheap and less reliable devices is common in IoT to bring the overall cost of a system down resulting in missing values, out of range values or impossible data contributions. The phrase "*garbage in garbage ou*t" fits perfectly for many machine learning algorithms.

Functionalities

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

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

Interfaces

Following interfaces for the pre-processing component running at different levels are provided.

- **VE Level using CEP:** An interface will be provided for application developer to define the pre-processing using Dolce language for the CEP. The output will be published to under the specific topic on message bus;
- **Object Storage using Storlets:** An application developer can code their own storlets in java or can use generic storlets provided by COSMOS for aggregation. Aggregation storlet will take the aggregation factor and the input features as input;
- **Platform level using Apache Spark:** Different pre-processing methods are provided which can be specified when choosing a particular data mining method.

8.1.4. IoT Service FG

8.1.4.1 IoT Service Resolution FC

Description of the component

As already mentioned, VEs are built on top of IoT Services but also extend their functionality through not-IoT services. One of the advantages of this approach is that a VE can incorporate different IoT Services even if these services have different providers.

One of the problems facing the developers of IoT applications is the lack of a standard interface description adopted by the majority of the IoT Service developers. As a result we can encounter IoT services exposed as REST services (which also come in different flavours), some are exposed through different implementations of a publish subscribe mechanism and even through proprietary protocols.

Since COSMOS emphasises the use of semantic descriptors for the VEs and their capabilities and later for their retrieval, the problem mentioned above translated also into the description of the IoT Services used by the VEs as well as their retrieval.

The COSMOS ontology provides the model for describing COSMOS “flavoured” IoT Services exposed either as REST services or through a message bus. This endpoint description model is based on a set of COSMOS defined conventions regarding these interfaces (e.g. parameter transfer modes, data types, data encapsulation, schemas, etc.). These conventions are meant to guarantee interoperability as well as the proper description of the interfaces and the service resolution.

As described in the section 9.1.1, which is dedicated to the COSMOS Ontology, the endpoints are described from both the data type perspective as well as from the semantic one.

While adhering to a project defined interface description convention does guarantee service and component interoperability, supporting only these COSMOS “flavoured” IoT services would limit the scope of the COSMOS platform. This would have been against one of its major goals: integration of different IoT Services under the same VE.

In order to address these conflicting constraints, the semantic model has been build to support both the description of IoT Services adhering to the proposed conventions but also to provide references to external descriptors. This means that whenever a service which does not adhere to the COSMOS interface conventions is integrated into a VE, the VE developer can reference the interface descriptor (if available) even if this is not stored into the COSMOS registry. This is one of the advantages of adopting the linked-data paradigm for the VE description.

The IoT service resolution component in based on the functionality of the VE registry as well that of the Semantic Topic Management and provides IoT service resolution. It provides a query interface allowing service retrieval based on the user defined criteria.

Functionalities/Interfaces

The description of the IoT Services following the COSMOS interface conventions is performed through the VE Registry, since it provides the API as well as a front-end for endpoint descriptions. As expected, external IoT service descriptions are expected to be provided by their developers and made publicly accessible once the IoT service is deployed.

The semantic model provided through the COSMOS ontology allows references to external registries (if required) so that the VE developer knows where the IoT service descriptions are published and has the ability to use that registry.

The IoT service resolution component exposes an API to allow the retrieval of COSMOS "flavoured" IoT services whose descriptions are stored into the VE Registry.

As for the external registries providing SPARQL endpoints, they can be queried as well if the VE developer knows the description format of those services. If not, the developer is at least forwarded to the external registry application which typically exposes its own REST endpoint for querying or an user interface.

The above mentioned interactions are mainly applicable during the VE development and involve the VE developer thus can be considered static.

Nevertheless in some applications the binding of the VEs to IoT services is dynamic and can change during the operation of the VE. For instance, IoT Services which where bind during the static phase could be replaced by others during its operation. This replacement can only be done if the replacing service is compatible with the existing one. This compatibility must apply to both the data types as well as the semantic types of the endpoints. The resolution of the services adhering to the COSMOS conventions is provided through the VE registry. This provides the mechanisms to retrieve similar services based on the provided criteria and the interface description of the service to be replaced.

The replacement of the externally described services can be achieved dynamically only if the registries which describe them provide this functionality. It falls into the responsibility of the VE developer to understand and use these querying interfaces if provided.

The IoT service resolution component exposes a REST interface to facilitate the retrieval of services meeting the provided search criteria. This has the advantage that the complexity of the SPARQL querying is hidden to the user.

8.1.4.2 Data Mapper FC

Description of the component

The Data Mapper FC is a component which subscribes to the topics which are flagged as persistent in the Message Bus FC, reads periodically data published from the VEs, aggregates and transforms them into a format suitable for persistent storage in the cloud, annotating them with enriching metadata.

Metadata like Id, timestamps, geo-location etc. are extracted from the raw data whereas the social ones are calculated from the Social Analysis component.

Functionalities/Interfaces

Message Bus topics can be marked as persistent in order to denote that their data be stored persistently in the Cloud Storage. The Data Mapper component needs to be notified of all topics marked as persistent in order for it to subscribe to all such topics.

The Data Mapper receives data from the Message Bus in Json format, supported by COSMOS. It writes the data to the Cloud Storage component using the OpenStack Swift REST API. Swift is used to create the data objects with their associated metadata, get the object details, get its metadata and update them.

Social indexes are extracted from the Social Analysis component through GET or POST REST requests.

8.1.4.3 Cloud Storage FC and Metadata Search FC

Description of the component

The purpose of the COSMOS Cloud Storage component is to persistently store COSMOS data and make it available for search and analysis. The open source OpenStack Swift object storage software is used in order to implement the COSMOS Data Store. Data is organized into containers and stored as objects. In Year 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 extended Swift to index metadata and to support searching for objects (containers, accounts) according to their metadata keys and values.

Functionalities/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.

8.1.4.4 Storlets FC (Data Analysis close to the storage)

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 is to perform the computation near the storage thereby reducing the network bandwidth.

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.

Functionalities/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 supported invoking a storlet as part of a Swift GET request using the Swift REST API. Storlet parameters are provided using certain headers. In Year 2 there are additional mechanisms for invoking storlets.

8.1.4.5 Inference and Prediction FC

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 are deployed in remote places and connected using less reliable wireless links. In order to prolong their battery life, data provided by these devices may be sporadic and less reliable. Data itself is of no value until it is processed intelligently to extract high-level knowledge which can be used to make decisions. Data mining methods based on machine learning and statistical analysis techniques have the potential to extract knowledge from unreliable and incomplete data. In this regard, we have explored several machine learning and statistical methods for providing functionalities which application developer can use to get more insight and value from the data.

Functionalities

This component is responsible for providing high-level knowledge from raw IoT data using different pattern recognition techniques. In this context, we have explored several supervised machine learning techniques including different variants of Support Vector Machines and K-Nearest Neighbour in addition to statistical techniques such as *Hidden Markov Model* (HMM) which were explained earlier. In short, it provides following two main functionalities.

1) If labelled historical data is available (raw data with labelled high-level knowledge), it provides the functionality to train the model and provides capability to deploy the model in order to predict the output for real-time data;

2) If the labelled historical data is not available (incomplete data), it exploits the temporal patterns of the data and learns using statistical properties of the data to train the model which can be used to predict the output for real-time data.

Interfaces

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

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

Application Developer-Inference/prediction Block API:

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

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

Client/VE-Inference/Prediction Block API:

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

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

Storage-Modeling Block Interface:

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

8.1.5. Security FG

8.1.5.1 Security Management FC

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.

Functionalities/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[21] 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.

8.1.5.2 Hardware Board FC

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 H/W 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.

Functionalities/Interfaces

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

- Ethernet
- WiFi
- ZigBee
- I^{2}C
- 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 [21] key exchange algorithm. With the help of SSH or HTTPS as transport layer and a RESTful interface, the HW Security Board is configured.

8.1.5.3 Authentication FC

Description of the component

The Authentication FC is responsible for authentication services in COSMOS. By default the Authentication FC confirms the identity of a third party by using one of the following factors of authentication:

- The user himself – in case of a human users the authentication can be checked by means of physically unique properties such as fingerprints or retinal scans;
- The user’s knowledge – in case of a human user, the Authentication FC confirms the identity of the user by checking a secret only known between the users and COSMOS (e.g. a password or a PIN);
- The user’s “belongings” – in case of both human and non-human users the authentication can be checked by means of a physical “belonging” such as an ID card, token or security key.

For COSMOS we consider:

- In case of human users – knowledge based authentication (i.e. user name/password pairs).
- In case of VEs and H/W Board – the user’s belongings (i.e. encryption keys/ssh).

The Authentication FC is triggered by all incoming or outgoing communication within the COSMOS platform. In case of human users the authentication takes place at the beginning of the communication session (i.e. when the user loges in COSMOS) and in case of VEs (e.g. "machines") for every data package (please see use-cases).

The Authentication FC checks:

- The nonce of the message;
- The authenticity of the message/users using the ACL.

The Authentication FC is interlinked with the Authorization FC – they both act as security enforcers. The Authentication FC is notified by the communication FC of all external data traffic within COSMOS.

Functionalities/Interfaces

A high level API provides access to the Authentication component which responds to `authentication_request` with `authentication_ack` or `authentication_nak`. Internal operations such as nonce check-ups are not accessible over the API for security reasons.

8.1.5.4 Authorisation FC

Description of the component

This FC provides authorization primitives, that is it permits the generation of new permissions/access rights associated to encryption keys (e.g. security tokens) which are based on existing/predefined security policies existing in COSMOS or on new ones, generated by the key owner. Permissions only affect the owner's data! Obsolete or invalid keys, marked as such by the Key Exchange and Management FC have the permissions removed therefore revoking their access to the platform.

This FC can be used on two occasions:

- **New VE enrolment:** the owner triggers the enrolment process and wishes to authorize the new VE by associating new permissions or deriving/editing permissions from another VE;
- **VE authorization during runtime:** the VE wishes to communicate to COSMOS which triggers the authorization process, that is the permissions of the accessing VE are checked and thus the VE is allowed (or not) to communicate to the platform.

Functionalities/Interfaces

The Authorization FC API can be used for a`uth_request`, `auth_grant`, `auth_denied`, `auth_limited` operations on incoming communication. For temporary operations the Authorization FC supports token generation and distribution (via the Key Exchange and Management FC).

8.1.5.5 Key Exchange and Management FC

Description of the component

The KEM FC has the role of generating, distributing and maintaining the security keys and associations within COSMOS.

Key generation

Based on the enrolment process, the FC will generate a new security key (e.g. AES128) which will be associated with a new VE. The key will be generated on-demand and if not used within a certain period of time will be marked as invalid and stored accordingly.

Key distribution

After the key has been generated it will be distributed using a key exchange protocol such as Diffie-Hellman [21]. This key agreement mechanism assures that the exchanged key is only known to the 2 communication partners. The key exchange is confirmed by executing a series of predefined commands which ensure that the target device has received the desired key.

Key management

After the key has been generated and successfully distributed it is enrolled into COSMOS. This process consists of associating, using the Authorization FC, of permissions to a key and of managing the key – function overtaken by this component. The key management is responsible for the life-cycle of the key – changes to permissions go through this component and are reflected accordingly into the ACL. Once the key is marked as obsolete or deprecated, the associated rights are removed and the key is stored for security and history/tracking reasons.

Functionalities/Interfaces

The key distribution API is based on the standard Linux driver model (i.e. `ioctl()`) where the H/W operations are mapped into virtual files. The API supports key generation - start, stop, reset and status (polling mode) operations executed.

The key management REST API can be used to enroll, mark as live or obsolete/removed, publish and revoke operation performed on the previously generated keys.

8.1.5.6 Cryptographic Non-repudiation FC

Description of component

This FC is responsible for the cryptographic primitives within COSMOS. In case of the H/W Board it relies on the hardware cryptographic primitives which offer a higher security level.

This FC is triggered for both incoming and outgoing data traffic within COSMOS.

This FC provides 2 functionalities which depend on each other:

- *Cryptography*: based on the authentication and authorization processes, the Cryptographic Non-repudiation FC selects the unique key and decrypts the data (using the H/W modules in case of the high secure H/W Board or using pure software implementations in case of S/W implementations). If the decryption process takes place successfully the message is pushed within COSMOS, otherwise the message is discarded and the reputation index is decreased;
- *Non-repudiation:* in case of incoming communication the sending partner will receive an acknowledgement message. In case of outgoing communication the Communication FC will expect an acknowledgement message. Non-repudiation is assured by using a combination enforced security policies (e.g. using strong security and ACLs), strong security (e.g. encryption keys) and acknowledge messages.

Functionalities/Interfaces

The cryptographic accelerators are exposed into the Linux OS using the standard driver model thus are accessible as virtual files using the standard I/O API of Linux (i.e. `ioctl()`). The File

I/O API can be used to encrypt, decrypt, read the output and the status of the cryptographic accelerators and to perform emergency reset of it.

8.1.5.7 Checksum FC

Description of component

This FC is directly connected to the Communication FC – all data traffic going into COSMOS reaches this FC. It's basic functionality is:

- To check data integrity – based on typical checksum computations the Checksum FC re-computes checksums and checks them against original ones;
- To correct bit-wise errors – based on the above computations it identifies and corrects bit-wise errors.

This FC is used in the first place to check the correctness of received data traffic and to report any findings, that is repeated errors are a good indicator that a communication partner is experiencing problems.

Functionalities/Interfaces

The API is based on the standard Linux driver model (i.e. `ioctl()`) which is used also for the cryptographic accelerators. The API offers direct access to the H/W modules with support for operations like start, stop, reset, ready (polling mode), load and read.

9. COSMOS Information View

Based on the Information Model (part of the IoT Reference Model) the Information View aims at providing details about how the information is actually coded, serialised and handled within the target IoT system. Indeed the IM does not give any indication on how objects, resources, devices and associated attributes and description must be encoded. It stays at an upper level, giving only indications concerning what needs to be modelled and which inter-concepts associations need to be implemented within the IoT system. Implementation matters stay at architect side, who in turn enjoy some freedom as far as her choices remain compliant to the IM constraints. The Information Model was briefly introduced in Section 5. This section now adopts several Viewpoints according to the Rozanski & Woods [6] terminology and elucidate several aspects pertaining to information and information flows within the COSMOS architecture.

9.1. Ontologies

COSMOS emphasizes the reuse of data, knowledge and experience among VEs and this is supported through its components. Not only the interoperability between all of this components had to be considered but also the fact that different actors are involved in the development and exploitation of COSMOS enabled applications. These actors include the VE developers, the COSMOS enabled application developers, COSMOS extensions developers or the platform owners and require a common “language” in order to work properly together.

To facilitate this, in addition to the data interoperability considerations (mainly the interface compatibility), also the semantic dimension of the data exchanged between various components has been addressed. Using semantic technologies, a new level of description has been introduced.

The main semantic model is provided through the core COSMOS ontology or *COSMOS ontology* and is supplemented through the *social-related ontology*. While the former provides the model for describing the complete chain from physical entities, to resources, IoT services, VEs and final endpoints, the latter addresses the social dimension of the VEs. In addition to this, domain dependent ontologies can be integrated into the description of the VEs, topic and other COSMOS related entities. depicts a block diagram of the VE registry whit its main components. Figure 10 presents a block diagram of the information model[4], including the key concepts supported by the COSMOS ontologies.

