COSMOS

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement Nº 609043

D5.1.3 Decentralized and Autonomous Things Management: Design and Open Specification (Final)

WP5: Decentralized and Autonomous Things Management

Version: 1.0
Due Date: 30 April 2016
Delivery Date: 30 April 2016
Nature: Report
Dissemination Level: PUBLIC

Lead partner: 4 (ICCS)
Authors: Orfefs Voutyras (ICCS), Panagiotis Bourelos ICCS), Juan Rico Fernandez (ATOS), Juan Sancho Murgui (ATOS), Bogdan Sorin Tarnauca (Siemens), Adnan Akbar (UNIS)
Internal reviewers: Francois Carrez (UNIS)

www.iot-COSMOS.eu

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

Version Control:

Version	Date	Author	Author's Organization	Changes
0.1	08/04/2016	Orfefs Voutyras	ICCS	Merged D5.1.1, D5.1.2, D5.2.1 and D5.2.2. First version for internal use.
0.2	11/04/2016	Orfefs Voutyras	ICCS	Revised Section 2 and 3. Added Executive Summary and subsection 3.1.
0.3	13/04/2016	Orfefs Voutyras	ICCS	Revised Section 4.
0.4	18/04/2016	Orfefs Voutyras	ICCS	Revised Section 5, 6 and 7.
0.5	20/04/2016	Orfefs Voutyras	ICCS	Revised Section 7. Added new content in 8.4.
0.6	21/04/2016	Juan Sancho	ATOS	Revised Section 9. Added subsection 9.7.
0.7	21/04/2016	Adnan Akbar	UNIS	Added subsection 8.3.2.5.
0.8	24/04/2016	Panagiotis Bourelos	ICCS	Added subsection 4.5.
0.9	25/04/2016	Orfefs Voutyras	ICCS	Revised Sections 10 and 11. Completed Section 1. Reviewed the document. Version for internal review.
0.10	26/04/2016	Bogdan Tarnauca	SIEMENS	Revised Sections 6 and 7. Added subsection 7.2.
0.11	27/04/2016	Francois Carrez	UNIS	Internal review.
1.0	28/04/2016	Orfefs Voutyras	ICCS	Version for submission.

Table of Contents

List of Figures

List of Tables

Table of Acronyms

Acronym	Meaning
API	Application Programming Interface
APP	Application
CB	Case Base
CBR	Case Based Reasoning
CEP	Complex Event Processing
CIDN	Collaborative Intrusion Detection Networks
CRUD	Create Read Update Delete
D	Deliverable
DIKW	Data Information Knowledge Wisdom
DNA	Dynamic Network Analysis
DoS	Denial of Service
FM	Friends Management
GUI	Graphical User Interface
GVE	Group of Virtual Entities
HTTP	Hypertext Transfer Protocol
ID	Identifier
IoT	Internet of Things
IT	Information Technology
KB	Knowledge Base
KM	Knowledge Management
K-NN	K-Nearest Neighbor
KPI	Key Performance Indicator
LHS	Left-Hand Side
M2M	Machine-to-Machine

MAPE-K	Monitoring Analyzing Planning Executing-Knowledge
MBR	Model Based Reasoning
ML	Machine Learning
MQTT	Message Queue Telemetry Transport
NRA	Network Runtime Adaptability
ORMS	Open Reputation Management Systems
P2P	Peer-to-Peer
PaaS	Platform as a Service
QoS	Quality of Service
RBR	Rule Based Reasoning
REST	Representational State Transfer
RHS	Right-Hand Side
SA	Social Analysis
SIoT	Social Internet of Things
SNA	Social Network Analysis
SPARQL	Simple Protocol and RDF Query Language
T&R	Trust & Reputation
TTL	Time To Live
URI	Uniform Resource Identifier
VANET	Vehicular Ad-hoc Networks
VE	Virtual Entity
VIP	Virtual IP (Internet Protocol address)
VM	Virtual Machine
WP	Work-Package
WSN	Wireless Sensor Network
XML	Extensible Markup Language
XP	Experience

Executive Summary

WP5 is focused on providing component-based methods, frameworks and tools for the development of large-scale autonomous distributed systems monitoring, controlling and managing physical processes. In this specific deliverable (D5.1.3), we present the final version of the Design and Open Specification aspects regarding the Decentralized and Autonomous Things Management model we have developed.

D5.1.3 is the result of the aggregation of the content introduced in all the previous WP5 deliverables (D5.1.1, D5.1.2, D.5.2.1 and D5.2.2) and, as a result, it gives a global and final view of the work done during the whole COSMOS project. In addition to the revision of the pre-existing sections, some **new subsections** were added (subsections 3.1, 4.5, 8.3.2.5, 8.4.4.2, 7.2 9.7) and some **new content** was provided in previous subsections (mainly in subsections 6.1 8.4.3).

The document is structured in such a way that future IoT developers and "challengers" who would like to follow the path of decentralization and autonomicity of IoT systems will find a useful **guideline** across the following pages.

In Section 1, we introduce the main high-level goals of WP5. In Section 2, a short analysis regarding the requirements of the WP takes place. Section 3 elaborates on the main ideas which serve as a basis for our initiative. The next six (6) sections present all the design and technical decisions that have been made, as well as all the details regarding each and every component of WP5. More specifically, in Section 4 we introduce the main tools we provide Things with, in order to make them more knowledgeable and cognitive. In Section 5 we elaborate on the basic decentralized discovery mechanism we have developed. Sections 6 and 7 analyze the ontologies created within the COSMOS project and the main mechanisms for the registration of VEs. In Section 7, we present the social approach that the COSMOS project introduces in order to achieve enhanced services like discovery, recommendation and sharing between Things enriched with social properties. Section 9 discusses the main concepts regarding Network Runtime Adaptability. Section 10 summarizes the services provided by the functional components of WP5 and presents the general architecture. In Section 11, through the description of an application, we attempt to show-case our framework as a whole. Finally, the last section concludes the deliverable.

1. Introduction

The Internet of Things (IoT) is very challenging as it leads to networks connecting a huge number of Things that operate on different administrative domains. The scale and the complexity of the formed networks require new approaches that will make objects able to cooperate in an open and reliable way. Taking into consideration the rate at which IoT devices are deployed and used in different applications, one of the main challenges refers to the efficient and optimized management of these entities. Future internet applications tend to exploit a big number of devices, which highlights the need for **distributed management approaches** given that centralized mechanisms are either non efficient (for a huge number of Things) or not applicable (e.g. due to communication problems). Furthermore, the Things are owned and operated by different administrative domains, thus centralized approaches in many cases cannot be used for their management, given the diversity in access rights. What is more, management decisions usually do not take into account the context under which the Things operate (e.g. a specific Thing may be used with different configuration parameters in different applications). Approaches are required that will allow management decisions to incorporate situational awareness and propose management actions based on them. Finally, an additional challenge with respect to IoT management relates to the **autonomous reasoning** of Things on a context-aware basis. Autonomous management will integrate different types of knowledge (e.g. device-specific, situational, application-specific, administration-related etc.) and trigger decisions accordingly.

In order to face all these challenges, WP5 provides a framework for the **decentralized and autonomous management of Things** based on principles inspired by **social media technologies**. The integration of social networking concepts into IoT systems is a burgeoning topic of research that promises to support novel and more powerful applications.

To sum up, this WP focuses on three different and complementary tasks:

- Network of things management and coordination framework (Task 5.1),
- Autonomous and predictive reasoning of things (Task 5.2) and
- Network of things run-time adaptability (Task 5.3).

Using these tasks as a first analysis step, we can identify the main objectives of COSMOS from the WP5 point of view:

- Providing tools and mechanisms for **decentralized management** of networks of things;
- Extending current management mechanisms by exploiting **new semantic structures** that capture information regarding **objects relationships and communication** with other objects, **importance** within the network of objects and **events** that may affect the network;
- Developing techniques to provide **coordination decisions based on the relationship of objects**;
- Building on the top of the outcomes of Task 6.3 (**Experience sharing**), providing techniques that allow things to **act in an autonomous way** following the experiences of other things with the **same object-related attributes** and the **same situation-dependent attributes**.

In the next section (Section 2), using as guideline the above goals, the most crucial requirements are extracted and divided into several categories, depending on the functional components that address them.

2. Requirements

The requirements relevant to this WP are listed in the Annex 1 of "D2.2.3. SotA Analysis and Requirement Definition (Final)". "COSMOS_Requirements (v3)" is the third and final iteration of the Requirements list which is being checked regularly in order to ensure that no requirement is left behind.

Since many requirements have been identified by studying the results of the IoT-A project [1] which has listed generic requirements called UNIfied requirements (UNIs) that can be reused by specific IoT projects in order to generate their own specific requirements, the ID of these requirements has the form "UNI.#", whereas the ID of the novel requirements of COSMOS WP5 has the form "5.#".

In the following sections it becomes evident that all the identified WP5 requirements have been addressed satisfactorily by the various components of the WP. More specifically, the mapping between the requirements and the corresponding mechanisms is as follows:

- **Section 4:** UNI. 253, 5.7, 5.8, 5.9, UNI. 251, 5.24, 5.25, UNI.714, 5.26, UNI.027, UNI.245, UNI.03, 5.28, UNI.015, UNI.100, UNI.508, 5.29, UNI.010, UNI.704, UNI.706, UNI.708, UNI.715, UNI.719
- **Section 5:** UNI. 422, 5.30, 5.31, UNI.031
- **Section 6/7:** 5.1, UNI.509, 5.3, 5.4, UNI.425, UNI.432, UNI.414, UNI.416, UNI.401, UNI.415, UNI.409, 5.6, UNI.426, UNI. 422, UNI. 406, UNI. 408
- **Section 8:** 5.11, 5.12, UNI. 237, 5.16, UNI.214, 5.16, 5.19, UNI.235
- **Section 9:** 5.13, 5.14, 5.15, 5.17, UNI.701, 5.18, 5.20, 5.21, 5.22, 5.23

3. High-level architecture

In this section, we follow a top-down analysis, describing the steps through which the high-level architecture of WP5 was derived.

3.1. Decentralized vs Centralized Things Management

Generally, the next generation of IoT applications, depending on the trends and techniques they follow, are separated in two main categories: those that follow centralized architectures and those that depend on decentralized ones.

Most emerging IoT PaaS (Platform as a Service) models are generally associated with centralized architectures in which a hub (typically powered by the cloud) controls the execution of nodes (smart devices) and provides a series of back-end services to them. In this architecture, smart devices act as recipients or consumers of data, while the central hub enables centralized services and capabilities. Some of the key centralized capabilities of an IoT platform are: Events Processing, Events Notifications, Real Time Analytics, Devices Discovery, Devices Management, etc. COSMOS does provide such services, as presented in the WP3, WP4 and WP6 technical deliverables.

However, while this model is certainly essential to implement IoT solutions, it may not be sufficient. In addition to integrating back-end services, there are many scenarios that require autonomous communication between smart devices in an IoT topology without the need of a central hub. In other words, decentralized computing models in which nodes in IoT systems interact without the control of a central authority are also fundamental to enable the next generation of IoT enterprise solutions. From the functional standpoint, decentralized IoT models can help to enable some of the following capabilities: Peer-to-Peer Messaging, Decentralized Auditing, Decentralized File Sharing, etc.

Implementing decentralized IoT capabilities requires an infrastructure that enables nodes in a distributed topology to perform **autonomously**. With autonomous computing comes the need to maintain trusted relationship between the nodes without a centralized authority.

3.2. Autonomous Things Management

One of the main goals of WP5 is achieving a satisfactory level of **autonomicity** of the system and its components. Autonomicity is the art of self-managing, given a set of high-level objectives from administrators. These objectives define for the system goals that it attempts to accomplish in the best manner.

3.2.1. Autonomic Computing Framework

Autonomic computing was first coined by IBM in 2001 [2]. The term refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity to operators and users. **Self-management** is central to automatic computing encompassing various self-* aspects, including self-configuring, self-healing, self-optimizing and self-protecting. Autonomic Computing is a comprehensive approach to build more automated IT infrastructures that require minimal intervention.

In 2002, IBM announced a new autonomic deployment model [3] that outlines a staged approach for helping customers chart a course toward establishing an autonomic IT environment. This model can be used as a guide to developing an autonomic infrastructure, in other words, it can be used as a staged approach to plan and measure the progress towards the autonomy of the systems developed. This high-level framing is essential to mapping the progression of true autonomic functionality and to help information systems work together in a way to better manage themselves.

This autonomic deployment model defines five levels of increasingly sophisticated self-governance of systems. These levels are:

i. **Basic:** It represents the starting point where a significant number of IT systems are today. Each element of the system is managed independently by systems administrators who set it up, monitor it and enhance it as needed.
ii. **Managed:** It uses systems management technologies to collect information from disparate systems into one, consolidated view, reducing the time it takes for the administrator to collect and synthesize information.
iii. **Predictive:** It introduces new technologies that provide correlation among several elements of the system. The system itself can begin to recognize patterns, predict the optimal configuration and provide advice on what course of action the administrator should take.
iv. **Adaptive:** It is reached when systems can not only provide advice on actions, but can also take the right actions automatically, based on the information that is available to them on what is happening in the system.
v. **Full Autonomic:** It is attained when the system operation is governed by business policies and objectives. Users interact with the system to monitor the business processes and/or alter the objectives.

> From WP5 point of view, the lower level that had been set as a goal was the **Predictive level**. Since this level was covered satisfactorily from the technologies presented in the next sections, steps have been taken towards the **Adaptive level**. The development of a Full Autonomic system was out of the scope of COSMOS, considering that this level is extremely demanding and other first-class goals, such as the adaptation of social technologies, had priority.

3.2.2. Self-management attributes of system components

In a self-managing autonomic environment, system components can include embedded control loop functionalities [4], [5], [6], [7], [8], [9], [10]. Although these control loops consist of the same fundamental parts, their functions can be divided into four broad embedded control loop categories. These categories are considered to be attributes of the system components and are defined as:

i. **Self-configuring:** Self-configuring components adapt dynamically to changes of the environment, using policies provided by IT professionals. Such changes could include the deployment of new components or the removal of existing ones or dramatic changes in the system characteristics. Dynamic adaptation can ensure the continuous strength and productivity of the IT infrastructure, resulting in business growth and flexibility.

Indicatively, the system must adjust the logical topology of the network to improve its reliability and scalability.

ii. **Self-optimizing:** Self-optimizing components can monitor and tune themselves automatically to meet end-user or business needs. The tuning actions could mean reallocating resources—such as in response to dynamically changing workloads—to improve overall utilization, or ensuring that particular business transactions can be completed in a timely fashion. Without self-optimizing functions, there is no easy way to divert excess server capacity to lower priority work when an application does not fully use its assigned computing resources. The system can measure its current performance and is able to compare it with the known optimum level of performance.

iii. **Self-healing:** Self-healing components can detect system malfunctions and initiate policy-based corrective actions without disrupting the IT environment. A corrective action could involve an entity altering its own state or initiating changes in other components. Self-healing can be implemented in two different ways (reactive and proactive). In the reactive approach, the system detects and recovers from faults as they occur and tries to repair the faulted functions if possible. In the proactive approach, the system monitors its state to detect and adjust its behavior before reaching an undesired state.

iv. **Self-protecting:** Self-protecting components can detect hostile behaviors as they occur and take corrective actions to minimize vulnerability. The system defends itself against internal and external threats, which can be either accidental events, such as cascading failures, or malicious attacks against the system.

When system components have these attributes, it is possible to automate the tasks that IT professionals must perform today to configure, heal, optimize and protect the IT infrastructure. Systems management software then can orchestrate the system-wide actions performed by these embedded control loops.

> COSMOS aims at providing developers and users with tools to enable them maintain such self-* aspects for their systems. As it is presented in Section 4, this requirement is covered at a satisfactory level through the use of the COSMOS **Case-Based Reasoning Planner**.

3.2.3. The MAPE-K approach

For a system component to be self-managing, it must have an automated method to collect the data it needs, to analyze those data in order to determine whether something needs to be changed or not, to create a plan or a sequence of actions that specifies the necessary changes and finally to perform those actions. When these functions can be automated, an **intelligent control loop** is formed. This definition describes the **Monitor-Analyze-Plan-Execute** functional composition of an autonomic manager. These key functions are well known and sometimes referred to as a **MAPE loop** [11].

In addition to the MAPE functions, an autonomic manager also contains a Knowledge block that is connected to all four of the MAPE functional blocks, producing a **MAPE-K control loop**. We can have autonomic managers for multiple resources arranged in a hierarchical fashion. In this case, lower level managers deal with resources at smaller granularity and/or smaller locality, while a Top-level autonomic manager deals with decision making and policies.

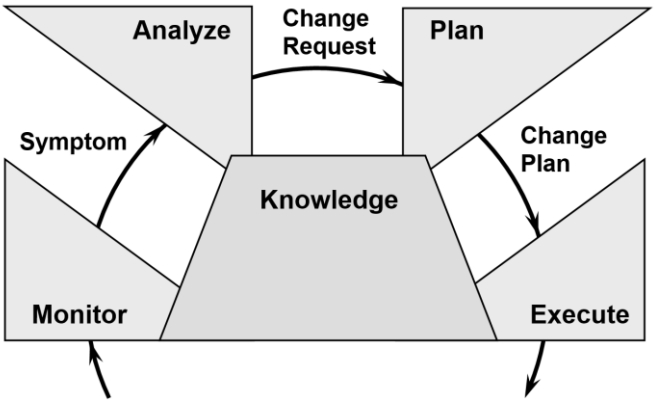

Figure 1: A typical MAPE-K loop.

The components of a typical MAPE-K loop and their functionality are further analyzed in the following paragraphs:

i. **Monitor:** The monitor component is in charge of capturing necessary measurements of the environment that are of significance to the self-properties of the underlying network. It collects the details from the managed resources via touchpoints and correlates them into symptoms that can be analyzed. The details can include topology information, metrics, configuration property settings, information about the managed resource configuration, status, offered capacity and throughput. The monitor component aggregates, correlates and filters these details until it determines a symptom that needs to be analyzed. Logically, this symptom is passed to the analyze component.
ii. **Analyze:** The analyze component provides the mechanisms to observe and analyze situations to determine if some change needs to be made. For example, the requirement to enact a change may occur when the analyze function determines that some policy is not being met. Autonomic managers must be able to perform complex data analysis and reasoning on the symptoms provided by the monitor function. If changes are required, the analyze component generates a change request and logically passes that change request to the plan function. The change request describes the modifications that the analyze component deems necessary or desirable.
iii. **Plan:** The plan component structures the actions needed to achieve goals and objectives. It creates or selects a procedure to enact a desired alteration in the managed resource which can take on many forms, ranging from a single command to a complex workflow. A plan component constructs adaptation plans based on the result of the analysis process and generates the appropriate change plan, which represents a desired set of changes for the managed resource and logically passes that change plan to the execute function.
iv. **Execute:** The execute component provides the mechanism to schedule and perform the necessary changes to the system. Once an autonomic manager has generated a change plan that corresponds to a change request, some actions may need to be taken to modify the state of one or more managed resources. The execute component of an autonomic manager is responsible for carrying out the procedure that was generated by the plan function of the autonomic manager through a series of actions. Part of the execution of the change plan could involve updating the knowledge that is used by the autonomic manager.
v. **Knowledge:** A knowledge base is the place where aggregated collected data are stored. The aggregated data (Knowledge) are shared among the monitor, analyze, plan and execute functions. The shared knowledge includes data such as topology information, historical logs, metrics, symptoms and policies. This repository serves for initial configuration of the network and guides the operation of other MAPE components.

3.2.4. The COSMOS MAPE-K model

Although in COSMOS we have adopted the MAPE-K analysis, we follow a slightly more complex approach, dictated by the social view that we introduce and the several different tools and new technologies we provide. This model consists of the following components:

i. **Monitoring (M):**
   a. **Environment Monitoring:** Related to the monitoring of the physical world and the surroundings of the physical entities, this functionality is strongly use-case dependent and is implemented indirectly through the applications and the IoT-services created by the VE developers.
   b. **System Monitoring:** Related to the monitoring of the devices characteristics and constraints such as energy and memory consumption. This functionality is presented in Section 9.
   c. **Social Monitoring:** This component contains all the main tools and techniques that are for the monitoring of the social properties of the VEs that COSMOS adds to their description. Social Monitoring forwards its data to the Social Analysis component. Social Monitoring is further discussed in Section 8.1.

ii. **Analysis (A):**
   a. **Environment Analysis:** This component contains many functionalities further explained in WP4 and WP6. Such functionalities are events identification, context awareness, pre-processing mechanisms, models extraction, implemented through COSMOS CEP engine and Machine Learning techniques. This component is not analyzed extensively in this deliverable.
   b. **System Analysis:** Building on top of System Monitoring and using the same tools with Environment Analysis, more complex events are identified. This functionality is presented in Section 9.
   c. **Social Analysis (SA):** This component is used for the extraction of complex social characteristics of the VEs, as well as, models and patterns regarding the behavior of the VEs and the relations between them. Its functionalities are presented in Section 8.2.

iii. **Planning (P):** At a first level, the main functionality of the planning component (from now on "Planner") is to match topics of the analysis components to certain actions of the VEs, implemented through IoT-services. Depending on how "smart" we decide that a VE has to be, the Planner evolves accordingly. For example, during Y3, the Planner was designed to also provide recommendation services. The Planner is further analyzed in Section 4.

iv. **Executor (E):** This component, based on the output of the planning process, executes certain actions such as IoT-services calls, M2M communication, **XP-sharing** etc. and sends feedback to the Social Monitoring component (and to the Monitoring component if the developers have created through their applications such a loop). In other words, the executor is an abstract component whose functionalities are shared between other tools used or provided by COSMOS (see subsection 4.5).

v. **Knowledge (K):** The knowledge base will contain all the data and objectives of the VEs provided by both the users and COSMOS. This component is further described in Section 4.

From the above analysis it becomes evident that through the provided technologies, Things can become more **Situational-aware** (Monitoring and Analysis), **Cognitive** (Planning and Knowledge) and **Social** (Social Monitoring, Social Analysis, XP-sharing).

4. Knowledgeable and Cognitive Virtual Entities

4.1. The concept of Knowledge and Cognition in COSMOS

The IoT will create a flood of real world information to the virtual world. Future applications will be considerably enriched, as they will be more and more aware of what happens in the real world, in real time, everywhere. With a trillion sensors [12] embedded in the environment, all connected by computing systems, software and services, the future IoT platforms have to deliver data and information management mechanisms to handle the exponentially increasing "born digital data". The transformation of this huge amount of raw data into knowledge is one of the biggest challenges behind the IoT. There is an entire cycle of data processing up to the generation of cooperative knowledge networks. These knowledge networks can feed complex hierarchical feedback control loops, since sensorial data is very important for decision making. Decisions made on the virtual side can be reflected on the real environment helping us to better use our resources. Hence, a first step to designing the general architecture of a project on the IoT domain and realizing its capabilities and chances for evolution is the definition of its own **Knowledge Management (KM) cycle**. Knowledge management is the process of capturing, developing, sharing and effectively using knowledge and summarizes all activities with the goal of using knowledge in a more efficient and effective manner, achieving certain objectives. A **Knowledge Pyramid**, the **DIKW Pyramid** [13], is usually used for the representation of purported structural and/or functional relationships between **data (D)**, **information (I)**, **knowledge (K)** and **wisdom (W)**. Generally, when we take data and put it in context we have information, when information becomes actionable it is transformed into knowledge and when pieces of knowledge are consolidated, with the help of experience, wisdom is born.

In COSMOS DIKW Pyramid:

- Data are the raw-data which are collected from the VEs through their IoT-services that expose their real-world services. Physical objects like buses or houses which are represented by VEs will have a huge number of embedded sensors, continuously "feeding" COSMOS with data regarding the temperature and humidity of the environment, the velocity of the buses etc.
- Information is the result produced by analyzing the raw-data. Suitable mechanisms make possible the detection of simple or complex events of the physical world around the VEs. For example, analyzing the data offered by the sensors of the buses or the houses, the detection of events like "fire" or "traffic" becomes possible.
- Knowledge includes problems or situations detected (e.g. "fire") associated with specific solutions, implemented through IoT-services. In other words, Knowledge includes directions that specify how the VEs are going to react in changes of their environment in a well-defined way. For example, a house may include in its **Knowledge Base (KB)** the scenario of the problem "fire" and "know" that the solution to the problem is "inform the fire department". Knowledge is a store of information proven useful for a capacity to act. This level gives the VEs the advantage of learning from previous experiences.
- Finally, Wisdom is born using high-level reasoning techniques, such as **Case Based Reasoning (CBR)** [14] and Rule Based Reasoning [15], which give to the VEs the ability to reason and understand their situation and take decisions on their own, thus producing Knowledge on their own. Things attaining this level could be characterized as cognitive, intelligent or wise, as they have the capacity to acquire, adapt, modify, extend and use knowledge in order to solve problems.

For a Thing to be autonomous and be able to utilize a full DIKW pyramid, its corresponding Virtual Entity should be equipped with:

i. its **own Knowledge Base**,
ii. a **Reasoner** (Planner) using one of the many reasoning techniques and
iii. a **Cognition Loop**, a cognitive process that can perceive its current conditions, plan, decide, act on those conditions and learn from the consequences of the actions, all while following end-to-end goals.

The issue of the knowledge representation is analyzed in the next subsection, while the reasoning technique we have adopted is presented in subsection 4.3 and 4.4. Finally, the way we manage to form a complete cognition loop is discussed in subsection 4.5.

4.2. The concept of Cases

The proposed approach provides the VEs with the advantage of learning from previous experiences. Experience is usable knowledge acquired through the use of collaborating communication techniques between two or more individuals. Different types of experiences are defined, arising from the correlated phases of our control loop approach, which is adopted for the implementation of the project regarding VEs' management. Experience can be a piece of knowledge described by an ontology, a model resulting from Machine Learning or contextual information. However, we focus mainly on the representation of experience through **Cases** as defined in the CBR technique. The reason the CBR technique was chosen among other reasoning techniques is further explained in subsection 4.3.1.

A case can be considered as a combination of a **Problem** with its **Solution**, whereas a problem consists of one or more **Events** or **Goals** (State Vectors). In other words, a case is a kind of rule for an actuation plan, which is triggered when specific events are identified [16].

Each VE may maintain its own **Case Base (CB)** locally as part of its KB. Storage of experience in a local or central KB [17] depends on whether the individual's knowledge needs are constant or opportunistic. Such a categorization of needs is primarily based on the "domain" membership of individual VEs as well as technical limitations that may be present. A KB can be shared between VEs with suitable social characteristics, something that improves the decision making mechanisms. Moreover, VEs representing weak devices that do not have their own KB can take advantage of the KB of their social group.

4.2.1. Description of Problems

Taking under consideration that the State Vectors constituting the Problems may describe current or future situations that a VE has to face, there are basically two different categories of problems:

- **Event driven:** The Problems consist of Events that have to be identified (e.g. by a local mini-CEP engine or the central COSMOS CEP engine) to trigger the Solution. This description of Events may for example be linked with the corresponding Topics of the COSMOS Message Bus. The problem can be simple (one Event) or complex (a list or sequence of Events).
- **Goal driven:** The Problems consist of Events describing the current situation of the VE as well as Goals that describe a desired deviation from the current state, in other words, a desired future situation. Yet again, the Solution is triggered by the Events. However, in this case, the Solution can be evaluated in an autonomous way (and not by human-users necessarily) as the expected/desired final state can be used by a feedback loop.

4.2.2. Description of Solutions

Generally, a Solution is a list or sequence of actions (actuations) to be undertaken given an initial state in order to reach a final state where the Problem has "disappeared" (Events) or has been solved (Events and Goals). Moreover, the Solution could contain some extra information regarding the actions undertaken (e.g. the description of the final state in case we study an Event driven Problem and not a Goal driven one).

In the case of COSMOS, in its simplest form, a Solution can be the **URI of an IoT-service** which provides some kind of actuation. Of course, more advanced solutions should contain some "logic" like algorithms. A Solution can be primitive (1 task- IoT-service) or complex (list or sequence of IoT-services). Three types of solutions have been identified:

a) **Type 1** - Solutions that are just a **recommendation/notification service**, a **message**: For example, the Solution to the Problem *"fire"* (detected as a complex Event by the CEP) at a VE-house could be just a message sent to the user, saying "The roof, the roof, the roof is on fire! Call the fire department!". This Solution can be shared between similar VEs facing similar Problems, without having to be changed. Moreover, since there is no kind of actuation regarding the physical world (we don't open/close doors/windows) these scenarios are the safest.

- **Type 2** - Concrete Solutions that consist of **IoT-services from 3rd party VEs**: In the same scenario, the VE-house facing the Problem *"fire"* will actuate the IoT-service (through its URI) *"send request to the VE-fire department"*. This solution can yet again be shared between similar VEs (similar means that they have close geographic location too) with no serious issues. A VE2-house that will take this Solution will inform the same VE-fire department by using the very same URI.
- **Type 3** - Generic Solutions that consist of **IoT-services provided by the VE itself or 3rd party VEs and need some logic**: In this case, if the VE1-house uses as a Solution a URI that represents the IoT-service *"open MY (of VE1) doors"*, it is evident that this Solution cannot be shared with other VEs for this scenario. Obviously, a VE2-house has to use the IoT-service *"open MY (of VE2) doors"* and it should not try to open the doors of VE1. This means that the **structure/logic** of the Solution has to be shared in this case. As a result, the **generic description of IoT-services** becomes necessary.

Regarding the 3rd type of Solutions, this kind of generic description of IoT-services should be defined by the application itself (like the rest types of solutions). In this case, the **Execution/Actuation level** would consist of replacing the generic fields ("I, VE-X, need an IoT-service that is mine and has these characteristics and three IoT-services from other VEs that have these characteristics") with concrete ones. The development of the semantic description of the Applications in parallel is necessary.

A scenario were VEs share between them Solutions of Type 1 and 2 has already been demonstrated during Year 1. Achieving to demonstrate a scenario where generic Solutions are shared and used by VEs is one of our main future goals.

4.2.3. Cases Representation

In order to use the previous experiences in the CBR cognition loop known as CBR cycle (Section 4.3.2), Cases must be represented in a structural manner. Several methods of representation can be used in case-based reasoning and the choice of a representation method depends on the domain that the system is modeling and the types of similarity assessments and retrieval which are chosen according to the requirements of the system.

The simplest format to represent the cases in the case base is to have simple feature-value vectors which are good for cases with attributes of nominal or numeric values. Some techniques of structural-based representation [18] are:

- **Object-oriented based:** One way to represent cases is in the form of objects where each of the attributes can be of simple type (e.g. "integer" or "string") or of type "object". This forms a hierarchy of the object structure within which cases in the same classes of the hierarchy can be compared.
- **Spreading activation method:** In this method, the CB is organized as an interconnected network of nodes which makes the case attribute value combinations. The network representing the CB consists of i. feature-value nodes and case nodes which are interconnected and ii. edges with weights that show the relevance of two different nodes.
- **Generalized cases:** Generalized cases can be viewed as the implicit representation of a set of closely related point cases.
- **Graph-oriented:** Graphs are commonly used for representing complex domains like planning and design. These graphs can be attribute graphs, semantic nets or conceptual graphs.
- **Ontology-based:** Ontologies can be used to make the case base where the cases are the instances of the ontology [19].

COSMOS uses ontologies for the description of many entities like the VEs and the IoT-services. Consequently, we chose the same type of representation for the Cases, thus adopting the **ontology-based representation of the CBR**.

Ontologies provide a rich vocabulary for the general domain knowledge enabling the users or application developers to better express their requirements and submit queries, leading to greater precision and recall rates. Moreover, formalization of ontologies improves retrieval and similarity adaptation and learning, concepts that will be discussed later. That is why the ontology-based representation has been chosen and used.

Since we have analyzed the concept of Cases and the method of their representation, we can provide an example of the structure of a Case. Let's take a scenario where the owner of a flat, while away from it, wants to set its internal temperature to a certain degree, before he/she arrives to it. Thus, he/she notifies the corresponding VE-flat by using a COSMOS-enabled Application and provides as input i. the desired temperature and ii. the estimated time of arrival to the flat. This is a problem regarding energy management where we want the desired temperature to be achieved right before the owner arrives to the flat. The Cases that will be produced and used by the Application can have as a Problem the inputs of the user and the current situation of the house (room temperature before) and as a Solution the URI of the IoT-service that corresponds to the actuation of the boiler and the required input. In the Case depicted at Figure 2, the boiler should be set to 32 °C.

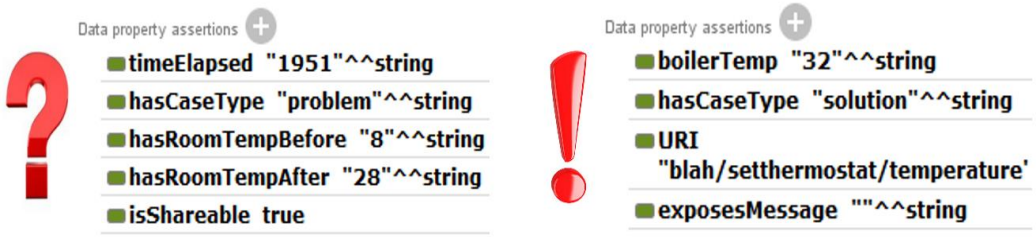

Figure 2: Example of the semantic description of a Case with the problem statement.

4.2.4. Case Memory Models

CBR is one of the tools that COSMOS provides to enable VEs run Applications. Whenever a VE "downloads" a COSMOS-enabled Application, a set of initial Cases is provided by it and stored in the CB of the VE. While the VE is facing new Problems related to this Application, new Cases are created as it will be described later.

The CB should be organized in a manageable structure that can support efficient search and retrieval methods. Several groups of case memory models have been proposed [20], [21], [22] like the:

- **Flat Memory model:** In a flat memory model, all the cases are organized at the same level. Retrieval time in this memory organization is very high since for each and every retrieval all the cases in the case base must be compared to the target case. Thus, this method is unacceptable for large case bases. However, advantages of this approach which include maximum accuracy and easy retention have led to its use in many applications.
- **Hierarchy or Shared-Feature Network Memory model:** Cases in this model are structured in the format of a three layer network. The first layer nodes are the feature- values, the second layer contains the problems and the third layer is the solution layer. Separating problems from solutions makes it possible for different problems to share a solution and for a problem to have alternative solutions. The connections between the first layer and the second one show the feature-values for each problem. The weights of the second set of connections represent how important a solution is for a problem case, if it is a potential candidate solution. The restriction for this kind of memory model is that it assumes simple nominal and discrete numeric attributes and it cannot cover complex and continuous attributes.
- **Network Based Memory Models:**
  - ✓ **Category exemplar model**: The case memory in this model is a network structure of categories, semantic relations, cases and index pointers. This organization has three types of indices: feature links which point from problem features to a case or a category, case links that point from a category to its cases and difference links which point from categories to the neighbor cases where the differences to the current category are small. In this organization, the categories are interlinked within a semantic network that represents a background of general domain knowledge which supports having explanation for some CBR tasks.
  - ✓ **Case retrieval nets:** The fundamental item in case retrieval nets is the information entity (IE). IEs represent any basic knowledge item, such as attribute-value pairs. A case consists of these IEs and the case base is a net with nodes for the IEs in the domain and additional nodes which are for cases. IE nodes may be connected by similarity arcs and relevant arcs connect the case nodes to the IE nodes which make the case. Construction of this organization is expensive, but these nets can handle partially specified queries.

Since the CBR technique has been designed to run locally on the devices level, case memory model we adopt is the **Flat Memory model**. However, the organization of the Case Base can evolve. For example, a very good candidate for later use in bigger, central CBs is the **Category Exemplar model**.

4.3. Reasoning Technique: Case Based Reasoning

4.3.1. Selecting a Reasoning Approach

The main reasoning approaches in our domain are model-based reasoning (MBR), rule-based reasoning (RBR) and case-based reasoning (CBR) [16], [23], [24], [25], [26], [27] [28], [29]. Many hybrid models can be developed by combining these approaches. Each approach has advantages and disadvantages, which are proved to be complementary in a large degree [30]:

- MBR [31] is an approach in which general knowledge is represented by formalizing the mathematical or physical relationships present in a problem domain. With this approach, the main focus of application development is developing the model, something that is time consuming, as it is necessary to make the model as deep, complex and detailed as possible to achieve the best results. Once a working model has been established, it may also require periodic updates. Then at run time, an "engine" combines this model knowledge with observed data to derive conclusions such as a diagnosis or a prediction. In other words, model-based reasoning is the use of a working model and accompanying real-world observations to draw conclusions. In an example of MBR derived from our use-cases, a bus company could develop a working mechanical and electrical model of its buses. Data about “symptoms” of malfunctions could be built into the system, using observations to create a matrix of known information. A user could potentially interact with the model by inputting symptoms (e.g. excessively high temperature of the engine) and the model would return a potential diagnosis (e.g. broken fan clutch).
- A rule-based system [23] may be viewed as consisting of three basic components: a set of rules (rule base), a data base (fact base) and an interpreter for the rules. In the simplest design, a rule is an ordered pair of symbol strings with a left-hand side and a right-hand side (LHS and RHS). The rule set has a predetermined, total ordering and the data base is simply a collection of symbols. The interpreter in this simple design operates by scanning the LHS of each rule until one is found that can be successfully matched against the data base. At that point, the symbols matched in the data base are replaced with those found in the RHS of the rule and scanning either continues with the next rule or begins again with the first. A rule can also be viewed as a simple conditional statement and the invocation of rules as a sequence of actions chained by modus ponens. In that sense, we can understand that rules usually represent general knowledge.
- Case-Based Reasoning [14] is the process of solving problems based on past experience. In more detail, it tries to solve a case (a formatted instance of a problem) by looking for similar cases from the past and reusing the solutions of these cases to solve the current one. Cases encompass knowledge accumulated from specific (specialized) situations. The advantages of case-based reasoning include the ability to encode historical knowledge directly, to provide shortcuts in reasoning and to make predictions about how much the suggested solution may help. An extensive analysis of domain knowledge is not required and CBR systems can learn over time.

By studying carefully the several reasoning techniques, **Case Based Reasoning** was chosen as the most appropriate reasoning approach for the case of COSMOS.

Indicatively, the CBR reflects human reasoning and is the easiest approach for the developers and the one that can be used in a huge variety of domains. Moreover, the concept of adapting past solutions is one of the main requirements of COSMOS. In that sense, the cases produced via CBR become part of the Experience of the VEs, which can not only aid to the planning of future solutions, but can also be shared between different VEs. One of the main disadvantages of CBR (store/compute trade-offs because of large case bases) is tackled, as most of the VEs are going to have their own light-weight case base, as it is going to be described later.

The functional component that enables the VEs to use CBR is the **Planner** (a Reasoner), a component that becomes part of the VEs during their registration time and runs locally, on the devices. Its functionalities are further described in subsection 4.4.

Although CBR is a useful technique for solving a wide range of problems, there are occasions that it is not the most appropriate methodology to employ. The following questions can be used to determine whether case-based reasoning is an applicable technique to solve a problem or not:

- **Are there exceptions and novel cases?** Domains without novel or exceptional cases may be modelled better with RBR or MBR. On the other hand, in a situation where new experiences and exceptions are encountered frequently, it would be difficult to maintain consistency among the rules and the models in the system, so CBR becomes more appropriate.
- **Do Cases recur?** If the similarities between cases are very low, then the experience gained may not help to solve a new problem.
- **Is there significant benefit in adapting past solutions?** We have to consider the significance of our benefit from making adaptation to old solutions compared to the benefit from creating a new solution from scratch.
- **Can we record data that have the necessary and relevant characteristics of past cases?** Is the solution recorded in sufficient structure with ample detail so it can guide to clear suggestions and be adapted in the future to provide better results?

4.3.2. The CBR cycle

In 1994, Aamodt and Plaza [14] proposed a life cycle for CBR systems to make evident the knowledge engineering effort in CBR. This cycle is used by other CBR researchers as a framework and consists of four main parts; **retrieve, reuse, revise** and **retain**. Each of these parts includes a set of tasks and different methods have been proposed for each of them. A brief presentation of the four phases of the CBR cycle follows:

1. **Retrieve**: Retrieve the most similar Case or Cases to the new one, in other words, given a target Problem, retrieve from the memory Cases relevant to solving it.
2. **Reuse**: Reuse the retrieved Cases by copying them completely or by integrating the Solutions of the Cases retrieved.
3. **Revise**: Revise or adapt the Solution of the Cases retrieved trying to solve the new problem. Having mapped the previous Solution to the target situation, test the new Solution in the real world (or a simulation) and, if necessary, revise.
4. **Retain**: Retain the new Solution once it has been proven that it is correct and brings successful results. Store new Case and found Solution into CB.

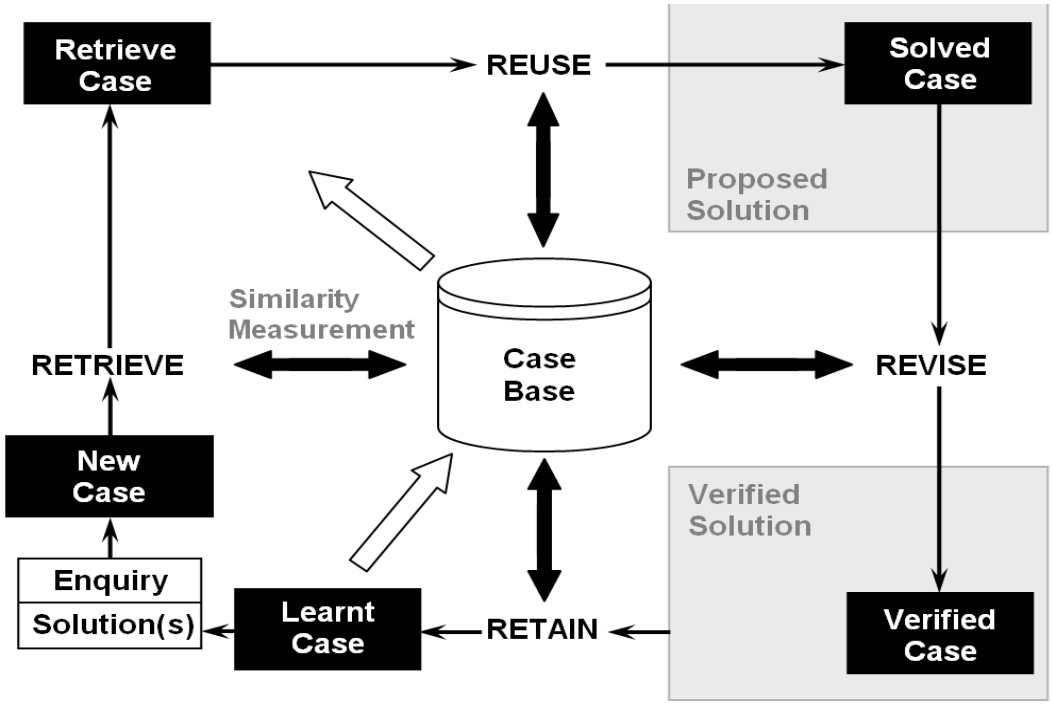

Figure 3: The CBR cycle.

4.3.3. The stage of Retrieval

CBR is based on finding Solutions to new Problems by reusing older Cases. As a result, the only component needed besides the CB in order to achieve CBR at its simplest form is a reasoner (the Planner) that can identify **similarities** between Cases. This step represents the first form of the Planner. In other words, at the early stages of the project, the Planner acted as a simple **Retriever**, matching the various problems to their solutions and identifying similar problems to a problem given, recommending in parallel the "best" solutions (in the sense of similarity between the problems and the reputation, trust and other social characteristics of the VEs sharing their Experience or offering their IoT-services).

The retrieve step is the key step in CBR because the quality of the adaptation depends on the quality of the retrieval. For retrieval of cases from the case base, different retrieval techniques have been proposed [21], like the similarity based retrieval, adaptive guided retrieval and ontology based retrieval. Factors that play major roles in determining the performance of a CBR system are the complexity and the accuracy of the case retrieval phase.

Although for most of the retrieval methods the type of application and needs of that application are the criteria to select a retrieval method, when more than one option for retrieval is available, the following aspects can help to decide:

- Efficiency of the method in both the speed and the efforts for searching in the case-base.
- Quality of the solution with measures like precision, overall length of dialog with user and methods of dealing with problems like noise, missing values or cases with different attributes.

Every **retrieval method** is a combination of:

a) a **similarity assessment procedure**, which determines the similarity between a target Case and a Case in the CB and
b) a **procedure for searching the case memory** to find the most similar Cases.

A retrieval method used by the COSMOS Planner has been defined:

a) Regarding the similarity assessment procedure, in order to measure similarity between Cases, we follow two steps:
   - ✓ First of all, the similarity of the attributes of the new Case (that consists of a Problem and no Solution) with the various Cases in the CB has to be measured. The method used takes as input the names of the parameters of a new incomplete Case and the percentage of acceptable similarity. The similarity is calculated by using the **Jaccard similarity coefficient** [32].
   - ✓ If there are Cases in the CB that have sufficiently similar attributes to those of the new Case, the Planner class is calculating a new similarity index in order to find the most appropriate Case. This method takes as input a list of the names of the attributes describing the new Case, a list of the values of the attributes, a list of the weights of these attributes (if any) and a new similarity threshold regarding the values of the attributes and returns the most appropriate Solution. In this case, the dissimilarity is calculated by using the **Bray-Curtis distance** [33].

b) Regarding the procedure for searching the case memory, the retrieval technique highly depends on the CBR knowledge representation and Case Memory model we have chosen. Since we are going to adopt the Flat Memory model, we chose the **simple flat memory k-Nearest Neighbor retrieval (K-NN)** [34]. In this approach, the assessment of similarity is based on a weighted sum of features. The problem with this method is the retrieval time, so this method is not suitable for large case bases. However, this is not a problem for COSMOS (subsection 5.4), since the cases are going to be distributed (each VE may have its own CB) and, as a result, the CBs will not be large.

It should be noted that when the Category Exemplar model is used as a Case Memory model, the retrieval method changes. Since each new Case is the result of a specific Application (e.g. App 1), it is compared only with the Cases that belong to the category “App 1”. That means that the first step of the similarity assessment procedure becomes unnecessary, lowering the retrieval times, and the procedure for searching the case memory changes.

4.3.4. Evaluation of Cases and Feedback loops

In a CBR system, learning is done in the retention step. In this step, the new Case will be added to the CB according to some policies in the system. Retention includes adding knowledge and new Cases to the CB, which need to be indexed, as well as deleting cases from the CB in order to restrict its growth. In order to choose whether a new Case should be stored or not, its evaluation is necessary, and evaluation can take place if **feedback loops** are available.

There are four main issues that have to be discussed in order to define a feedback loop:

- **Who** will give the feedback? The feedback may come from **a)** the **system** itself through an autonomic loop (e.g. by comparing the initial Problem with the final state) or **b) human-users**. In the first case, the Executor should trigger the Monitoring component and the corresponding logic should be added to the Application.
- **When** will the evaluation take place? Feedback may be given **a) before** or **b) after** the execution of the Solution. Evaluation from the system can take place after the actuations of course. Evaluation from human-users can take place both before and after the execution of the Solution. It should be noted that in many scenarios where the execution of the Solution will result in the usage of actuators, for safety reasons, feedback may have to be given by human-users before this execution.

- **What** will be evaluated? There are some factors that can be evaluated only by the system and other factors that can be evaluated only by human-users. For example, the exact temperature of a heated room can be evaluated according to a schedule by the system, but the comfort that a human feels inside the room can be evaluated only by human-users.
- **How** will the evaluation take place?
  - ✓ For feedback given by human-users, the **five-star rating system** can be used.
  - ✓ For feedback coming from the system, two mechanisms can be used depending on the nature of the Problem. If the Problem is **event-driven**, then the Solution is considered successful when the corresponding Event is no longer detected (e.g. Event "fire"). If the Problem is **goal-driven**, then the Solution is considered successful when the corresponding Goal (desired situation) is reached or unsuccessful if the desired situation has not been reached after a predetermined amount of time.

4.4. Functionalities of the Planner

The main functionality of a planner is "solving" problems. As such, its nature is depended on the reasoning approach that we follow. As it has been mentioned in the previous analysis, we have developed an **ontology-based CBR planner** and adopted the **Flat Memory** and the **Category Exemplar model**. When the planner "senses" a situation or accepts a query, it means that there is a new problem to solve. The planner creates a target case and compares it with the cases available in the CB. If the Planner does not find a case similar enough to the target case, the VE can ask other VEs for assistance (Section 5), thus experience sharing and communication between autonomic managers takes place. The choice of the solution is made by taking under consideration social criteria too, such as the reputation of the VEs that offer the solutions.

As it is stated in [22], **CBR is a methodology, not a technology**. As a result, the Planner can offer various functionalities.

4.4.1. Retrieval modes

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. The developers should also be able to define the logic of their Applications, the corresponding feedback loops needed, how the Cases are going to be stored in the CBs, etc.

Each VE will have its own Planner and CB and will be able to facilitate M2M communication. The Planner of the VEs acts as part of a MAPE-K loop (hence in an autonomous way) as well as "manually", in other words, on demand. Consequently, the Planner has **two retrieval modes**:

- 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 other components, the corresponding solution is forwarded to the Executor.
- 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 CB or the CBs of other VEs, choose the most appropriate Solution and create new Solutions (e.g. services composition).

Both of these modes were demonstrated during the first review of the project by the two demos presented by WP5. In the first demo, the Planner of a VE-house was following Topics of the COSMOS Message Bus regarding sensor malfunctions and was reacting to them, while in the second demo, the Planner was reasoning on Problems defined by the end-user who wanted to set the temperature of his/her house at a specific degree after a specific amount of time (see Figure 2).

4.4.2. Levels of self-governance

CBR is a simple and flexible yet powerful reasoning technique when it is used at the right domains. As such, it can be used for many purposes. Moreover, the persistent nature of the ontologies used enables proactiveness and robustness to 'ignorable events' while their unitary nature enables end-to-end adaptations. These characteristics enable the VEs to achieve high levels of self-governance. As stated in subsection 3.2.1, the autonomic deployment model proposed by IBM defines 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.
- **Predictive:** This characteristic has not been presented yet and is the outcome of a "peculiar" usage of the Cases. 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).
- **Full-autonomic:** The operation of the VE is governed by business policies and objectives (expressed by Goals).

4.4.3. System or COSMOS Cases and self-management

For each Application a distinct set of Cases will be created and stored in the CB of a VE. However, the Planner and the CBR cycle can be used beyond the domain of application specific scenarios focused on the end-users. They can also be used for the self-management of the (COSMOS) system itself. The COSMOS community could identify weaknesses and fails of the system and create **System Cases** (**COSMOS Cases**) to solve them. An example is given below.

As it described in Section 5, an infinite loop during XP-sharing is avoided because of the introduction of TTL. However, if the request somehow returns to the initial VE, there is no meaning for the VE to forward the request again, since the very same VEs will be reached. This problem can easily get solved by updating the mechanism via adding a name/identifier to each request. Then, the initial VE (if it has any pending requests) will check whether an incoming request is its own or not. In case it is, then it will not forward the request again, thus avoiding the creation of a loop.

This **update** could take place if the VEs were to change their source-code of the XP-sharing component. A disadvantage of this approach is that this kind of update could make the service unavailable for a while. However, because of the Planner, there is a second option. This update could be forwarded to the VEs not as a change in the code of their functional components but a change in their CB: new Cases. In the case we are studying, the triggering Problem would be the action of a) sending or b) receiving an XP-sharing request. The Solution would be a) adding a field in the request with an identifier and marking the request as a pending request or b) checking this field from a forwarded request and deciding whether it should be reforwarded or not.

Of course, this specific mechanism will be added at the source code since it has already been identified. Another simple example is the case were the COSMOS platform has to inform the VEs that they should change their TTL from e.g. 6 to 5, because of a massive registration of new VEs to the network.

This new kind of Cases could use 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 (subsection 3.2.2): **Self-Configuring Cases**, **Self-Optimizing Cases**, **Self-Healing Cases** and **Self-Protecting Cases**.

Thus, the System Case presented in the previous example can be characterized as a Self-Configuration/Optimization Case. Generally, System Cases can be used to control critical parameters of the system (such as the TTL).

4.4.4. The Planner as a generic Ontologies Comparator

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 is 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.

In COSMOS, ontologies are used for the semantic description of VEs, IoT-services, Applications, etc. If such descriptions are stored locally, the Planner can be used for answering to queries regarding these entities. Let's take as an example the problem of the 3rd type Solutions presented in subsection 4.2.2. These Solutions consist of generic descriptions of IoT-services. The Planner could reason on its local IoT-services Base in order to find IoT-services of the VE with specific characteristics, replacing that way generic fields of these Solutions with concrete ones.

CBR is naturally applicable to web services. A service case can be defined as w = (d, s) and consists of the service description d and its service solution (or functions) s as well as other information including functional dependencies among web services. Web service retrieval, reuse, adaptation and retention in web services correspond easily to the CBR cycle. Our Planner can be involved into a unified approach for case-based web service **discovery**, **recommendation** and **composition**. Osman et al. [35] present an approach that uses CBR for modeling dynamic Web service discovery and matchmaking. Lajmi et al. [36] propose an approach called WeSCo CBR that aims at enhancing the process of Web service composition by using a CBR technique. In [37] Fouad and Baghdad present an approach to dynamically produce composite services (Figure 4).

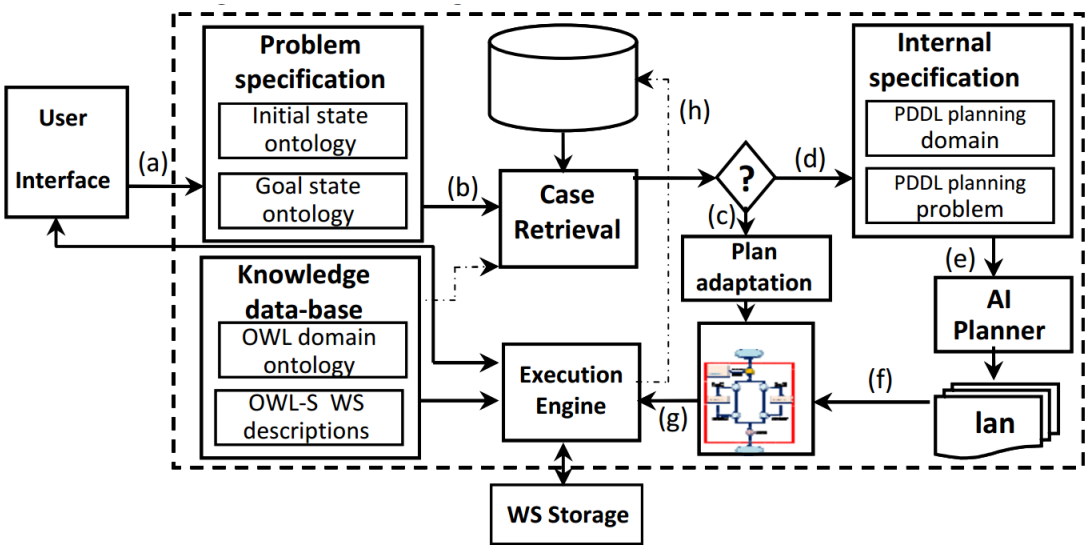

Figure 4: Producing Composite Services with CBR.

4.5. COSMOS Cognition Loop: Node-RED flows

Implementing the concept of the cognition loop is a process that can be achieved through the use of specialized tools. Implemented in the context of WP5, the requirements of any solution must conform to the nature of portability and lightweight deployment. In order to find a matching nature of execution flow management, the Node-RED tool [38] was chosen as the implementing core of our approach.

Node-RED is an event-processing engine that, while it does not take away the need to write code altogether, it does reduce it, and, in the process, both lowers the technical bar and allows IoT innovators to focus on the “creating”, rather than on the “doing”. Node-RED provides a GUI where users can drag-and-drop blocks that represent components of a larger system (such as devices, software platforms and web services) and connect these nodes with each other. Further blocks can be placed in between these components to represent software functions that wrangle and transform the data in transit.

This focus on the flow of information as it passes through different functionalities, whether internal or external, allows us to transform the way VEs perform their functionalities as autonomous entities. By utilizing divergent flows of information and data gathering, we can focus on compartmentalizing the sources and using their input in a dynamic way to implement cognition loops. In the context of WP5, we can rely on Node-RED implemented solutions to act complementarily to the VE-side services we have developed. A major path of action is the integration of the CBR cycle with parts with direct user involvement, whether that is through End-User or Application Developer perspectives. Node-RED blocks can be used to connect front-end, graphical solutions that provide an immediate form of input, especially in the case of Case planning. By utilizing a GUI of parametric Case creation, the Application Developer can determine the combinations of source/sensor data, which are available to be used as the Problem vector of a Case and by perusing the available interactions, there is a concrete way to connect these attributes with VE actuations. The information and plan gathered in this way can then be forwarded through Node-RED message instances in the interconnected VE Services responsible for creating Case templates and eventually using historical data in order to populate the CB with the initial seed of Cases.

Furthermore, the concept of knowledge evaluation can be achieved again through the use of Node-RED. Whether End-User, or automatically initiated, the sum of "events" that can trigger actions through the functionality flows can be used to initiate control loops for the performed actuations through sensor data feedback at the most basic level.

Departing from the CBR cycle, another approachable characteristic is the implementation of the MAPE-K loop, through the connective tissue of Node-RED and the "brains" of the µCEP. By using specialized flows, we can implement, as demonstrated in Year 2, monitoring capabilities for the detection of Events. By processing and forwarding these Events into the Complex Event Detection component of the VE, we take advantage of the numerous inter component communication capabilities given through the implemented nodes. Whether information travels by HTTP or MQTT or other supported protocols, Node-RED is the bridge that brings together the COSMOS loop reasoning behind the handling of any internally monitored and analyzed activities. By also applying reasoning techniques on the combination of readings constituting a Complex Event, we create appropriate action plans, which then pass back into the respective system actuations, by specialized flows, exposed as isolated access points. An early example of that was the handling of a sensor malfunction, which, once internally detected, could be translated into execution of a planned action, by shutting down affected applications.

It is also behind this example that another functionality of the provided tool is demonstrated. Interconnection with VE external COSMOS components is also possible through Node-RED. Creating privacy oriented buffer zones of functionalities is as simple as importing sub-flows which, given the fact of data compatibility, can be seamlessly added into more general data exposing flows. Entirely new additions of connectivity with the persistent Cloud Storage can also easily be introduced, or modified, and the actual data capabilities of the VE's physical manifestation, through the introduction of new input data nodes can be enhanced.

Figure 5 demonstrates one of the flows with the capability of receiving remote actuation plans, into maintaining, creating and updating an existing filter file with the ability to guide the autonomous system in a series of actions, culminating with the disabling of VE Applications. The Event produced is reasoned on by the deep back-end system and the extraction of a parametric COSMOS solution may guide the VE to contact a wide variety of input sensing flows. In this case, there is also the capability for monitoring extra input from both MQTT and HTTP protocol compliant nodes.

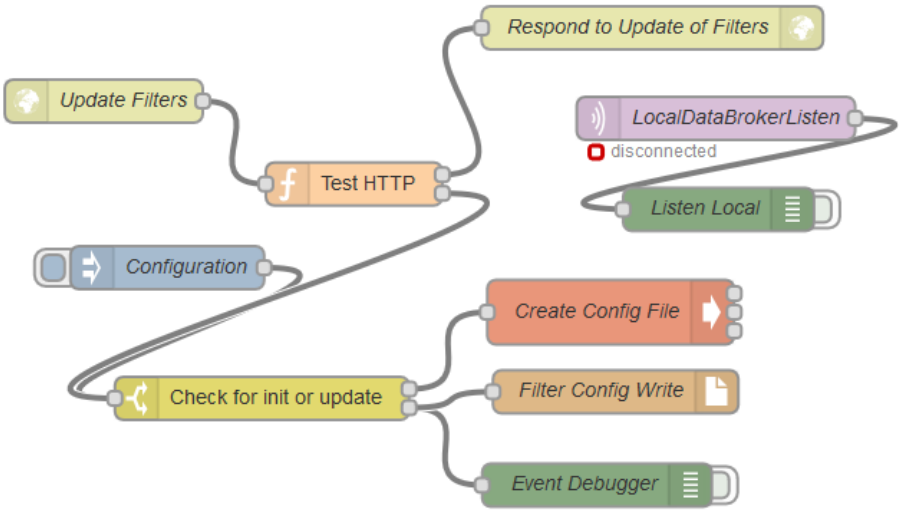

Figure 5: VE Service implementation example.

Figure 6 illustrates the actual data ingestion and filtering mechanism which, in a semi static and predefined way, may lead a continuous stream of information into a quantization of individual Events. In this specific implementation, a series of readings are being extracted for monitoring along with a periodic pulse which can lead into the detection of reading absence. This logic can be used by the µCEP to extract Event of sensor malfunctions, which will then be processed by the internal Planner reasoning capabilities, implemented with CBR through COSMOS designed Cases.

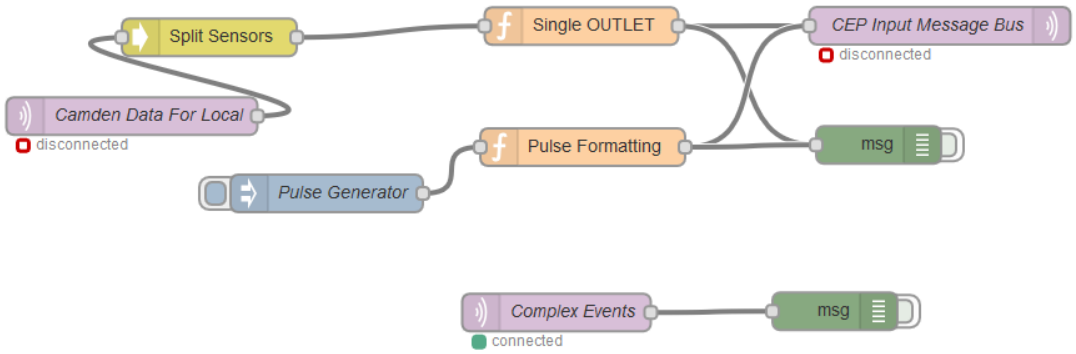

Figure 6: Data feed processing and Event forwarding.

4.6. Types of Learning

VEs have three types of learning cycles which are complementary and may occur in parallel [39]. These may interact with each other in complicated ways and are the following:

- **individual learning:** Individual learning takes the form of Cases creation and storage inside the VEs. By utilizing sensor readings and actuation values, each VE is capable of creating complete Cases of a complexity proportional to its technical abilities. The individual enrichment of the local CB can serve as a basis for the second stage of learning.
- **learning through communication:** This second stage comes into play when the locally stored knowledge is not sufficient for the needs of a VE. In this case, a VE uses the experience sharing (XP-sharing) service and targets a group of Friends that may have the required knowledge. This procedure is further analyzed in section 5.
- **learning through a central knowledge repository:** Finally, if both previous knowledge acquisition mechanisms fail, VEs possess the ability to connect to a central KB. It is worth mentioning that experience, as a final resort, could be stored centrally in the COSMOS repositories so that a “purge” of acquired knowledge does not result in loss of experience.

If we want to experiment on the concept of CBR, we can choose to focus on the first and third option. By adding the second option, we produce a Social IoT system. This system is further described in Section 7.

5. Decentralized Discovery in Distributed Systems

5.1. Cases Discovery - Learning through Communication

The concept of adapting past solutions is one of the main requirements of CBR. For this reason, we introduce the idea of allowing VEs to share their Cases, thus producing an environment where the knowledge is distributed and knowledge flow is supported. In that sense, the Cases produced via CBR become one form of Experience of the VEs, which can not only aid in the planning of future solutions, but can also be shared between different VEs. Because of the great distribution of the knowledge among the VEs, the development of decentralized discovery mechanisms becomes essential.

Learning through communication comes into play when the locally stored knowledge is not sufficient for the needs of a VE (Figure 7A for VE I). Such needs may be constant or opportunistic in nature, a distinction which helps segregate the actions taken on the provided knowledge. In this case, a VE uses the **experience sharing (XP-sharing) service** and forwards its request for new knowledge to a group of other "friendly" VEs (**Friends**) that may have the required knowledge. Friends are maintained in **Friend Lists** in the KB (one Friend List for each Application). The request contains the description of the Problem (**parameter names and values**) for which the VE needs a Solution, the **threshold values** for the corresponding similarity indexes and a **TTL number** (see subsection 5.3).

Upon receiving the request, a Friend extracts the data (Figure 7B for VE I) and uses its Planner in order to check if a similar Problem to the requested one exists inside its own local CB. In case a similar enough Problem is found, the Friend returns to the initial VE the similarity percentage of the similar Problem (or the similar Problem itself) and the corresponding Solution (parameter names and values, URIs, Messages, etc.).

If a similar Problem is not retrieved, then the Friend checks the value of the TTL number and, if after reducing it by 1 the TTL number is above zero, the Friend becomes a "broker" by initiating recursively a new call of the XP-Sharing mechanism (Figure 7C for VE I). Therefore brokers are dynamically designated taking into account that Friends are willing and able to act as such for their respective Friends. This approach is related to the "six degrees of separation" concept that has become quite popular at the domain of social networks [40].

All of the similar Cases that are found are forwarded back to the initial VE only through its direct Friends. In order to be possible for the initial VE to evaluate the answers coming from Friends of Friends, the ID of the VE from which a similar Case is found may be part of the answer to the XP-sharing request. The Planner chooses the best Case amongst the similar Cases. However, the procedure followed for choosing the best Case is not the same with the one followed during individual learning. During learning through communication, the Planner takes under consideration not only the **similarity indexes** of the Cases but also the **social indexes** of the VEs that provided these Cases. For this to happen, **weights** for similarity and social indexes must be defined. In other words, the best Case is the one that has a similar Problem to the initial request and derives from a **trustworthy VE**. The matter of the trustworthiness of VEs is discussed in subsection 8.3. Finally, through feedback loops, the selected Solution is evaluated and the same happens through the **Social Monitoring** component for the VEs from which this Solution was shared (Figure 7D for VE I). Social Monitoring is analyzed in subsection 8.1.

As a final note, the technical solution used for learning through communication involves RESTful interfaces [41] that connect individual VEs on a peer to peer basis.

Learning through communication was successfully demonstrated during the first two reviews of the project.