[4] Information Model is taken here in the classical sense, not the ARM one.

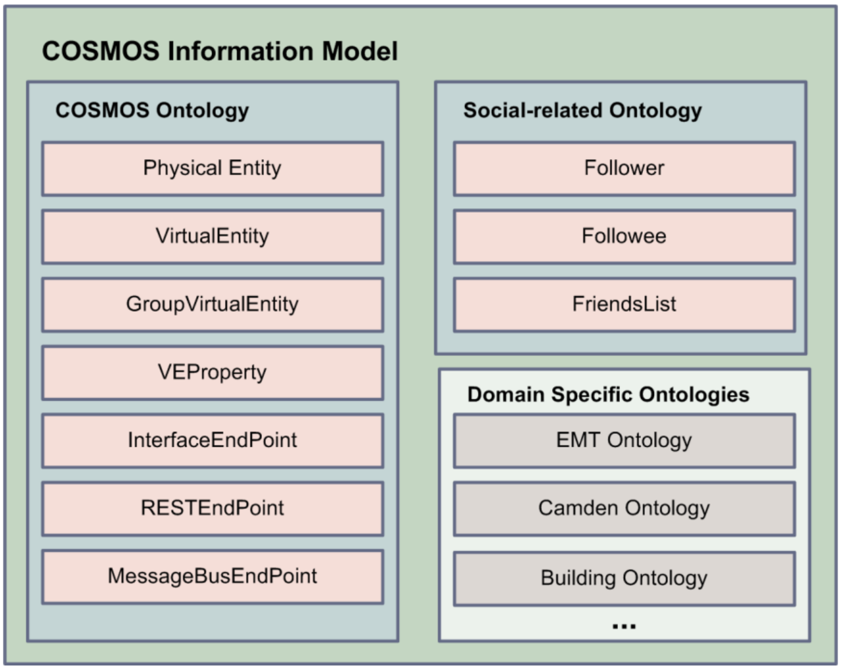


Figure 10: COSMOS Information Model

The general upper COSMOS ontology will be briefly described in the following section. However the Domain Specific (one per scenario) and Social Ontologies are still work in progress and will be fully detailed in the final iteration of the Architecture deliverable. A more comprehensive description of the COSMOS ontology can be found in deliverable D5.1.2.

9.1.1. COSMOS Ontology

COSMOS Ontology is the core ontology of the project and is meant to address the description of the relevant COSMOS entities from a domain independent perspective. This domain independence provides the ability to address various applications without the need to redesign or extend the ontology whenever a new scenario is considered.

The ontology is centred around the description of VEs and the main goal is to provide adequate support for VE retrieval when COSMOS enabled applications are built. COSMOS applications can use VEs developed and deployed by different developers. Moreover, depending on the scenario, the binding to a specific VE can be dynamic (it is not hardcoded or preconfigured into the application) thus requiring an adequate manual or automated VE retrieval mechanism.

A simple example would be that of a COSMOS application which requires localized indoor temperature readings. Based on its current location the application should be able to retrieve the appropriate VE capable of providing temperature readings in ℃. As we can see, the search criteria are not confined to a double value which would ensure the data compatibility between the producer and the consumer of that data. They are enriched with the semantics of that double value (temperature measurement, location, unit of measurement etc.).

Based on the linked data paradigm, the description of the VE is supported through the domain independent COSMOS ontology which allows references to concepts described through domain specific ontologies.

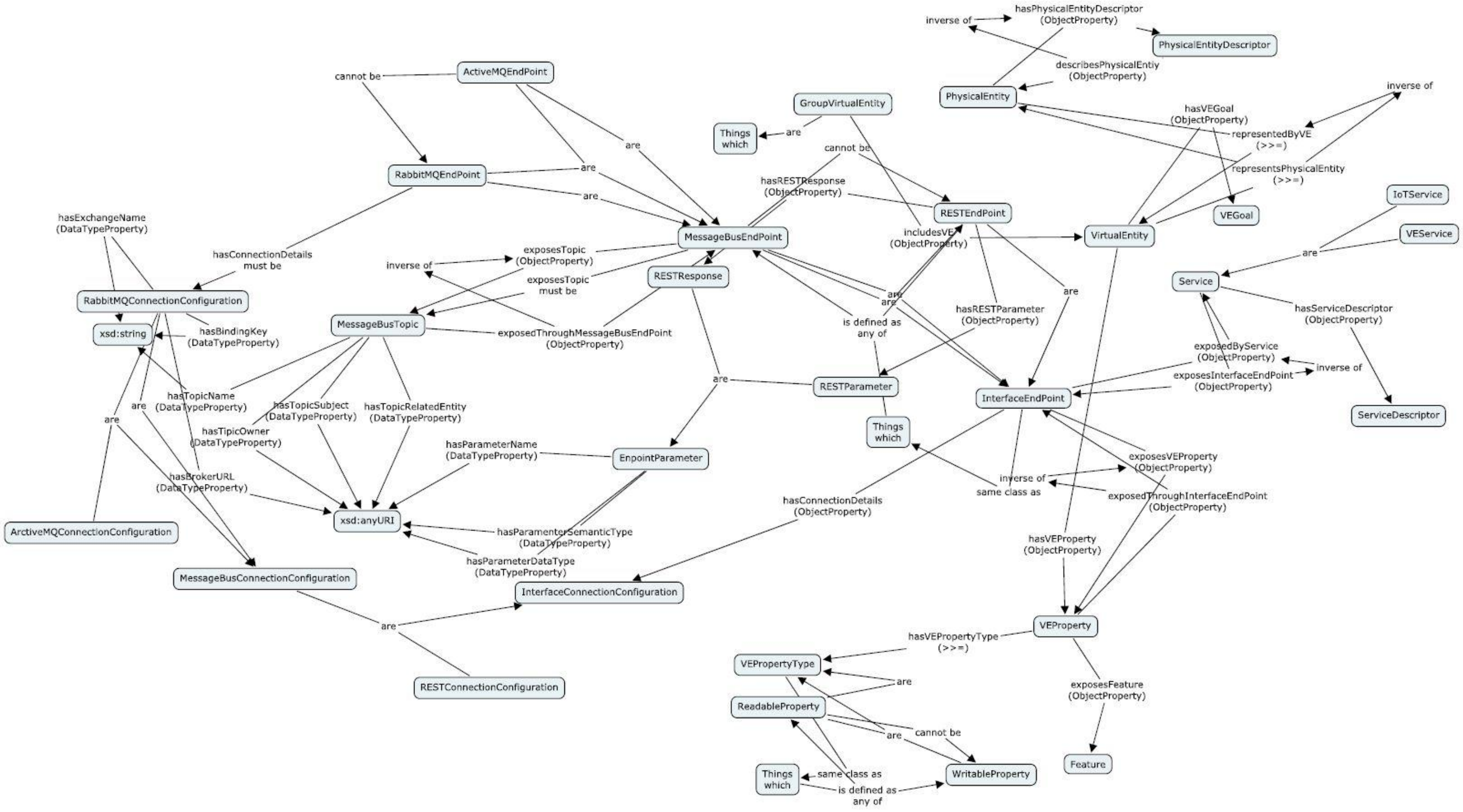

hasPhysicalEntityDescriptor (ObjectProperty)
inverse of
PhysicalEntityDescriptor
describesPhysicalEntiy (ObjectProperty)
PhysicalEntity
inverse of
hasVEGoal (ObjectProperty)
representedByVE (>>=)
representsPhysicalEntity (>>=)
VEGoal
IoTService
VEService
GroupVirtualEntity
Things which
are
cannot be
hasRESTResponse (ObjectProperty)
RESTEndPoint
includesVE (ObjectProperty)
VirtualEntity
ActiveMQEndPoint
cannot be
are
RabbitMQEndPoint
are
are
MessageBusEndPoint
RESTResponse
hasExchangeName (DataTypeProperty)
hasConnectionDetails must be
RabbitMQConnectionConfiguration
xsd:string
hasBindingKey (DataTypeProperty)
exposesTopic (ObjectProperty)
inverse of
exposesTopic must be
MessageBusTopic
exposedThroughMessageBusEndPoint (ObjectProperty)
are
are
is defined as any of
hasRESTParameter (ObjectProperty)
are
Service
hasServiceDescriptor (ObjectProperty)
exposedByService (ObjectProperty)
exposesInterfaceEndPoint (ObjectProperty)
inverse of
ServiceDescriptor
hasTopicName (DataTypeProperty)
are
are
hasTopicSubject (DataTypeProperty)
hasTopicRelatedEntity (DataTypeProperty)
hasTipicOwner (DataTypeProperty)
are
RESTParameter
Things which
InterfaceEndPoint
hasParameterName (DataTypeProperty)
EnpointParameter
exposesVEProperty (ObjectProperty)
inverse of
same class as
exposedThroughInterfaceEndPoint (ObjectProperty)
hasBrokerURL (DataTypeProperty)
xsd:anyURI
hasConnectionDetails (ObjectProperty)
ArctiveMQConnectionConfiguration
hasParamenterSemanticType (DataTypeProperty)
hasParameterDataType (DataTypeProperty)
hasVEProperty (ObjectProperty)
MessageBusConnectionConfiguration
InterfaceConnectionConfiguration
are
VEProperty
hasVEPropertyType (>>=)
VEPropertyType
are
RESTConnectionConfiguration
ReadableProperty
exposesFeature (ObjectProperty)
are
cannot be
Things which
same class as
is defined as any of
WritableProperty
Feature

Figure 11: COSMOS (Core) Ontology

By using the ontology we can describe, as mentioned above, the entire chain starting from the physical entity to the endpoints which provide access to the underlying data. Figure 11 above depicts the COSMOS core ontology with all the concepts and relations connecting them.

VEs are representing physical entities whose properties can be sensed and upon which actuation can take place. These actions are taken using devices accessible through IoT Services.

VEs not only encapsulate IoT Services and the description of the underlying physical entities and resources, but also include additional functionality. These additions extend the capabilities of the IoT services and also provide support for the social dimension of the VEs.

In order to provide unified accessibility and descriptiveness, VEs expose both the IoT as well as non-IoT related functionality through *VE properties*. VE properties act a binding element between the above mentioned functionality and the endpoints. This binding is insensitive to the fact that a property is resources or non-resource related.

The endpoints are the final element in this chain since they provide the means to access these properties using application level interfaces, such as REST endpoints or Message Bus topics.

The introduction of the VE properties and their rich descriptive attributes facilitate the exact retrieval of VE functionality using a dedicated and easy to use query interface. By using the semantic description approach, COSMOS exposes the benefits of the linked data paradigm and allows not only the use of project specific components but the integration of external services, as long as they are also semantically described and their description is accessible for public querying.

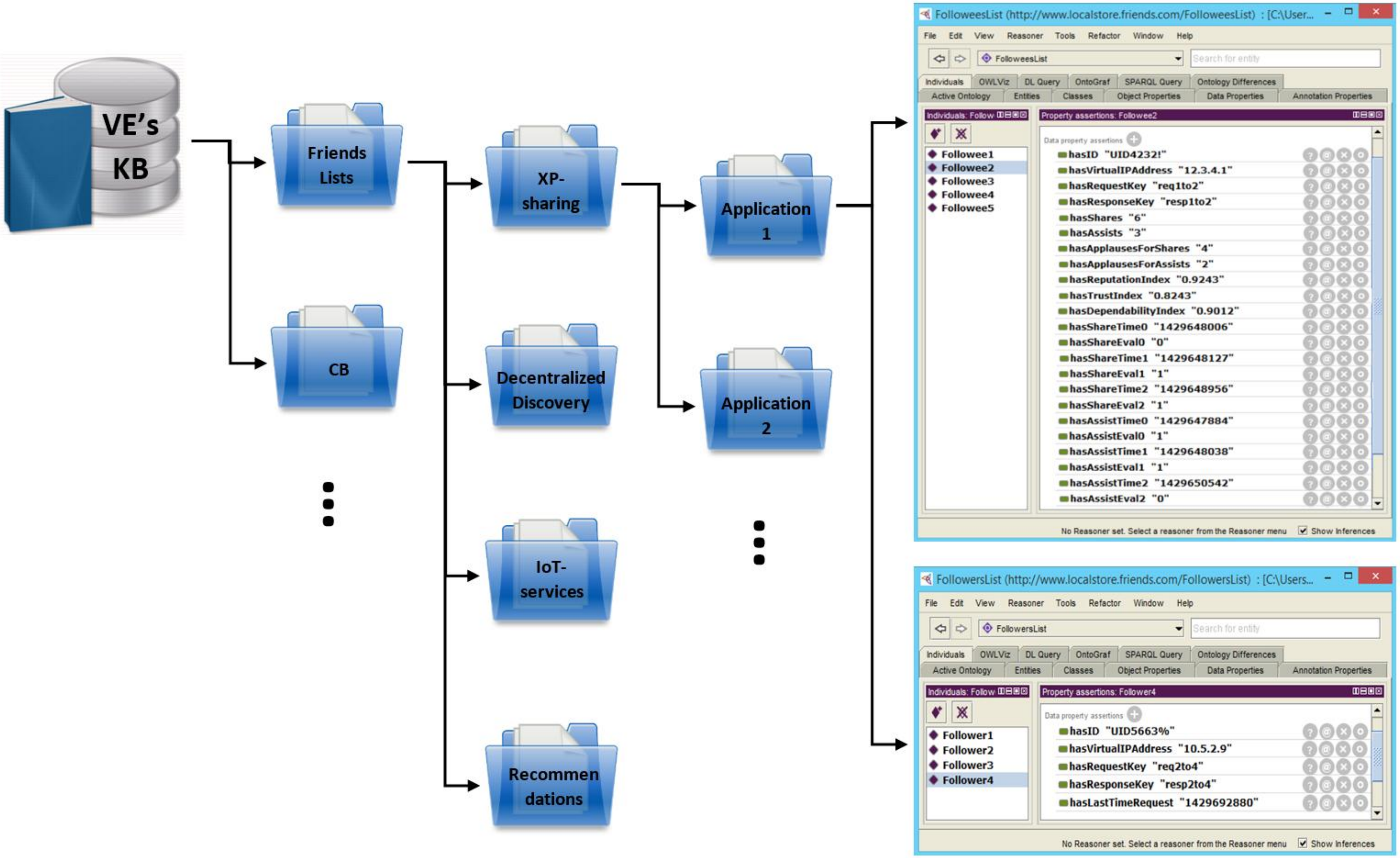
Besides the description of the VE semantics, which are needed for effective retrieval, the endpoints as well are described, so that interface compatibility can be checked prior to the use of a VE. As a result, the ontology includes the concepts and the properties which allow the user to describe endpoints, be it a REST endpoint or a Message Bus topic. The interface type, required parameters, expected schemas, URLs are included into the description.

The COSMOS core ontology is extended with the social-related ontology which is described in section 9.1.2. Still, but both are domain independent, therefore do not include any application specific descriptions. To facilitate application development and benefit from the linked data paradigm, VE attributes can be linked to domain specific ontologies, especially when it comes to the description of the physical entities and of the resources. Section 9.1.3 provides a description of such a sample ontology which is derived from the Madrid use-case.