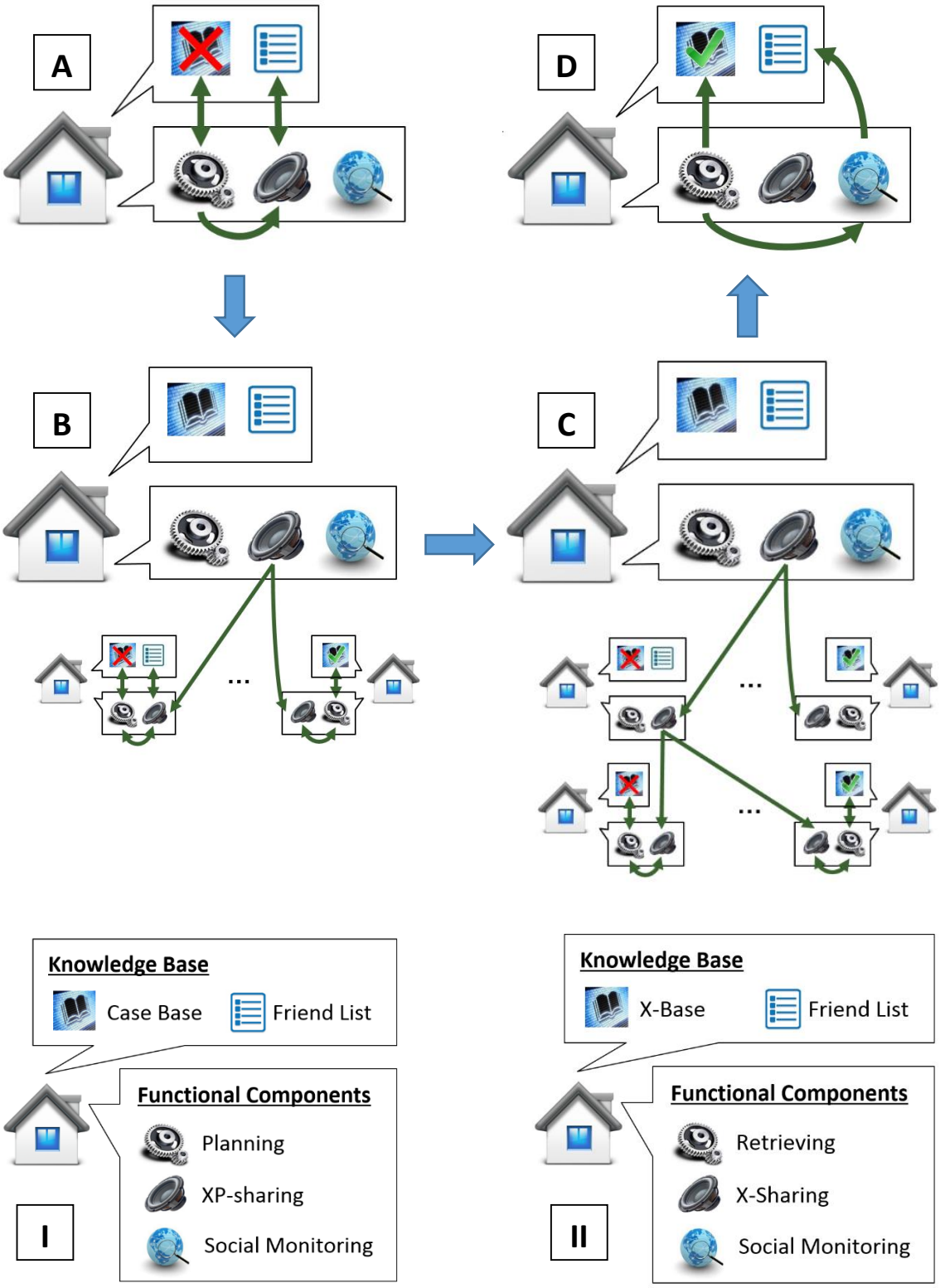

Figure 7: Decentralized Discovery mechanism.

5.2. Discovery of other entities

As it is discussed in subsection 4.4.4, the Planner can reason not only on Cases, but on other parts of the Knowledge Base too, as long as ontologies are used. Because of this characteristic of the Planner, the mechanism used for the decentralized discovery of Cases can be extended to the discovery of IoT-services, VEs, Applications, Friends or any other entities. This is depicted in Figure 7 for VE II, were the Planner acts as a simple Retriever.

For each Application based on the CBR technique that a VE uses, the VE should maintain a separate Friend List. The Friends contained to these lists should be similar to the VE. However, in the case of discovery of other entities besides Cases, it is evident that **a)** the usage of one (per VE) **general Friend List** is enough and **b)** the Friends populating this list do not have to be similar to the VE. The criteria taken under consideration for the selection of Friends used for discovery or other purposes are further analyzed in subsection 8.3.2.

5.3. Issues regarding the Discovery Request

When a VE decides to initiate the experience sharing mechanism with its Friends, it specifies the "depth" of communication. That is important mainly because of the recursive way the experience sharing method works, meaning that if a Friend of the original VE does not locate a suitable case inside its own local CB, it will check the depth required (mentioned TTL/time-to-live of the experience query) and initiate a new version of experience sharing, this time directed at its own Friends. Thus, we ensure that the entire process will operate until either a suitable solution has been discovered or until it reaches a dead end (TTL time out, no case present in our friend VE cluster).

For each one of the discovery mechanisms described previously, one of the main goals is refining their recursive abilities. A big step in this direction is the use of the TTL number not only as a static although customizable input but as an actual dynamic representation of the in-between stages of "information brokering" based on the theory of the six degrees of separation and the actual social capabilities of affected VEs. When a TTL number is high and the recursive discovery uses nodes (broker VEs) with many outward connections, this can lead to **network overhead**, a situation we want to avoid. If we take into account social criteria of the VE, we can adjust the TTL number. For example, by initially using a simple mathematical function of a logarithmic nature, we can adjust the TTL number of each kind of discovery request by using the **number of Friends** (of the corresponding Friend List) of a VE as well as a "target audience" to be reached (**maxhits** variable). Further code development and experimentation with real data will refine the function used. Moreover, if the experimental data points to such a possibility, we will use an absolute upper limit for TTL in the sense of the six degrees of separation theory.

Although the TTL number can solve the problem of network overhead and makes sure that the creation of infinite loops is not possible when the initial request is forwarded back to the initial VE (through e.g. Friends of Friends), it does not solve the problem of **"discovery overlap"**. If a discovery request somehow returns to the initial VE, there is no meaning for the VE to forward the request again, since the very same VEs will be reached. This problem can easily get solved by adding a name/identifier for each request. Then the initial VE (if it has any pending requests) will check whether an incoming request is its own or not. In case it is, then it will not forward the request again.

It is worth noting that when a VE accepts a Case discovery request, then it retrieves only Cases in its local CB that are marked as **shareable** by flags. These flags can be set either by the corresponding Application developer or the owner of the VE.

5.4. Stress Tests of Component's Functions

The VE specific code is developed having as a basis the Java programming language, in order to ensure that it will be cross platform compatible. Java was also selected since it holds a good balance between the intermediate computational requirements a gateway should possess and the availability of frameworks and technologies for the implementation. **Raspberry Pi 2** [42] was selected as a deployment Testbed since it holds a good balance between cost and gateway capabilities.

Running on Raspberry Pi 2 is the Raspbian OS [43] which is a Linux version for Raspberry based on the ARM hard-float Debian 7 'Wheezy' architecture port. The latest version of the Raspbian OS comes bundled with Java version 1.8 (as of 12/2015). This part of the Testbed acts as the Hardware Gateway of a Flat on which a Smart-Home Application is running and is being operated on by the End-User. We are simulating two additional interconnected VEs by using two remote Virtual Machines which have been retrofitted with the required code (application and VE code).

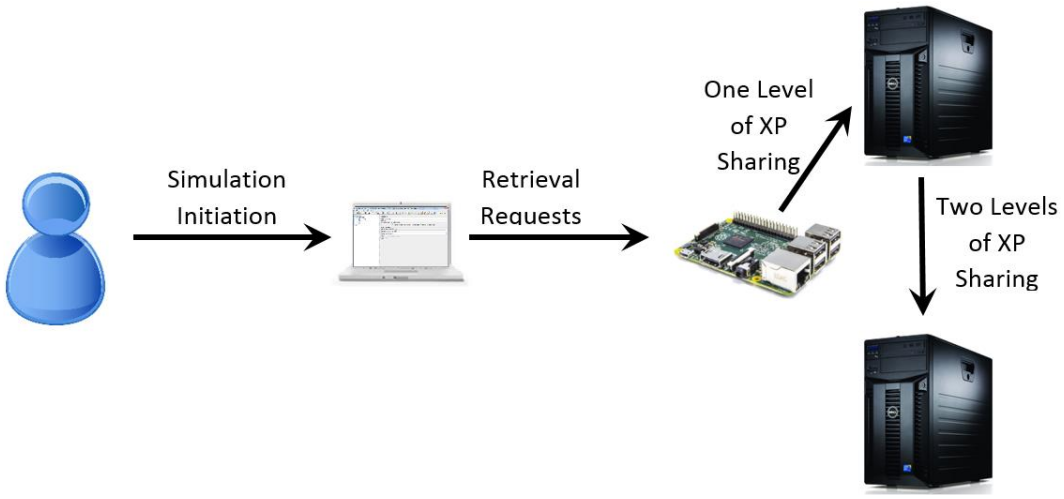

Figure 8: Our Testbed.

Our simulation approach focuses on the ability of the Testbed at the Raspberry side to provide a consistently stable environment for remote VE components being used by an Application to communicate and share their Experiences as well as reason on their stored Knowledge in order to provide answers to each other's queries. Our tests are demonstrating how differentiated workloads of queries may affect VE performance in resource constraint environments. The Raspberry running the VE Service is contacted and attempts to locate a Solution to the incoming Problem are made. In the first attempt, the VE searches in its own CB (local retrieval) and, if nothing is found, it contacts the first VM (first level of XP-sharing). The first VM then searches in its own CB and, if nothing is found, it contacts the second VM (recursive Experience Sharing). Of course, the response time to the query is considered to be the overall delay.

The tool used for the simulation is **Apache JMeter®** [44]. It is located in an external workstation dedicated to the metering process and simulates requests from End-Users. Our tests are divided into four categories, each one corresponding to a different possible traffic condition of the Network of Things. In all categories, differentiations of query values which lead to XP-Sharing were made in order to test the effect that the depth of the queries has on the retrieval and response times.

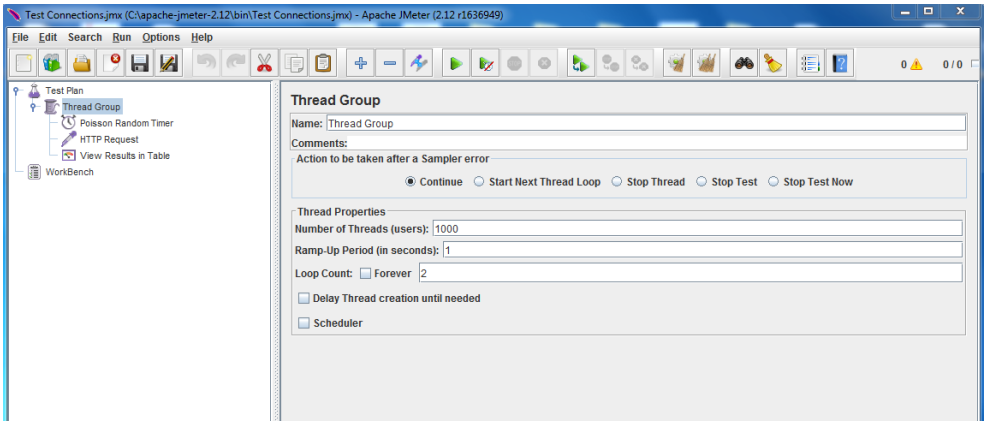

Figure 9: Apache JMeter®.

The first category is the **Low Volume** category which is characterized by a single Query Thread running infinite loops, with a constant 10 second delay between calls. Results of the three versions of this category of simulations are demonstrated in Figure 10. In the Low Volume category, results in all three versions demonstrate that the lightweight design of our approach can produce response times of less than a second from the initiation of the query to the eventual return of a valid answer.

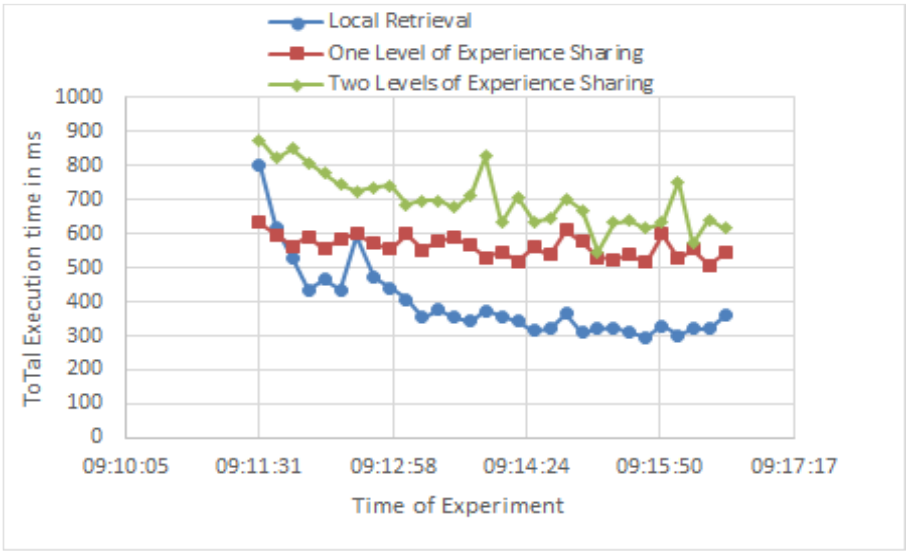

Figure 10: Low Volume Simulations.

The second category is the **Medium Volume** category, which we deem to be the normal/common volume of communications expected by the COSMOS system. In this category of simulations, ten Query Threads are used, starting their operation in intervals of two seconds, with infinite loops for each one and a Poisson timer of query delay with a lambda value of 10,000 ms. These settings lead to an increased load of queries which, due to the nature of Java parallelization in handling incoming HTTP requests, are serviced in less time than the serial arrivals of the Low Volume category. This is the case for all three versions of the simulation. Results are demonstrated in Figure 11.

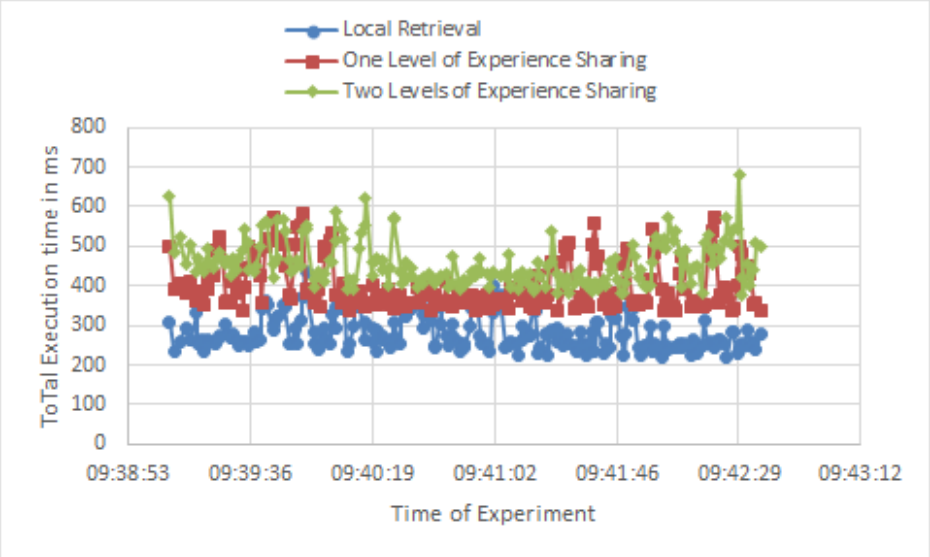

Figure 11: Medium Volume Simulations.

The third category is the **High Volume** category. In this category we used one hundred Query Threads, operational in two second intervals, running infinite loops with a Poisson timed query delay of 5,000 ms. This category initially presents similar response times to the previous two categories, but as more Query Threads come into operation, response times increase (especially after 50 concurrently running Threads) and eventually reach averages of 12s, 16s and 17s. In this category, deviation is also increased with results in the third version reaching as high as 28s and as low as 8s. Results are demonstrated in Figure 12.

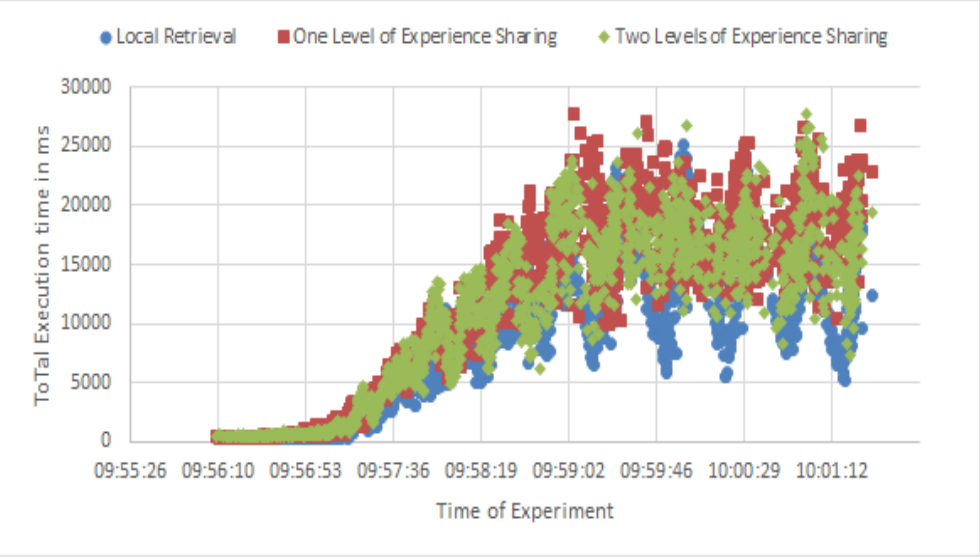

Figure 12: High Volume Simulations.

Finally, we attempted tests in **the DoS (Denial of Service) Volume** category, which includes the use of 1,000 Query Threads running infinite query loops, starting simultaneously and with no delay between queries. As was expected, response times were greatly increased with the VE responding to queries at an average of 200s. However, it was proven that the VEs are robust and reliable enough to sustain such an increased load of operations without denying service. Comparative levels of throughput between categories as measured by JMeter indicate that the

request service rate was similar to that of the High volume category. Results are demonstrated in Figure 13. Specifically in this category, the query versions implementing XP-sharing were not used because of permission issues pertaining to the use of the VMs under heavy incoming loads. Therefore only the use of the Raspberry running VE and Application code is demonstrated.

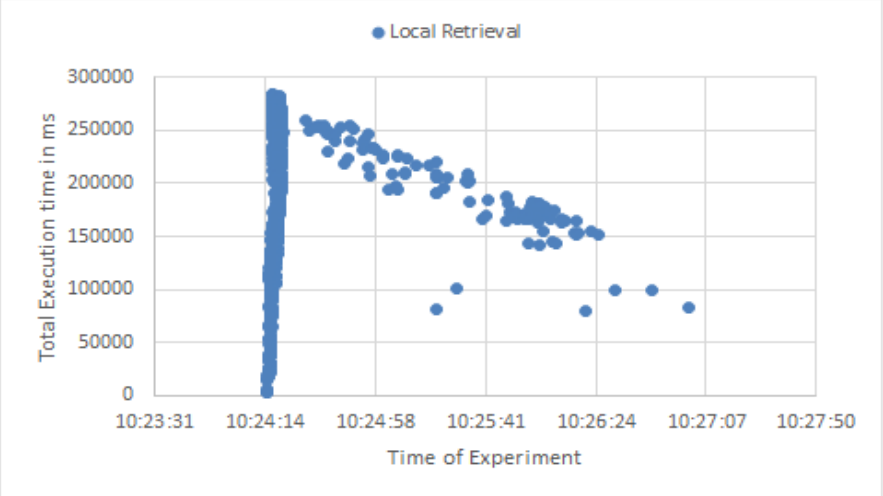

Figure 13: DoS Volume Simulation.

It should be noted that the measurements for XP-sharing of one or two levels are biased, considering the fact that the auxiliary VMs used are not Raspberry Pi 2 themselves. Therefore, further simulations are performed in order to clarify the amount of time differences from the previous measurements. The approach followed is to simulate local search of a Case inside the VM, in order to compare it with the first simulation version of the Low and Medium Volume categories (with no XP-sharing taking place). Also measured is the actual time the VE spends in searching and retrieving the Case. Results of the simulation are demonstrated in Table 1.

Table 1: Raspberry Pi 2 and VM Time Comparisons.

	RASP2	VM
Request Low (ms)	392	201
Search and Retrieval Low (ms)	378	139
Request Medium (ms)	360	212
Search and Retrieval Medium (ms)	346	132

These results demonstrate that while the efficiency of the VM is increased by as much as 68% in internal calculations in comparison to the Raspberry Pi 2, by adding the amount of time the request takes to travel to and from the request originator, the actual simulation gain is a little less than 50% for each VM used. Therefore the results have to be considered under this light.

While these findings, may adversely affect the Application performance under heavier load, the delays in normal and low volume are not greatly affected, considering their low order of magnitude.

5.5. Types of Communication

We can make a distinction between two forms of learning through communication:

- **supply driven learning:** In supply driven learning, an individual VE acquires new experience and communicates it to the Groups of VEs (GVEs) it belongs to.
- **demand driven learning:** In demand driven learning, a VE comes along a new event/problem and asks its Friends whether they have a solution for this problem.

In both cases, two factors should be taken into account:

- **overhead:** the number of useless messages that are acceptable from the recipients side.
- **hit rate:** the amount of VEs that get the message compared to the amount of VEs that should have received it.

Regarding the dissemination mode, there are three options to choose from (adopting the terminology from the advertising, marketing and communications domain):

- **Broadcasting:** Sending the message to every available VE. This way, the hit rate is maximized at the cost of a large communication overhead. An advantage of sending the message to a large audience is that it creates redundancy in the knowledge assets of the system, which facilitates knowledge development through combination. However, for most IoT use-cases, this is not an option due to scalability issues. As such, COSMOS does not provide any broadcasting mechanisms.
- **Narrow casting:** Sending the message to every VE that may be interested in a specific topic. This option combines the advantages of the other two, but it requires the VEs to state beforehand which kinds of messages they are interested in (e.g. by means of a user profile). This in turn requires that there is a predefined set of possible topics or, otherwise, that there are guidelines for creating new topics. In our case, data can flow through the system via a Message Bus which is organized into Topics. Each VE can publish and/or subscribe to them. The whole process is supported by a Complex Event Processing (CEP) component which is responsible for processing data and analyzing them in real time, according to applications' specific logic. If a certain event is detected by the CEP component, this may trigger the generation of certain messages to a new topic.
- **Personal casting:** Sending the message only to VEs that are directly involved to its content. This is the most efficient way of communication, as only VEs that can directly help or can offer the new required knowledge are informed. In this way, the communication overhead is kept to a minimum, which is important for maintaining the communication channel alive. That is why the VEs need Friends and we should develop a social environment that can support their discovery.

6. Ontologies

6.1. The COSMOS Ontology

One of the key goals of COSMOS is to facilitate the development of IoT Applications by providing sharing mechanisms for data, experience, models etc. The main pillar supporting these goals is the VE. VEs extend IoT-services adding new functionalities as well as a social dimension to these services.

One of the requirements which had to be considered was the efficient retrieval and the precise addressability of VE properties. These requirements are enforced also by the fact that different actors are involved in the development and deployment of a COSMOS enabled Application (VE developers, platform owners, platform extension developers, COSMOS Application enabled developers etc.) since resources are shared and reused.

To support these requirements we have decided to use semantic technologies and the linked data paradigm [45] in the description of the COSMOS entities. We have therefore designed the core **COSMOS ontology** which is supplemented with the **social-related Ontology**. Application or domain specific descriptions can also be integrated through **domain specific ontologies**. Figure 14 presents a block diagram of the information model, including the key concepts supported by the COSMOS ontologies.

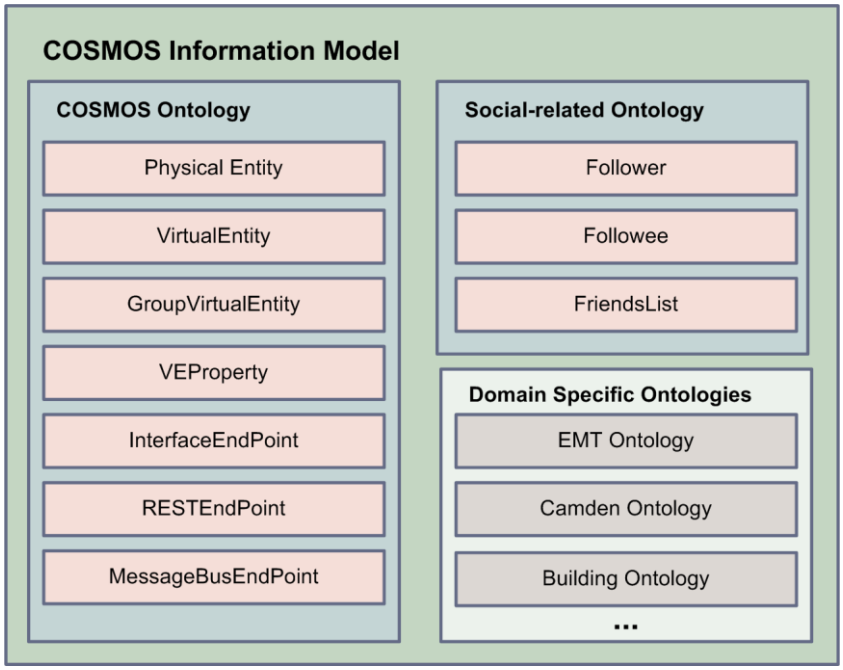

Figure 14: COSMOS Information Model.

The COSMOS ontology is a light weight domain interdependent ontology centered on the description of the VEs. It provides the model for describing the complete chain from physical entities, to resources, IoT-services, VEs and final endpoints. The ontology has been developed and updated during the project in order to meet the extended description requirements.

Figure 15 depicts the final version of the COSMOS ontology with all the concepts and relations connecting them.

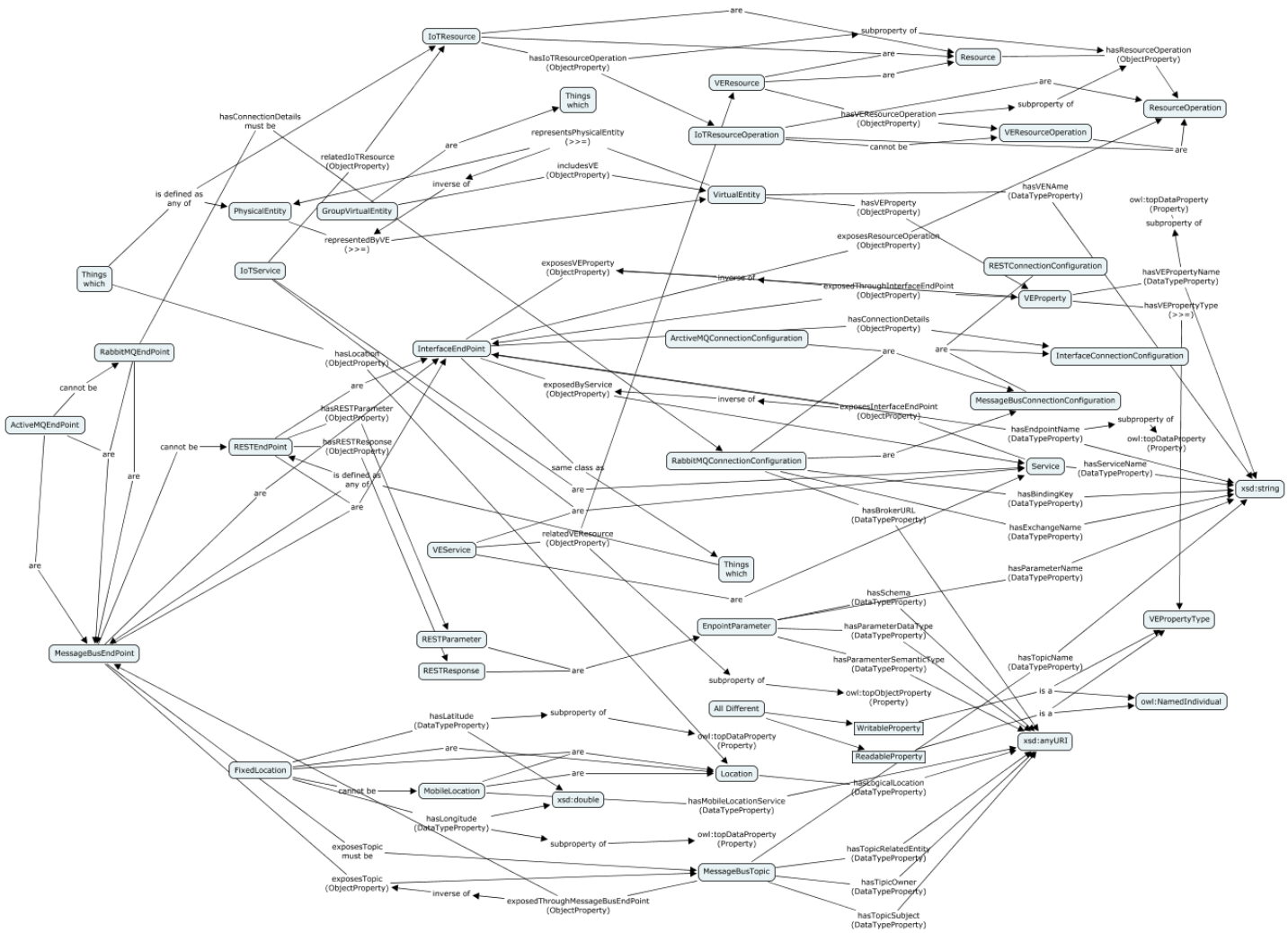

Figure 15: COSMOS Ontology.

In the following paragraphs we will describe the key concepts of the ontology and their relations. Since the complete depiction of the ontology is hard to be followed by the reader, we will depict the ontology in a cloned manner in Figure 16, Figure 17, Figure 18 and Figure 19, meaning that some classes are duplicated in order to facilitate viewing.

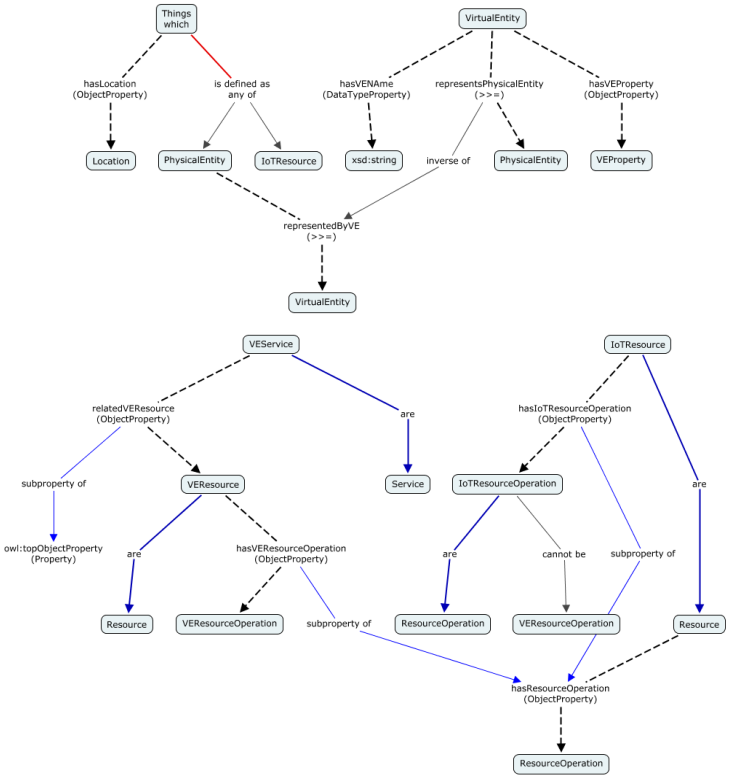

Figure 16: COSMOS Ontology - partial view 1.

VirtualEntity – It represents the key concept of COSMOS and the ontology is centered on it. A VE provides an abstraction above the IoT-services, encapsulating additional services, social relations and functionalities.

PhysicalEntity – A Virtual Entity represents an entity from the physical world which could be anything; a bus, a building or even a person.

VEProperty – VEs expose values which not only relate to the raw readings of sensors or the control of actuators but also to additional functionalities which the VE concept introduces. For instance, the physical entity of a bus might be represented by a VE which would expose: current speed, location, number of hard brake events in the last 60 minutes, the condition of the traffic based on situational awareness, etc. In other words, the VE is able to expose data which are not necessarily IoT related but the result of an internal computation, the aggregation of external data, a prediction model, etc. VEs integrate VEProperties and VEServices in order to expose raw or processed IoT data but also VE provided services. The VEProperty is the concept providing the bridge between the physical entity, all these features (including also non IoT related aspects) and the actual access interfaces which will be later described. In other words, it references to descriptions of ***what*** is exposed and ***how*** it is exposed.

Feature – As already mentioned, a VE exposes IoT and non-IoT related data. The *feature* concept is meant to describe these capabilities, like sensor readings, actuator commands, aggregated data, historical models, etc. It indicates ***what*** is being exposed by the VEProperty.

VEService – A VE Service exposes VE related services which are not related to a specific IoT resource. Any extended, non IoT resource related functionality of a VE can be described and exposed as a VE Service.

IoTService – An IoT Service is related to an IoT resource. It can be used to provide direct access to the properties of an IoT resource as well as extended functionality related to that resource.