9.1.2. Social Ontology

The integration of social networking concepts into IoT systems is a burgeoning topic of research that promises to support novel and more powerful applications. In the COSMOS project we introduce a social approach in order to achieve enhanced services (like *discovery*, *recommendation* and *sharing* between Things enriched with *social properties*) and we investigate how typical notions and modes of interactions of social networking can be extended to the networks of Things, providing a *Social Internet of Things* (SIoT) platform. As such, the creation of a *Social Ontology* for defining concepts and terms used when dealing with Trust & Reputation and social relationships between VEs becomes necessary. To this direction, we define three types of descriptors:

- types (classes) of relations between VEs;
- social indexes;
- social (interaction) metrics.


VE's KB
Friends Lists
CB
XP-sharing
Decentralized Discovery
IoT-services
Recommendations
Application 1
Application 2
File Edit View Reasoner Tools Refactor Window Help
Search for entity
Individuals OWLViz DL Query OntoGraf SPARQL Query Ontology Differences
Active Ontology Entities Classes Object Properties Data Properties Annotation Properties
Individuals: Follow
Data property assertions
No Reasoner set. Select a reasoner from the Reasoner menu
Show Inferences

FolloweesList (http://www.localstore.friends.com/FolloweesList) : [C:\User...

FolloweesList

Followee1
Followee2
Followee3
Followee4
Followee5

Property assertions: Followee2

hasID "UID4232!"
hasVirtualIPAddress "12.3.4.1"
hasRequestKey "req1to2"
hasResponseKey "resp1to2"
hasShares "6"
hasAssists "3"
hasApplausesForShares "4"
hasApplausesForAssists "2"
hasReputationIndex "0.9243"
hasTrustIndex "0.8243"
hasDependabilityIndex "0.9012"
hasShareTime0 "1429648006"
hasShareEval0 "0"
hasShareTime1 "1429648127"
hasShareEval1 "1"
hasShareTime2 "1429648956"
hasShareEval2 "1"
hasAssistTime0 "1429647884"
hasAssistEval0 "1"
hasAssistTime1 "1429648038"
hasAssistEval1 "1"
hasAssistTime2 "1429650542"
hasAssistEval2 "0"

FollowersList (http://www.localstore.friends.com/FollowersList) : [C:\Users...

FollowersList

Follower1
Follower2
Follower3
Follower4

Property assertions: Follower4

hasID "UID5663%"
hasVirtualIPAddress "10.5.2.9"
hasRequestKey "req2to4"
hasResponseKey "resp2to4"
hasLastTimeRequest "1429692880"

Figure 12: Example of a Followees List and a Followers List of a VE for XP-sharing

Inspired from the social media domain, we define and monitor the following basic friendship notions (an example of which is shown in Figure 12 above:

- **Followees:** The VEs that are being tracked by a specific VE. Followees Lists define the receivers of the VEs requests for services;
- **Followers:** The VEs that track a specific VE. They are held in the Followers List and indicate the credibility and reputation of a VE.

Regarding the further classification of the Followers and Followees, it should be noted that, for each type of service request (XP-sharing, decentralized discovery, IoT-services, recommendations) different lists are created, as some Followees are more useful e.g. for XP-sharing while others are more useful (and trustworthy) e.g. for recommending the services of other VEs. These classes of lists consist of subclasses, further specifying the relationships of VEs. For example, a specific Followers List may be used specifically for the sharing (XP-sharing) of Cases (and not of any other kind of XP) regarding a very specific Application (e.g. Application 1) which defines the structure of the Cases shared.

Apart from the Follower/Followee relationship, we have defined and studied many other kinds of VE relationships (Relational Models) like “Conflict of Interest Relationship”, “Replacement Relationship” etc. Actually, the number of classes of relationships that can be defined is bounded only by our imagination and the specific use cases we study. In other words, our model is extensible, meaning that new kinds of relationships can be defined and used, exploiting the several services that COSMOS provides.

The relation between a VE and its Followees is trust-based and non-mutual. In order to face security threats that could undermine the social network of VEs, we introduce the *COSMOS Trust and Reputation model* which defines some core indexes for the social characterization of VEs. These social indexes are *Popularity*, *Trust* and *Reputation* and are used from VEs in order to anticipate whether the services that other VEs provide can be trusted or not. Trust and Reputation management is a very useful and powerful tool in environments where a lack of previous knowledge about the system can lead participants to undesired situations, specifically in virtual communities where users do not know each other at all or, at least, do not know everyone. It is in those cases where the application of trust and reputation mechanisms is more effective, helping a peer to find out which is the most trustworthy or reputable participant to have an interaction with, preventing thus the selection of a fraudulent or malicious one.

Finally, it should be noted that, in order to extract the social indexes of other VEs, VEs have to keep a track of their interactions and the results of these interactions with other VEs. Thus, depending on the type of relationship, we define different interaction metrics (e.g. *Shares*, *Assists*, *Applauses*) monitored by the Social Monitoring component of a VE and stored locally in its Friends Lists. These metrics are calculated in a distributed manner by the VEs on a per-VE basis and are the main input for the services provided by other components. For each metric identified we develop/choose the corresponding KPIs and tools that should be imported into the VE during the phase of registration.

More details regarding the social notions we have introduced (and the rationale behind introducing them) can be found in D5.1.2 where the social model of COSMOS is presented.

9.1.3. Domain specific ontologies

Besides the use of the COSMOS core ontologies, VE description can be augmented through the use of domain specific ontologies and the use of the linked data paradigm. VE referenced physical entities can be described using such domain specific ontologies thus allowing a rich descriptiveness and extended support for the application developers.

The EMT, HILDEBRAND and III ontologies are such a domain specific ontology and will be fully developed in the final iteration of this deliverable, fully connected to the COSMOS core ontology.

Specifically for the EMT Domain Ontology, work has already been accomplished in translating characteristics of Physical Entities which can provide a richer semantic description. Work focuses not only on describing the value oriented data properties of Physical Entities but also their actual links and connections in the form of ownership relationships, subordination or other interactions. By using specific schemas developed by EMT, we can store “Knowledge” not pertaining directly to the implementation of these Entities as VEs but also their actual physical limitations and characteristics.

Also, in following with the linked data paradigm we strive to provide clear identification of individuals and their relationships using URIs and by defining a structured vocabulary to make use of SPARQL querying and RDF descriptive capabilities. These relationships and extra VE descriptions of the Entities will be available through connections with the main COSMOS ontology.

For instance, connections include static or semi-dynamic memberships of buses on lines as well as connections of buses with depots. Additionally ownership of buses in the context of an eventual multi-company approach may be investigated. Characteristics stored include seating capacities, types of fuel consumed etc.

The EMT, HILDEBRAND and III ontologies will be fully developed in the final iteration of this deliverable, fully connected to the COSMOS core ontology.

9.2. Data Structures

9.2.1. Message Bus data structure

We adopt the following conventions for the Message Bus data format.

1. VEs can send messages to topics on the Message Bus. Each message should be in Json format.
2. Each message MUST have the following fields (at top level):
    a. *C*OSMOS_VE_ID (Virtual Entity Id) – the associated value should have type string. Note that the VE IDs will need to be consistent with those used in the Registry component.
    b. COSMOS_TIMESTAMP – the associated value should have a date/time format[5] as described here
        i. Note that time zones are supported.

[5] Date/time format is available at the following URL: http://joda-time.sourceforge.net/api-release/org/joda/time/format/ISODateTimeFormat.html#dateOptionalTimeParser() (last accessed 30/4/2015)

ii. Precision of milliseconds is supported

c. `COSMOS_DATA` – the associated data, which should contain a nested JSON value.

i. The Cloud Storage can optionally store a schema for this data on a per topic basis. This schema should be specified using the Avro format [20]. A reference to the schema should be associated with the Message Bus topic by the Registry component.

ii. The data can contain more than one field.

3. Each message can OPTIONALLY have the following fields

a. `COSMOS_LOCATION` – the location in `(lat,lon)` format, for example `"41.12,-71.34"`

9.2.2. Object data structure

We assume the following (see Deliverable 4.1.2 for more details)

1. Each COSMOS application is mapped to a Swift account.
2. The data from a VE in a COSMOS application is typically published to a particular Message Bus topic. If this topic is specified as persistent by the Registry component then it is mapped to a Swift container under the corresponding account;
3. VEs periodically publish ‘messages’ to the Message Bus in Json format. Multiple such messages are collected by the Data Mapper and published as a single Swift object which contains multiple ‘records’;
4. In addition objects can be annotated with metadata such as the start and end timestamps for a Swift object;
5. Objects may be stored in their original Json format or they may be transformed into another format such as Parquet. See deliverable 4.1.2 for more details;
6. As written in the previous section the Cloud Storage can optionally store a schema for Message Bus data on a per topic basis. This schema should be specified using the Avro format.

9.3. System Use-Cases

The system use-cases described in this section follows another view point used to describe the Information View. They show information flows and interaction steps between the components with a low level of detail. Another view point that consists of using Sequence charts would provide an higher level of detail; however they will be described in the WP deliverables dealing with those components. Elaborating on System Use-cases is an essential step and group activity of the Architecting process as it allows to understand better the interactions existing between the components (and to share this understanding between all parties involved) and detect potential issues or inconsistencies. This sub-section follows the ARM Functional Group structure already introduced in the previous section.

Please note*: The following list of system use-cases is not exhaustive (and hardly can be by nature). It shows typical usages (patterns) which can be made of the Functional Components* *described in the previous section. More examples will be added in the latest iteration of this Deliverable during Year 3.*

9.3.1. Management FG System Use-cases

This section provides typical use-cases pertaining to the Management FG, like secure account creation and setting up of access rights.

9.3.1.1 Registering a new Account (COSMOS User)

Each user/owner of VEs needs to have a COSMOS account – an all-in-one stop where he or she can manage all COSMOS related activities. The COSMOS account is the umbrella for personal settings as well as the gateway users are offered for interaction with the platform.

New users are registered in COSMOS while going through a process of registration. By this the new users must create a user name/password pair on the account creation page. Via an email address the user receives a confirmation link which, when clicked, activates the user's account while generating a user unique encryption key – a key unique the users and his present account. If a user has more than one account he or she will have a unique key for each one of them.

Users are not allowed to self-generate a new user key – if they suspect an attack or loss of key has happened then they must contact the system administrator to get a new key generated.

Users leaving COSMOS are not allowed to erase their keys, instead the keys are marked as obsolete and their access rights are removed.

9.3.1.2 Setting-up Access right

After setting up the account the user must set up his access-rights – what are others allowed to see and do with the new user's data. By default users have R/W rights over their own data and unless otherwise specified R rights over publicly available data within COSMOS. Users can:

- Share their data with R or R/W rights to other users or a group of users;
- Restrict access to their data (R rights or no rights at all);
- Restrict access to parts of their data (e.g. R rights over VE_A and no rights over VE_B);
- Set up access rights of their VEs to COSMOS.

Each VE a user own has its own access rights. Users can derive rights of an existing VE and use them on a new VE without affecting the original one.

9.3.2. Service Organisation FG

9.3.2.1 VE Trust and Reputation ranking

On the subject of the calculation of the Dependability Index, the developed solution is a platform specific service that is initiated by the SA component and entails the querying of Followers of a specific VE. The first action of the SA is to acquire the Followers List of the Evaluated VE. The SA extracts the group of Followers of the Evaluated VE and then randomly decides which ones and how many of them to use as a querying basis (if their number is too large). This element of randomness is essential in the development of the mechanism, as it can prevent collusions which may alter the final result of the evaluation process. After this step, the SA requests the stored Applauses, Mentions Assists and Shares for the Evaluated VE from the Followees List of each Follower of the VE. After receiving the requested metrics, the SA component calculates both the Trust and the Reputation Indexes which, combined with certain weights defined by the user, result to the Dependability Index. It should be noted that for the calculation of the Dependability, the Evaluated VE itself does not provide any information at all, but instead, all the information needed is offered by its social environment.

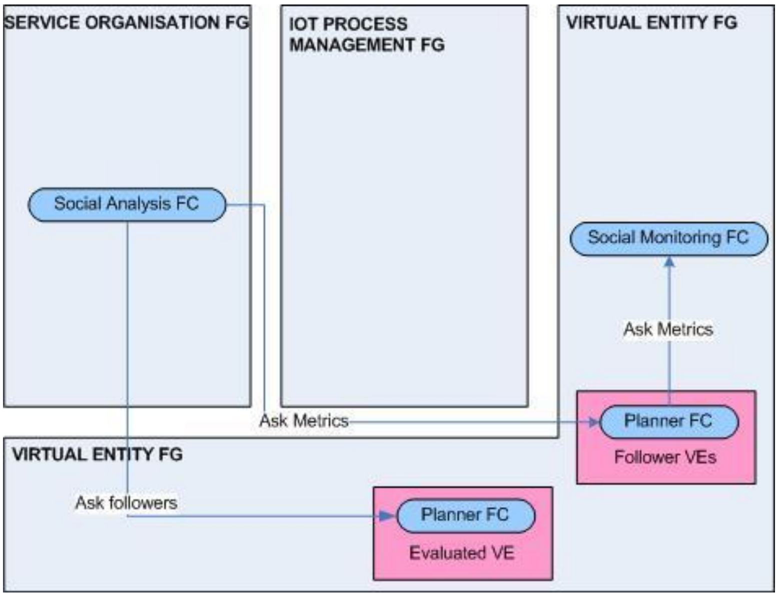


Figure 13: Interaction diagram for VE Trust & Reputation ranking

9.3.2.2 Extraction of relational-models

One of the functionalities of the Social Analysis component is the extraction of Relational Models. Such models are the Communal Sharing (behaviors of VEs with collective relevance e.g. a service offered by an entire swarm of VEs), Equality Matching (VEs operate as equals and request/provide information among them in the perspective of providing IoT-services to users while maintaining their individuality), Authority Ranking (established between VEs of different complexity and hierarchical levels) and Market Pricing (VEs working together in the view of achieving mutual benefit and participating in this relationship only when it’s worth doing so).

9.3.2.3 Recommendation of VEs: Friend-recommendation

Another way of acquiring Followees is through a discovery mechanism, which is based on recommendation. Discovery through recommendation is more reliable and provides protection from malicious behavior. New Followees can be recommended to a VE by its current Followees or by the SA component.

In the first case, transitivity is used (e.g. VE_1 recommends to VE_2 its own Followees as new Followees, after VE_2 has asked all VEs for Followees). After the VE_2 acquires a number of recommended Followees (from VE_1), it asks the Social Analysis component for their Dependability Indexes. The Social Analysis FC then calculates the indexes and forwards the result back to the VE_2. Finally, the VE_2, based on the thresholds set by the user, decides whether it will accept the new recommendations or not.