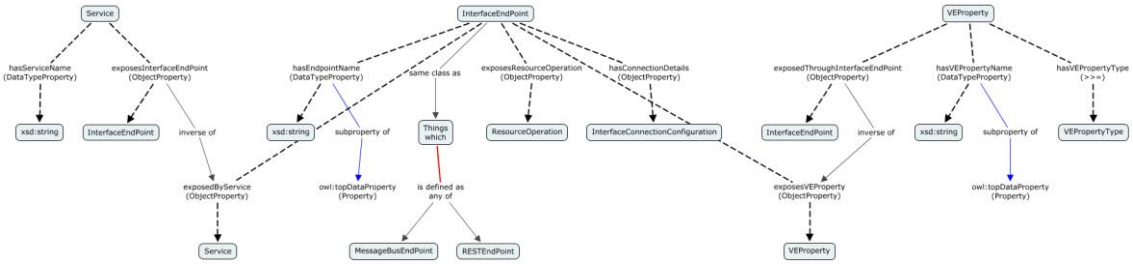

Figure 17: COSMOS Ontology - partial view 2.

InterfaceEndPoint – The concept of *Interface EndPoint* provides the transition from the conceptual description of the VEs to the actual interfaces providing the access to the features exposed by the VE properties. A VE will most probably have multiple VE properties which are exposed by services and have to be uniquely addressable. These can be IoT-services or other kinds of services and their endpoints can be REST endpoints or message bus endpoints.

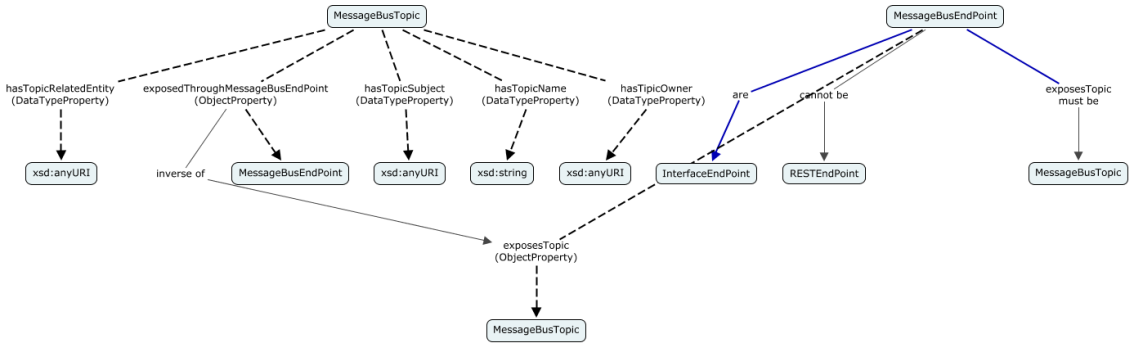

Figure 18: COSMOS Ontology partial view 3.

The ontology can be extended to support different interfaces. Figure 19 indicates some of the descriptors for different endpoints such as the **RESTEndpoint**, **RabbitMQEndPoint** or **ActiveMQEnpoint**.

As seen from the above descriptions, the ontology is domain independent. The advantage of domain independence is that the ontology will not require any changes which have not been considered during the design whenever a new Application is developed for a use case. In fact, such a goal would have been impossible to achieve since the range of use cases is infinite. Instead, the ontology supports the extension of the VE descriptions using domain ontologies.

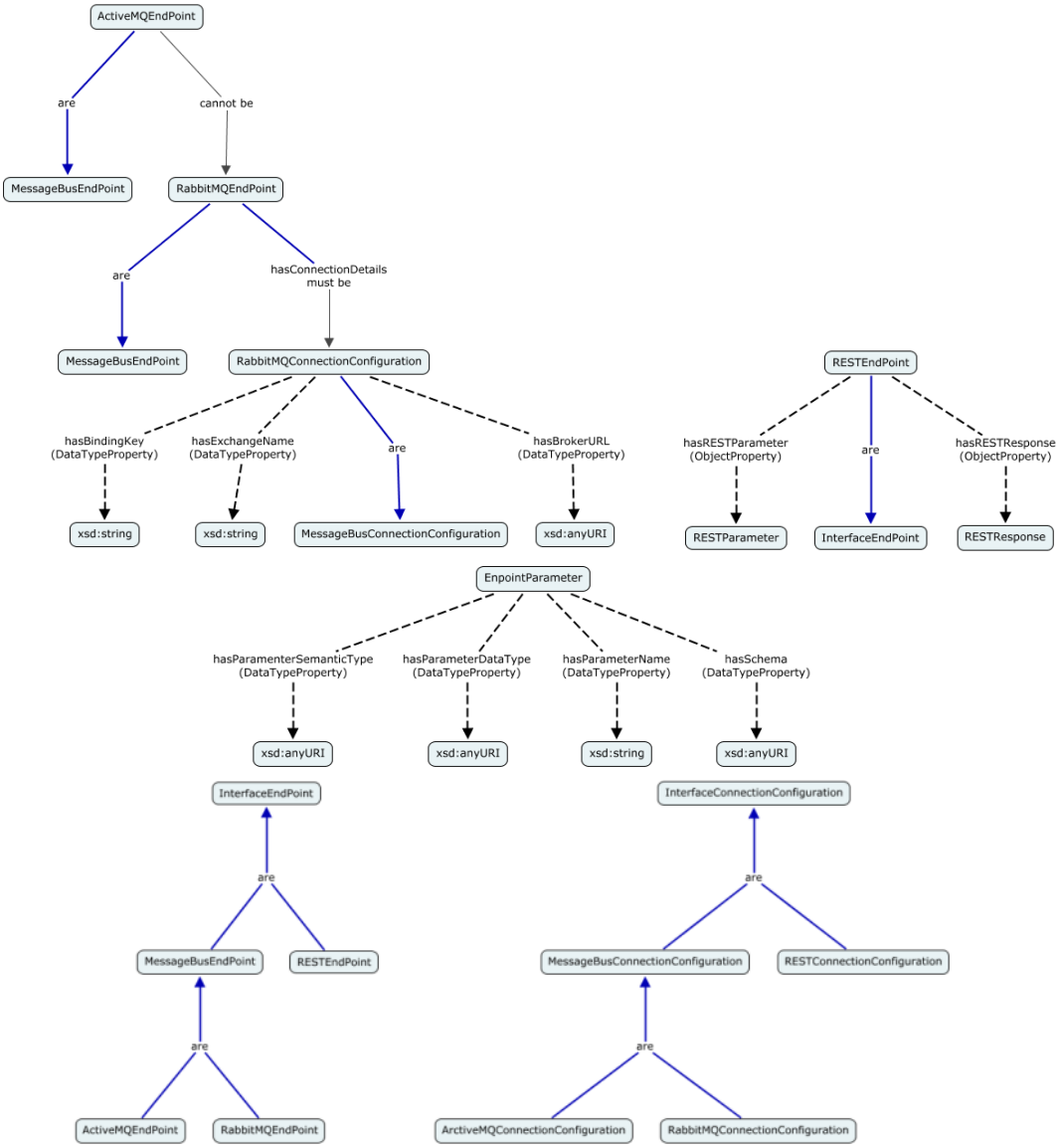

Figure 19: COSMOS Ontology partial view 4.

6.2. Domain Ontologies

VE descriptions can be augmented through the use of domain specific ontologies. This makes use of the linked data paradigm which is a key concept of the semantic technologies. While the COSMOS ontology is the product of the COSMOS project and is intended to be the main model for the VE description, domain ontologies can be built by VE developers, COSMOS enabled Applications or COSMOS-extensions developers to support the description of their use cases and most probably of the physical entities which their VEs are representing.

While in some use cases new domain specific ontologies will be required, this is a mandatory requirement since in some cases existing ontologies can be reused. In fact, from the perspective of the core COSMOS ontology, there is no constraint regarding domain specific ontologies which are referenced in the VE descriptions. D2.3.2 contains the description of a sample domain ontology which has been built around the EMT use case scenarios.

7. Registry and Centralized Discovery

As already mentioned, proper retrieval of the VEs' and other COSMOS related entities' descriptions is a key requirement for the sharing mechanism.

7.1. VE Registry structure

To support this requirement we have designed a Registry whose role is to provide the adequate functionality for the retrieval of VEs. This Registry is in fact a semantic Registry and is backed by the core COSMOS ontology. Figure 20 depicts a block diagram of the VE Registry with its main components.

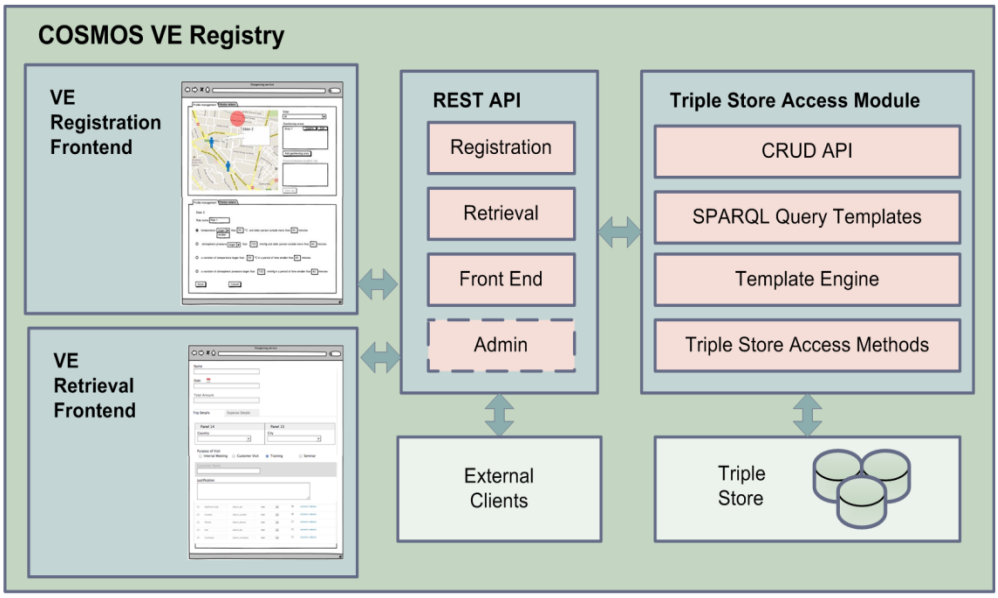

Figure 20: VE Registry block diagram.

While the backing of the COSMOS ontology provides a common "language" for the description of VEs, users are not required to have a thorough understanding of the semantic technologies used in order to describe and retrieve VEs. This is due to the fact that the Registry exposes a front-end which hides the complexity of the publishing and retrieval mechanisms.

The Registry's back-end is responsible for adequate storage, retrieval and maintenance of these descriptions. As also described in D2.3.2, triple stores provide the support for the persistence of the semantic descriptions made for VEs and other related entities and components. Moreover, the Registry exposes an API which can be used without any knowledge of the underlying technologies. Nevertheless, the exposed REST API also allows external clients (such as other components, Applications, VEs) to connect to the Registry and access its functionalities.

The stores are accessed using SPARQL queries but the complexity of these queries is hidden to the user of the VE Registry. The API provides access to CRUD operations from a higher level (VE description) and handles transparently the triple creation, removal, update and deletion.

7.2. VE Registry Suite

We have developed a suite of modules integrated into a web application which provides complete access to the COSMOS entities, as described by the COSMOS ontology. This suite provides the ability to annotate, retrieve and visualize VEs and their associated components.

The start page of the suite allows the user to select a task which might be the annotation of a COSMOS entity or a search.

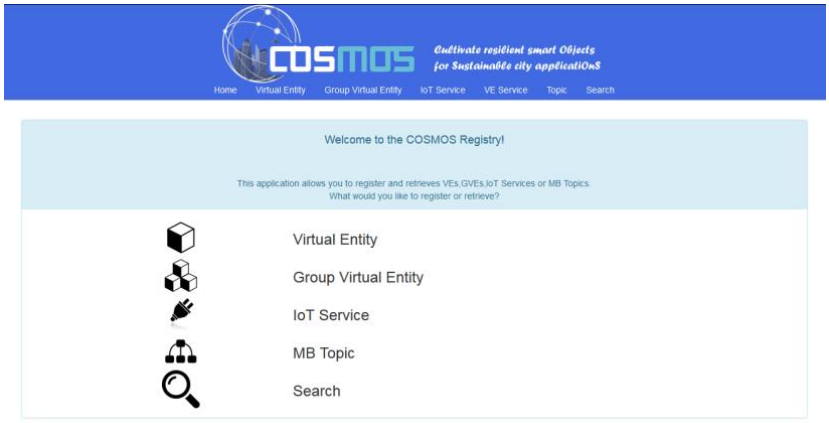

Figure 21: VE Registry start page.

As a recommendation, the descriptions should be performed using a bottom-up approach, with the VE definition being the last. The reason for using this approach is that, when describing a COSMOS entity, references to the lower level entities are required. A search functionality is provided to ease the annotation process and the linking of entities.

A good starting point is the IoT Service annotation since this is linked to the actual IoT resource whose functionality is encapsulated into the VE.

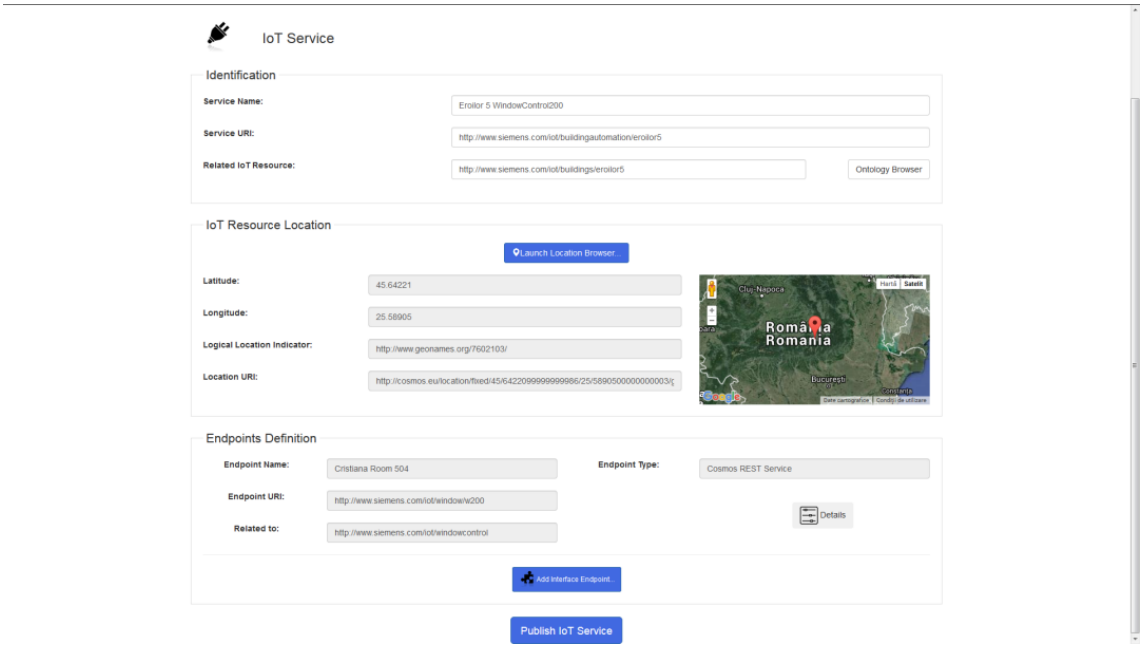

Figure 22: IoT Service annotation.

After entering the service identification data, the user can link the service to the related IoT Resource by using to Ontology Browser integrated into the suite. The Ontology Browser allows the user to select from the desired domain specific ontology the class or the instance which properly describes the associated IoT resource.

The location of the associated IoT resource can be introduced in three different ways, according to the user's needs: using a GeoNames Records, using a Google Maps based custom location indicator or pointing it to a location indication service in the case of mobile IoT resources.

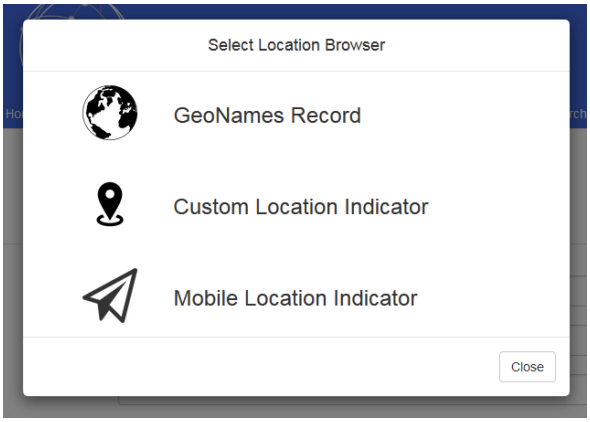

Figure 23: Location Browser Selector.

GeoNames records are useful when pointing to known locations where as Custom Location Indicators are meant to support besides the physical location (latitude and longitude) references to user defined semantic descriptors (usually part of its application domain specific ontology).

Figure 24: Sample location indicated using a GeoNames entry.

Once the identification data is entered, the user can describe the endpoints exposed by the IoT Service. The type of the interface, the input and the output parameter data types, semantic types and schemas can be described as well. The Ontology Browser can be used to search for a suitable description of the associated resource operation if a domain specific ontology is available.

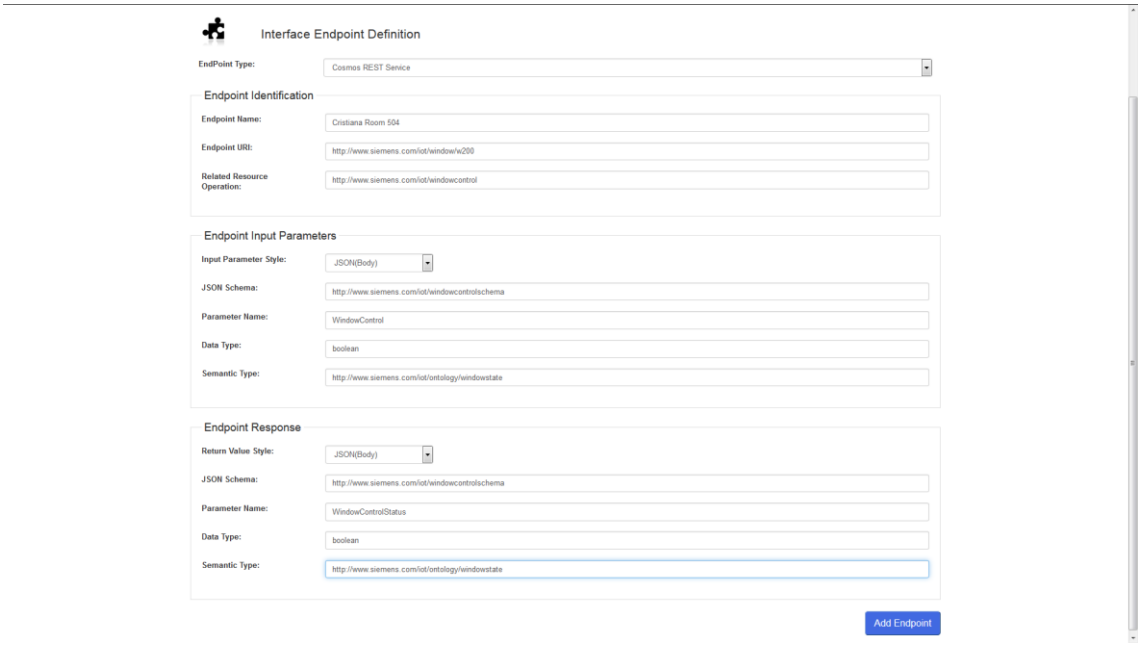

Figure 25: Interface Endpoint Definition.

Once the IoT services have been described, the definition of the VEs encapsulating these services can be done. A similar description pattern is followed.

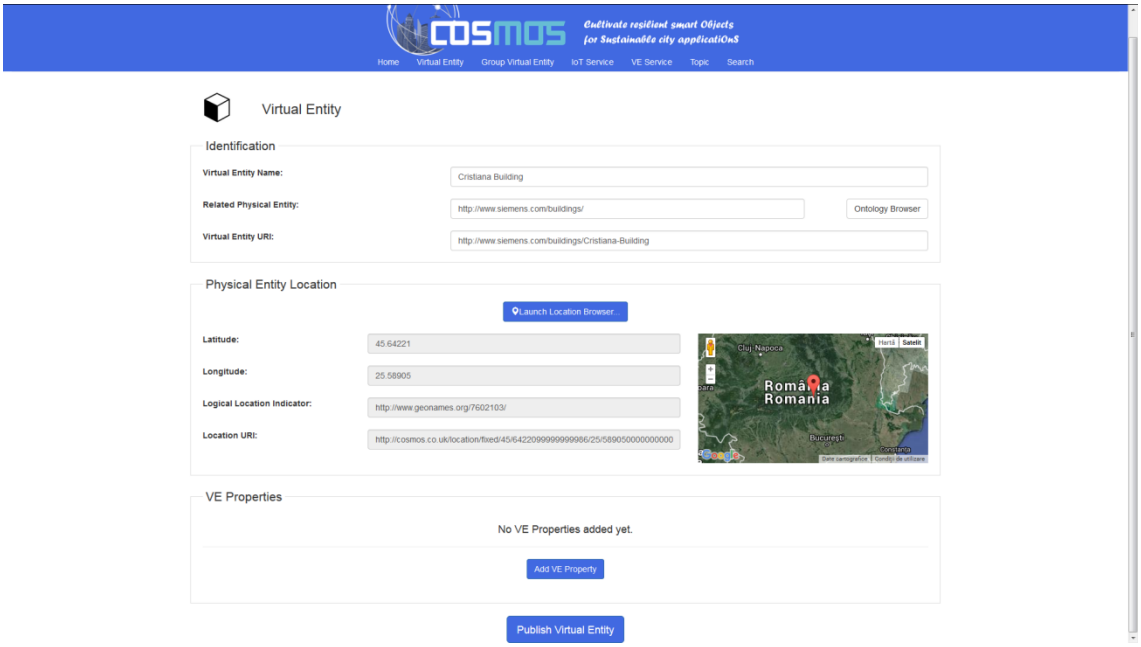

Figure 26: Virtual Entity Annotation.

Since the IoT services have already been annotated, the user can search for existing endpoints and associate them to the VE Properties. To facilitate this process, an endpoint searching functionality is provided by the tool.

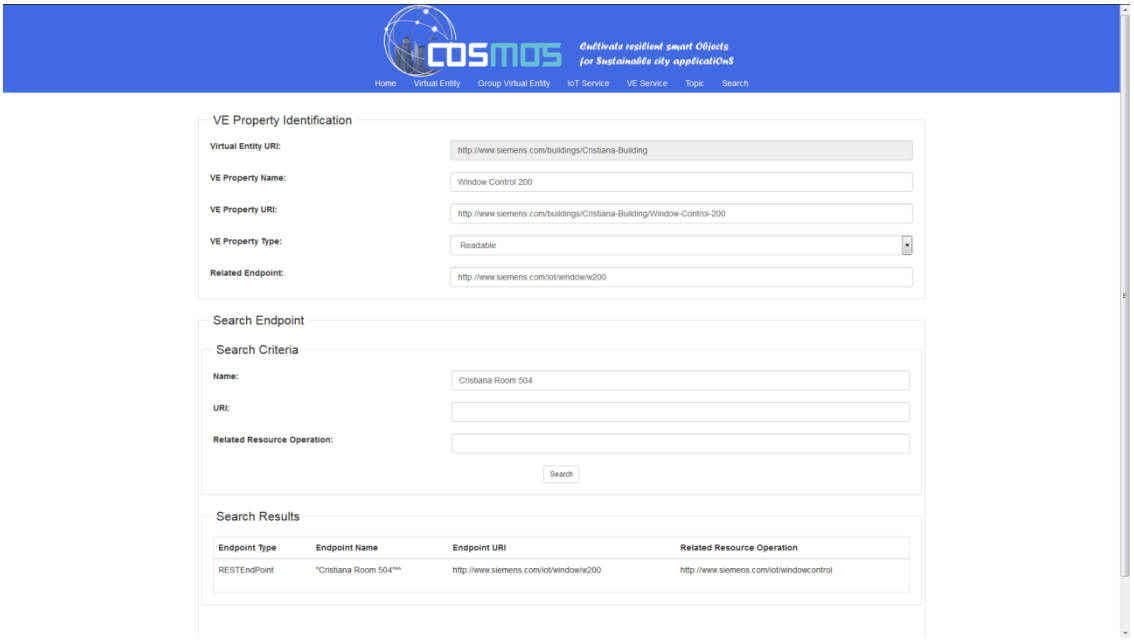

Figure 27: VE Property Annotation.

An important tool for building COSMOS application is the Entity Search Tool. This is a flexible component of the suite allowing the search for any type of COSMOS entity, not only for the VEs, by using multi-criteria, location based search algorithm.

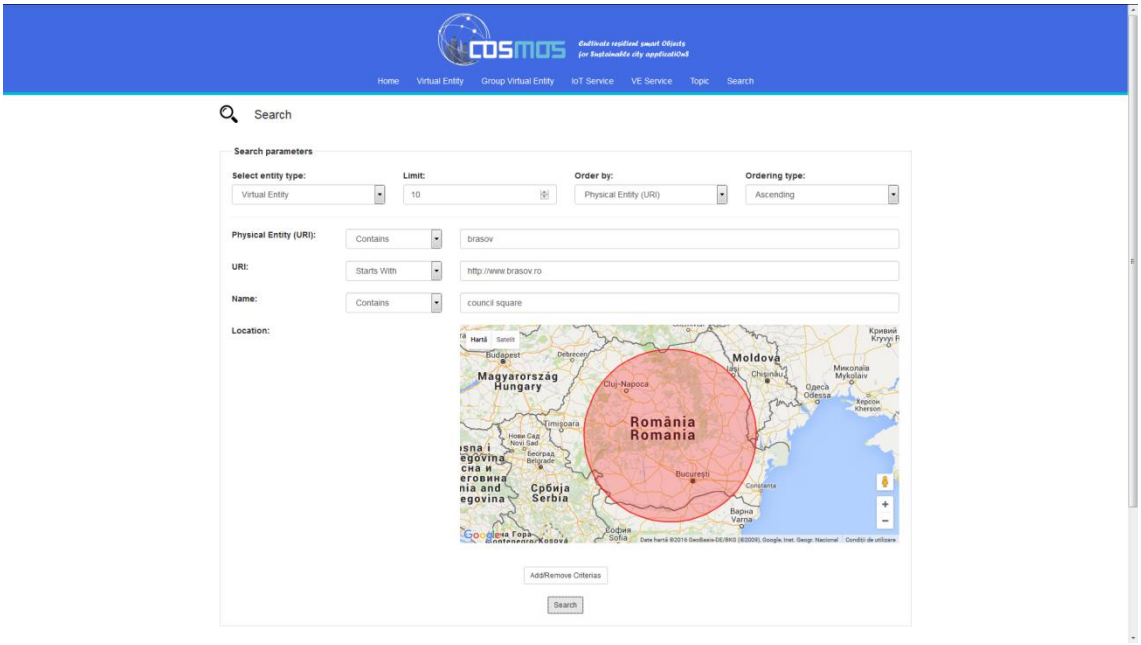

Figure 28: Multi-criteria, location based search tool.

The results of the search can be displayed in and table format but also in a navigable graph representation in order to facilitate the visualization of the related entities.

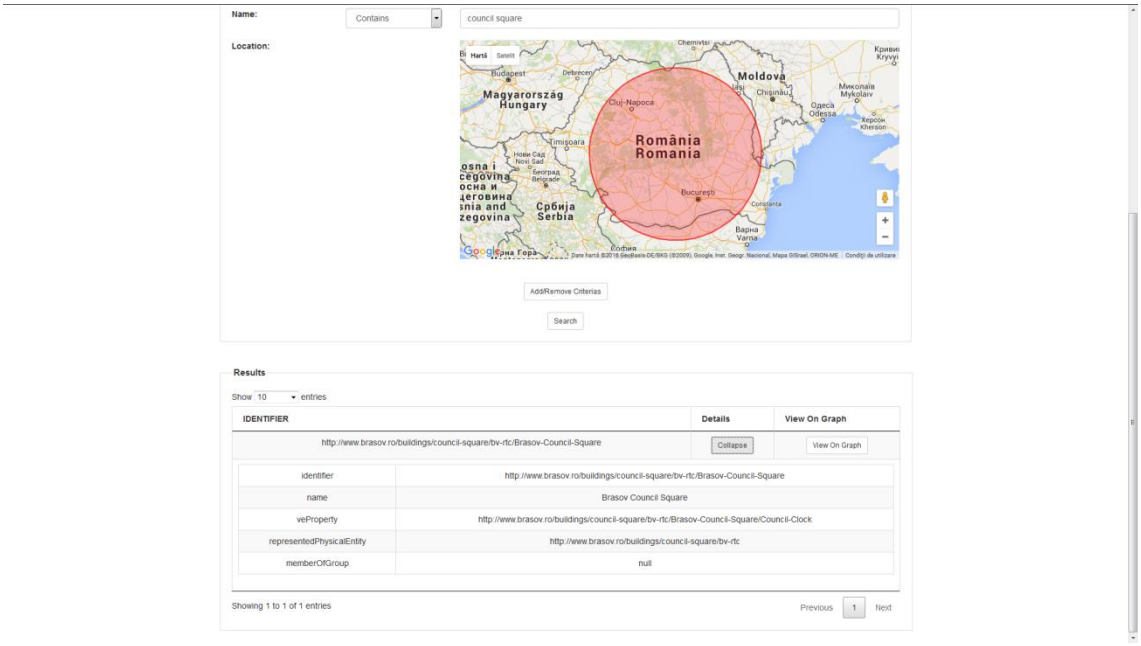

Figure 29: Search results in table format.

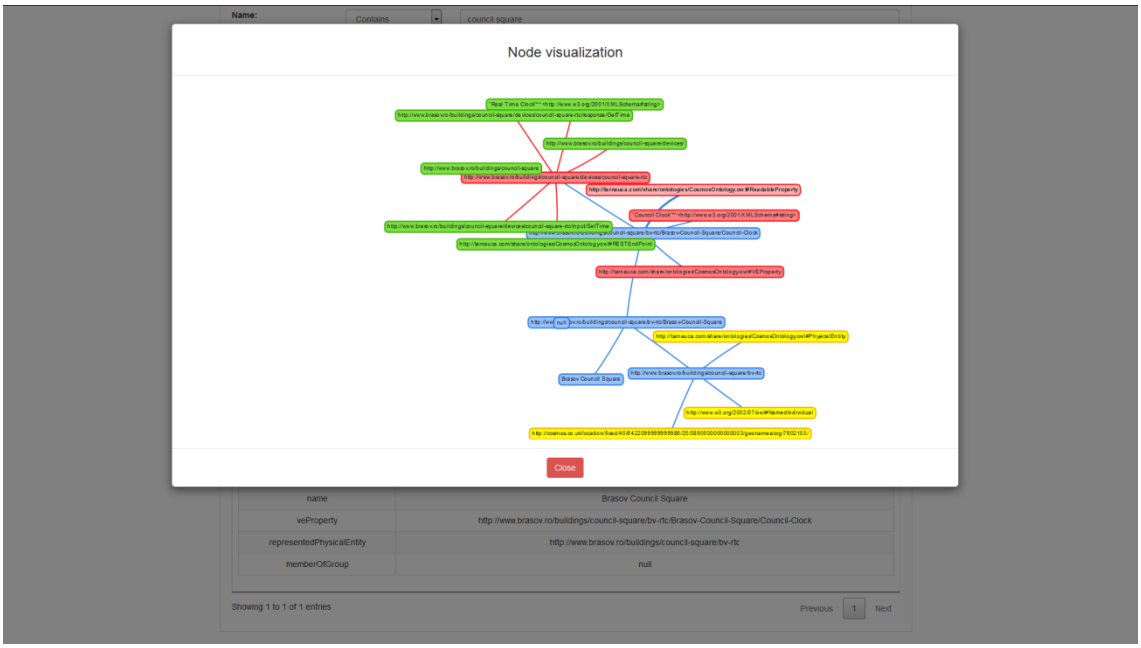

Figure 30: Search result in graph format.

All the functionalities exposed through the front-end modules are also accessible using the provided REST services API allowing the programmatic use of the VE registry by the other components of COSMOS or by external clients which might implement additional functionalities.

8. Social Virtual Entities

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 this section we present the social approach that the COSMOS project introduces in order to achieve enhanced services like discovery, recommendation and sharing between Things enriched with social properties. 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) [46] system. There are three elements that would justify the characterization of a system as a SIoT one:

- it maps the social relations and interactions of the individuals/organizations to their Things,
- it defines, monitors and exploits social relations and interactions between the Things,
- it uses technologies and exploits services from the domain of the social media.

The COSMOS platform as it has been designed comes under the two last categories.

8.1. Social Relations & Monitoring

Instead of trying to map various social relations and characteristics of the real world to the IoT, a more concrete approach would be to identify the various interactions that could exist between VEs. A VE1 can send to VE2 a **service request** for:

- **XP-sharing**
- decentralized **discovery** of other entities
- access to **IoT-services**
- **recommendations** (analyzed in subsections 8.3.1 and 8.4.3)

Inspired from the social media domain, we define and monitor the following **relations**:

- **Followees:** The VEs that are being followed by a specific VE. The Followees Lists define the receivers of the VE's requests for services.
- **Followers:** The VEs that follow a specific VE. They are held in the Followers Lists and indicate the credibility and reputation of a VE. Moreover, their number can act as a rough indicator of the frequency of requests for services from other VEs.
- **Groups:** To how many GVEs a VE belongs and how many VEs and GVEs (separately and in total) a GVE contains. A VE belonging to many GVEs gives us a measure of its "centrality" in the whole system and the real world, whereas the members of a GVE provide the main data needed to grasp its size and complexity.

The relation between a VE and its Followees is trust-based and non-mutual (low reciprocity [47]). This means that VE1 may use the experience of VE2, but on the other hand, VE2 may not do the same for VE1. 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.

The interaction metrics (Shares, Assists, Applauses and Mentions) monitored by the Social Monitoring component of a VE are stored locally in the corresponding **Followees Lists**. For each type of service request (XP-sharing, decentralized discovery, IoT-services, recommendations) and for each application, different Followees Lists are created. 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 the **Social Analysis (SA)** and **Friends Management (FM)** components. For each metric identified we develop/choose the corresponding Key Performance Indicators (KPIs) and tools that should be imported into the VE during the phase of registration. Based on the chosen configuration, different levels of reporting granularity could be possible in order to keep the monitoring tasks as light-weight as possible but still to be able to perform in depth analysis whenever needed. 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.

From these metrics, the **social indexes** of the several kinds of Friends are extracted. These are the **Trust Index** and the **Reputation Index**. More details about these indexes can be found in subsection 8.3. Our main goal is to combine the Reputation Index, the Trust Index and the **Reliability Index** (an absolute indicator of the performance of the corresponding Physical Entity of a VE) of a Followee and express them by only one social measure, the **Dependability Index** of the Followee. This measure, which is further discussed later, is a crucial indicator that will lead a VE to take decisions regarding the selection of new experience or new Followees. If a VE presents good social indexes, then other VEs will keep following it (sending services requests to it). If the social indexes of the same VE worsen (because it has started providing bad services), then the other VEs may decide to remove it from their Followees Lists, place it in a **Black List** and stop following it. This may have as a result even the **social exclusion** of the VE. In this case, if the social indexes of the VE become better, then **social reintegration** will take place, as other VEs will start following the VE once more. The social exclusion and reintegration are a subject of the sensitivity of the actors to good or bad experiences from services. The component responsible for the renewal of the Friends Lists according to these indexes is the Friends Management component.

Since the social indexes will constantly change, it is quite important to take under consideration their **evolution**. Although a VE may have a low Reputation Index when we study a wide time-window, it may have a much greater Reputation Index when we study a smaller and recent time-window. This means that the specific VE is improving and this improvement should be evaluated fairly by the system and the community. For this reason, for each Followee in each Followees List the **timestamps** (unix time) and the evaluation of the last e.g. 3-10 interactions may be kept, so that, when applying simple rules, the evolution of the indexes can be studied.

The main functionalities of the Social Monitoring component were demonstrated during the two first reviews of the project.

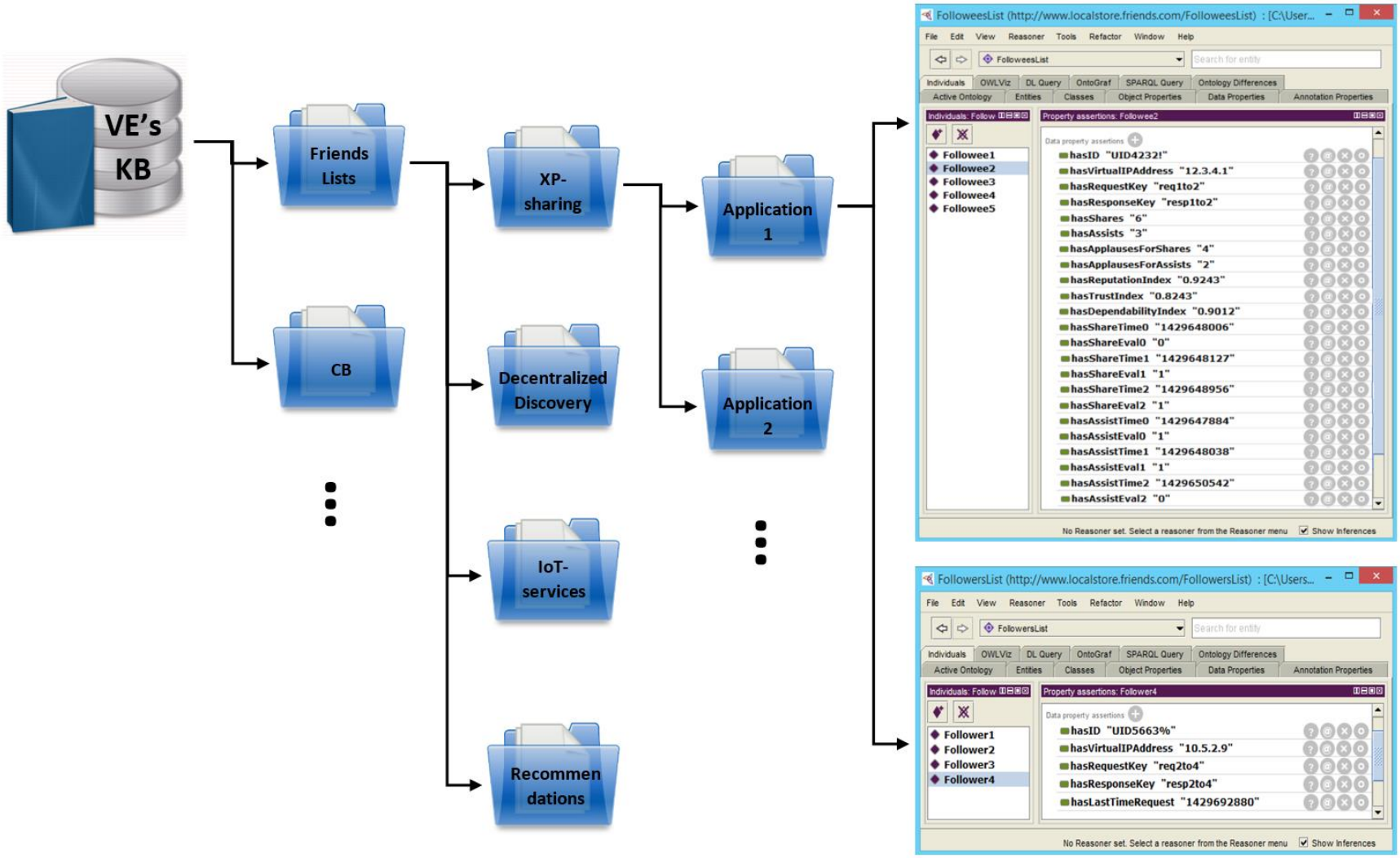

Figure 31: Example of a Followees List and a Followers List of a VE for XP-sharing.

8.2. Functionalities of the Social Analysis component

The VEs 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 levels (node level, group level or system wide) against a set of rules. The evaluation results can be used by the Planner or other COSMOS components (**internal use**) and can be used as a form of service for the users (**external 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 could 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:

- **Recommendation of VEs:** By finding the similarities between VEs, calculating their trustworthiness and reputation and identifying their needs, it is possible to produce many recommendation services. One representative example is the Friends Recommendation of COSMOS. This feature, can provide through both centralized and decentralized methods lists 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. The Social Links Establishment functionality through recommendations is presented in subsection 8.3.
- **Trust & Reputation Management:** The main problem that a Thing will face with the information acquired from its social circle is that it cannot always ensure its validity. There needs to be confidence over the information provider i.e. there needs to be Trust. This problem can be tackled by using a Trust and Reputation (T&R) model. In subsection 8.4 we introduce a new scalable hybrid T&R model for the SIoT, COSMOS TRM-SIoT.
- **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 colors, size and other advanced properties to nodes. Collaboration graphs could be used to illustrate good and bad relationships between VEs based on characteristics such as the evolution of the Trust and Reputation of the VEs. Some pre-existing tools that could be used for such services are presented in subsection 8.3.2.
- **Extraction of structural characteristics of the networks:** There are many properties of the networks that could be analyzed 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. 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. Some of these metrics and pre-existing tools that could be used for such services are presented in subsection 8.3.2.
- **Extraction of relational-models:** One other functionality of the Social Analysis component could be the extraction of Relational Models. Such models are the Communal Sharing, Equality Matching, Authority Ranking, Market pricing, as well as Parental relationship, Co-location and Co-work relationship, Ownership etc. The theoretical analysis of such models is presented in subsection 8.5.

8.3. Social Links Establishment/Recommendations

8.3.1. Followees Acquisition

The process of Followees acquisition begins at the phase of registration. The user can manually set the Followees Lists of the VE (e.g. in order to link his/her VEs with each other). This is the most basic way a VE forms social bonds. Such friends will have a number of benefits during the social monitoring or discovery phases (e.g. greater priority).

Another way of acquiring Followees is through a discovery mechanism, which is always 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 **Social Analysis component** of the COSMOS platform:

- **Neighborhood Method:** In this method the Thing ranks its friends in descending order of recommendation Trust indexes and asks them if they can suggest a potential service provider. This is done for one friend at a time until the Thing gathers several proposals. When the proposals are collected their (the potential service provider's) Reputation indexes are calculated and the most reputable one is kept, along with the Thing that proposed it. Then, the service is requested and, if the result is satisfactory, a new log entry is created to accommodate the new friend. The recommender is positively or negatively assessed based on the final satisfaction. In this case,
- **Platform Method:** The platform can have a partial view of the several interactions between the Things. Upon a proposal request the Platform uses the feedback received by the Things to calculate the Reputation indexes of the service provider and caches them to minimize the response time. After this calculation, the Platform returns one of the four (4) most reputable service providers to put a balanced load over them. If the proposed Thing returns a good service it will become a new friend.

A case of a Sybil attack with over 50% affected Things will lead to wrong proposals from the Platform since the malicious nodes will reverse the satisfaction in their feedback and propose their malicious friends as trustworthy. This problem is tackled by incorporating the random surfer notion [48] in the algorithm of the Platform. Hence the four (4) most reputable nodes have 80% chance to be proposed. The remaining 20% chance is for the Platform to propose a Thing with very low or 0 Reputation index. That way, even in the worst case scenario, the Platform will eventually propose a good service provider and after a few interactions there will be enough Trust from the Thing that requested the new friend and, as a result, the beliefs of the malicious society will be of no importance.

8.3.2. Recommendation Criteria

The choice of suitable Followees, no matter for which service they will be chosen, is based on three composite criteria: Relevance, Dependability and Structural Power. Relevance aggregates concepts like these of Homophily (Domain and Physical Entity attributes in the VEs' ontologies) and Distance Proximity (Location & GeoLocation), Dependability refers to Reputation, Trust and Reliability, while Structural Power is evaluated by several structural network characteristics. Our goal is to combine all these criteria in order to obtain the "Social Power" of a VE [49].

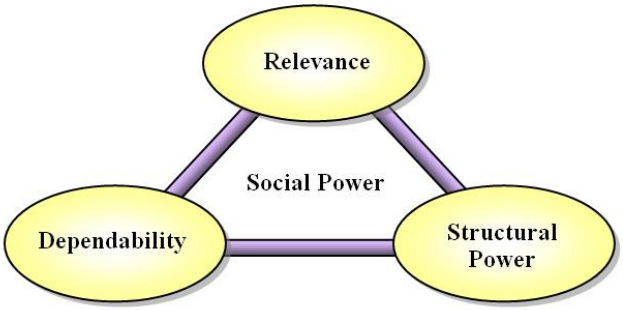

Figure 32: The Social Power of a Followee.

8.3.2.1 Relevance/(Dis)Similarity

The basic criterion for choosing a new Followee, which will most probably provide good and useful services, mainly depends on its similarity (regarding both its nature and its environment) to the VE asking for new Followees. As a result, two properties have to be studied:

- **Homophily/Heterophily:** the degree to which VEs and GVEs form ties with similar/dissimilar others. Similarity can be based on domain-dependent characteristics of the VEs as they are described in their ontologies. Homophily leads to the formation of homogeneous groups, where the relationships are easy to build. However, for VEs associated with more domains, heterophily is desirable. These two aspects aim at providing more beneficial friend recommendation services. For example, the homophily value is taken under consideration from friend recommendation services when the requests concern XP-sharing. In the case of requests concerning IoT-services or Decentralized Discovery, the opposite characteristic, the heterophily value, is extracted. In general, various recommendation algorithms could be developed incorporating both values [50].
- **Propinquity:** The tendency for VEs to have more ties with geographically close others. The mobility of Things should be taken under consideration [51].

8.3.2.2 Dependability

The Dependability Index of a VE can be calculated from: **i.** other VEs by taking under consideration the social evaluation provided by the community and **ii.** the COSMOS platform (central SA component) by gathering data from **a.** Followers directly or **b.** the COSMOS Registry.

Indicatively, the second developed solution is a platform specific service that is initiated by the SA component and entails the querying of Followers of a specific VE. We refer to this VE as “Evaluated 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 EvaluatedVE and then randomly decides which ones and how many of them to use as a querying basis (if their number is too great). 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 and Shares (and Assists or Mentions if applicable) for the EvaluatedVE from the Followees List or Black Lists 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 EvaluatedVE itself does not provide any information at all, but instead, all the information needed is offered by its social environment. More details are given in subsection 8.4.

8.3.2.3 Structural/Social Analysis Properties and Metrics

As we already discussed, the analysis and formalization of social relations among Things is critical and the computation of social measures is necessary. They improve the effectiveness of the system as they help in taking decisions at many levels. However, the analysis of social networks is basically related to their structural analysis. **Social Network Analysis (SNA)** is the analysis of social networks viewing social relationships in terms of network theory. These relationships are represented by nodes (individual actors within the network) and ties (relationships between the individuals). The computation of structural power is considered to be a step of high importance.

At a first level, it is important to refer to some Social Network Analysis properties and discuss the way they can help us identify the key nodes, the relationships strength (ties strength) and the cohesion of our social networks. There is a great variety of metrics that could be used under the functionalities of the Social Analysis, offering more detail and information about the networks being analyzed. The main metrics that have been identified and whose role is evident are:

- **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 centrality metrics include degree centrality, closeness centrality, betweenness centrality, eigenvector centrality, alpha centrality, etc. [52]. Centrality is one of the main metrics that should be taken under consideration from the recommendation services.
- **Distance (Shortest Path):** The minimum number of ties required to connect two particular VEs, as popularized by Stanley Milgram's small world experiment and the theory of 'six degrees of separation'. This theory introduces the idea that each node in a freely emerged network can be reached by propagating items of information via six hops. Distance is important as it is involved in various other measures, e.g. closeness centrality and betweenness centrality. Closeness centrality is distance-based and increases when the distance between nodes decreases, while betweenness centrality is a measure of the number of shortest paths in a network that traverse the node.
- **Tie Multiplexity:** The number of content-forms contained in a tie of a dyad of VEs, in other words how many relationships represents a tie. For example, two VEs that can share knowledge will have a tie with multiplexity of 1, whereas, in case they can share IoT-services too, they will have a tie with multiplexity of 2 and so on. Multiplexity is associated with relationship strength and durability and may be an indicator of network effectiveness. Some other kinds of relationships that can be defined and affect the multiplexity of ties are presented in [46] and are the Parental object relationship, the Ownership object relationship, the Co-location object relationship etc.
- **Cohesion:** The degree to which VEs are connected directly to each other by cohesive bonds. Structural cohesion refers to the minimum number of members who, if removed from a group, would disconnect the group. This characteristic is quite important when reconfiguration of a group of VEs must take place.
- **Density:** The proportion of direct ties in a network relative to the maximum possible number of them. When density is close to 1, the network is dense and can resist to tie failures more easily, otherwise it is sparse. For density 1 the network is called a clique. Density is related to the speed with which information is diffused among the actors and is useful in comparing networks or different regions of a single network.
- **Centralization:** An aggregate metric that characterizes the amount to which the network is centered on one or a few important nodes.

- **Clustering coefficient:** A measure of the likelihood that two randomly selected neighbors of a node are connected to each other. It represents the density of a node's neighborhood. A higher clustering coefficient indicates a greater 'cliquishness' [53]. For an entire network it is computed as the average of all its nodes' clustering coefficients and represents the tendency to form clusters and groups.
- **Structural holes & Bridges (Mediators):** The structural hole concept, developed by sociologist Ronald Burt and sometimes called social capital [54], refers to the lack of ties between two parts of a network. The structural holes theory introduces the concept of bridges or mediators as individual actors that fill a structural hole, thus connecting two previously unconnected (or at least loosely coupled) actors and gain valuable insights in others work. Finding and exploiting a structural hole can offer novel and competitive innovation opportunities [55]. In our approach, mediators could be used to facilitate the communication between VEs and GVEs. Mediators can influence partners and build high reputation. Structural holes & Bridges can enable effective decentralized discovery mechanisms.

Other network level analysis metrics include average distance (average distance between all pairs of nodes), metrics that integrate attribute data with network data (for example, metrics that measure homophily), etc.

The structural metrics discussed above give us the opportunity to identify the key VEs in the network. The centrality metrics bring out the nodes of great 'importance'. For example, the degree centrality is a measure of a VE's connectedness and represents the number of friends it has in its neighborhood. The betweenness centrality (BC) is a measure of how often a VE is the most direct route between two other VEs and represents its potential to act as mediator. The closeness centrality (CC) represents how fast a VE reach every other in the network, whereas the eigenvector centrality (EC) represents how well a VE is connected to other well-connected VEs.

Of course, the importance of a VE depends on the context and use case. For example, in the home automation domain, high degree centrality may indicate the key room in a building, in the smart city domain, VEs of high betweenness that bridge various communities may be of great importance, while in the transport domain, VEs of high eigenvector that influence the whole network may be of greater significance (Table 1). The various metrics are correlated and they do not necessarily have the same tension. A VE with high eigenvector may not have high closeness and/or high betweenness, meaning that it does not have the greatest local influence and/or it has low bridgering potential.

Table 2: Types of Centrality and VEs' Roles.

Metrics	High BC	High CC	High EC
VE Roles	Mediator	Group Leader	Network Influencer

Reciprocity and multiplexity metrics bring out the tie strength. However, the aspects of heterogeneity, such as diversified bonds and/or dissimilar nodes, add more complexity. SNA could be enhanced by the use of tie weights (weighted network). Ties weighted in relation to frequency of interaction, influence, capacity etc. provide a more real world indication of the dynamics of a particular network. They affect the path that information takes, the speed it travels and may characterize the nodes that use them as more or less key ones. Thus, centrality measure results are affected by tie weightings. Moreover, centrality is affected by ties between

dissimilar VEs (multimodal network). A metric such as betweenness centrality applies to uni-modal networks and there is no clear definition in a multimodal one. All these mean that new algorithms are required for the calculation of centrality for weighted and multimodal networks.

At this point, it should be noted that, while some VEs relationships will be defined by the users, a great percentage of them will have to be appointed by the COSMOS platform. As a result, the case we study here is not just of an independent system from which some metrics are extracted for research, but instead a system where many times new relationships are appointed dynamically depending on known desired values of these metrics. In other words, instead of extracting the properties of an existing graph, we create a graph based on the desired values of the properties. Consequently, the creation and maintenance of the VEs' social environment is reduced to determining the proper metrics and their corresponding values.

In general, VEs' social networks will be self-organized, emergent and complex [56], 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 the 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 analyzed at the proper scale, depending on the application or the needs of a user or a functional component of the platform. Generally, there are three scale-levels into which networks may fall: micro-level, meso-level and macro-level [57]. At the micro-level, social network research typically begins with tracing inter-VE interactions in a small group of a particular domain. Meso-level analysis begins with a population size that falls between the micro- and macro-levels and may also refer to analysis that is specifically designed to reveal connections between micro- and macro-levels. At last, macro level analysis generally traces the outcomes of interactions over a large population. It becomes quite evident that, in case we have to take decisions at system level regarding e.g. the reconfiguration of some entities, this kind of analysis becomes really important.

8.3.2.4 Computational Models and Social Network Tools

There are many tools that can be used from the Social Analysis component and a need for the development of new tools may exist. Some of the main tools that we have taken under consideration are:

- **Representation Formats, markup languages and ontologies:** DyNetML [58] and GraphML [59] could be used as a reference model.
- **Dynamic Network Analysis and Social Network Analysis:** Tools for reasoning under varying levels of uncertainty about dynamic networks, their vulnerabilities and their ability to reconfigure themselves, choosing **Dynamic Network Analysis (DNA)** metrics and then using one or more of the available optimizers to find a design that more closely meets an ideal as well as exploring network graphs. The dynamic network visualization has been a challenging topic due to the complexity introduced by the extra dimension of time [60]. Some tools that have been studied are ORA, IGraph, Networkx and Pajek [61], [62].

The tool that has proven to be the most useful to us is **Gephi** [63]. Gephi is an open-source and free interactive visualization and exploration platform for all kinds of networks and complex systems, dynamic and hierarchical graphs. Its statistics and metrics framework offer the most common metrics for SNA and scale-free networks. Gephi is the ideal platform for DNA, since Dynamic structures, such as social networks can be filtered with the timeline component.

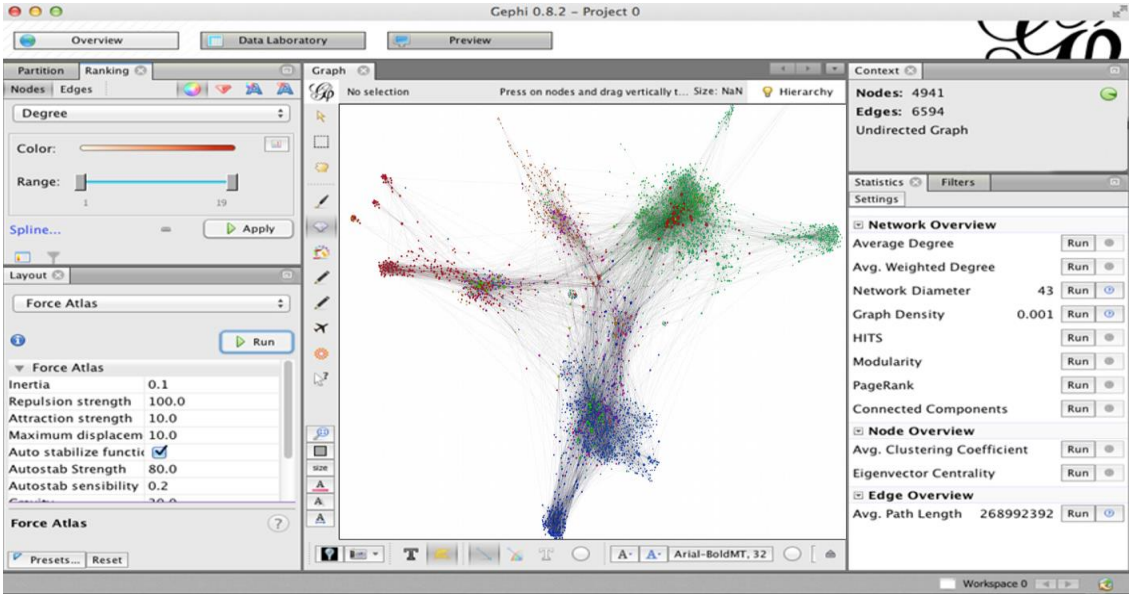

Figure 33: Network Analysis and Visualization using Gephi.

8.3.2.5 Finding similar VEs using ML

During Year 3, the idea of finding similar VEs for experience sharing using machine learning is introduced. It is evident that the behavior and actions of VEs will be more relevant to the VEs having similar characteristics and hence experience sharing will be more optimized in such cases. In this regard, we proposed to **group** VEs with similar characteristics using clustering mechanisms. Clustering is an unsupervised ML technique which is used for grouping similar objects on the basis of pre-defined metrics such as distance or density. There are several variants of clustering algorithms found in the literature. The choice of a particular clustering algorithm and parameters (such as type of metric, threshold and number of clusters) is driven by the problem scenario and related data-sets. Cluster analysis generates a hypothesis about the given data, whether the data under observation have distinct classes, or overlapping classes or classes with fuzzy nature.

As an example, consider XP sharing between different smart-homes for a heating schedule (see Section 11). It will be more optimized to share the experience between two flats which are located closer to each other and have similar characteristics (such as their size, the number of rooms, the floor location, their exposure to sunlight, number of occupants). For example, a heating schedule for a 2-bedroom flat located in London on the top floor of a building facing the sun cannot be applied efficiently to a flat located in Madrid which has a single bed room on the ground floor and is not facing the sun.

This is one way of looking at the problem, where we assume prior knowledge about VEs **characteristics** is given and using it we can group the various entities into similar groups. Such a source of information can be the COSMOS Registry.

Another approach to the same problem is to group the VEs depending on the pattern of their **data usage**. In this way, similar VEs can be grouped together even if knowledge about their characteristics is not given. In the previous scenario, the flats with similar electricity and heating consumption patterns will be grouped together. Flats with high consumption of electricity and heating energy data will be in a same group whereas flats with low energy usage will be in a same group.

To sum up, there are two approaches for clustering we can follow:

- **Finding similar VEs on the basis of static characteristics:** In this method we exploit static characteristics of a VE which can be provided during the registration phase of the VE (from its ontology). It does not need to be updated often unless the location of the VE is changed.
- **Finding similar VEs on the basis of dynamic data:** Grouping the VEs on the basis of the data they are measuring is more complex. In this regard, there are two options:
  a) **Time series clustering:** In this method, VEs with similar time series pattern usage over the time are grouped together.
  b) **Clustering on aggregated data:** In this method, data is aggregated over defined time such as a day, a week or a month and then clustering is applied on the aggregated data.

8.3.3. Social Contracts and Contacts Maintenance

When a Followee(s) recommendation request is sent to the platform and a new VE is chosen by the SA, the platform has to act as an intermediator to help the two VEs form a new bond and agree on a social contract. As such, the platform generates two passwords (a password and a countersign) and forwards them both with the corresponding VIP to the two VEs (the one asking for new Followees and the one recommended as one new Followee). The passwords do not have to be too complicated or even unique (Figure 31) since brute-force cannot be used against them as, once a wrong key is given during a request or a response to a request, the system/community gets informed. Through this mechanism, many security threats can be tackled. This concept is analyzed in D3.1.2 too. It should be noticed that these keys cannot be used at the opposite direction since the Follower-Followee relation is not symmetric. This means that in the case of two mutual Friends, four keys will be needed.

When the platform sends to the initial VE the VIP of the new Followee and to the new Followee the VIP of the initial VE, then the corresponding Followers and Followees lists are filled in. In order for a VE that changes its VIP dynamically to re-establish its bonds, it can forward its Friends Lists to the platform which will act yet again as an intermediator and re-establish the old connections by informing the Followers and Followees of the VE for this change. Of course, the platform has to make sure that the new VIP really belongs to the older VE (the platform can query the Friends presented in the lists; this information is only available to the original VE, unless it gets hacked).

A **decentralized approach** for the creation of social contracts can be used too. In this case, the intermediator helping two VEs to have their first contact will be another VE (probably a common Friend of theirs). The intermediator will create the two simple keys and forward them together with the corresponding VIPs to the two VEs. The two VEs, after their first contact, will agree on two new keys so that no other VE (even the intermediator) will be aware of them

While designing these mechanisms, the concept of **Opportunistic IoT** [64] should be taken under consideration. VEs (and their owners) should have some motives to agree on sharing their services with other entities of the network. A simple model would be to set a rule according to which a VE may access services of other VEs if and only if it provides some services of its own to the other VEs too. This leads to the idea that the social contracts could be used as a form of coin and the transactions between VEs could be "**monetized**" [65].

8.4. Trust & Reputation Management

8.4.1. Introduction

Trust and Reputation (T&R) models have been recently proposed by many researches as an innovative solution for guaranteeing a minimum level of security between two entities of a distributed system that want to have a transaction or interaction. Thus, many studies, works and models have been designed, carried out and developed in this direction, leading to a current solid research field on which both academia and industry are focusing their attention. Many methods, technologies and mechanisms have been proposed in order to manage and model trust and reputation in systems such as P2P networks [66], ad-hoc ones [67], wireless sensor networks [67] or even multi-agent systems [69]. Such methods have been used in many environments like P2P networks, Wireless Sensor Networks (WSN), Vehicular Ad-hoc Networks (VANETs), Identity Management Systems, Collaborative Intrusion Detection Networks (CIDN), Cloud Computing Systems, Application Stores and of course the IoT.

T&R 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. Most of the current T&R models in the literature follow four general steps which are described by Marti and Garcia-Molina [70] (Figure 34):

1. **Collecting information** about a certain participant in the community by asking other users their opinions or recommendations about that peer.
2. **Aggregating all the received information** properly and somehow computing a score for every peer in the network.
3. **Selecting the most trustworthy or reputable entity** in the community providing a certain service and effectively having an interaction with it, assessing a posteriori the satisfaction of the user with the received service.
4. **Punishing or rewarding** according to the satisfaction obtained, adjusting consequently the global trust (or reputation) deposited in the selected service provider.

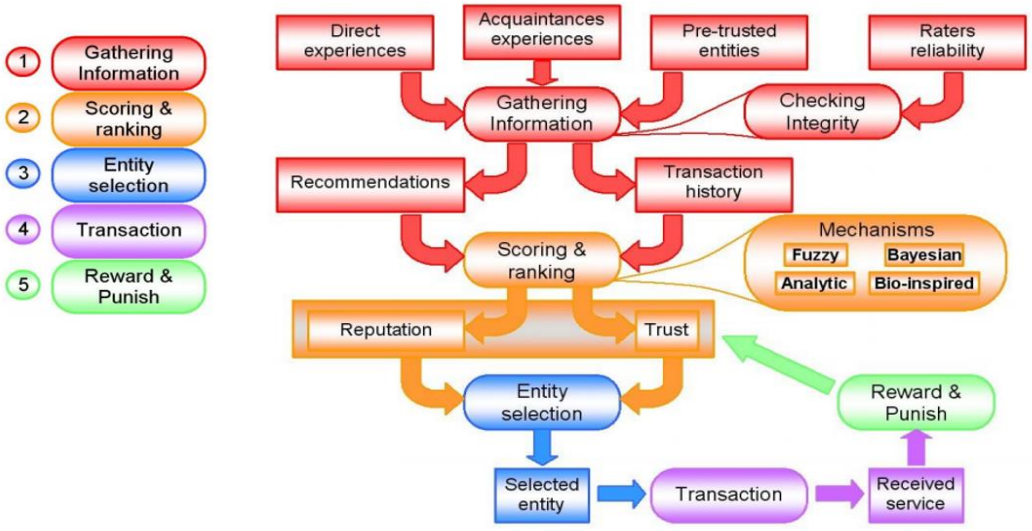

Figure 34: General steps followed in T&R models.

8.4.2. The COSMOS T&R model: TRM-SIoT

Different models manage concepts such as Trust or Reputation in many different ways. For instance, some models like PTM [66], [71] or AFRAS [72] make use of fuzzy logic in order to deal with those topics, Bayesian networks are used by MTrust [73] and BNBTM [74], bio-inspired algorithms are used in AntRep [75] and TACS [76], while other models like Eigentrust [77] or PeerTrust [78] just give some analytic expressions. Although there are some generic data structures for the domain of T&R provided for example by the Open Reputation Management Systems (ORMS) of OASIS [79], **there are no standards** for concepts like Trust and Reputation. In this subsection we try to provide some clear definitions of the main concepts that build up the COSMOS T&R model, TRM-SIoT, and the main features that characterize it. In TRM-SIoT we define Trust and Reputation as follows:

- **Trust:** The expectation that an interaction will be satisfactory based on *our personal experience*.
- **Reputation:** The belief that an interaction will be satisfactory based on the experience of *our social circle*.

Node A will have a high Trust index for Node B if the services provided from Node B to Node A have been evaluated from Node A positively. Node A will have a high Reputation index for Node B if the services provided from Node B have been evaluated from the social circle of Node A positively.

8.4.2.1 Definitions

The distinction between a trust and a reputation model is not always clear. However, in our opinion, those models making an explicit use of other participants' recommendations could be categorized as reputation models while the rest could be considered just as trust models.

Let's assume that VE1 wants to find out some social characteristics of VE2 for a specific service offered. The following terms can then be defined:

- **Popularity (P):** A counter which monitors how many times VE2 has received or may receive a request (how many "hits" it has). The Popularity Index is a property which indicates the total Shares and Mentions VE2 has. It is an accumulative and comparative indicator, and is used to determine the stability of Reputation and Trust.
- **Trust (T):** The belief of VE1 that VE2 is going to deliver the correct service based on previous interactions of VE1 with VE2. The Trust Index of VE2 provided by VE1 is a property which states how many times VE2 has successfully shared its services with VE1 and it can be calculated as the ratio of the Applauses from VE1 to the Shares to VE1. Trust is subjective, because it is estimated from perspective of the individual trustor (VE1 in this case).
- **Reputation (R):** The belief of VE1 that VE2 is going to deliver the correct service based on previous interactions of other VEs (in the social neighbor of VE1) with VE2. The Reputation Index can be calculated from the Trust that other VEs (apart from VE1) have on VE2. In other words, this metric determines the belief of others on a VE and is useful especially when VE1 does not enough data to extract a Trust Index for VE2 (because e.g. there are no interactions between the two VEs yet).
- **Reliability (R')**: An absolute indicator of the performance of the VE that quantifies its efficiency to offer successfully its services relatively to its ideal or normal operation. The Reliability Index should be based on criteria like: response time upon request, ability to communicate (depending on how constrained is the VE or how constrained are the sensors/actuators associated with the VE), quality of service provided, etc.

- **Dependability (D):** A social measure combining all the above social measures. It can be simply derived by the expression $\boldsymbol{D = a \cdot T + b \cdot R + c \cdot R' + d \cdot P}$ where a, b, c and d (non-negative integers) are the weights of the measures and $\boldsymbol{a + b + c + d = 1}$. For this calculation, Popularity has to get normalized. By selecting the appropriate weights, we can provide the expression of the Dependability Index that we want. For example, when there are only a few interactions between VE1 and VE2, then the Trust Index should have a low weight and the Reputation Index should have a high weight. This means that the weights should change dynamically and be set according to the users or developers preferences.