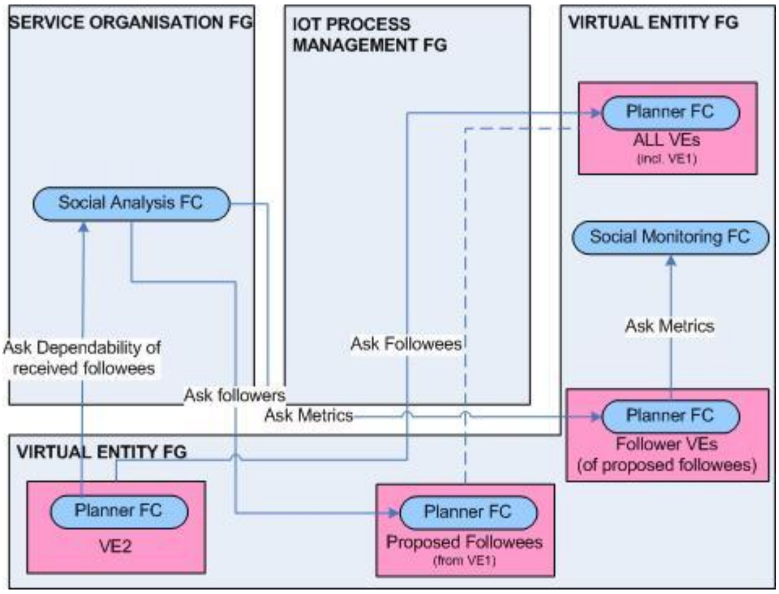


Figure 14: Interaction diagram for Friend recommendation (case 1)

In the second case the Original VE sends a Followee recommendation request to the Social Analysis component. Practically, this leads to the renewal of its Followees List. The VE sends the request, passing as parameters weights for calculating the Dependability Index, a minimum acceptable limit of its value and the current Followees List. The Social Analysis FC calculates the Dependability of the Followees, based on the above input. If the new indexes are below the limit, the Social Analysis FC purges these VEs from the list, replacing them with more reliable ones, and a new Followees List is returned. Followees that have been set by users and do not have high Dependability Index anymore are not thrown away from the Followees List, but are isolated (are not used by the sharing-mechanisms) till their Dependability Index gets high enough.

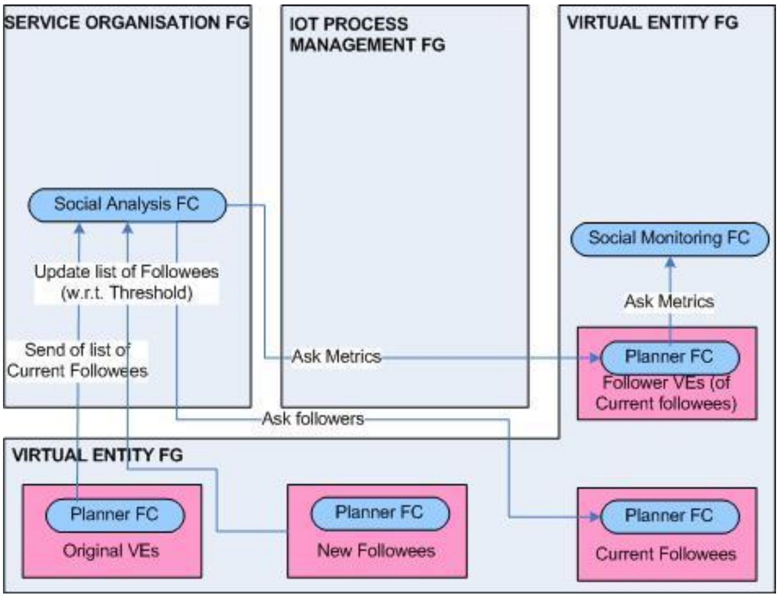


Figure 15: Interaction diagram for Friend recommendation (case 2)

9.3.2.4 Modeling and Visualization of networks

Visual representations of social networks help to understand features of the network that are not easily identifiable and convey the result of the analysis. Collaboration graphs are used to illustrate the quality of relationships between VEs based on characteristics such as the evolution of their Trust and Reputation. Moreover, the network propagation modeling can be included in this functionality. This work is till in progress and will be fully illustrated in Year 3 iteration.

9.3.2.5 Extraction of structural characteristics of the networks

There are many properties of the networks that could be analyzed without direct modeling 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 that knowledge flows, whether a team has any weak points that can be structurally overcome etc. A representative example is the discovery of structural holes. Networks rich in structural holes are a form of social capital in that they offer information benefits. COSMOS could make recommendations to fill in these structural holes and exploit the social capital.

9.3.3. Virtual Entity FG System Use-cases

9.3.3.1 Registering a new VE

VE registration is a process executed by the VE developer once a VE has been deployed. The process itself involves the description of the VE and its capabilities as described in Section 9.1 The goal is to make the VE retrievable by others in order to support the sharing mechanisms of COSMOS (e.g. for data, experience, models).

The same front-end can be used for other related operations such as VE description updates and deletion.

The registration involves the front-end provided by the registry but could also be done programmatically through the registry's REST interface, if other registration clients are developed.

The VE developer uses the VE registry front-end's registration form in order to fill in the description of the new VE. Once the description is complete, the VE developer submits the form which is processed by the back-end and translated into triples persisted into the triple store.

9.3.3.2 Retrieval of VE

In addition to the VE registration functionality, the VE registry also allows the retrieval of VEs based on user defined search criteria. The main target for this functionality are the COSMOS enabled application developers. Application developer need to search for suitable VEs to be integrated into their applications. This is mainly a process executed during the design time of the application and is facilitated by the dedicated front-end which the registry exposes.

After registration, VEs are available for retrieval either by application developers using the VE registry front-end but also by other VEs, and components using the query REST API. Any such request is translated by the registry back-end into a query of the triple store which retrieves the VEs meeting the search criteria.

9.3.3.3 Creation of Cases from historical data

Creating Cases from historical data will be done by accessing the Cloud storage on a per Application basis. The Application Developer will have to specify which data are used and in what combinations, in order to structure Cases from previous information produced by the VE through its sensors or actuators. The required Storlets will have to be coded and used in order to retrieve said data and on the VE side, the Planner functionalities will be used to create the actual Cases inside the local CB.

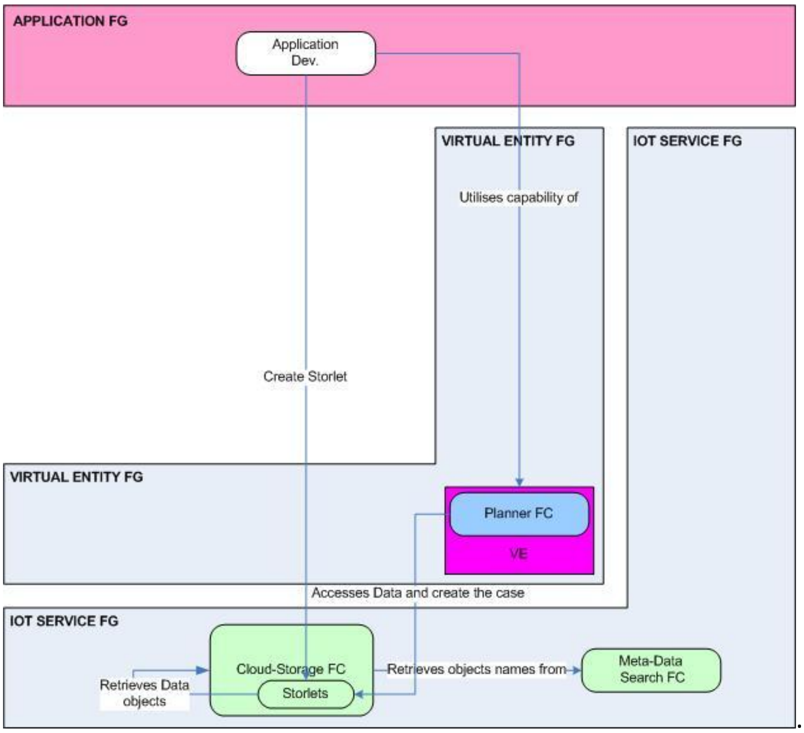


Figure 16: Interaction diagram for creation of a Case from historical data

9.3.3.4 Creation of Cases through the CBR cycle: Retrieve, Reuse, Revise, Retain Cases

The Planner component acts as the creator of Cases on a per Application basis. The Application Developer is responsible for using the Planner methods of updating the local Case Base, by providing the structure of his Cases. The creation of Cases through the CBR cycle is thus performed by Application specific code making use of the Planner functions required. Retrieving a specific Solution to a given Problem may lead to the creation of a new Case as similarity calculations between Problems during reasoning will invariably involve subtle differences in values. Retaining these new Cases will enrich the Knowledge present in a VE. Reuse of Cases will be presented when the VE has need locally for a Solution to a provided Problem, or when responding to an Experience Sharing request. The VE may also be required to modify the retrieved Solutions, during the revise stage, at which point the process can be based on how changes in the Problem values affect Solution values (linear, exponential) and on which value ranges if applicable.

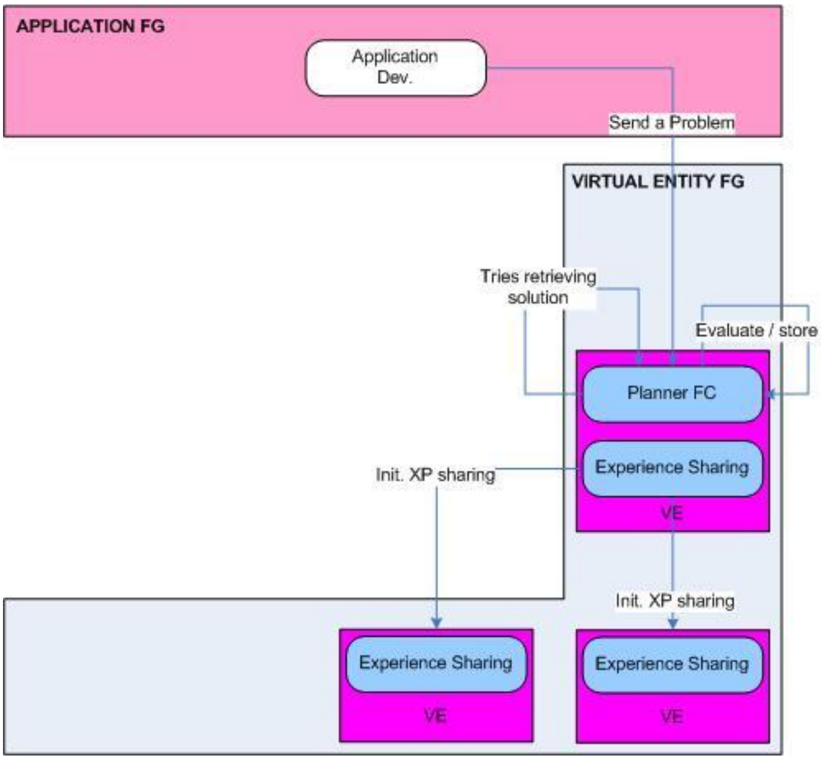


Figure 17: Interaction diagram for Creation fo a Case through CBR cycle

9.3.3.5 Connection of Cases with Topics and their activation

The Planner uses functions for connecting to and listening on the Message Bus. This connection implies that the events detected and propagated on the MB are corresponding to the Knowledge present inside the VE in the form of Cases. Therefore the detection of an incoming event will trigger the retrieval attempt of a similar Case first locally and if not present then through Experience Sharing. Solutions of retrieved Cases are thus presented or directly used, depending on their nature.

9.3.3.6 Monitoring of social interaction metrics (Shares, Assists, Applauses)

These metrics are connected to the evaluation process of a VE when a Case is retrieved through the use of Experience Sharing. The End-User and the system itself (or possibly either one) will provide a positive or negative feedback. If the feedback was negative, then the only metric which increases is the Shares metric. If the feedback was positive, the VE will act depending on whether the remote VE was the actual Experience Provider or a “broker” during the process. In the first case, the VE will increase the Applause metric along with the Shares metric. In the second case, the VE will increase the Assists metric along with the Shares metric.

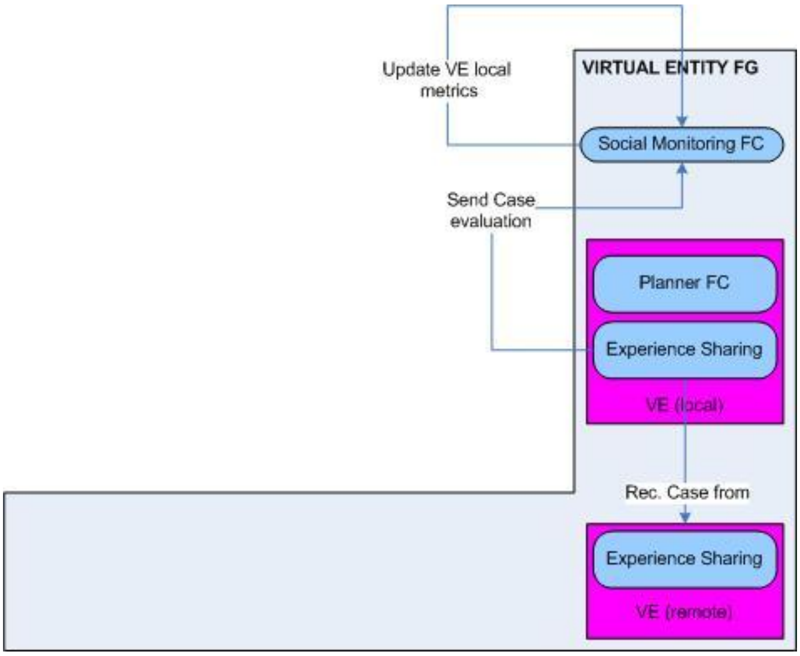


Figure 18: Interaction diagram for Monitoring of social interaction metrics

9.3.3.7 Experience Sharing evolution; Models, Pro-activeness and Request Handling

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

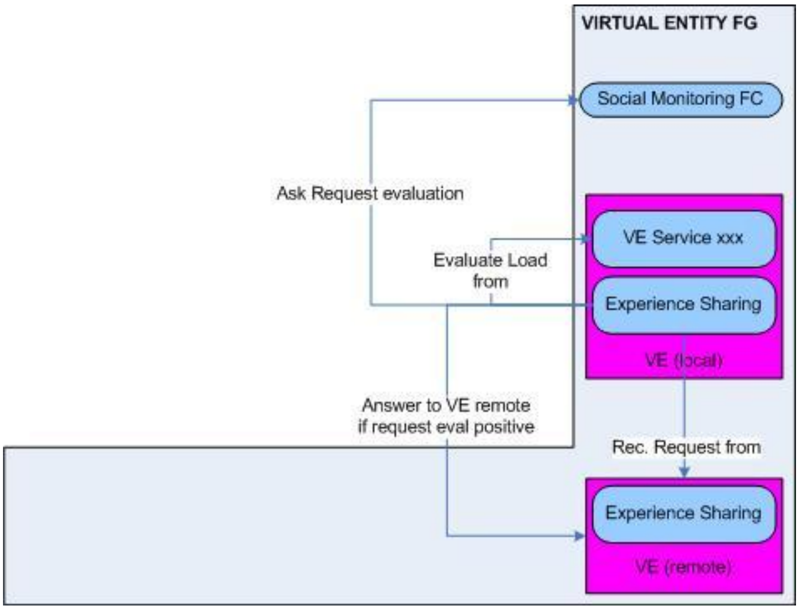


Figure 19: Interaction diagram for Experience Sharing (case a/)

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

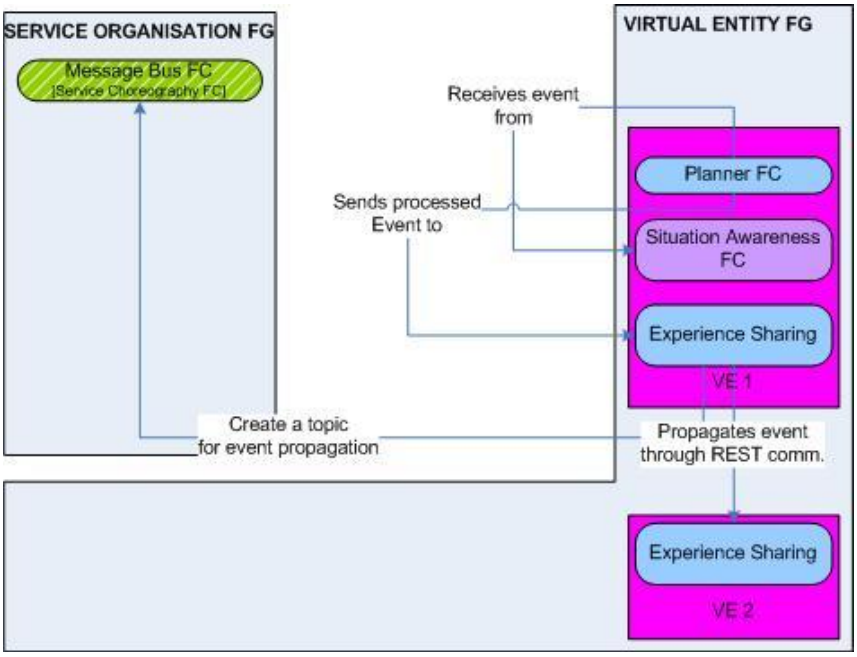


Figure 20: Interaction diagram for Experience Sharing (case b/)

Future considerations for Year 3 also include the possibility of COSMOS to accept Models as a Service on the Platform level, connected to Model Developers coding and uploading them centrally. A semantic description of those Models in the form of Services, will lead to the Experience Sharing component diffusing them as is the way with Cases currently and informing other VEs of their existence.

9.3.3.8 Configuration of Privelets

The VE Developer is responsible for filling in the `Privelets` configuration file (part of Member FC within Management FG), structured in Json format. An example of this file is shown below:

```
{
   "interval_between_requests":"10",
   "data":{
      "temperature":"public",
      "id":"private",
      "latitude":"private",
      "longitude":"private",
      "domain":"public",
      "humiditiy":"public",
      "ipAddress":"private"
   }
}
```

- **`interval_between_requests`:** if VE_1 sends a second request to VE_2 within this time interval then the VE_2 refuses to give back the data requested. The unit of measurement is the second and the value is an Integer;
- **`data`:** all the data that are generated by the VE. They can be tagged either as public or private. Only the public ones are available through the message bus or through VE2VE communication. (GET or POST REST requests)

9.3.3.9 Access (from another VE) to any kind of VE-data (including VE property, IoT Service, Experience Sharing, etc)[Privelet FC]

VE_1 needs some kind of information so it searches in its followees list where it finds VE_2. VE_1 sends a GET or a POST request to VE_2 alongside with a request key, which is used during the decentralised authentication process. VE_2 gets the request and searches in its followers list to find VE_1 and tries to match the request key with the one already stored in the list. If the matching is successful, VE_2 reads the `hasLastTimeRequest` of the VE_1, which is a Data Property of the followers ontology. If `currentTime – hasLastTimeRequest > interval_between_requests` then VE_2 reads its configuration file to check whether the requested data is public. If yes, then VE_2 provides its temperature alongside with a response key which is used for authentication in a similar way as above. Finally, VE_1 confirms the identity of VE_2 using the response key.

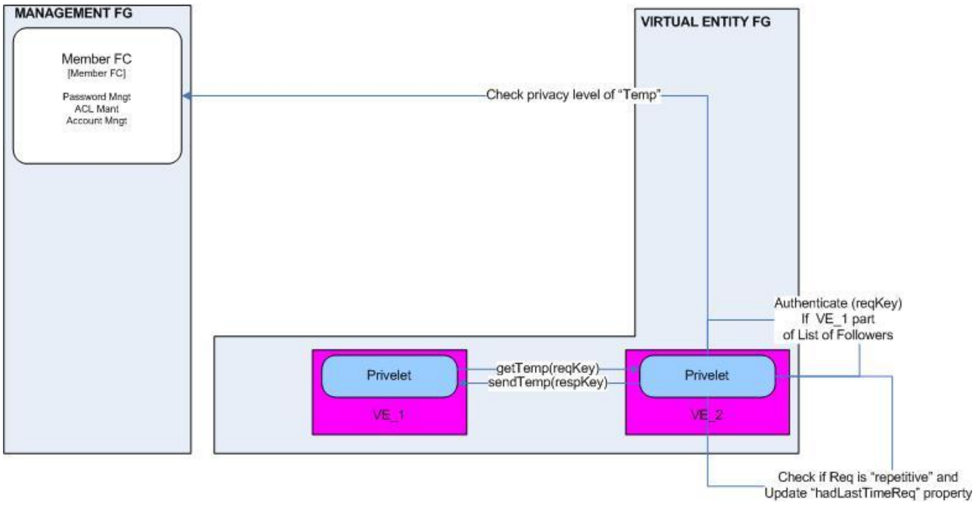


Figure 21: Interaction diagram for VE accessing service from another VE (Privelet)

9.3.3.10 Detecting Complex Event using Event (Pattern) Detection FC

COSMOS intends to provide the functionality of detecting a complex event at VE level in real-time using light weight version of CEP which is able to update its rules automatically. An application developer will define the initial set of rules, input data streams and the complex event. COSMOS platform will gather historical data and later keep updating the rules by exploiting historical data and finding optimized threshold values. Figure 22 shows the data flow and high level architecture for detecting complex event using Event Detection FC.

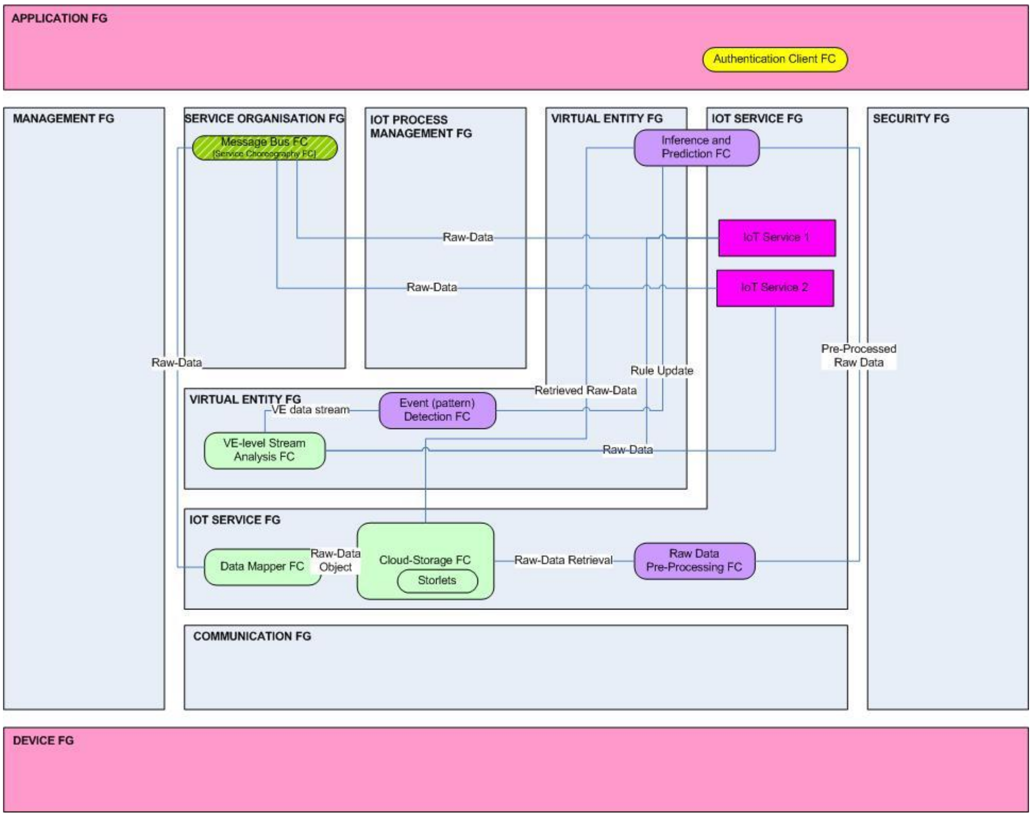


Figure 22: Interaction diagram for Detection of Complex Event using events

9.3.3.11 Extracting high-level knowledge using Inference and Prediction FC

Application developer defines the input features and interested output quantity for the data mining models. Depending on the historical data, application developer chooses specific data mining algorithm. If labelled data is available, then supervised machine learning models can be trained otherwise unsupervised statistical model is trained. After training the specific model, the model can be deployed for extracting high-level knowledge which can be used for other applications. Figure 23 shows the data flow and high level architecture for extracting high level knowledge using Inference and Prediction FC.

The Inference and Prediction FC can take both raw data and VE data as input depending on the scenario and the usage required by application developer. Figure 23 shows the connection of components for inference and prediction on VE data. The flow of data is almost the same for raw data as input with the additional capability of carrying basic pre-processing on raw data before storing it.

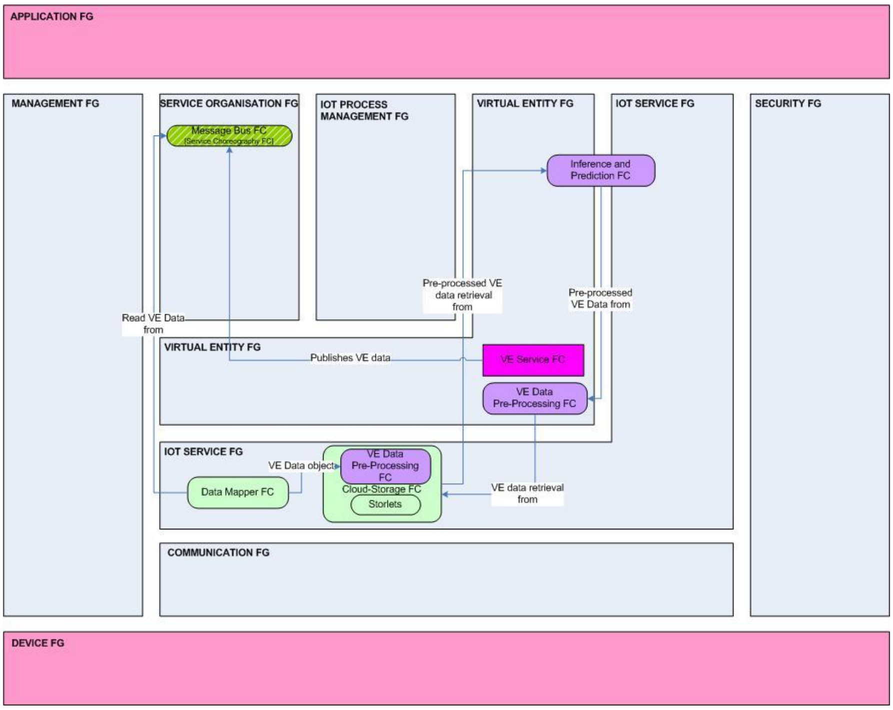


Figure 23: Extracting high-level knowledge using Inference/Prediction FC

9.3.4. IoT Service FG System Use-cases

9.3.4.1 Registering a new IoT Service

An IoT service is registered in COSMOS either as part of the VE registration or prior to it. Either way, its registration is based on the VE registration front-end, and the process is similar to that of a VE registration: attributes are described semantically and interfaces include both the semantic as well as the data type description.

COSMOS ontology also supports the use of IoT services which were not developed according to the COSMOS design patterns but are still semantically described. This is possible by specifying - when the binding to a VE property is done - the type of the IoT service interface, and the URI of the semantic description of that specific service. This mechanism facilitates the interoperability with IoT services developed and described outside the COSMOS environment.

9.3.4.2 Accessing an object with known identifier

We use OpenStack Swift as our base framework for object storage and Swift has a REST API which allows CRUD operations on objects. In order to retrieve an object from Swift with a known identifier (denoting the Swift account, container and object names) one needs to generate a REST request using the identifier and provide the necessary credentials in order to retrieve the object. This can be done using the OpenStack Swift client software. This is an operation which should be done by applications developed to use COSMOS. In future a generic COSMOS application could be developed using this scheme which would allow end users to access VE data in this way.

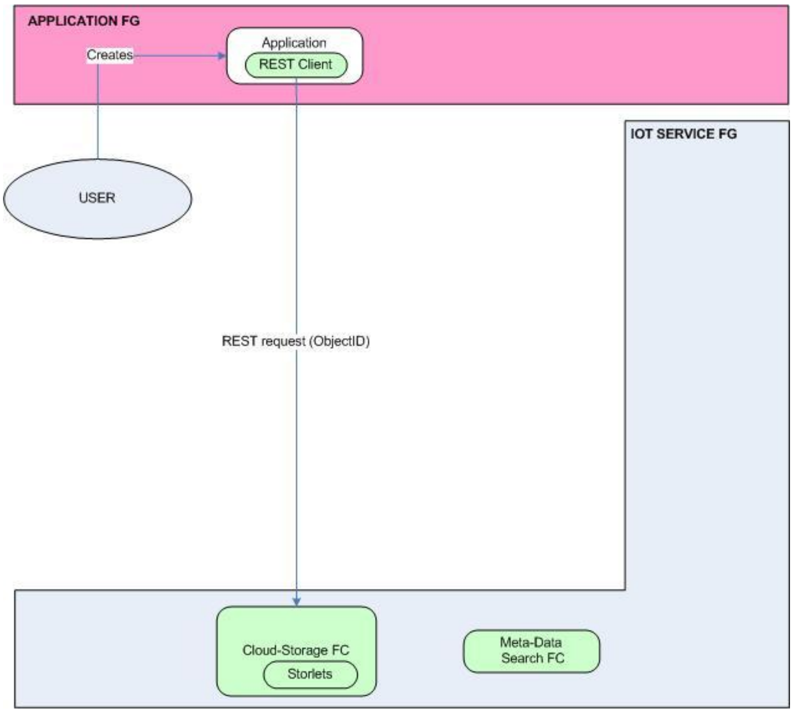


Figure 24: Interaction diagram for Accessing an Object with known Object ID

9.3.4.3 Analytics (and connection to Storlets and Cloud Storage)

Cloud Storage (OpenStack Swift) was integrated with an analytics framework (Apache Spark) in order to enable analytics on data produced by VEs. In order to access data in the Cloud Storage using Spark, the Application Developer needs to use the "`swift://`" namespace and access the data via its container name/object name. If a storlet needs to be invoked then there is a special syntax for including this in the URL. The details are provided in deliverable D4.1.2

9.3.4.4 Accessing an object through meta-data search

One can search for objects which have certain metadata values. The search returns a list of object identifiers meeting the constraints. Then one can retrieve each object using case 9.3.5.2

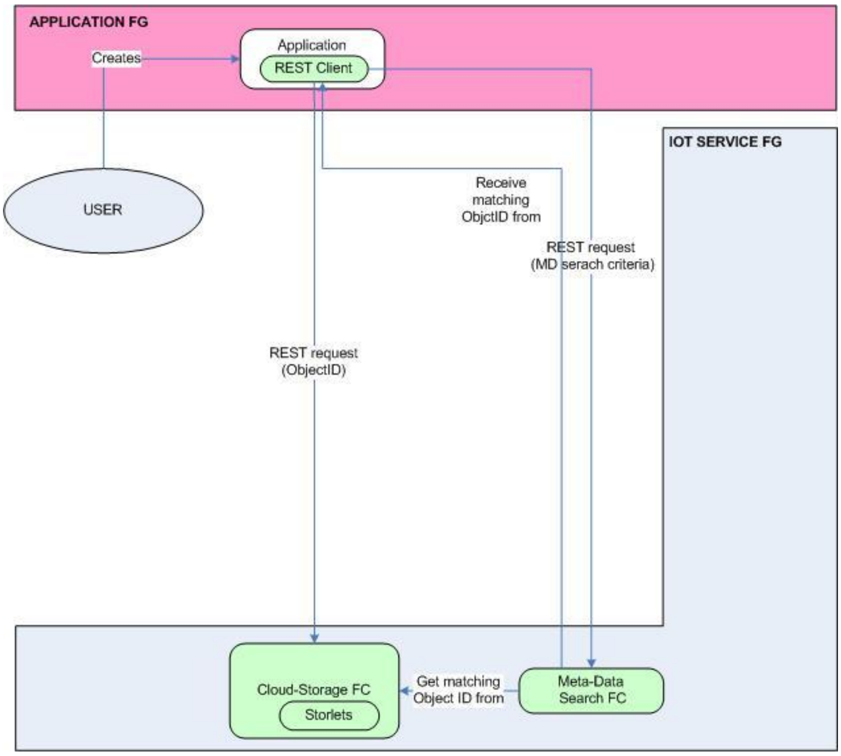


Figure 25: Interaction diagram for Accessing an Object using the Meta-Data Search

9.3.4.5 Creating and uploading a storlet

An Application Developer develops a storlet by writing Java code which conforms to the Storlet interface. The compiled Java code is uploaded to the Swift object storage as a `jar` file in a specific system defined container and having certain metadata. The details are provided in deliverable D4.1.2.