8.4.2.2 General Features

Reputation connects closely to the concept of Trust, but there is a clear difference, which can be illustrated by the following two scenarios:

- VE1 trusts VE2 because VE2 has a good Reputation. This reflects that Reputation can be used to build Trust.
- VE1 trusts VE2 despite the bad Reputation of VE2. This reflects that even if VE1 knows the Reputation of VE2, VE1 has its own private knowledge (e.g. direct experience about VE2) which is considered to be more important

Generally, a VE can be evaluated only by information gathered from other VEs. Its Dependability can be calculated by each and every other VE of the community (a subjective estimation) or by the platform (a more, but not totally, objective estimation). Depending on its (subjective or objective) Dependability and a threshold set by the user or the (VE or Application) developer, it can be chosen as a member of the Friends Lists or the Black Lists of other VEs. In case the VE is added to the Black List of another VE, a given amount of time has to pass for its redemption to take place.

Both big and small time-windows are used to quickly detect malicious or unsatisfactory behavior and avoid the fast redemption of blacklisted VEs. Moreover, feedback from recent interactions has a higher weight than this of older actions.

Benevolent VEs should have more **opportunities** than newcomers. As a result, newcomers with 0 interactions with other VEs will have Reputation equal to 0. However, an extra rule has to be applied to the model we have designed to give the opportunity to newcomers that have a low Reputation (because of the small number of interactions with other VEs) to be chosen as service providers at some point and start building their Reputation. For example, 10% of the recommendations from the platform should introduce newcomers to the rest of the community. The same applies for VEs which have low Reputation due to malicious or unsatisfactory behavior in the past. In other words, this rule enables the **social integration** and **reintegration** of the VEs to the system. Moreover, this rule is necessary for the first moments of the social community that will be born from COSMOS, as the network, at its **initial state**, will not have any VEs with high Reputation.

It should be noticed that, in contrast with many T&R models, we choose to use **different** Trust and Reputation **scores** for different services provided by the members of the network. This feature helps as face quite many security threats, described in subsection 8.4.4. For example, abuse of a high achieved Reputation is easily avoided.

From all the T&R models we have studied, the one that is the most similar to the model we propose is SORT [80], a self-organizing trust model for P2P systems.

8.4.3. Calculation of Trust & Reputation

8.4.3.1 The Trust Metric

In TRM-SIoT only the idea of subjective Trust is modelled, as we claim that subjectiveness is embedded in Trust's meaning. Strong Trust on a Thing cannot and should not be affected by claims of a third party. In order to model Trust, the experiences based on which the Trust is calculated need to be modelled. Thus, we need memory. Each Thing has log files where its experiences are stored (Figure 31). The structure of the log files is crucial. In order to deal with the malicious nodes that oscillate their behavior depending on the service they provide, one log file per service is kept. Moreover, the experiences are also grouped per service provider for lower computational overhead. It should be noted that Trust on a Thing for a service provision is decoupled from Trust on the same Thing for recommendations provision. This should be done because providing recommendations is a subjective act based on personal criteria and should not be mixed with the evaluation of a service provider. Also, this way, the detection of malicious recommendation behavior can be accomplished even in the case of good service providers. Some crucial attributes that have to be stored in the Log Files are:

- **Satisfaction (s):** This value is essentially a subjective QoS indicator and is being monitored by the Applause metrics. The Satisfaction is automatically derived by the absolute values of the service based on their correctness. For example, a sensor that suddenly reports a really high temperature will be assigned a satisfaction rating based on the correctness of this report. If it happens to be a fire, the Satisfaction is high, but if that is not the case, the Satisfaction is zero. Since a Thing that regulates alarms can consult more than one sensors, a malicious or faulty sensor will quickly lose any trust. If the sensor is fixed, the Social Reintegration part of the system will allow it to build trust again as we mention below.
- **Weight (w):** This is a value indicating how crucial the service is for the well-being of the Thing. It is used in order to prevent a malicious Thing from providing a minor service well and then exploiting the built Trust and providing a crucial service poorly. Due to this value, it is difficult for the Trust index to increase just because of minor services, whereas it can drop quickly in case of a crucial service with low quality.
- **Fading factor (f):** When new interactions take place, the importance of older ones should decrease. The fading factor addresses this issue and forces peers to stay consistent with regard to their previous behavior. Old interactions have lower fading factor values, so a node cannot misbehave relying on its good history, otherwise its malicious behavior would be detected and the node would be quarantined from its social circle and be deleted from the corresponding Followees Lists (Social Exclusion). Furthermore, the fading factor is used to increase the scalability of the system, since interactions that get a fading factor of 0 are deleted from the Log Files. The fading factor makes the Social Reintegration of ex-malicious nodes possible, meaning that if they become benevolent, it is possible for them to get a second chance and form new ties within the network. Of course multiple incidents of misbehavior can get a node permanently black-listed. This fading factor can be set by the system administrator so that the node stores the last N interactions with any other node. It is decreased every time a new interaction is attempted. If a node is unavailable for a significant period of time, the fading factor will lead to the deletion of any previous history, so that new nodes get chosen for the same service. The N value determines how fast oblivion comes. If the node becomes active again, then the Social Integration mechanism together with its good behavior will lead to it building its Trust and Reputation scores again. Finally, the owner of the Thing has the ability to pin friends so that they are preferred even if the Trust value is zero, so that they are not forgotten because of the fading factor.

When a Thing wants to calculate the **Trust Index** of a Followee, it looks into the appropriate Log File and calculates the trust value as the weighted average of the log entries using:

$$\mu_t^k = \frac{\sum_{i=1}^{N}(s_i \cdot w_i \cdot f_i)}{W} \quad (1)$$

where W is the normalization co-efficient which ensures that the trust value will be between [0,1] and is calculated by:

$$W = \sum_{i=1}^{N}(w_i \cdot f_i) \quad (2)$$

The **mean value (μ)** is a measure of the overall observed behavior of the Followee and indicates the expected satisfaction value of the next interaction. However, it is needed to know how confident we can be about the value of μ i.e. how much the satisfaction from the service may actually deviate from μ. Thus, the **standard deviation (σ)** of the behavior is also calculated. To reduce the computational overhead, the calculation of the later occurs simultaneously with the calculation of the mean value following the formula:

$$\sigma_t^k = \frac{\sqrt{\sum_{i=1}^{N}(s_i{}^2 \cdot w_i \cdot f_i) \cdot W - \left(\sum_{i=1}^{N}(s_i \cdot w_i \cdot f_i)\right)^2}}{W} \quad (3)$$

Finally, we define Trust as:

$$\boldsymbol{T^k = \mu_t^k - \sigma_t^k} \quad (4)$$

To sum up, μ shows the satisfaction that VE1 should expect from VE2, while σ shows how predictable the behavior of VE2 is. This means that if T = 0.5 then there is an 84% probability that the satisfaction for the service will be 0.5 or greater. That way the service providers that are not consistent and have an ever changing and oscillating behavior will have lower Trust indexes even if their μ value is higher.

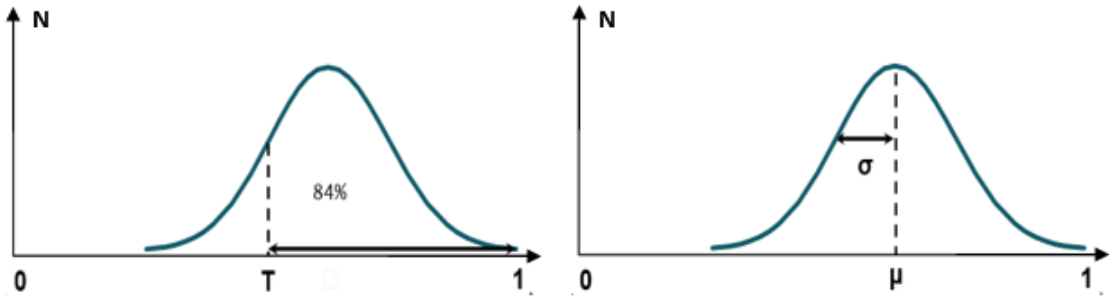

Figure 35: Calculation of the Trust Index of a VE.

When a Thing decides that it needs a service, it knows how crucial for its well-being this service is. Thus, it can decide how willing it is to risk being exposed to malicious acts. This value is used as a threshold for the Trust index. For example, if this value is 70% (low risk), the Thing is going to request the service only from its friends with Trust index of 0.7 or higher.

After defining the threshold for the requested service, the Thing attempts to find a suitable service provider from its Followees List by calculating the corresponding Trust indexes. While doing so, the Thing checks if size(log_file) > 5, where **size(log_file)** is the number of interactions for the specific service between the Thing and its Followee. If this is true, it calculates **two different Trust indexes**; one for a small time-window and one for a big time-window.

Using equations (1)–(4), it calculates $\mathbf{T_{long}}$ from N=size(log_file) interactions (all the interactions) and $\mathbf{T_{short}}$ from the last N=[size(log_file)/10] interactions Finally, the Trust is calculated as:

$$\boldsymbol{T = min(T_{long}, T_{short})}$$

T_{long} shows the long term satisfaction on the service provider while T_{short} shows its recent trend. By assigning Trust as the minimum of these two values, the Thing manages not only to quickly detect the service deterioration of a trustworthy service provider, but also to remain unaffected by the sudden amelioration of the service of an ex-malicious provider, since it is protected from temporal changes by the value of T_{long}.

After the calculation of the Trust indexes the Thing decides to send the service request to its most trustworthy Followee, unless its Trust index is below the threshold that was set. There is no need for load balancing since every Thing will probably have a different service provider as the most trustworthy one. If the service is not critical for the Thing, TRM-SIoT gives a 10% chance for the Thing to trust a new Followee whose Trust index is 0 or really low. This is done in order to allow **Social Integration** of new active nodes and Social Reintegration of ex-malicious nodes.

If the above procedure fails due to an insufficient number of Followees, the Thing can either propagate a service request message to its social circle as described in 8.3.1 and wait for one of its Followees to indicate a more trustworthy service provider or search for new Followees.

8.4.3.2 The Reputation Metric

In TRM-SIoT, the Reputation of each node is not calculated globally. Thus, if two Things have a **Reputation index** for a third one, the two values most probably will not be the same. Hence, the Reputation index is also individual information for each Thing. In order to ensure that each Thing can extract the Reputation index of another Thing, we present a novel mechanism for Reputation calculation. The Reputation may be extracted through two independent methods:

- **Neighborhood method:** The main method used. It follows a decentralized approach, thus avoiding any bottlenecks on the system's performance. Each Thing can use its social circle to find suitable information regarding another Thing.
  - ✓ **Step 1:** Node A finds the 4 best recommenders in its Followees List that are also connected with B and asks for the Trust index they have for Node B. Then, the recommenders return the Trust index (T) for Node B, and Node A collects this information and calculates an initial Reputation distribution using equations similar to (1), (2), (3), where the weight is A's Trust in each recommender ($\mathbf{T_{rec}}$). Consequently:

$$\mu_r^k = \frac{\sum_{i=1}^{N}\left(T_i^k \cdot T_{rec}^i\right)}{W'} \qquad (5)$$

and

$$\sigma_r^k = \frac{\sqrt{\sum_{i=1}^{N}\left(T_i^2 \cdot T_{rec}^i\right) \cdot W' - \left(\sum_{i=1}^{N}\left(T_i^k \cdot T_{rec}^i\right)\right)^2}}{W'} \qquad (6)$$

where

$$W' = \sum_{i=1}^{N} T_{rec}^i \qquad (7)$$

The Reputation index is defined as:

$$\boldsymbol{R^k = \mu_r^k - \sigma_r^k} \quad (\mathbf{8})$$

✓ **Step 2:** Having computed the initial reputation distribution (μ_r, σ_r), Node A asks the rest of its Followees (recommenders) about their experience with Node B, until it collects a sufficient number of opinions. However, these opinions are not automatically accepted. First of all, Node A attempts to exclude malicious recommendations. Assuming that the distribution is normal, we want the feedback to be at the non-malicious range [μ_r–0.7·σ_r, μ_r+0.7·σ_r]. In retrospect, statistical analysis for each system can take place and the range can change to become more accurate. After collecting this feedback, Node A continues with the calculation of the final Reputation distribution by integrating the new benevolent recommendations into equations (5)-(8). Finally, the initial recommenders used in step 1 are also evaluated based on the final Reputation values.

It should be noted that when the recommenders forward to Node A their Trust values for Node B, they do not reveal their μ and σ values. As a result, the manipulation of their Trust value by a malicious friend becomes a lot harder.

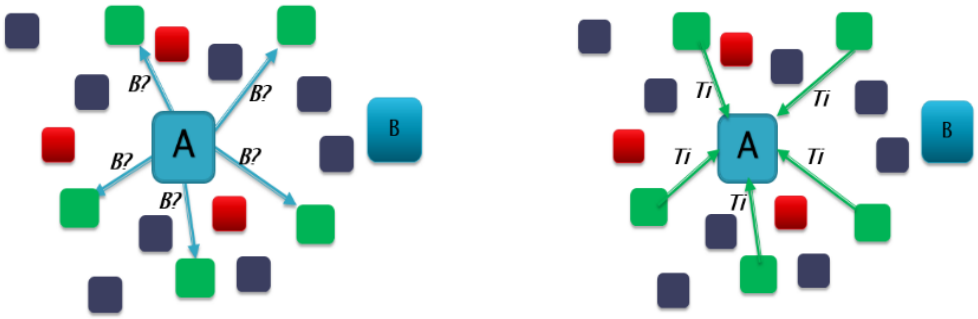

Figure 36: Step 1 in Neighborhood method - Ask close friends/recommenders.

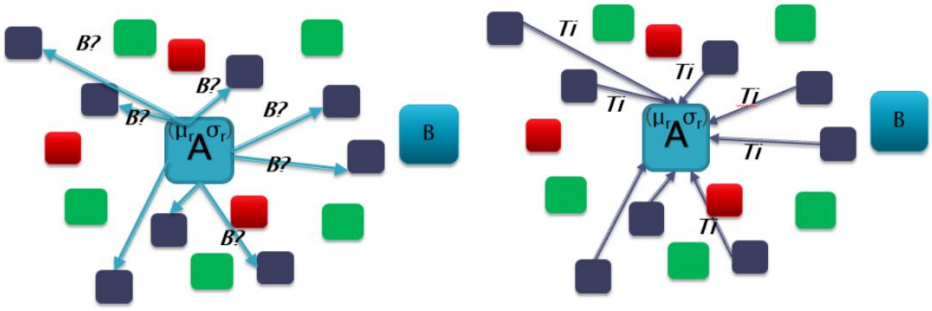

Figure 37: Step 2 in Neighborhood method - Ask and judge the social circle.

- **Platform method:** This method is used when the social circle cannot provide feedback for a Thing, as it may be new to this specific neighborhood. The Platform, which has a wider view of the VE-ecosystem, is asked to provide an index. One of the services that the Platform (centralized management mechanism) offers is calculating Reputation indexes for Things that cannot compute them with the neighborhood method due to lack of sufficient feedback from their social circle. The platform is aware of the activity of all the Things that have registered to it. However, in order to be able to manage the huge number of Things, it has only partial overview of the interactions between them by receiving, at random time intervals information from Things about their experiences (e.g. parts of their Log Files), in the simple form of Trust indexes. Using as weight the centrality of a node (how many friends it has) the same equations as above can be applied to extract Reputation indexes. This index is not entirely "correct", but it can show the trend of the behavior of a Thing.

8.4.4. Security Threats Scenarios

Trust and reputation management over distributed and heterogeneous systems has emerged in the last few years as a novel and accurate way of dealing with some security risks related to these environments. Thus, many models and theories have been developed in order to manage T&R in those communities effectively and accurately. Nevertheless, very few of them take into consideration all the possible security threats that can compromise the system and most models involve the arising of new threats that should not be underestimated.

In [81] Marmol and Perez, in order to provide a reference guide for developing secure trust and reputation models, present and describe the most common security threats applicable in the field of T&R management over distributed environments. Every accurate and T&R model should have some mechanisms to effectively overcome all the threats that could be applied to it. After studying several T&R models like EigenTrust [77], PeerTrust [78] and PowerTrust [82], we realized that not all the models address all the possible threats and, in fact, some of them do not even deal with these risks at all. In our opinion, this is an issue that should not be underestimated when designing and developing a new T&R model over distributed and heterogeneous systems, since an inaccurate management of these threats could result in important security deficiencies and weaknesses.

In the next subsections, we analyze the main security threats that can be applied to our T&R scheme. Moreover, we develop a complete taxonomy of those threats by describing several possible categories of attacks over our T&R system and categorizing them according to those properties. Finally, we discuss the solutions that our model offers for overcoming these threats and will suggest possible extra ways of tackling each one of these risks in the design phase.

The model we provide can be tested through simulations. Experimental results will be necessary in the future to find out how it reacts against certain attacks.

It should be stressed out that these kinds of security threats are strongly linked to the T&R model we designed. Therefore, their analysis must be done within the “jurisdiction” of WP5 and not this of WP3 which focuses on hardware security, cloud storage security and the concept of Privelets. Of course, there are some cases that discussions between the two WPs become necessary. One representative example is the threat of Sibyl attacks which will be described in the next sub-section.

For the analysis that follows, we consider scenarios where several participants (entities, nodes, peers) belong to a virtual community where a certain set of services is offered. When a specific participant is requested to provide one of the services it offers it can either provide the offered service with a good quality and act in a benevolent way or provide the offered service with a bad quality and act fraudulently or maliciously.

Special attention should be given to the fact that not all the bad services are a product of maliciousness, but may also result from wrong applications logic or malfunctions. For example, when a VE shares “wrong” cases, this does not mean necessarily that it is a malicious VE, since these cases may have been produced while sensors of the VE were malfunctioning. However, this detail does not affect the analysis we present later on.

Finally, we also make a distinction between the recommendation services and the rest of the services provided by the VEs. When a VE provides recommendation services, it will be considered to be a **Recommender**, while, when it provides any other kind of services, it will be considered to be a **Service Provider**.

8.4.4.1 Description of Security Threats Scenarios

The main threats that have been identified are:

a) **Individual malicious VEs:** A malicious VE always provides false services and information when selected as a service provider or a recommender. This is the simplest threat that can be found in a trust and reputation system (Figure 38a).
b) **Malicious groups of VEs:** Malicious VEs that provide bad services collude to unfairly provide high ratings about each other. That way the platform or individual VEs of the network can be manipulated to believe that the collective is trustworthy. Not many trust and reputation models treat the problem arisen from the constitution of a collusion among malicious peers, having thus an important security deficiency (Figure 38b).
c) **Malicious VEs with camouflage:** When malicious VEs are selected as service providers or recommenders, they provide bad services and recommendations in p% of all requests. This is, in many cases, a threat which is not always easy to tackle, since its resilience will mostly depend on the behavioral pattern followed by malicious peers. For instance, it is not the same battling against an oscillating pattern (being fully benevolent for a period of time, fully fraudulent for the next period and so on) with battling against an increasing and decreasing one or even a random one (Figure 38c).
d) **Partially malicious VEs:** A partially malicious VE for certain services it behaves properly while for other specific services it acts maliciously. For example, a VE could increase its reputation by providing good services to many requests of minor importance while it would provide bad services to a few important requests. In such a situation some distortion can emerge, considering a peer as fully or quite benevolent although it can also provide some fraudulent services (Figure 38d).
e) **Malicious Spies:** Malicious spies always provide good services when selected as service providers, but they also give the maximum rating values to malicious nodes too. In this threat, the malicious spies may gain a high level of trust and reputation, since they always provide good services. Then they may be able then to easily subvert the trust and reputation mechanism applied in the system (Figure 38e).
f) **Sybil Attack:** In this case, an attacker may initiate a disproportionate number of malicious VEs in the network. Each time one of these VEs is selected as a service provider or recommender, it provides a bad service and, after its reputation gets low, it is disconnected and replaced with a new peer identity [83] (Figure 38f).
g) **Slanderers:** Malicious VEs that provide bad services collude to unfairly provide high ratings about each other, but also to provide low ratings about benevolent VEs. In such a situation, if an interaction has never been performed with a VE which is actually benevolent but whose reputation has been driven down by malicious participants, then that VE will not probably be chosen as the one to have an interaction with (Figure 38g).
h) **Man in the middle attack:** In this case, a malicious VE intercepts the messages from a benevolent service provider VE to the requestor and replaces them with bad services, making therefore the reputation of the benevolent VE to decrease. This is a great threat especially when the COSMOS decentralized discovery mechanisms are used, like in the case of XP-sharing with TTL>1 (Figure 38h).
i) **Malicious pre-trusted VEs:** Pre-trusted VEs that provide positive recommendations about malicious nodes and negative ones about benevolent nodes (Figure 38i).

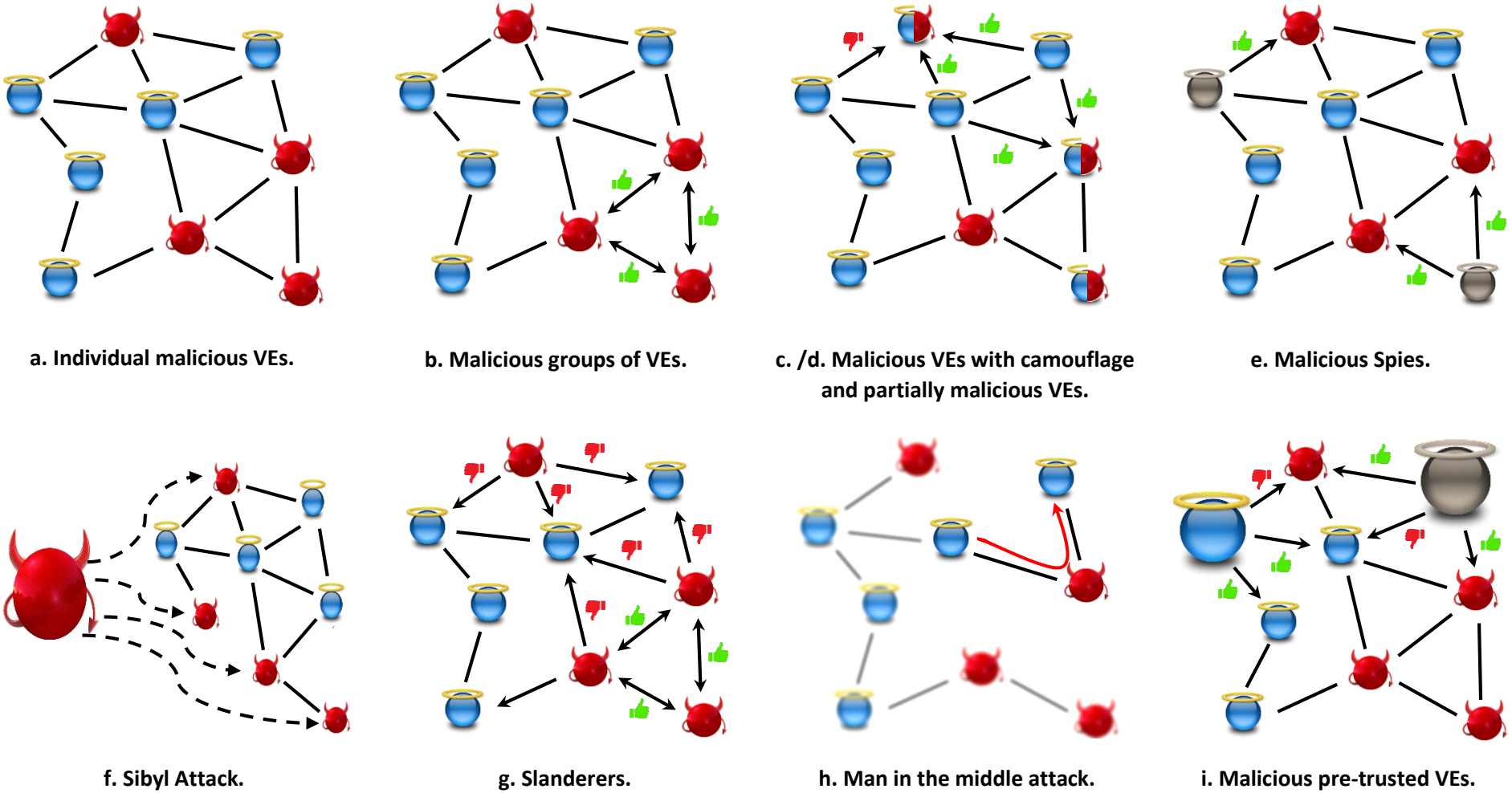

Figure 38: Security threats in the COSMOS T&R model.

8.4.4.2 Identifying the basic behavioral anomalies

From the previous threats, we identify three (3) basic behavioral anomalies and characterize easily malicious and benevolent nodes using a 3-bit number. Table 3 presents all the possible behaviors and how well TRM-SIoT detects and excludes them. The basic malicious behaviors are:

- **Malicious service provision:** This behavior is encountered when a node that is highly trusted for a given service gets a new request for that specific service and, taking advantage of this high Trust, provides a lower than expected quality of service. This way, the node can benefit by serving its self-interests. For example, a malicious smart-car may provide false information to other cars for the route it follows so that it can avoid future traffic. Such actions can cause serious problems (e.g. delay of emergency vehicles).
- **Malicious recommendations:** This behavior is encountered only when T&R models are used. A Thing may ask for recommendations or aggregate feedback from its social circle when searching for a new service provider (discovery phase) and deciding whether to interact with it or not (decision phase). In this case, a malicious Thing A could share with other Things falsified evaluations of the services provided by Thing B in order to lower or increase its reputation. A malicious node may be a part of a smaller society of malicious nodes which collude to increase their reputation, by recommending each other at the discovery phase and by providing good feedback for each other and bad for everyone else at the decision phase. These nodes may not demonstrate malicious service provision.
- **Oscillating behavior:** A malicious Thing that demonstrates this behavior is trying to trick the collective "mind" of its ecosystem by causing conflicts between the observations of individual nodes when providing services or recommendations. The oscillation can happen between different services or within the same service. In the first case, a Thing can act maliciously for one service while being benevolent for another one. This case is not of a big concern in our system as it is shown in Section IV. In the second and harder to identify case, a node generally provides good services/recommendations but not always. This way, when nodes try to collectively decide the reputation of the node, they might disagree with each other, leading to a decrease in the trust between the individual benevolent recommenders of the group.

Table 3: Combinations of Behavioral Anomalies

Malicious service provision	Malicious Recommendations	Behavioral Oscillation	Comments
0	0	0	Benevolent behavior.
0	0	1	Non-existent state.
0	1	0	Hard to detect with QoS as a metric.
0	1	1	Easy to detect severe threats.
1	0	0	Trivial detection.
1	0	1	Easy to detect severe threats.
1	1	0	Hard to detect if malicious nodes exceed 50%.
1	1	1	Higher percentage of malicious nodes increases detection difficulty

8.4.4.3 Security Threats Taxonomy

After identifying the several threats that our T&R model has to face, it is useful to categorize them according to some properties that will help as evaluate them more efficiently. For this reason, we are going to adopt some ideas presented in [84] regarding the analysis of a generic security threat for T&R systems. So, dimensions of the threats that should be taken under consideration are:

- **Attack intent:** While the direct result of attacks is generally that the evaluations made by the T&R system are manipulated in some way, there are three straightforward intents behind them:
  i. fraudulently praising entities in order to increase their reputation in the system,
  ii. driving down the reputation of reliable entities and
  iii. damaging the reputation system as a whole, so that users will decrease their trust in it and, eventually, stop using it.

  For instance, malicious groups' and spies' attacks will try to unfairly praise and increase the reputation of some entities which actually do not deserve it.

- **Targets:** Security threats may focus their efforts on:
  i. specific individual entities belonging to the system,
  ii. subsets of entities and
  iii. the whole community (making no distinctions).

  For example, individual malicious VEs and man in the middle attacks can be classified as individual attacks. It is in an attacker's best interest to limit the effect of an attack to a small target set of entities in order to be more subtle and try to avoid detection by the system.

- **Required knowledge:** Attacks may require some level of knowledge about the items, users, ratings and algorithms in the T&R system being attacked. Some threats require a comprehensive knowledge about the whole system or about some particular entities, while some other threats work properly with a small knowledge about it (its users, the T&R model applied, ratings distribution etc.). For example, creating a collusion will need more information about the system (each member of the collusion needs to know the rest of them) than an individual attack such as the individual malicious VE attacks. Generally, further knowledge about the system can help in choosing which attack to employ and in tuning attack parameters to maximize effectiveness and minimize detectability.
- **Cost:** The less expensive an attack is, the more beneficial is its application. Once again, the cost of running an attack is not necessarily economic, but it can be also measured in terms of resources or time requirements. Thus, some threats will have a higher associated cost and will be therefore more difficult to be performed, while others will be easily applicable, since their corresponding cost will make them worthy. In most cases, the cost of applying an attack is directly related to its associated amount of required knowledge. The only case where both dimensions do not match is for the Sybil attack, because although it needs (nearly) no knowledge about the system, it is not usually so easy to create a disproportionate number of entities enough to cause a really important damage to the community. The following factors contribute to the cost of a given attack:
  i. size of attack,
  ii. difficulty of interacting with the recommender system,
  iii. difficulty of obtaining required knowledge about the algorithm, users, entities and ratings in the recommendation system,
  iv. any other resources required for attack planning or execution, such as additional logistical, computational, or technical requirements.

- **Algorithm dependence:** Some attacks may be specifically designed to exploit a particular weakness in a specific algorithm or class of algorithms, while others might be more general and can be effective against a variety of algorithms. More specific attacks will intuitively require fewer resources for the same effectiveness, but require detailed knowledge of the algorithm being used and its parameter settings. For example, malicious pre-trusted peers, is a specific threat related and, therefore, only applicable to those T&R algorithms or models which actually make use of pre-trusted peers.
- **Detectability:** Finally, an attack over T&R systems is desired to be as less detectable as possible. The later an attack is detected, the higher might be the damage it will cause. That is the reason why most of the threats act trying not to induce suspicion as much as possible, (i.e. they do not cause drastic changes in the system but they rather make slight ones). In some way, the detectability of an attack or threat is a measurement of its resilience and effectiveness. Thus, the easiest threat of the previously presented ones to be detected would be the individual malicious VEs. As the collaboration between attackers and their gathered knowledge about the system increases, those attacks become more and more undetectable. That is the reason why all the threats based on a collusion are, generally, more difficult to tackle.

Based on the available literature, we can make an estimation about the several properties of the attacks that our T&R system will have to face. This is depicted in the following table:

Table 4: Security threats and their corresponding properties.

Security Threats	Properties of the Attacks					
	Intent	Target	Required Knowledge	Cost	Algorithm Dependence	Detectability
Individual malicious VEs	(iii) Damage	Individuals	Low	Low	Generic	High
Malicious groups	(i) Praise	Subsets	Medium	Medium	Generic	Medium
VEs with camouflage	(iii) Damage	Individuals	Medium	Medium	Generic	Low
Partially malicious VEs	(iii) Damage	Individuals	High	Medium	Generic	Low
Malicious Spies	(i) Praise	Subsets	High	High	Generic	Low
Sybil Attack	(iii) Damage	System	Low	High	Generic	Low
Slanderers	(ii) Slander	System	High	High	Generic	Low
Man in the middle attack	(ii) Slander	Individuals	Medium	Medium	Generic	Medium
Malicious pre-trusted VEs	(iii) Damage	System	High	High	Specific	Low

8.4.4.4 Solutions for overcoming Security Threats

a) **Individual malicious VEs:** The way of preventing such a misbehavior is by decreasing the level of trust or reputation of those participants who always provide bad services, categorizing them, therefore, as malicious VEs. The model we provide faces this simple problem successfully.

b) **Malicious groups of VEs:** The most crucial step in overcoming this threat is making a distinction between the trust indexes used for the evaluation of the VEs when they act as recommenders and the trust indexes used for the evaluation of the same VEs when they act as service providers. Many T&R models do not make this distinction. However, as it is discussed in the previous sections, COSMOS does make this distinction, thus this threat can

be faced satisfactorily, since the malicious groups will be regarded as bad service providers and recommenders from the benevolent VEs.

c) **Malicious VEs with camouflage:** From their nature, these VEs have to provide sometimes good services, since their total number of bad service transactions is bounded by p%. Thus, these VEs will not necessarily cause heavy damage, especially in the case where the trust standards (thresholds of the acceptable trust indexes) set by the other VEs are high, ensuring a high p%. Indicatively, the variable behavior of these VEs is not even considered as a threat in many T&R models in the sense that they do not punish that kind of behavior, but they just try to adjust the trust and reputation given to a peer to its real and current goodness. Other models, however, demonstrate the uselessness for malicious peers to behave in this way. It should be noted that, in the case of COSMOS, Trust incorporates the standard deviation of the behavior of the VEs, ensuring that their bad behavior will be limited even more. Finally, the variable behavior of a peer, when detected, could be punished and avoided. However, to achieve this, storing a part of the transactions history becomes necessary.

d) **Partially malicious VEs:** There are some trust and reputation models which are not resilient to this kind of attack since they just perform a global computation of the trust and/or reputation of a peer, regardless the service they are providing. On the other hand, COSMOS, by just considering a different score for every service offered by a VE (e.g. different Friend Lists for different Applications), can mitigate this threat most of the times. However, it should be tested whether this distinction is always practical, since in some environments (for instance, those with a great amount of services offered) it could lead to some scalability problems. Finally, different weights could be added for the evaluation of different services depending on their importance.

e) **Malicious Spies:** Like in previous threats, an accurate management of the reliability of the peers, not only as service providers, but also as recommendation providers may effectively help to prevent this kind of abuse. This is the case for COSMOS, so Malicious Spies will be used as service providers but not as recommenders.

f) **Sybil Attack:** Yet again, not many T&R models deal with such an important and potentially dangerous threat, thus leading to an underestimated but great risk. A reputation system's vulnerability to a Sybil attack depends on how cheaply Sybils can be generated, the degree to which the reputation system accepts input from entities that do not have a chain of trust linking them to a trusted entity and whether the reputation system treats all entities identically. One of the most common solutions proposed in the literature for this kind of threat consists of associating a cost to the generation of new identities in the community. This cost is not necessarily economic, but it can also be a cost in terms of time or resources for instance. Another suggested way of dealing with this problem [85] makes use of a central entity managing (virtual) identities in the system or even a set of identity providers ensuring that every participant in the community has a unique and immutable identity. The dual nature of TRM-SIoT which combines the Platform and the social circle together with the adaptation of the **random surfer ideas** first seen in PageRank [48] allows the system to provide an excellent quality of service even when the number of malicious nodes exceeds the number of the benevolent ones.

g) **Slanderers:** The differentiated management of the trust given to a participant when supplying services and the reliability of its recommendations can be very useful in this scenario as well. Moreover, in the case of COSMOS, the platform or even some benevolent VEs have a 10% chance of recommending a random VE below the reputation thresholds. That means that no matter how low the reputation of a benevolent VE becomes because of slanderers, it will eventually be recommended to new benevolent VEs which, after their first interactions, will give it a high trust rating and ignore its bad reputation in the future (if it remains bad after the new connections).

h) **Man in the middle attack:** Once more, this is a threat which has not been associated with T&R systems traditionally. Most models consider or assume the authenticity of the service-providers/recommenders. Nevertheless, this attack can cause a great damage in the system if its application is possible. A solution proposed in the literature is the use of cryptography schemes in order to authenticate each user in the system. However, it is not always feasible to apply such a solution, above all in highly distributed environments like those we are trying to build. **COSMOS introduces a new solution, by using the concept of Assists described earlier, an interaction metric that is used for the evaluation of VEs as Mediators**. In our case, only the direct Friends of a VE are evaluated, meaning that only direct service providers or mediators are evaluated by the VEs. This means that a man in the middle attack would result in the low rating only of the malicious VE (which would gather many negative Assists) thus helping the benevolent VE to remove this Friend from its Friend List, and maintaining the good reputation of the VE whose services where changed by the mediator.

i) **Malicious pre-trusted VEs:** It is not always feasible to find a set of peers that can be trusted before any transaction is carried out in the system. Some models base their strategy on this kind of participants. However, and maybe in a paranoic way of thinking, every user in a virtual community can behave inappropriately at some point. If such a thing occurred with a pre-trusted peer, the system would be at a risk. For this reason, we have decided not to use pre-trusted VEs in the COSMOS system, since their existence does not seem necessary for now.