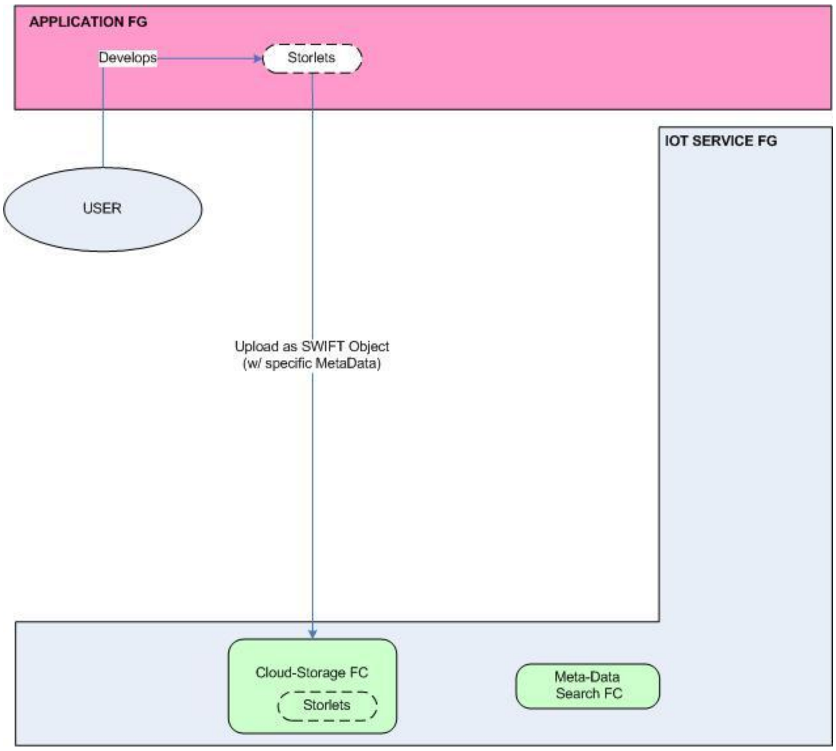


Figure 26: Interaction diagram for the creation and upload of a storlets

9.3.4.6 Using data-mapper to create and store an object

Data Mapper runs continuously inside the COSMOS platform. Firstly, it reads the configuration file and checks whether its mandatory fields `(size, period, mandatory metadata)` are properly filled in by the VE Developer. If yes, then the component reads these values alongside with the `optional_metadata` ones. After that, it subscribes to the Message Bus and keeps consuming messages. The messages are sorted depending on their origin (VE Id) and are stored in the buffer as aggregated messages. When a *period* of time elapses, the Data Mapper checks all these aggregated messages and those, which are larger than the *size* value, are stored in the cloud as data objects, annotated with the metadata mentioned above.

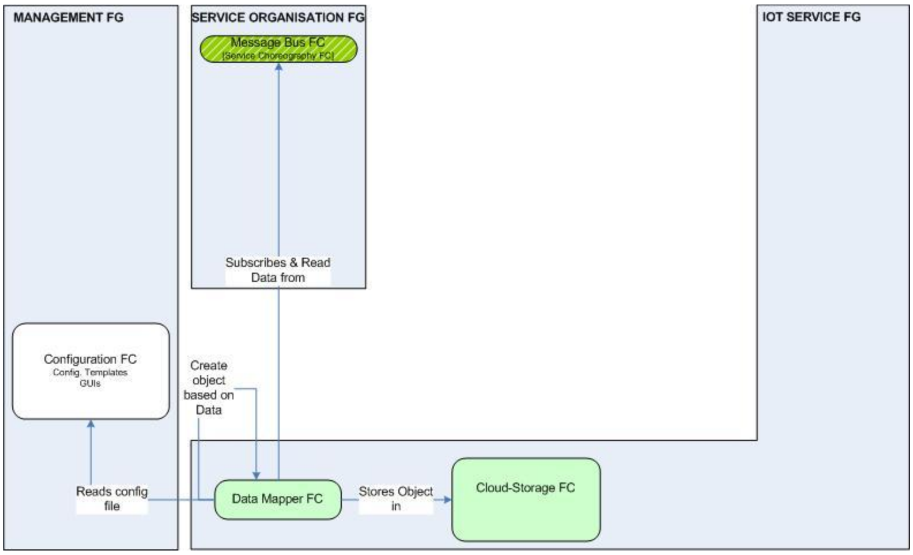


Figure 27: Interaction diagram for Creation and Storage of an object using the Data Mapper

9.3.4.7 Data-mapper configuration / policy (period and size in Year 1)

The VE Developer must fill in the Data Mapper's configuration file, structured in Json format. Please see an example below:

```
{
   "period":"1",
   "size":"1000",
   "mandatory_metadata":{
      "Timestamp":"ts",
      "Id":"hid"
   },
   "optional_metadata":{
    "Estate":"estate"
   }
}
```

The `period` key indicates how often the Data Mapper sends the data to the cloud storage. The unit of measurement is the minute and the value is an Integer.

The `size` key defines the lowest threshold of the message to be stored. If an aggregated message is smaller than the threshold, it has to wait for the next period before being stored. The unit of measurement is the kilobyte and the value is an Integer.

The `mandatory-metadata` mean that the Data Mapper does not store any data that do not contain this kind of information, whereas for the optional ones, we give the VE Developer the capability to annotate the data with enriching metadata.

9.3.5. Security FG System Use-cases

This section provides typical system use-cases pertaining to the Security FG, like secured VE to VE communication, plugging in a new H/W board, authentication and authorisation process, including setting-up the access rights and revocation.

9.3.5.1 Enrolling a H/W Board

New H/W Security Boards need to be enrolled into the system, operation by which a device unique key is generated and associated with the H/W board. A H/W Board owner needs to have a COSMOS account which will allow the enrolment process to take place.

The H/W Board allows for 2 types of key enrolment:

- Manual: the owner loads the key manually in the board (either types is in a local terminal or load it via a USB stick);
- Automatic: the owner connects the H/W Board to the COSMOS platform and selects it in his account after the which the key will be automatically loaded in the H/W Board.

Please note that the automatic enrolment process assumes a H/W Board with a generic COSMOS public key and a device unique ID known and recognizable only by the rightful owner. If this automatic mechanism does not work it falls back to the manual one.

The key exchange mechanism is realized by the Diffie-Hellman[21] key exchange protocol.

When a new H/W Board is to be enrolled the owner/user needs to log into COSMOS using a user/password pair. The credentials are verified by the Member FC which in turn fetched a new configuration template from the Configuration FC and triggers a new key generation Key Exchange and Management FC. If a manual enrolment is to be performed the owner will manually take the key (e.g. save it to a file) and install it to the H/W Board. If an automatic enrolment is performed the user will need to connect the H/W Board to COSMOS. The owner will need to select his H/W board from a list of [new] available boards. This triggers the system to authenticate the H/W Board using the standard COSMOS key. After the board is authenticated and authorized, the key exchange takes place. At the end of this procedure the H/W Board will have a new key which is enrolled into COSMOS with the owner selected credentials and authorizations.

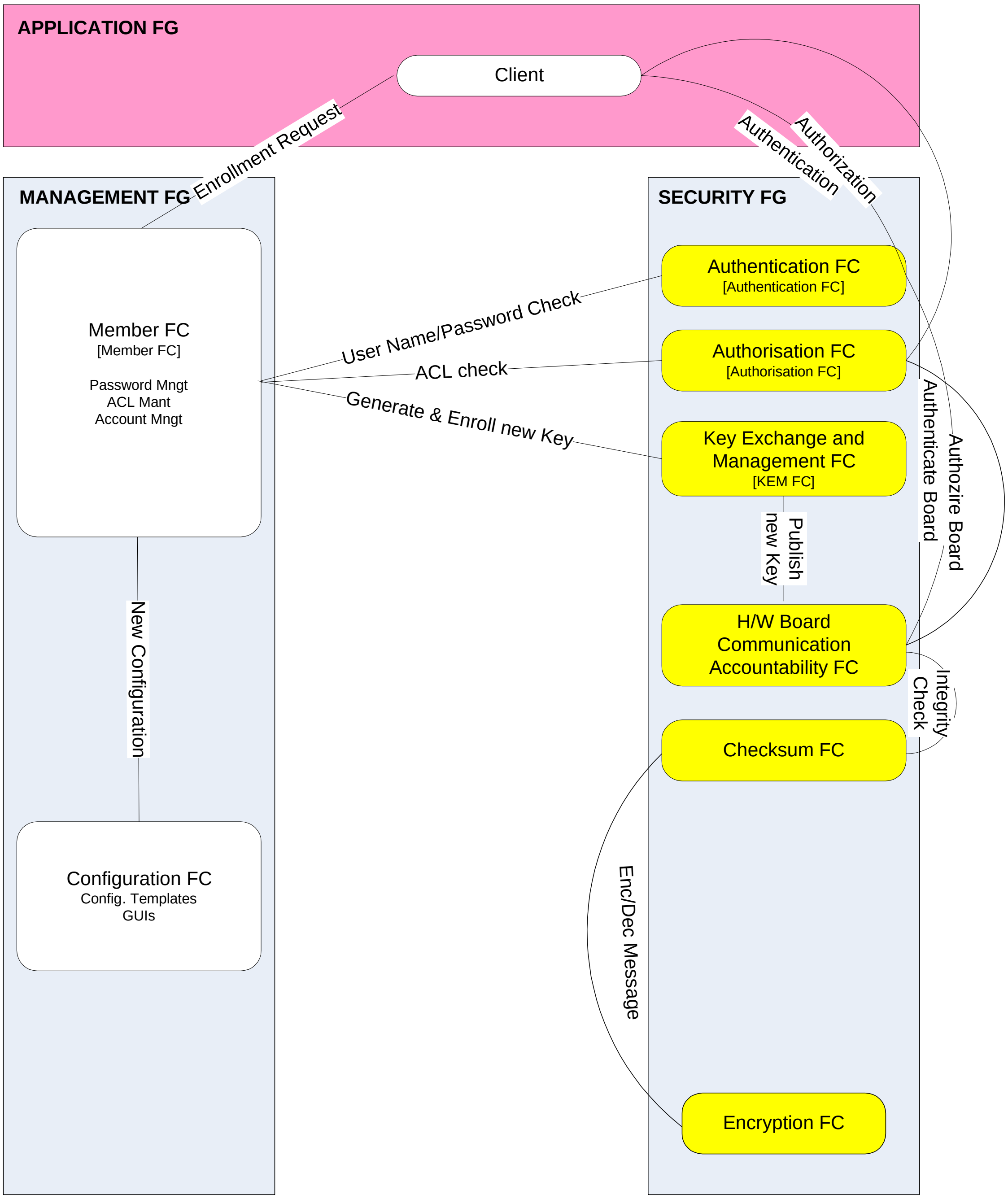


Figure 28: Interaction diagram for H/W Board enrolment

9.3.5.2 Authentication process

Is the process by which a VEs identity is verified. The authentication process covers both the authentication of human users and of VEs. In case of human users authentication is realized by user-password pairs which are unique to each user. In case of VE authentication the process is realized by means of encryption. Each data package, consisting of a timestamp (nonce), device unique ID, actual data payload and data hash, is encrypted with the device unique key. The data package can only be decrypted if the key has been generated and enrolled within the COSMOS platform and if the VE is authenticated by means of its unique ID. Non authenticated packages are scraped but are used in order to compute the reputation index of the VE.

In case a human user logs in, he or she will be using a user name/password pair. In case a VE communicates to COSMOS each data packet will be authenticated. Each authentication uses the Member FC/ACL.

In case a VE communicates to COSMOS, the data packet is checked of accountability by the H/W Board Communication FC. After the data packet is checked for its integrity the appropriate key is selected as indicated by the `communication_id` header. The data packet is afterwards decrypted and authenticated. If any of the steps fails or produces an error (e.g. the key selected by the `communication_id` cannot decrypt the message or the decrypted `message/device_id` does not match the ACL then the reputation index of the VE is decreased and the data packet is scraped.

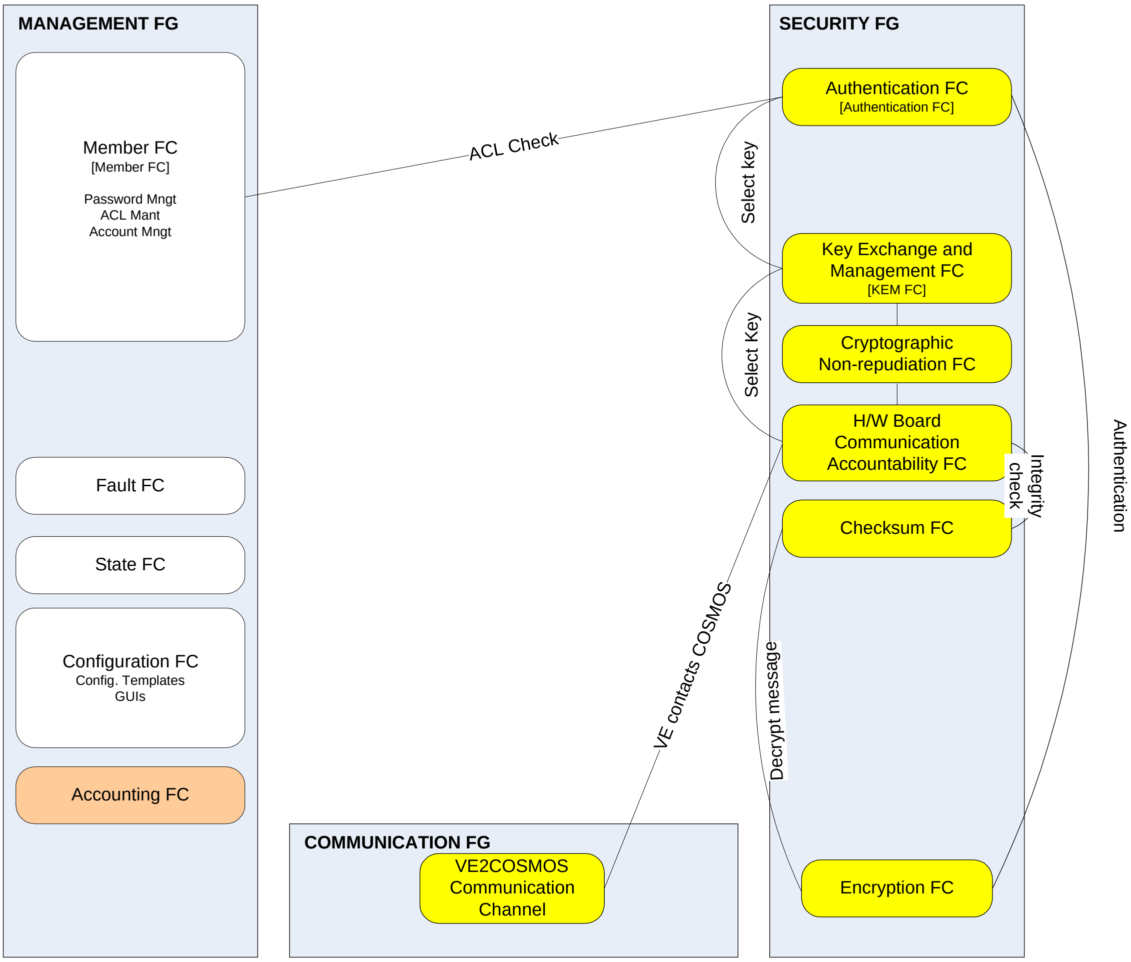


Figure 29: Interaction diagram for Authentication process

9.3.5.3 Authorisation process

Is the process by which an entity (human user or VE) is allowed to perform only permitted operations.

During enrolment each entity is associated with a set of permissions (e.g. `read_data` or `write_back`) which determine the allowed operations. Permissions are enforced for each data package in order to mitigate man-in-the-middle attacks.

In order for a VE to perform operations within the COSMOS platform, both authorization and authentication need to be successfully completed!

The authorisation process is transparent to both human users as well as VEs – the process takes place in the COSMOS platform.

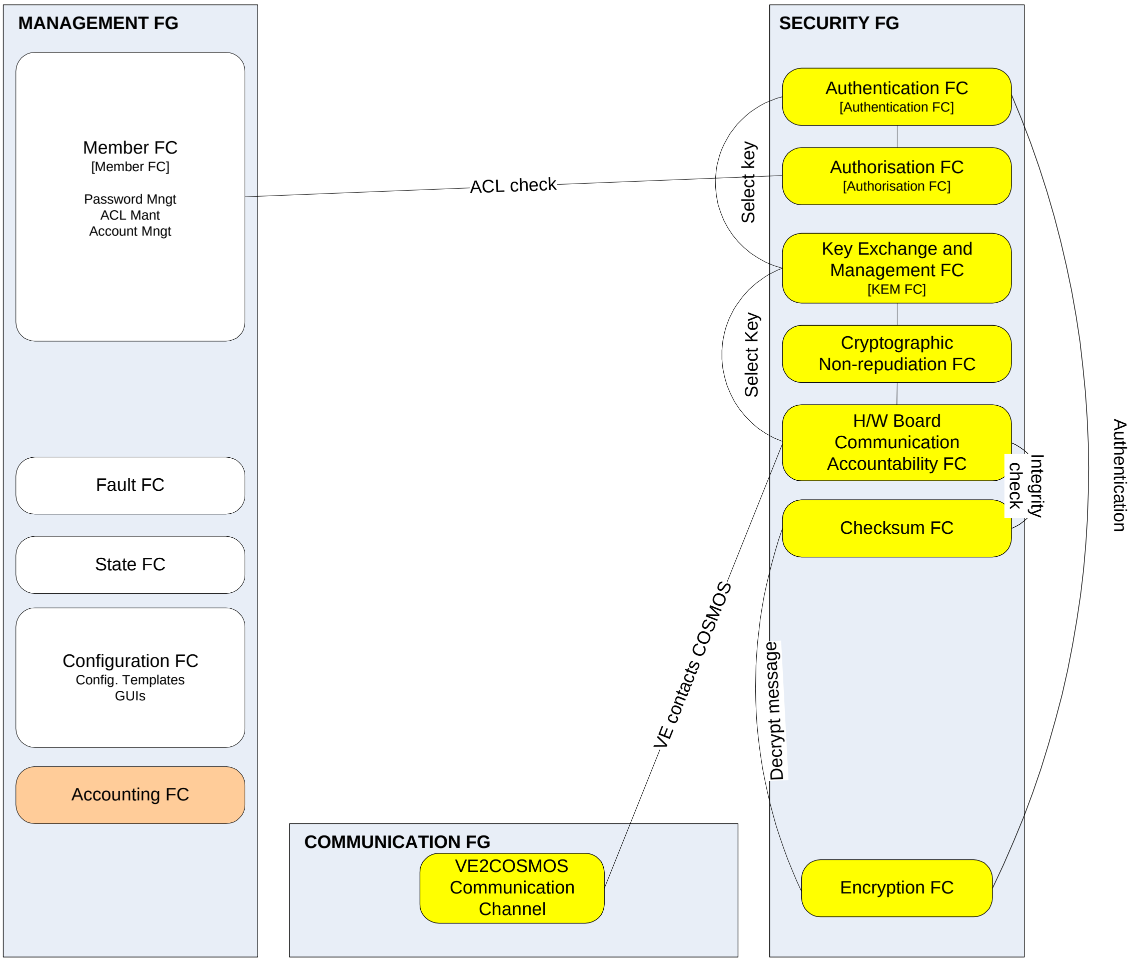


Figure 30: Interaction diagram for Authorisation process

9.3.5.4 VE accessing COSMOS

A VE accessing COSMOS is a typical use-case (e.g. push or pull data). After the VE has been enrolled is uses the received key to encrypt data packages. A data package has following typical structure:

- Timestamp (e.g. nonce) [ms resolution];
- Device Unique ID [bit stream which uniquely characterizes the device - hardcoded];
- Data payload [data structure];
- Data hash;
- Communication_ID.

Each data package is encrypted using the on-device AES cryptographic algorithm and is sent over the data carrier (e.g. public network) to the COSMOS platform. When the package reaches the platform, based on the communication ID COSMOS selects the appropriate key.

The message is decrypted and the device unique ID is verified as well as the timestamp (nonce). If the data package is authenticated and the VE is authorized, the data is processed and the VE receives back an acknowledge message consisting of the initially received timestamp and it's hash, both encrypted using the device unique key.

A basic communication flow is:

```
send(encrypt(timestamp,
             device_id,
             data_payload,
             hash(timestamp,
                  device_id,
                  data_payload,
                  communication_id)
             ),
     communication_id
)
receive(decrypt(select_key(communication_id),
                           message)))
answer_back(timestamp,
            hash(timestamp))
```

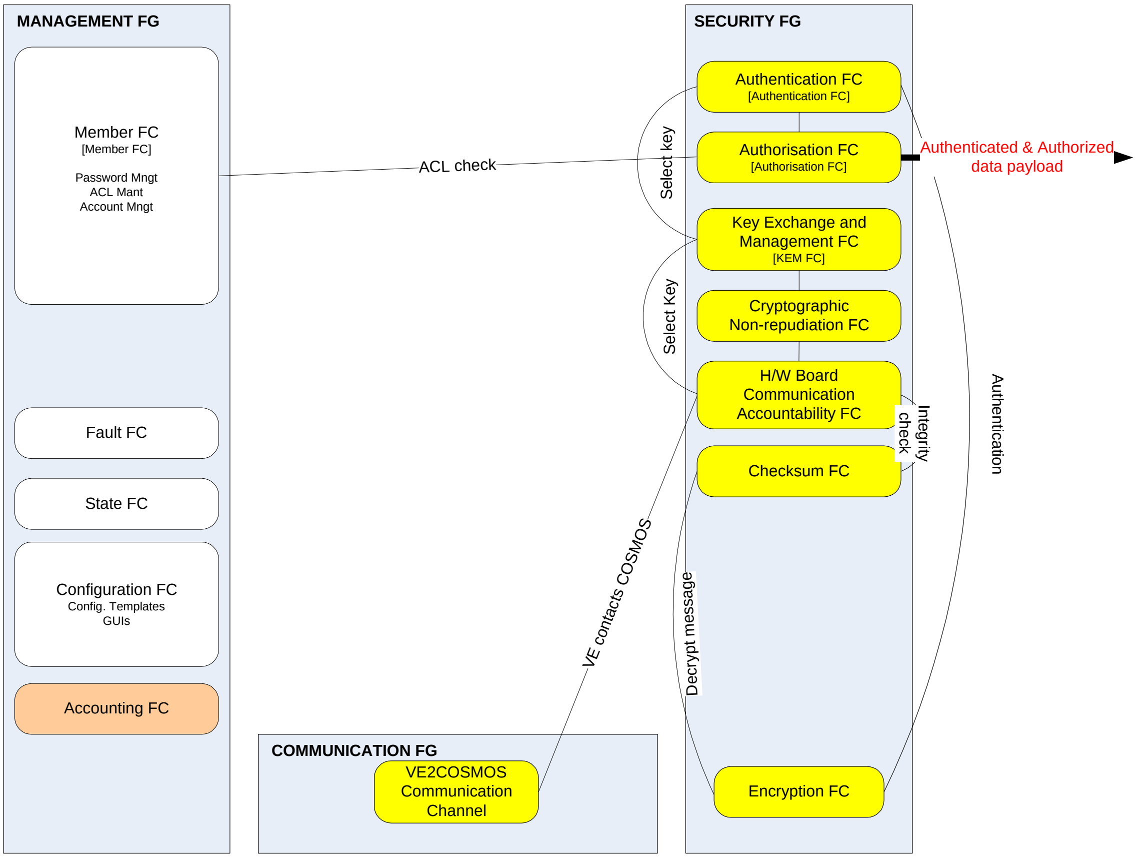


Figure 31: Interaction diagram for a VE accessing the COSMOS platform

9.3.5.5 VE accessing another VE

There are 2 options with respect to VE2VE communication:

- VEs located behind the same H/W Board;
- VEs located on different H/W Boards (e.g. different sub-nets).

Both cases begin with the same procedure where a VE (e.g. VE_A) contacts COSMOS in order to set-up the communication to another VE. After VE_A is authenticated and authorized to do so, COSMOS authenticates and verifies the authorization of VE_B. If both VE_A and VE_B are allowed to access each other, COSMOS verifies their location. If both VEs are connected to the same H/W Board, thus share the same device unique key, they are allowed to access each other unencrypted are notified of this.

If the VEs are not located behind the same H/W Board, COSMOS generates a session key which is shared to both VE_A and VE_B. The shared key is send to each VE encrypted, using the device unique key of each VE, individually.

9.3.5.6 Compromision detection and revoking

In case the a VE or a H/W Board is compromised, COSMOS revokes the access to the platform. The revocation process is similar to the enrolment process but the device unique key is marked as un-trusted and all its permissions are revoked. A VE can be compromised if its reputation index reaches 0 due to:

- mismatch of communication ID;
- mismatch of device unique ID;
- mismatch of timestamp;
- mismatch of hashes.

9.3.6. Application FG

9.3.6.1 Application Design Definition and Reusability of Template Interactions

One of the key features for application creation and design is to give the ability to application developers to create the application workflow that may utilize one or more COSMOS platform services, VE services or application logic. This functionality is exposed through the Service Orchestration FC. In this FC the developer should be able to define the flows needed to implement application logic and produce or consume information from the available services (Figure 32). A critical functionality here is the re-use of existing templates, either with regard to application designs or with relation to concrete system use cases, as these have been detailed in the previous sections.

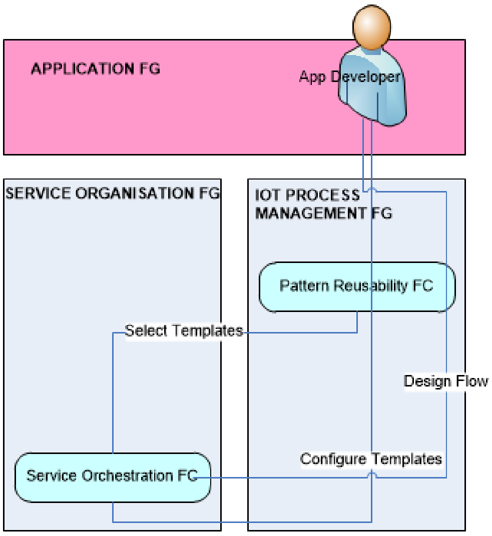


Figure 32: Generic Application Design

9.3.6.2 Different capabilities for VE/Topic Discovery

Based on their needs, mainly from a semantic point of view, Application Developers may need to include one or more VEs in their application logic. To do so, they might utilize a number of discovery mechanisms:

VE filtering through static capabilities/semantics

Based on queries on the VE registry, they may acquire the endpoints (in terms of REST interfaces or MB topics) in which specific VEs (identified through their IDs) publish their information. By querying or registering to these endpoints, they acquire the necessary pieces of information. This is handled through the relevant System Use Case described in 9.4.2.

VE filtering through value-based historical data

In case a specific VE is needed based on a higher level logic (e.g. based on similarities in the historical data or patterns), a suitable storlet should be in place that implements the according comparison logic and returns the VE ID that corresponds to the respective filter. This is handled through the relevant System Use Case described in 9.3.4.5.

VE Discovery through value-based real time data

Given that an application developer or the application itself may need to discover VEs that abide by a specific constraint (e.g. located in a specific radius of GPS location), they may define suitable CEP rules on an application logic component, that is registered to the according MB topics, listens and filters VEs that correspond to this constraint. This is handled through the relevant System Use Case described in 9.3.3.10.

VE behaviour discovery through the social structures of COSMOS

Another aspect of Discovery is the ability to find VEs based on experience sharing and the social structures of COSMOS (expressed mainly through the decentralized approach). This discovery is mainly on the specific shared aspects (e.g. similar problems and solutions of the Planner). This is handled through the relevant System Use Case described in 9.3.3.7.

9.4. Storage

9.4.1. Cloud Storage

The purpose of the COSMOS Cloud Storage (also known as the 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 is used in order to implement the COSMOS Data Store. Object storage allows defining CRUD operations (Create, Read, Update, Delete) on entire objects, and write-in-place is not supported. This can be suitable for storing historical IoT data (typically time series data) which does not change over time. Objects can be organized into containers, and each container belongs to an account. Account, container and object CRUD operations can be performed using the Swift REST API.

9.4.2. Triple Stores

Triple stores provide the support for the persistence of the semantic descriptions made to VEs and other related entities and components. The VE registry uses a triple store for persistence and retrieval of these descriptions and exposes an API which can we used without any knowledge about the underlying technology. The stores are accessed using SPARQL queries but the complexity of these queries is hidden to the user of the VE registry.

The API thus provides access to CRUD operations from a higher level (e.g. VE description) and handles transparently the triple creation, removal, update and deletion.