8.4.5. Evaluating and Testing our T&R model

It is not always easy to check the correctness and accuracy of a model and even more to compare it against other trust and reputation models. In order to test our T&R model, we used one of the best simulators for T&R models we came across, the **TRMSim-WSN** [86], a Java-based T&R models simulator aiming to provide an easy way to test a trust and/or reputation model over WSNs and to compare it against other models. It allows the user to carry out customized simulations by adjusting several parameters such as the percentage of malicious nodes or the possibility of nodes forming collusions. Moreover, it offers an API which provides a template for the users to help them easily load new T&R models to the simulator [87]. Finally, it is possible to load to the simulator new networks from a XML file. By loading to the simulator our own T&R model and the networks extracted from the COSMOS Use Cases, we were able to run many tests before the actual deployment of the corresponding components. That way, we were able to determine whether the security threats described above are faced successfully and whether our model covers the main goals we have set regarding the T&R management of VEs.

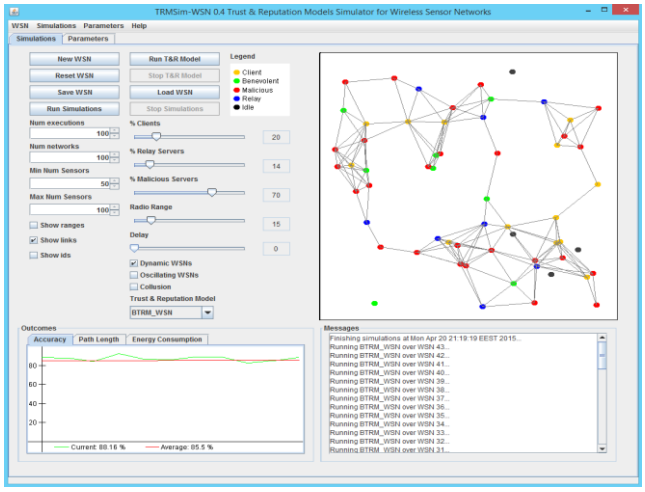

Figure 39: TRMSim-WSN, Trust and Reputation Models Simulator for WSNs.

8.4.5.1 Satisfaction Performance of TRM-SIoT

TRM-SIoT was tested using TRMSim-WSN, the state-of-the-art simulator for T&R systems. The simulations were run for one service, but due to the fact that the T&R management follows a per service basis, the results would be similar for a multi-service simulation. In essence, a multi-service system can be represented by the sum of several simulations where the nodes remain the same, but the distribution of their behavior (malicious, benevolent) is different, depending on the service that is studied.

In Figure 40 most of the arrows point to good service providers (green dots), while the only arrows that point to malicious service providers (red dots) derive from other malicious service providers. This is a visual way to prove the success of our T&R model. TRM-SIoT was tested in multiple simulation environments where the nodes can have collusive, oscillating and/or dynamic behavior. The collusive and oscillating behavior are already described in subsection 8.4.4 while the dynamic behavior is that of a node that dynamically enters and exits the system. Figure 41 depicts the convergence of random generated networks using our T&R model.

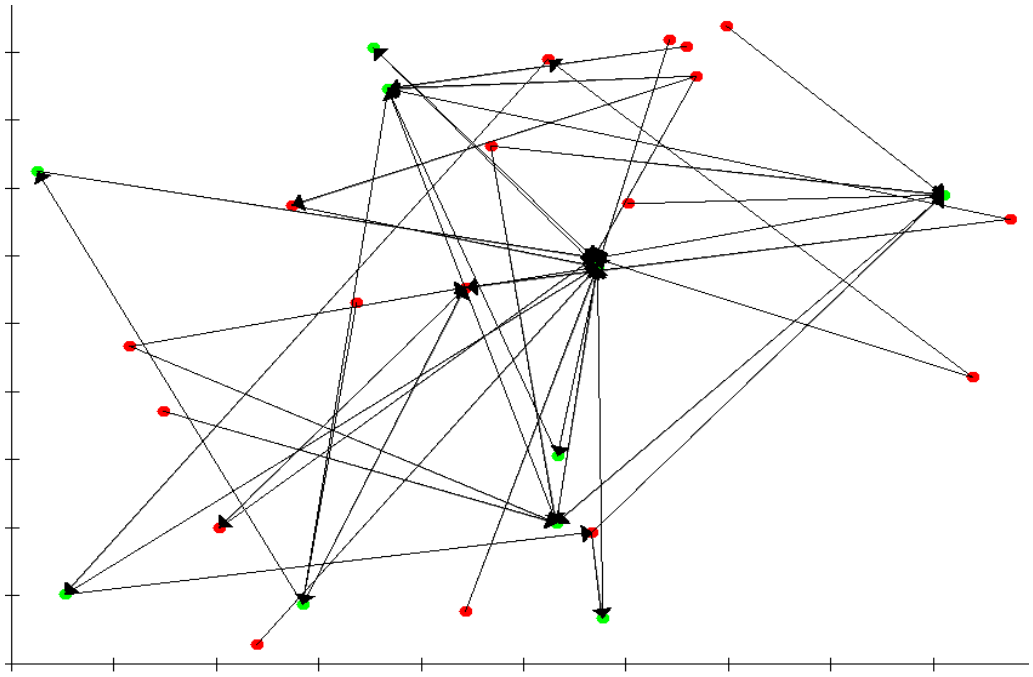

Figure 40: Trust relations simulation for TRM-SIoT.

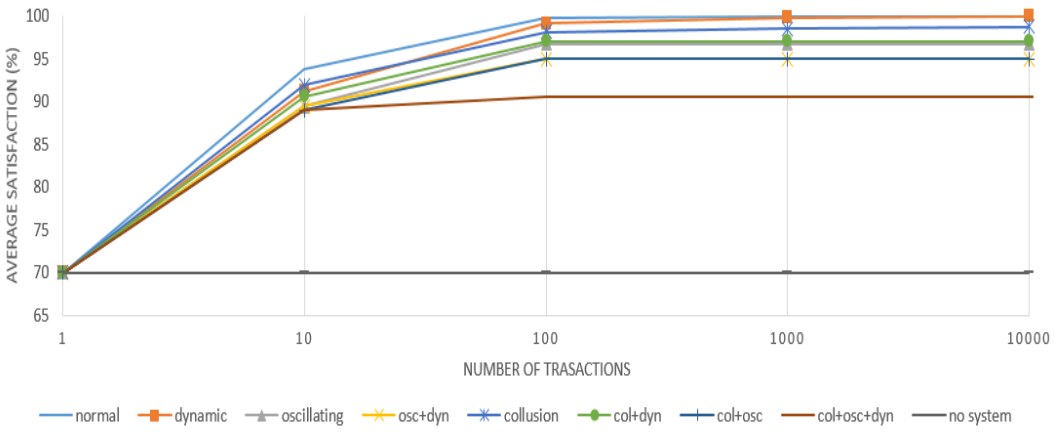

Figure 41: Convergence on network with 30% malicious nodes.

By studying Table 5 derived from a great number of simulations we can conclude that:

- In a normal system TRM-SIoT provides excellent security. Even in a network with 90% malicious Things, the average satisfaction is over 96%.
- In a network with an average percentage of malicious Things (20-40%) TRM-SIoT manages to provide satisfaction of about 90% even in dynamic, oscillating networks with collusive nodes (col. & osc. & dyn.).
- In networks where 50% of the nodes are malicious and the Platform might provide wrong proposals, the neighborhood method manages to keep the satisfaction to acceptable levels.
- On average, the system provides satisfaction levels equal to that of a network with 30% less malicious Things that does not use TRM-SIoT ("no system"), e.g. the satisfaction at a dynamic, oscillating network with collusive nodes and 60% malicious Things using TRM-SIoT is equal to that of a network without TRM-SIoT and 30% malicious Things.

Table 5: Average Satisfaction (%) for different types of networks with increasing percentage of malicious nodes.

%	10	20	30	40	50	60	70	80	90
normal	99.9	99.8	99.7	99.5	99.3	99	98.9	97.5	96.1
dynamic	99.9	99.3	99.1	98.4	96.7	94.8	93.4	90.5	77
oscillating	99.1	98.3	96.7	94.4	92.1	86	82	65	48.5
osc. & dyn.	98.1	96.3	95	92	85.2	82	69	59.2	43.4
collusion	99.8	98.8	98.1	97	94.2	90.7	86	68.8	47
col. & dyn.	99.5	98.5	97	90.7	91.2	85	79	62	45
col. & osc.	98.9	97	95	90.9	83.6	76.4	65	51.6	26.6
col. & osc. & dyn.	98.4	94.2	90.5	86.8	79.1	71	63	44.7	22
no system	90	80	70	60	50	40	30	20	10

8.4.5.2 Satisfaction Performance Comparison with other models

In this subsection TRM-SIoT is compared with three widely used T&R models, Eigentrust [77], PeerTrust [78]and PowerTrust [82], and a relatively new system, known as BTRM-WSN [86] which implements a bio-inspired algorithm known as Ant-Colony System (ACS). The simulations where run for normal, dynamic and oscillating networks, all combined with collusive behavior. The measurements were taken for varying percentages of malicious nodes (10, 30, 50, 70 and 90). Indicatively, from the bar diagrams of the following Figures, the TRM-SIoT appears to have comparable satisfaction performance with the other models (and sometimes slightly better) across all networks, thus, our hybrid approach, the combination of the Platform and the social circles, leads to a system capable of maintaining satisfaction at very good levels.

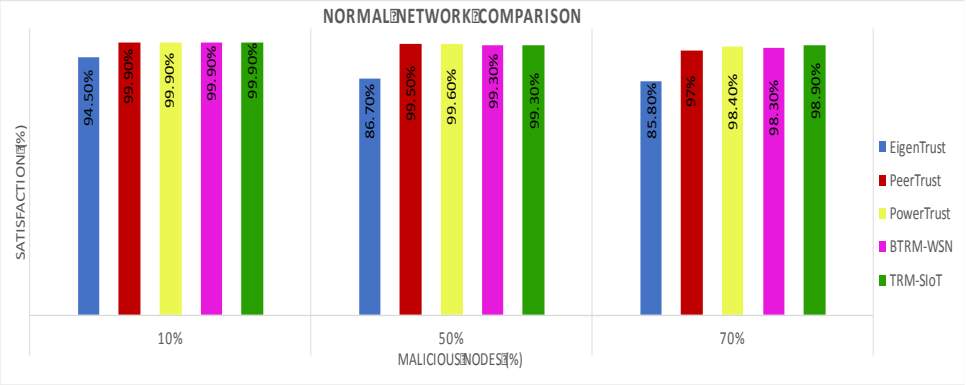

Figure 42: Normal Network Comparison.

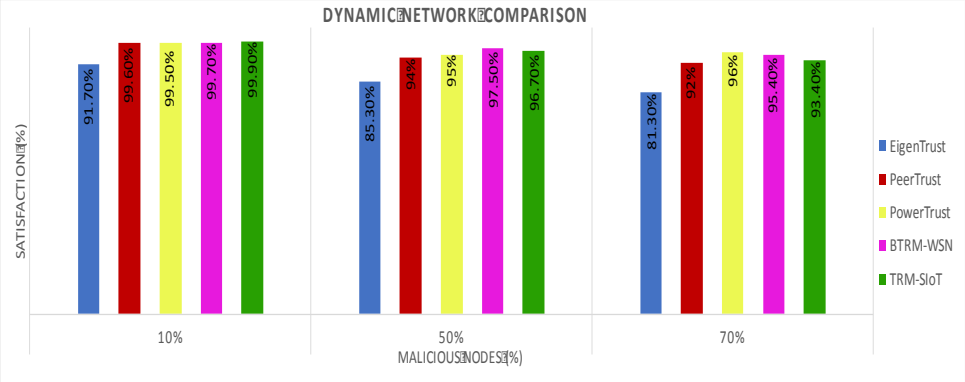

Figure 43: Oscillating Network Comparison.

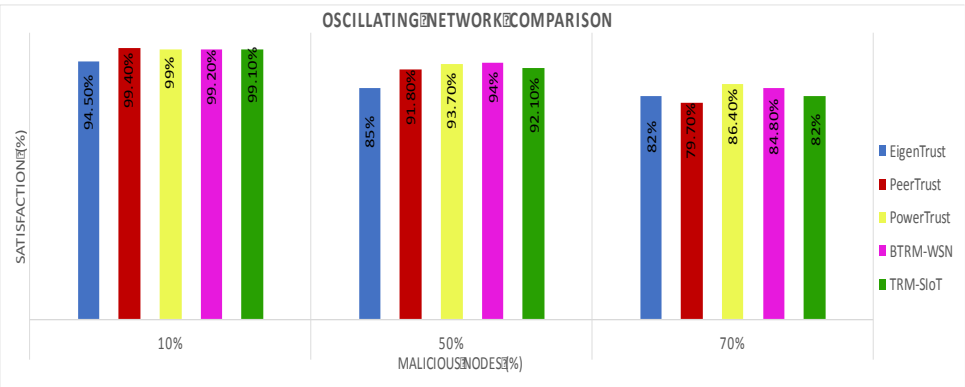

Figure 44: Dynamic Network Comparison.

8.4.5.3 Scalability Comparison with other models

TRM-SIoT provides similar satisfaction performance with other models, but there is a great difference in its design. Neither the entities/nodes nor the platform need to have a complete view of the several interactions within the system, a characteristic that offers a huge advantage when scalability has to be taken under consideration. The bar diagram of Figure 45 shows how the different algorithms managed to scale in the simulation environment. All measurements were taken on a system with an Intel Core i7 3630QM and 8GB RAM. The time limit per interaction round (1 step in the simulation) was set to 30s. The TRM-SIoT did not reach 30s/step but only 1s/step due to the inability of the computer system to spawn more threads (the limit was 8,500 threads). TRM-SIoT managed to run simulations with ten (10) times more nodes and with sufficient memory it could scale up to 250x. The results showed that the computation time of the nodes remains constant while the computation time of the Platform was linear to the size of its log file. Hence, the bottleneck of the system is on the side of the Platform which has to be a high performance server that could be upgraded or have multiple instances if needed.

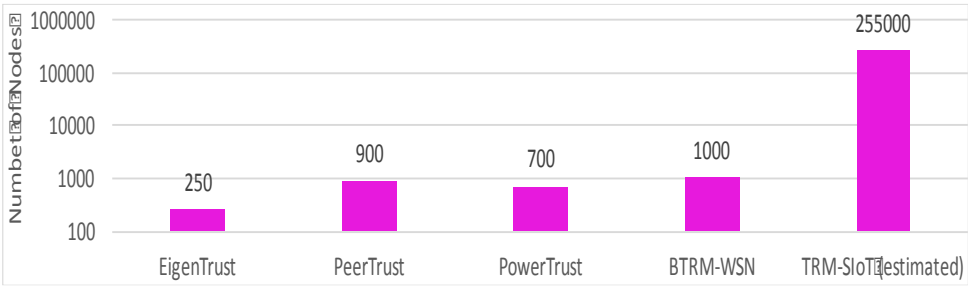

Figure 45: Scalability comparison with other T&R models.

8.5. Relational Models

Till now, the concept of Friendship between VEs has been discussed in detail. However, taking under consideration the SIoT paradigm introduced in [46], we can realize that other kinds of relationships between VEs can be defined too. The information that these relationships can provide when extracted may provide more options for the development of new mechanisms. A representative example is the Replacement Relationship that we are going to introduce.

Ownership Relationship: The Ownership Relationship is a simple relationship which is established between objects that share the same owner. It is primarily a **user based** relationship. This means that the users play an important role in the formation of these kinds of relationships. The rare change of the owner of a VE makes the relationship a **static** one. This means that Ownership Relationships are not subject to frequent changes and have a longer duration than other types of relationships. This duration also makes the relationship more **"reliable"** than others, as there is less room for errors during its detection. The Ownership Relationship is a **symmetric** one, meaning that if a VE1 has an ownership relationship with another VE2, then the same applies for VE2. The main property that is needed to establish an ownership relationship between VEs is the **Owner ID**, an identification that matches to the user that owns a VE. Extracting the relationship is fairly trivial. If two VEs share the same Owner ID, then they form an Ownership Relationship. The motivation behind extracting this relationship lies in the need of knowledge flow between VEs that share the same owner. Through interaction with the owner and with the use of sensors, VEs may be able to acquire information that could prove useful to the other VEs owned by the same user.

Common-User Relationship: The Common-User Relationship is similar to the Ownership Relationship, with a notable difference in the types of users that take part in its creation. This relationship is established between VEs that share a common simple user. A simple user of a VE is defined as the user who has partial access to the VE regarding its use, maintenance and permissions. Simple users can have various levels of access to the referred VE, however the extent of this access is decided only by the owner who has full control of the VE. A typical example of a simple user is a person who rents a car (VE-car). The Common-User Relationship is primarily a user based relationship. The frequent change of active users of VEs makes the relationship a more **dynamic** one. However, this relationship is more reliable than others, as there is less room for errors during its extraction. The Common-User Relationship is a **symmetric** one. The main property that is needed to establish such a relationship between two VEs is the **Simple Users List**, a list containing all the User IDs of simple users that are actively using a VE. Yet again, extracting the relationship is fairly trivial. If two VEs share a simple user, then they form a Common-User Relationship. However, providing a framework where a Users List is created is not an easy task, so actual real detection of this relationship may not be one of the features that we will provide. The motivation behind this relationship is the same with the one for the Ownership Relationship. As an example, let's assume that George who lives in Greece and drives an IoT-enabled vehicle, equipped with sensors gathering information regarding his driving habits, has at some point to travel to Italy for a business trip. When he visits Italy, he rents a car which is IoT-enabled as well. By establishing a Common-User Relationship, the two VE-cars may share any information available regarding the driving habits of George.

Follower/Followee Relationship: The followers/followees relationship may be either **VE based** or **Application based**. This means that the VEs themselves and Applications play an important role in the formation of these kinds of relationships as it has already been analyzed. The ephemeral nature of friendships makes this relationship a more dynamic one. The followers/followees relationship is an asymmetric relationship.

Parental Relationship: The Parental Relationship is the relationship that is established between VEs and their higher ranked counterparts. A higher ranked VE is defined as the VE that has permission to monitor and control, to some extent, VEs that are configured to be under its authority. The parental relationship is VE and Application based. Authorities are decided per application and are not that often subject to change, something that makes the relationship a more static one. The parental relationship is also a strictly asymmetric one. This relationship is closely related to the concept of GVEs that COSMOS introduces. The motivation behind this relationship is the introduction of the element of authority between VEs (per application) which is necessary to some extent to all communities aiming for autonomy. The existence of a VE with a more central role, able to organize and control the group can help prevent decision deadlocks and solve conflicts when and if they appear.

Co-Work Relationship: The Co-Work Relationship connects VEs that work on the same Application. Applications may need groups of VEs to form teams in order to share resources like specific knowledge or IoT-services. The importance of Applications in this relationship makes the Co-Work Relationship an Application based one. The type of an Application plays an important role in the nature of this relationship. Applications can be of fixed size, with a predefined number of VEs involved, or of an evolving size, with VEs joining and leaving the Application constantly. The duration of Applications can vary as well. Applications can have a predetermined duration or may be of "open time". All the above characteristics make the Co-Work Relationship a dynamic relationship and of variable reliability, depending on the Application characteristics. For example, a fixed size and fixed time Application will form more reliable Co-Work Relationships than an evolving in size, "open time" Application. The Co-Work Relationship is a symmetric one.

Conflict of Interest Relationship: The Conflict of Interest Relationship is established between VEs that may enter into conflict with each other. Conflicts are created when VEs vie for the same resources, either in the form of critical information or in the form of vital (IoT-)services that can be offered to a limited number of VEs. The importance of Applications in this relationship makes the Conflict of Interest Relationship an Application based one. This means that Applications define the formation of these kinds of relationships between VEs, by providing necessary information for their detection. The main information that has to be examined to extract this kind of relationship is the List of VEs aiming for a common resource and an Importance Table (e.g. a VE from the domain of e-Health will have greater priority than one from the domain of Home-Automation). The table is necessary for determining any kind of priority between VEs in order to ultimately resolve the conflicts.

Replacement Relationship: The Replacement Relationship is a relationship that can be established between a VE (VE1) and another VE (VE2) which has the capacity to replace the first one. This means that, for a specific Application and type of service, VE2 can be used in an identical way as the VE1 that is replaced. The Replacement Relationship is an Application based relationship. A challenge that all Application based relationships share is the consequences of VEs participating in more than one Applications at the same time. The amount of possible Replacement Relationships for all Applications is too high to ensure a scalable network. Therefore, to overcome this, this relationship will always be established for one particular Application and the corresponding service at a time. The replacement of VEs may have various levels of success. When multiple VEs are able to act as replacements, the most efficient and successful one must be chosen first. In order to achieve this, a **Replacement Rating** is introduced, a rating system that measures the replacement efficiency of a VE. Once a replacement is completed, VEs can rate their replacements based on their level of success. This level can be measured as the percentage of completed tasks during the replacement period. The motivation behind the Replacement Relationship is to ensure the smooth operation of Applications. The solution we propose is as follows. A VE that offers its services to some

Applications could maintain a Replacements List which would contain the addresses of other VEs that offer similar services in similar conditions and cover specific needs of the Applications. In case one of the services of the VE becomes unavailable, then the corresponding VE is informed. The main concepts and mechanisms used for the Follower/Followee Relationship can be applied here too.

Co-Location Relationship: The Co-Location Relationship is a relationship that is established between VEs that are close to each other. Proximity of VEs is defined as the distance between the two corresponding Physical Entities that satisfies a given lower threshold. This relationship can be used for example in Applications for weather forecasting or geological and geographical surveying. The Co-Location Relationship is a **Location** and **Application based** relationship. Calculation of location is a complex procedure which requires different levels of accuracy, depending on the scope and scale of the Application. The main property that needs to be examined is the **Geolocation** of the VEs, especially the **hasGeoLat** and **hasGeoLon** data-type properties. However, as mentioned before, the necessary threshold to determine proximity needs to be set by the Application. For that purpose, the **GeoBoundaries** property of the Application could be introduced. GeoBoundaries is a set of coordinates that create a geographical boundary, determining the area and therefore the threshold on distance for the VEs to be considered close to each other. The motivation behind the Co-Location Relationship lies in the plethora of location based information that can be gathered by a VE and may be deemed useful for other VEs as well.

Table 6: Types of Relationships and their dimensions.

Relationship	Ontology Focus	Type	Reliability	Direction	User Properties	VE properties	Application Properties
Ownership	User based	Static	High	Symmetric	Owner ID	-	-
Common-User	User based	Dynamic	High	Symmetric	User ID	Simple Users List	-
Follower/ Followee	VE/App based	Dynamic	-	Asymmetric	-	Subsection 8.2	Subsection 8.2
Parental	VE/App based	Static	High	Asymmetric	-	-	List of VEs GVE members
Co - Work	App based	Dynamic	Varying	Symmetric	-	-	List of VEs
Conflict of Interest	App based	Dynamic	Varying	Symmetric	-	Domain	List of VEs Importance Table
Replacement	App based	Dynamic	Varying	Asymmetric	-	Services Rating	List of VEs
Co - Location	Location/ App based	Dynamic	Low	Symmetric	-	Geolocation	GeoBoundaries List of VEs

9. Network Runtime Adaptability (NRA)

9.1. Participants in Network Runtime Adaptability scenario

COSMOS aims at enabling the customization of services offered to final users and automatizing (at least at some extent) the improvement of various parameters describing the performance of the network. In this context, the network refers to all the entities and systems that participate in the scenario of runtime adaptability. The next figure highlights them.

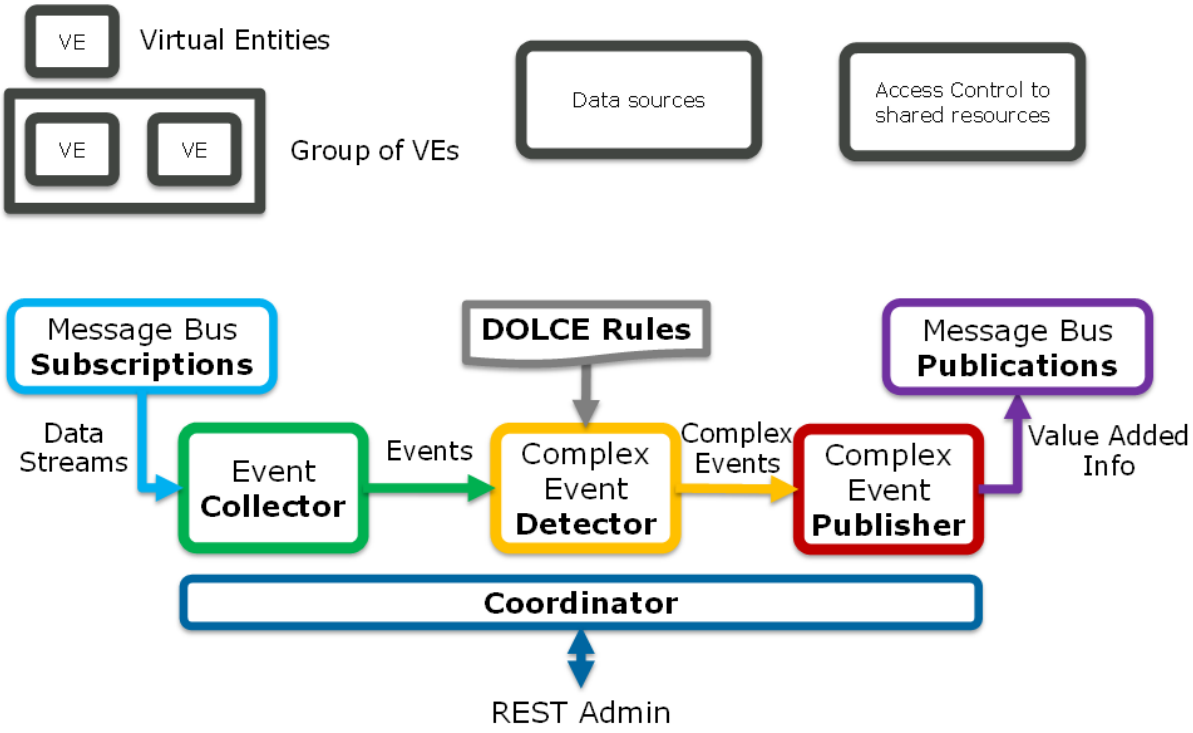

Figure 46: Components participating in the Network Runtime Adaptability scenario.

There are several components and entities that can play an important role in the optimization of the overall system performance by adapting to various situations:

- **Virtual Entities** are linked to Physical Entities. Their reconfiguration has as an implication changes in their parameters such as the costs related to their functional mode, their power consumption, the data provision and data processing. When a group of VEs can be affected by the same optimization, a GVE is addressed by the Network Runtime Adaptability mechanisms.
- **Data sources** are not limited to VEs providing the information they collect but extend to other elements (mainly open data platforms) that can provide extremely useful information for the development of a service. Nevertheless, sometimes the services could collect data in many different ways. The different ways that data can be collected represent another challenge that can be addressed by the Network Runtime Adaptability mechanisms.
- **Access Control** to shared resources. Access to the information provided by all the components defined in COSMOS can be dynamically modified based on several rules. For example, emergency situations or cases where specific areas of the system are under maintenance may lead to failures. These circumstances should be detected and the system should be alerted.

- The **CEP engine** plays a very important role in the mechanism we introduce and adaptation actions can be applied to each one of its modules.
  - ✓ **Event Collector:** The Event Collector is in charge of generating the events that the detector will evaluate based on the rules compiled in the CEP engine. Changes to the events in runtime can provide to the whole system an adaption capability that can increase its performance.
  - ✓ **Complex Event Detector:** The role of the detector does not vary. However, it should be mentioned that DOLCE rules can be updated in runtime via the coordinator module.
  - ✓ **Complex Event Publisher:** This module connects the outputted complex events of the CEP engine (that can modify the behavior of any of the aforementioned modules) to the Message Bus, assuring their delivery.
- The Message Bus is not an entity whose functionalities will be adapted under the Network Runtime Adaptability paradigm. However, it is a key module that guarantees the delivery of the messages to their destinations.

In the following subsections we analyze how all these entities and components can get adapted.

9.2. Network Runtime Adaptability with COSMOS components

In the COSMOS SotA deliverable, the project has described the different levels of Situational Awareness as well as the Context Information classifications that are available. The one that has been chosen and the rationale for its selection can be followed in D6.1.2. In summary, Situational Awareness is composed by:

- the **perception** level – get information from data sources
- the **understanding** level – process data received and react if needed
- the **predicting** level – compose received data with previous processed data and explore them looking for patterns that can be derived from/for predictions.

Regarding the context information, the classification selected is the so called general classification, which basically divides context information in four categories:

- **System context:** It corresponds, for a given Application, to the context information of the system (software and hardware) on which the Application runs as well as the contextual information of the used communication system (for example, the wireless network type).
- **User context:** It can be any information that can characterize a user. This may be his/her age, location, medical history, biometric information, emotions, activities, social relationships etc.
- **Environment context:** It represents information that describes the physical environment and is not covered by the system or user context (particularly the context information from external sources such as temperature).
- **Time context:** It represents all the information related to time (hour, day, month, year, …).

All these terms represent the conceptual components that enable the Network Runtime Adaptability. In this sense, the awareness level defines the actions that can be applied in the COSMOS system. The next figure summarizes how all these concepts are related to each other and what kind of decisions can be extracted from the evolution in the processing and analysis of the information.

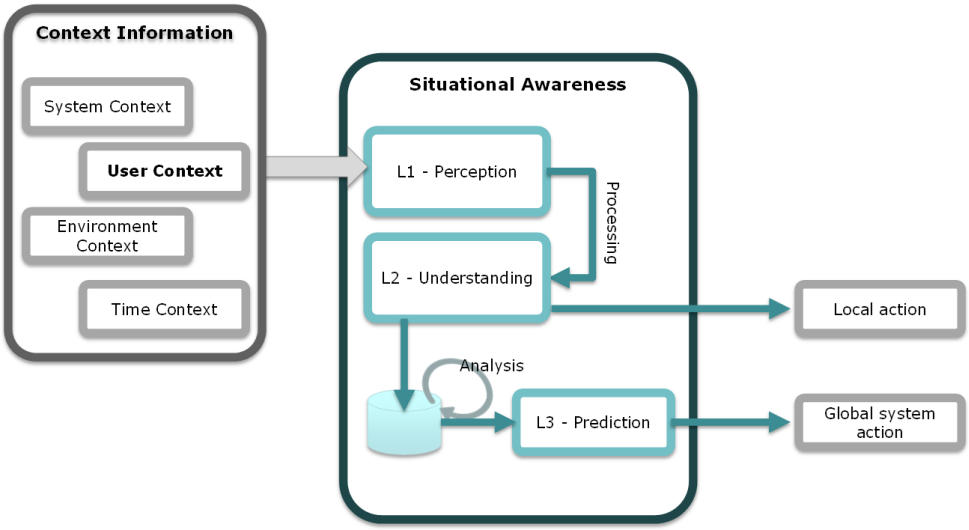

Figure 47: Flow towards Runtime Adaptability.