10. Deployment View

So far in previous sections we have introduced different models and Views which are rather abstract in the sense they are quite decoupled from any implementation / realisation concerns. When it comes to realizing an IoT System operating in the Real-World it is necessary to understand and show how and where the different parts of the system (IoT devices, Resources, VEs and all kinds of Services (IoT- or not) which were, until now talked about in a rather abstract, way will be deployed and how they will communicate. Part of the deployment view concerns also non-IoT aspects of the system like Servers, Micro-controllers, Gateways, Cloud, Data Bases etc…

The following subsection shows a preliminary Deployment View for the MADRID use-case scenario. This first view will be completed during Year 3 along with the HILDEBRAND and III scenario and featured in the third and last iteration of this COSMOS Architecture Document.

10.1. Deployment View for MADRID Scenario

Within the Madrid Data model we can distinguish three types of Virtual Entities. This classification is based on both the class and type. The first classification is related to the type of information provider infrastructure, differentiating between:

- Transportation: Defining the Virtual Entities involved in the public transport of passengers and their relationships.
- Mobility: Composition of Virtual Entities related to traffic and urban mobility.
- The third classification is related to the type of information and message structure and content. The data model contains polymorphic entities called events.

The Madrid use-case features the following Virtual Entities (for more details relate to Section 4.4 items 3) and 4):

- BusVE
- TrafficLightVE
- BusLineVE
- BusStopVE
- BusDriverVE
- SpecialPersonVE

In addition a collection of sensors are attached to the Bus-P-E represented by the BusVE.

The VEProt system (Virtual Entity Process for Reactive and Ontological Things), designed globally, is the set of processes and protocols that allow a low-level object within the paradigm of IoT, to communicate status and data and to connect sensors and react instantly to changing events.

The following three Figures give an outlook on how the Madrid scenario is currently deployed, focusing on BusVEs and TrafficLightVEs only. The two first ones are focusing on EMT system and show how it is implemented internally, while the third one shows how the EMT IT System as a whole (including the various P-Es) integrates smoothly with the COSMOS framework.

Figure 33 below shows the simplified (actuators like the BusPMV and BusDriverPanel are omitted) deployment of the various sensors (e.g. GPS and CANbus -which is an integral part of the Bus-), hub and IT system at the bus side. As it can be seen, the client part of the VEProt system is the main component deployed within the Bus that bridges the Sensors to the central platform. It also manages the communication between the Bus and the central party of the EMT IT system.

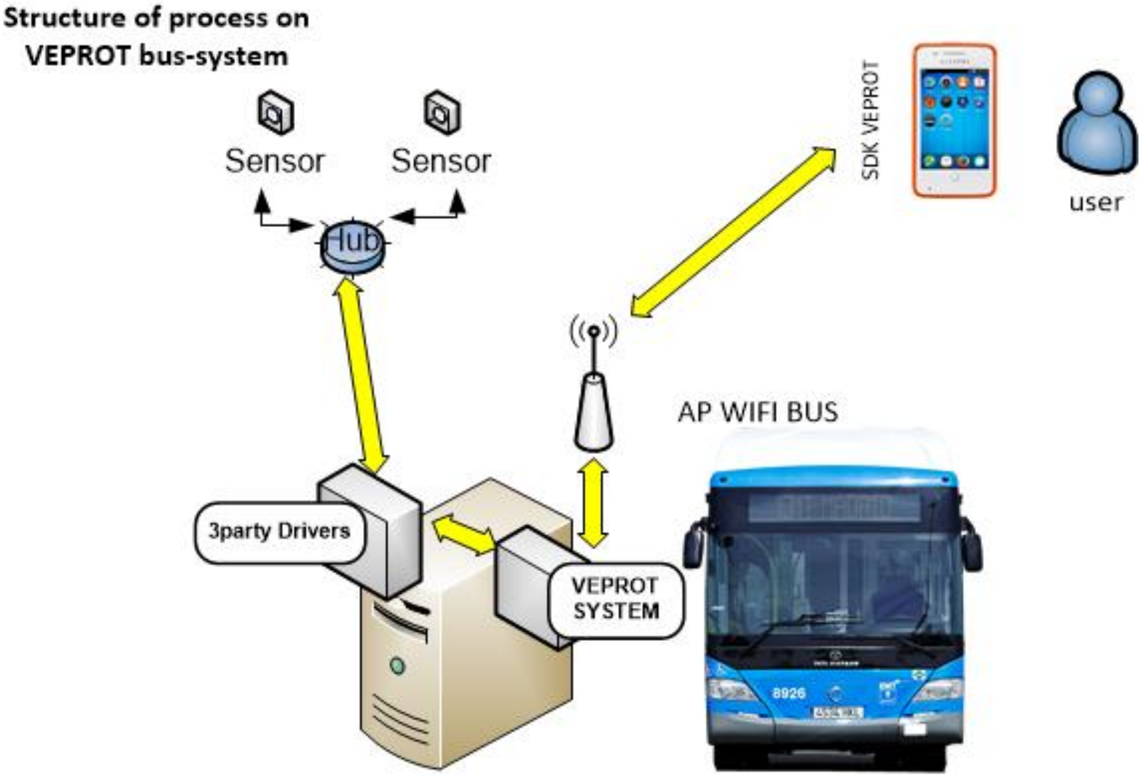


Figure 33: VEProt onboard architecture

This second picture below (Figure 34) focuses on the internal architecture of the server side of VEProt framework and how it links to the Buses and Traffic lights on the one side and to additional servers like the Traffic and Transport servers on the other side. This picture depicts the EMT IT system and does not feature any COSMOS features as such. Its main objective is to manage the way COSMOS components may access information about Buses and Traffic lights, and as such it provides COSMOS with a VE interface; BusVEs and TrafficLightVEs are therefore "logically" implemented at the server side. The ReactiveBox is responsible for publishing/subscribing events towards/from the COSMOS platform.

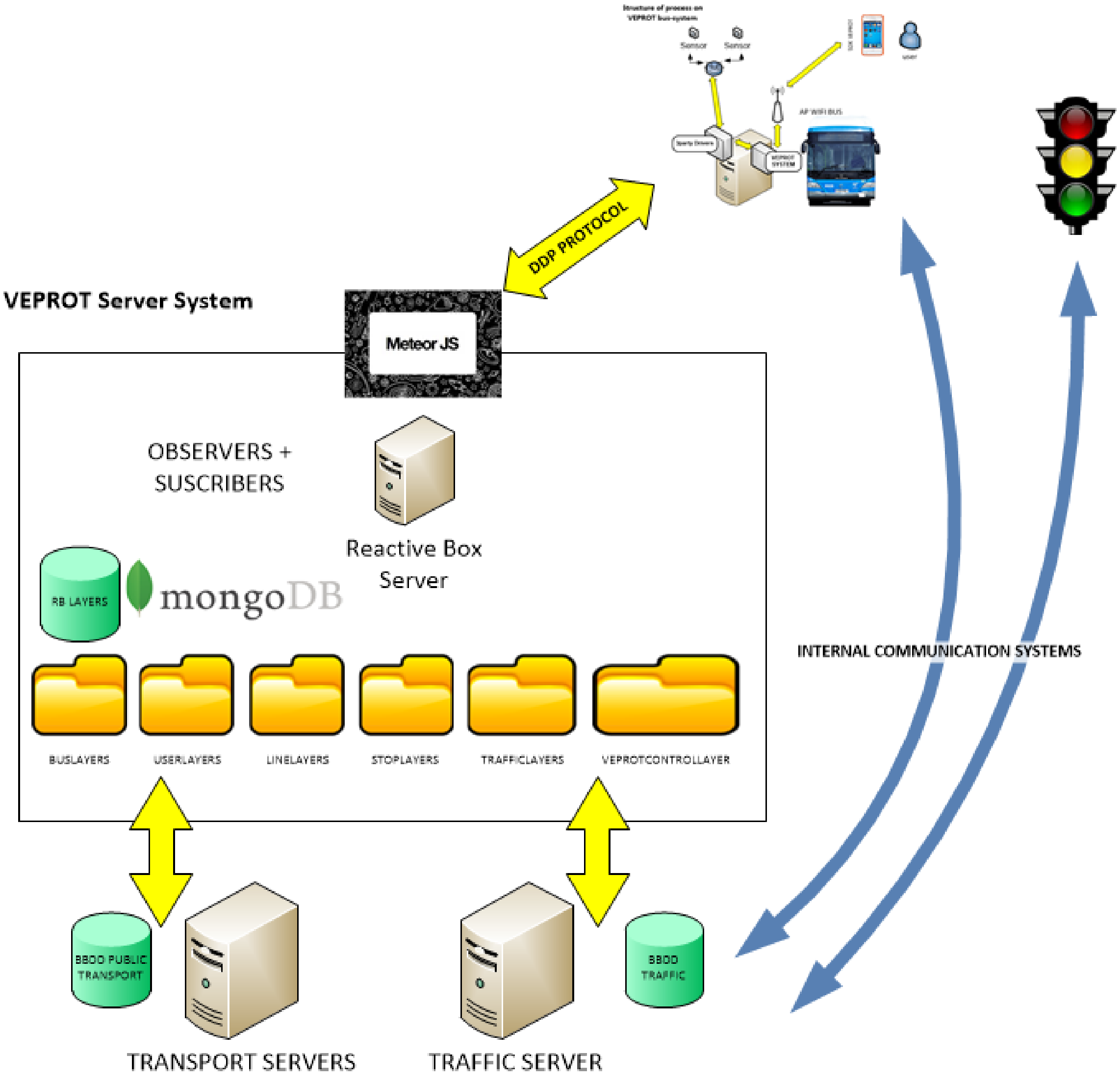


Figure 34: VEProt (server side) with ReactiveBox and Data layers

The final picture below (Figure 35) show how the EMT IT system integrates with COSMOS. The COSMOS system can be seen at the top-left corner of the figure, delivering events to the message bus and receiving events published from EMT BusVEs and TrafficLightVEs. In addition to Real-time events, the EMT system gains access to predictive data coming from COSMOS which are then integrated before being available to the users of EMT+COSMOS system (bottom-left corner) for the sake e.g. of monitoring and planning purposes.

More information about VEProt, ReactiveBox and in general integration of EMT platform with COSMOS can be found into WP7 deliverables.

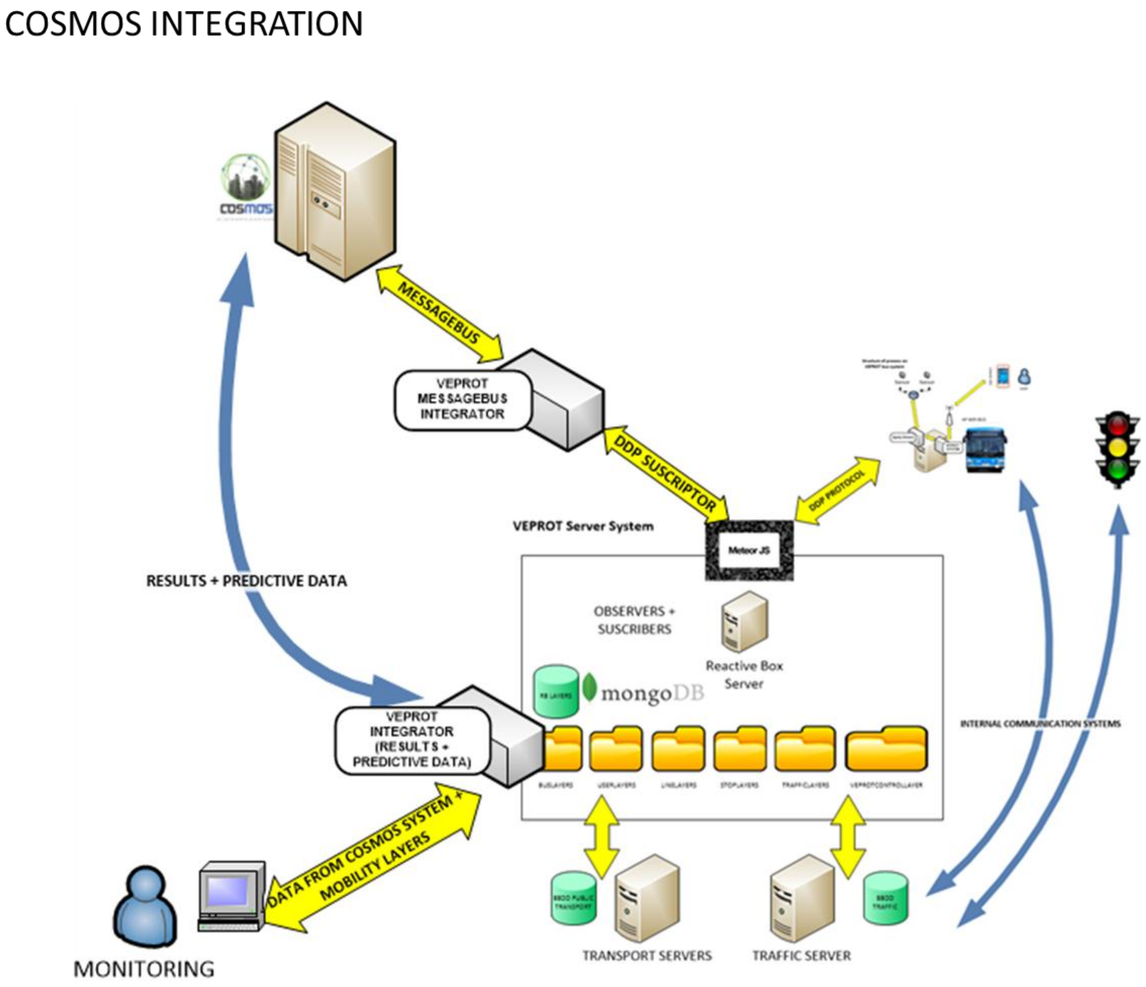


Figure 35: COSMOS Integration (COSMOS APIs and ReactiveBox)

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 20 project months.

Initially, in the 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. In the current version a major revision has been performed in the Methodology section, clarifying concepts and fully aligning COSMOS to the IoT-A ARM. COSMOS components have been grouped in relevant functional groups, extended functional components have been identified to match also the input from Tasks 2.1 Market Analysis and 2.2 Requirements and state of the art analysis, from the architectural point of view. Specifically the User Requirements and Use Cases were examined and the result was their prioritisation and association with specific parts of the platform. An example of the COSMOS Domain model for the Madrid UC is also presented as an example.

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 and the way these extracted capabilities may be adapted to the UCs. In the fully updated Information View, the system use-cases have been highlighted in terms of components and their interactions. The newly introduced Application FG aims at highlighting potential application templates in order to exploit the COSMOS capabilities.

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 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.

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

[17] BSI Threats Catalogue https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Grundschutz/download/threats_catalogue.pdf?__blob=publicationFile

[18] Verizon Data Breach Investigation Report http://www.verizonenterprise.com/DBIR/

[19] Microsoft STRIDE https://msdn.microsoft.com/en-us/library/ee823878%28v=cs.20%29.aspx

[20] Apache Avro https://avro.apache.org/docs/1.7.7/spec.html

[21] Diffie, W.; Hellman, M. (1976). "New directions in cryptography" (PDF). IEEE Transactions on Information Theory 22 (6): 644–654

[22] George Orwell “1984”, ISBN 0-547-24964-0, initially published in 1949