Taking as reference this diagram, it is easy to understand how the different levels of data processing can be transferred into actions that imply modifications of the behavior of components. In the next subsections, we provide a detailed analysis of the actions that can be performed and of the flow that the system will follow, from the time of detection until the time of execution of the needed updates/adaptations.

9.3. CEP actuation in Network Runtime Adaptability

The µCEP engine is a key element for the scenario of runtime adaptability. As it has been presented in D4.1.2, the engine can be implemented in a centralized way (all the modules running in the same machine) or in a distributed way (different machines running different instances of each module). In the worst case, what this structure allows is the application of local actions derived from the execution of rules in the Complex Event Detector module.

Initially, it is necessary to map all the actors in their positions inside our scenario. In order to do so, the next figure establishes where they are and the concepts they are linked with.

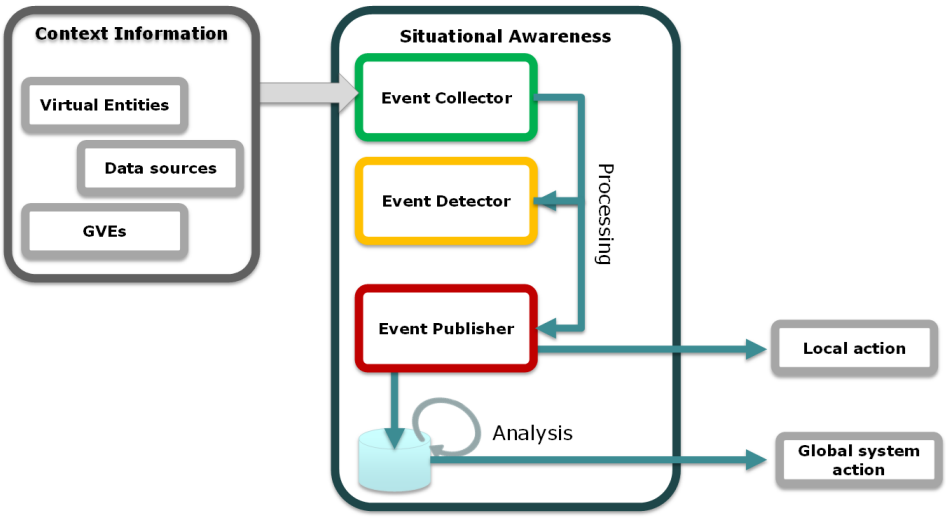

Figure 48: Concept mapping over real components.

The flow makes sure that data sources are processed at several levels, depending on the processing done, the data nature and the volume of data. The corresponding decisions taken have different impacts. The picture considers a generic implementation of µCEP. It should be noticed that we take under consideration the simplicity of the devices that will take part in the future IoT paradigm. Thus, the implementation is designed to fit in low power, constrained devices, opening new possibilities to automatize the advanced management of them.

In order to move from the conceptual world closer to reality, it is mandatory to establish the link between the actions and actuations and the real objects and machines that will implement them. The next figure presents the wide picture of how COSMOS concepts would be implemented in the most general scenario.

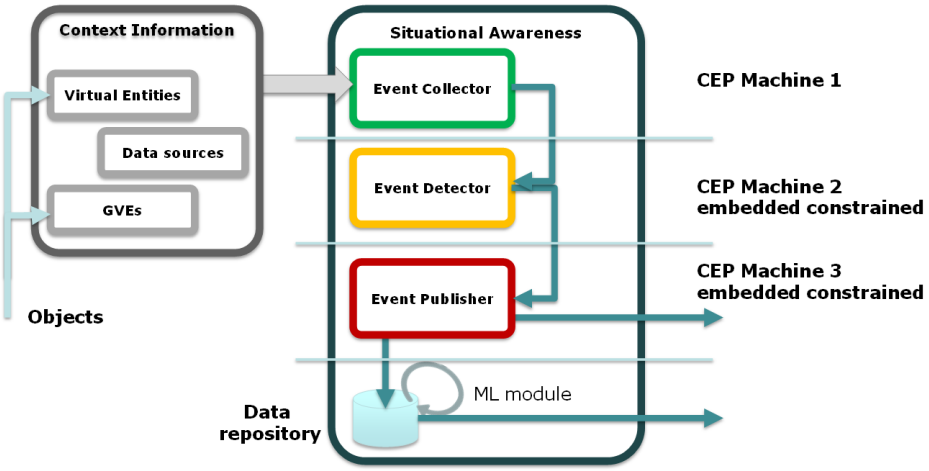

Figure 49: Mapping over real implementation.

Now that it has been explained how the different components fit in the COSMOS architecture, we can analyze the different actions that can be performed and the ways they can be shared in order to notify all the involved parties properly.

9.4. Decisions taken in Network Runtime Adaptability

In previous sections, the different elements that can be customized by the Network Runtime Adaptability mechanism are presented. For the sake of clarity, in the next figure the Machine Learning module has not been included even though it is co-working with the Complex Event Publisher module for distributing and triggering actions using the Message Bus.

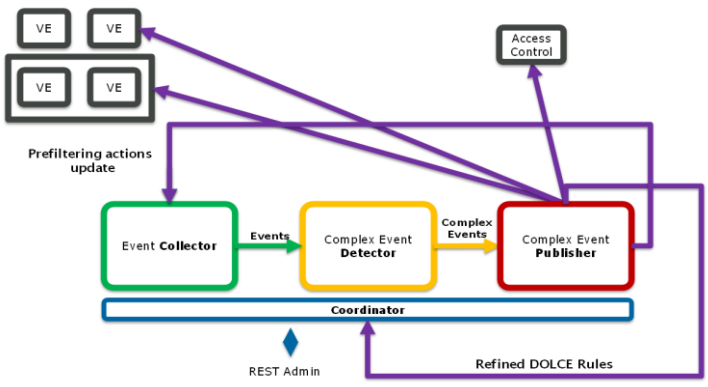

Figure 50: Decision distribution in Network Runtime Adaptability.

The decisions that have been defined are the following:

- Refinement of DOLCE rules in µCEP
- Refinement of pre-filtering actions and event generation
- Dynamic update of access control rules for sharing resources
- Modification of the behavior of VEs
- Modification of the group behavior of GVEs

The first three actions are related to the modification of the way that information is analyzed or accessed, whereas the last two actions focus on the optimization of functional processes related to the Physical Entities and the VEs.

Table 7: List of events that modify in runtime components' functionalities.

Action	Type	Destination	Impact
Refinement of DOLCE rules	New event detection	Complex Event Detector	Run time modification of the rules that applies to certain events. This also means adding new rules as long as the systems evolves.
Refinement of pre-filtering	New event generation	Event Collector	The Event Collector module could require to add more data sources in order to identify the new events that can be defined by DOLCE rules.
Modification of VE behavior	Change of individual entity behavior	VE – Object	A VE can have multiple functionalities from sensing to actuations over physical systems. In this sense, the modification of certain parameters may have a strong impact on the Physical Entities' performance, thus it is really important to have the capability of remotely managing this aspect.
Modification of GVEs behavior	Group modification	Several VEs – Object(s)	^The same rationale is applied to GVEs.

9.5. CEP adaptability implementation

The implementation of the Complex Event Processing Engine provides a REST interface to manage the files containing Dolce programs. As any change in a dolce program affects the programmatic structure of the Complex Event Detector module, any changes performed in a Dolce program will have as a result a soft restart of Complex Event Detector to rebuild the structure of the detector engine. In a Dolce program there will be three types of declarations or statements defined by three reserved words:

- **External:** to declare variables that can be modified in runtime.
- **Event:** to declare incoming events.
- **Complex:** to declare the rule defining a complex event and the output information.

As a REST interface uses JSON format in the body it is possible to change many parameters that define the way the CEP operates. The following table summarizes the methods available.

Table 8: Methods for the adaptation of the CEP.

Method	Description
Get DOLCE Rules files	Gets the list of DOLCE Rules files being processed in a CEP instance
Get a DOLCE Rules file	Gets the DOLCE Rules file details by the given name
Add or Modify a DOLCE Rules file	Add a new DOLCE Rules file or modify an existing one by the given name
Delete a DOLCE Rules file	Delete the DOLCE Rules file by the given name
Get the External declarations	Get all external declarations in a DOLCE program
Add/Modify an external declaration into a DOLCE Rules file	Add a new event declaration or modify an existing one into a DOLCE Rules file
Delete an external declaration into a DOLCE Rules file	Delete an event declaration into a DOLCE Rules file by the given name
Get the Event declarations	Gets all event declarations in a DOLCE program
Add/Modify an event declaration into a DOLCE Rules file	Add a new event declaration or modify an existing one into a DOLCE Rules file
Get the Complex declarations	Get all complex declarations in a DOLCE program
Add/Modify a Complex declaration into a DOLCE Rules file	Add a new complex declaration or modify an existing one into a DOLCE Rules file
Delete an event declaration into a DOLCE Rules file	Delete an event declaration into a DOLCE Rules file by the given name
Delete a Complex declaration into a DOLCE Rules file	Delete a Complex declaration into a DOLCE Rules file by the given name

By using of these methods, the adoption of the runtime modifications done over the components running a CEP becomes possible.

9.6. Hierarchical update of IoT devices

One of the most promising aspects of the Network Runtime Adaptability is the ability of VEs to dynamically modify the firmware they are running. In this sense, the management will be done by defining a hierarchy of devices and events. Additionally, the distributed implementation of µCEP will facilitate the continuous optimization of this scenario by simplifying the management procedure of the VEs.

The actions presented before may have as a result the partial or complete reconfiguration of VEs. This hierarchical update aims to exploit a two level cognitive system. First of all, the VEs with a µCEP engine are able to take their own decisions improving their performance. However, the view they have is limited as they are not aware of the full picture and, consequently, higher level entities that can access this information are needed. Secondly, following this approach, it is also possible to ensure the proper behavior of a VE regardless of its connectivity to other entities and higher layers. A set of hierarchical actions will be derived and applied where needed.

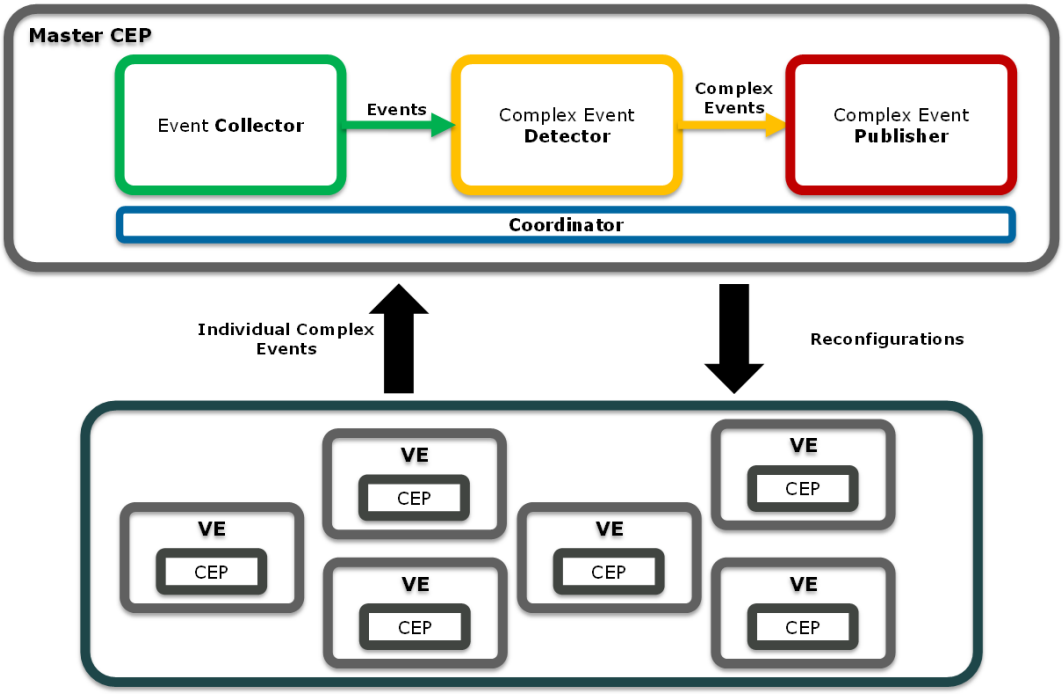

Figure 51: Hierarchical architecture of adaptability.

The modification of functional parameters or even of whole sets of functionalities of a VE in real life offers a huge number of opportunities. However, there are several threats that must be minimized. The two level approach assures that compromising the security of one VE –for example, injecting a malicious software version– will not have a huge impact into the others. Once the rules injected in a sandboxed-µCEP are securely set and stored, the individual modification by malicious software of VE just affects these specific individual components.

In the IoT world there are several mechanisms for performing the remote update of devices [88] and also for doing it in a secure, efficient and reliable way [89]. However, in all cases the update is triggered by end users, so this proposal goes a step beyond by adding autonomy to the system. The combination of efficient distribution with an effective decision making engine will have to be explored. It goes without saying that the usage of components described in the previous sections (e.g. the Planner) is in our initial plans.

9.7. Enabling NRA across heterogeneous VEs

Supporting a scalable and heterogeneous evolving scenario requires the implementation, deployment and monitoring of a set of software tools that will run not only on cloud servers but also on energy constrained embedded devices. Given that fast development of new IoT devices is nowadays commonplace and, even with the efforts made in protocol standardization activities, there is a huge plethora of commercial solutions, it is difficult for high-level applications to be ported easily and seamlessly from one environment to another. While it is true that IoT devices are meant to perform a reduced amount of simple tasks, the IoT paradigm requires them to be able to adapt to changing conditions and changing applications purposes, which in turn determines *what* the device is able to do, and *how* it is able to do it. COSMOS offers the developers the ability to adjust the analysis procedures running on the VE side by means of a µCEP whose implementation is tested in a reduced set of operating system

environments. In this sense, in order to facilitate and spread its usage across IoT communities, this tool is going to be *containerized*. Because an application not only consists of its own binary but an entire runtime environment enclosing its dependencies, libraries, configuration files and other binaries too, by containerizing the application platform and its dependencies, differences in operating system distributions and underlying infrastructure are abstracted away.

The term *container* extends the ability of a software component to be encapsulated and delivered in an isolated manner, ensuring that the operating environment would remain the same as the one in which it was created. Hence, containers are a solution to the problem of how to get software to run reliably when moved from one computing environment to another. This could be from a staging environment into production, from a physical machine in a data center to a virtual machine in a private or public cloud, or perhaps from an embedded device with high computing capabilities to one with fewer resources. While the functionality provided by container technology reminds certain similarities to the one offered by well-known virtualization systems such as Xen [90], KVM [91] or VMware [92], some subtle differences arise. Besides being the former a virtualization technique, it relies at the operating system level, while the latter operates at kernel-level. In this sense, containers do not provide a *virtual machine*, but a *virtual environment* with its own process and network space, typically sharing access to the main hardware across different containers, in addition to sharing the same kernel. Thanks to that, there is no additional overhead associated with having each guest executing a completely isolated operating system. This approach can also improve the performance since there is a single operating system dealing with hardware interruptions. Thus, the advantages offered by this technology make it very valuable to be implemented in the IoT domain, especially in low-level devices and gateways. Figure 52 depicts the architectural differences when implementing an Application + MySQL database using both approaches.

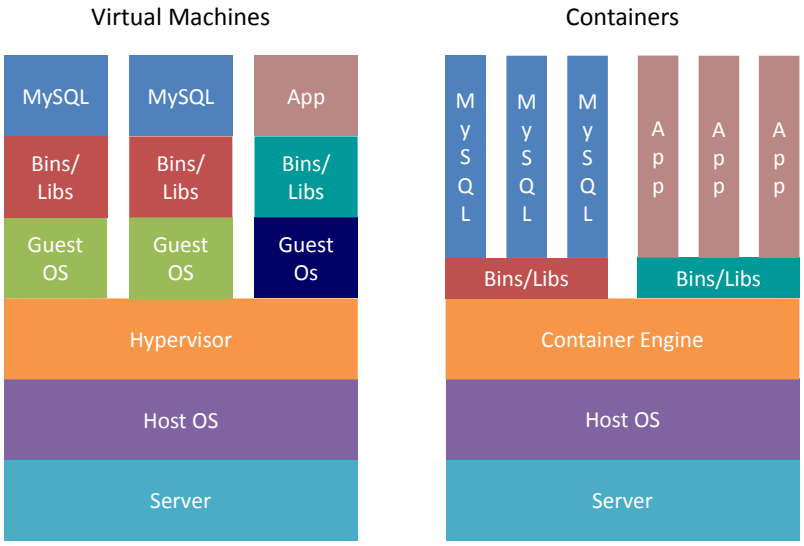

Figure 52: Virtual Machines vs Containers.

One of the most promising container systems nowadays is Docker [93], a technology that has made a strong impression in several communities, despite other also well-known container alternatives such as LXC [94], OpenVZ Virtuozzo [95], Solaris Zones [96] and FreeBSD Jails [97]. Among other characteristics, what made Docker different from other competitors was its simplicity procedures for installation and usage, well documented procedures, and being the

most successful at popularizing it. This yielded in a growing community of developers (DockerHub [98]) who are creating their own containers and sharing them with the community. However, Docker is not the only player in the game. Taking advantage of the container hype, CoreOS [99] comes into place as a tool that produces, maintains and utilizes open source software for Linux containers and distributed systems. In fact, the CoreOS team is developing its own container manager, rkt [100], which is able to handle not only its own rkt-like containers, but also Docker containers.

Following the evolution of the IoT regarding this domain and being willing to produce tools that are easy to be delivered (and easy to be deployed) by different communities of developers, the assets provided in this work are going to be applied for packaging and containerization, something that will be further detailed in the next deliverable, D5.2.3.

10. Management Components

In this deliverable we decided to focus on the analytic description of functionalities, models and mechanisms we developed and will develop instead of focusing exclusively on the description of functional components. However, in this section we provide a short description of all our functional components, addressing their core functionalities and relating them to the previous sections. All of the components are depicted in Figure 53. We do not analyze the components introduced in section 9 since they are depicted in detail.

Planner: The Planner of a VE is its brain, the component that enables it to reason on its Knowledge Base and take decisions depending on its situation. This functional component enables the VEs to use CBR, giving them the ability to learn from their own experiences and the experiences of other VEs. The Planner can be used for adaptation, but also for prediction too, depending on the way it is used. The Planner reasons on Cases defined by the Applications, but also on Cases for the self-management defined by the COSMOS platform (System Cases), enabling VEs to make reconfigurations and change some of their critical parameters. Finally, the Planner can be used generally as an Ontologies Comparator. This characteristic of the Planner combined with the functionality of the Decentralized Discovery component enables the evaluation of various COSMOS entities and the introduction of recommendation services. The functionalities of the Planner are analyzed in Section 4.

Decentralized Discovery component: This component is closely related to the XP-sharing functional component described in the deliverables of WP6. By using RESTful interfaces, individual VEs are connected on a peer to peer basis and can communicate with each other by sending and answering to requests for discovery of several entities. This component works closely with the Planner and uses the lists maintained by the Friends Management component. In Section 4.6 the mechanisms on which this component is based are described in detail.

Social Monitoring (SM) component: 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. Social Monitoring is analyzed mainly in subsection 8.1.

The Social Analysis (SA) component: Based on the results of the Social Monitoring component and taking advantage of Social Network Analysis (SNA) [101], 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 behavior 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, 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 this 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. This component offers both centralized and decentralized mechanisms. These functionalities are analyzed in subsections 8.2, 8.3, 8.4 and 8.5.

Friends Management (FM) component: This component is responsible for creating and maintaining all the Friends Lists and Black Lists that a VE has. In other words, it allows VEs to initiate, update and terminate their friendship with other VEs on the basis of the owner's or developer's control settings. It provides the owner with the option of setting new Friends to his/her VEs, offers friend-recommendation request services and monitors the Friend List of a VE regularly or on demand in order to find any Friends whose Dependability is no more the desired one and thus should be removed. For this purpose, the FM component communicates with the SA component. Features of this component are presented in subsection 8.1 and 8.3.

Profiling and Policy Management (PPM) component: This component is responsible for assigning a unique ID to the VE and enables the entry of all the information needed for the description of the physical entity through the domain ontology of the corresponding VE. Moreover, it enables the owner to determine the social "openness" of the VE: the IoT-services that can be used by other VEs, the kind of experience that can be shared, the sets of VEs which can access such information etc. However, the "openness" of VEs is affected by the social selfishness, a basic attribute of human beings. Thus, while designing this component, the concept of Opportunistic IoT [64] should be taken under consideration. Moreover, preferences regarding the thresholds, taken under consideration when social interactions take place (e.g. the Dependability thresholds), could be received through this component. This component is related to the Registry and the Consent Management component presented in D3.1.3.

Registry: The role of the Registry is to provide the adequate functionality for the retrieval of VEs and other COSMOS related entities descriptions. This Registry is in fact a semantic Registry and is backed by the core COSMOS ontology. The Registry and the corresponding ontologies are described in sections 6 and 7.

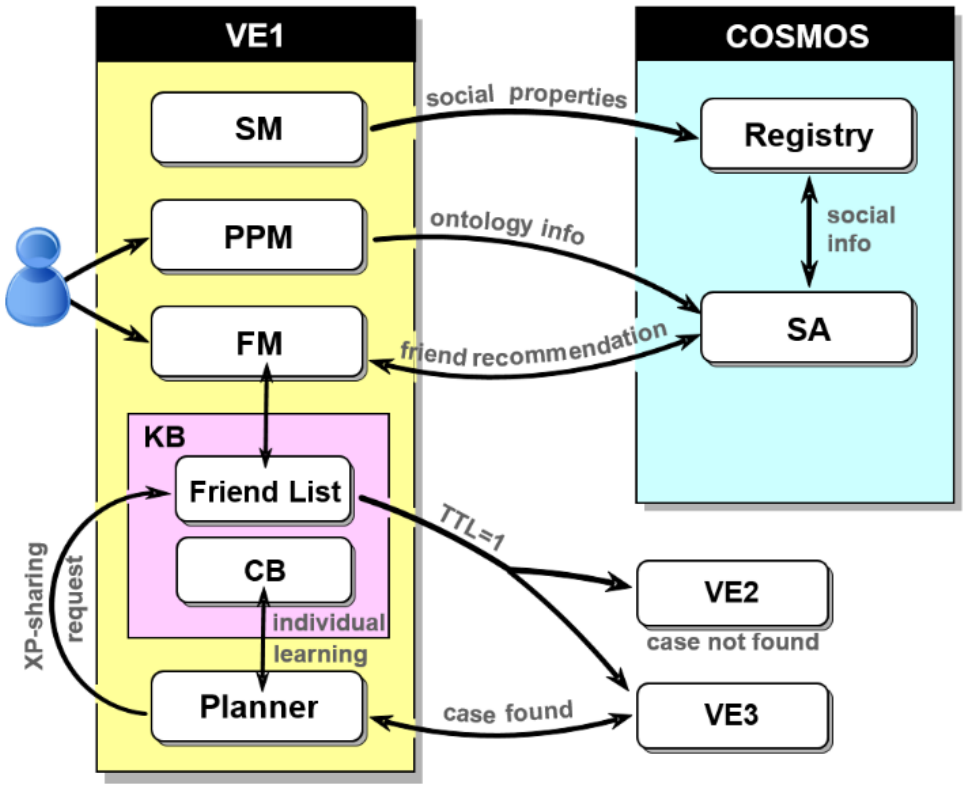

Figure 53: Conceptual view of WP5 components.

11. Show-casing our framework through an App

In this section we show-case our framework through analyzing the several steps needed to design a COSMOS-enabled Application. To do that, we adopt one of the Application Use Cases described in D7.1.2 (subsection 4.6) which is focusing on minimizing demand for heat energy production in smart-flats.

The **actors** that are taking part in this scenario are the:

- **VE Developer:** The VE developer is responsible for building COSMOS compliant VE-flats. He/she is responsible for respecting the APIs needed to register a VE with the platform and make it accessible to other VEs though well-known interfaces.
- **Application Developer:** The Application developer provides the (COSMOS-enabled) Applications utilizing the COSMOS offerings or mixing it with input from the VEs.
- **COSMOS Platform Provider:** COSMOS platform provides services through a standard interface to applications and VEs.
- **End-User (user):** The end-user is the entity that uses the COSMOS-enabled applications. The end-user runs the application which may use preconfigured VEs or VEs discovered at run-time.

The scenario described in D7.1.2 is as follows:

- Taking under consideration his/her **budget**, the owner of a flat (user) wants to set a **heating schedule** for his/her flat.
- To assist him, the IoT platform EnergyHive [102] designed by Hildebrand will use meters to report **real-time energy consumption information** automatically and remotely.
- Depending on these data, the user will change his/her consumption accordingly, thus reducing (sometimes) his/her **energy demand** (cost) and, as a result, minimizing **carbon production**.
- This scenario however has as a **postcondition** that the user can optimize his/her schedule. With COSMOS this condition is not needed. Moreover, COSMOS may be able to offer a **prediction** regarding the total budget needed for a schedule.

The scenario as it is now implies the monitoring of the readings of smart-meters by the End User and the need for continuous modification of settings of the program. Such an approach is time consuming and will eventually alienate users even if the data are provided in understandable monetary terms and not in consumption metrics. Taking under consideration the services that are offered by COSMOS, we can provide an improved scenario. According to it:

- Taking under consideration his/her **budget**, a user wants to set a **heating schedule** for his/her flat.
- The user plans a **program**, stating which is the **desired temperature** value for his/her flat for **specific time intervals** of the day/week etc.
- Because of COSMOS, the flat has a **knowledge base** of past programs that can be used and thus it is able to estimate **a)** how the actuators should be used to achieve the best possible consumption (not necessarily the optimal one) and **b)** the needed budget.
- Even if the flat does not have a good program in its knowledge base, it will be able to ask other similar flats for help.
- The user does not have to act in an optimal way but just states his/her desires. Moreover, a prediction regarding the total budget needed for a schedule is provided by COSMOS from the very beginning.

The main Physical Entity is the user's flat. Each flat should be equipped with:

- A **temperature sensor** measuring the temperature **inside** the flat.
- A **temperature sensor** measuring the temperature **outside** the flat. This is optional, as this information may be available by a weather website for example.
- The EnergyHive system providing real-time readings for the **energy consumption** (energy "**sensor**").
- A **valve up/down control system** (**actuator**).
- A device used for setting the schedule and (for example) displaying the readings of the sensors, e.g a **tablet ("interface")**.

The Virtual Entity (VE-flat) that will act as the virtual counterpart of the Physical Entity to the cyber-world will:

- expose the services of the Physical Entity (sensors/actuators) through **IoT-services**, using REST endpoints.
- be equipped with **functional components** provided by COSMOS like the Planner and the XP-sharing component.
- have a **Knowledge Base** consisting of:
  - ✓ a **Case Base** where cases are stored and retrieved or changed by the Planner and
  - ✓ one **Friend List** (for each application) with the addresses of other similar VEs used by the XP-sharing component.
- be linked to the **COSMOS platform** (using platform functional components) and other **Data Sources**, while the **Applications** will be using capabilities of the VE.

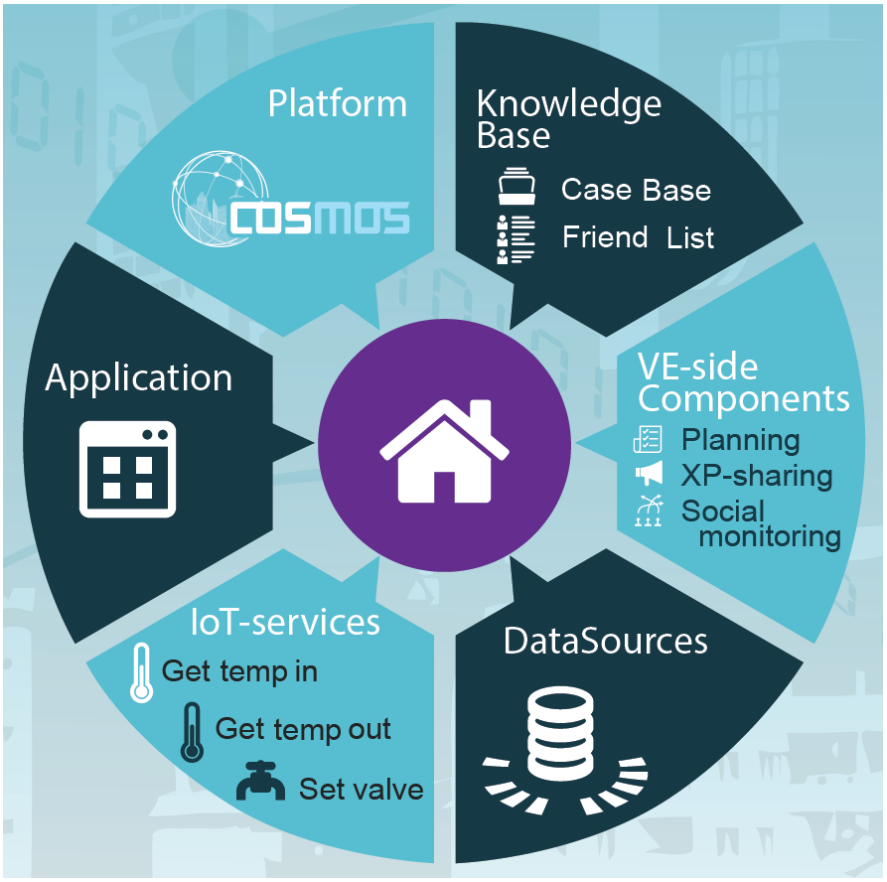

Figure 54: The COSMOS VE-flat.

The VE developer will have to:

- create the **IoT-services** of the VE.
- provide the **semantic description** of the Physical Entity and the services (VE and IoT-services) – Domain Ontology provided.
- **register** the VE to the COSMOS platform – COSMOS Ontology added.
- download the appropriate COSMOS "**system-code**" (Planner etc.) to the device where the VE "runs". In our case, this device could be a gateway or the tablet of the flat.

Regarding the Domain Ontology, properties of the flat needed in our scenario are:

- **location:** This property is needed for finding the right external temperature from data provided by weather-websites.
- **total surface** and **total volume (space):** may be needed for calculating the optimal (theoretical) energy consumption of the flat for a given schedule.
- **U-value:** An 'overall heat transfer co-efficient' which measures how well parts of a building transfer heat. U-values form the basis of any energy or carbon reduction standard.

These properties are also used as a **selection criteria for Friends**. The **Application developer** has to state which these criteria are and what their weights are.

Based on the Application logic, a CBR cycle will be formed:

- As it is mentioned in subsection 0, the CBR cycle has four steps: Retrieve, Reuse, Revise, Retain.
- The user wants a schedule for his/her heating needs. He/She inputs **a)** the **desired temperature** for **b)** a **period of time** (e.g. 1 day) and **c)** his/her **budget**.
- The application **creates** one or more new **Problems** based on this input.
- The Planner searches in the Case Base for Cases with similar Problems and returns the **Solution**. If nothing is found, XP-sharing is initiated (Retrieve).
- The user accepts the Solution or not (Reuse) and later he/she may provide feedback (Revise/Retain).

Regarding the **Problem Creation** defined by the Application logic, the inputs needed are **a)** the **desired temperature** for **b) periods of time** (e.g. throughout 1 day) **c)** for a given maximum **budget**. When the user is away from home, we may not care about the temperature. For every e.g. **half-hour** of the time-periods of interest, we create a **Problem** with properties:

- **Initial temperature inside** (provided by the corresponding IoT-service, needed only for the first problem)
- **Desired (final) temperature inside** (given by the user)
- **Temperature outside** (predicted by a weather website)

Thus, we create a chronological series of Problems. If we find Cases with similar Problems, we can use their Solutions.

Now, regarding the Solution Retrieval defined by the Application logic:

- A solution has as properties the **URI** of the IoT-service for setting (or not doing so) the valve and the **energy consumption** that corresponds to the problem.
- By executing the URIs at the corresponding time intervals, the **heating schedule** is executed.
- From the sum of the energy consumption of each individual Solution, we can find the **total (predicted) energy consumption**. This has to be lower than the **maximum energy**

consumption defined by the user's budget. This can be calculated by the budget and the cost of kWh provided by a website.
- If we want to evaluate the Solution, we can also find the **optimal theoretical energy consumption** using the HDD model [103].

Finally, regarding the Case Base that will be used, an **initial set of Cases** has to be created. The initial Case Base will be extracted from **historical data** derived from tracking the **energy-behavior of the user**:

- Over a period of time (e.g. six months) the COSMOS Cloud will accumulate readings for **Tin**, **Tout**, **consumption** and **flow rates** (for fixed intervals).
- From these data, for these specific time intervals (half-hour), cases will be created, having as:
  - ✓ **Problem: Tin_initial**, **Tin_final** (sensors), **Tout** (sensor/website)
  - ✓ **Solution: IoT-service** (up/down), **Consumption** (EnergyHive)

To sum up, the several steps that the Application Developer should take are:

- stating which data (and how) are forwarded to the **Cloud**: readings from sensors, the EnergyHive, the weather website etc.
- stating how the **CB** is created (what will be its **format**).
- stating how the **Problems** are **defined** by the user input.
- providing the internal logic/code of the Application. For example, he/she states how the **maximum consumption** is derived by the budget (e.g. using data from the energy-provider website).
- providing the GUI for the scheduling input and the report form for the outcomes.
- stating which properties of the VE (domain ontology) are important for finding similar friends.

On the other hand, COSMOS:

- offers to the App-developer **cloud storage**.
- by using CBR, helps the App-developers create **simple** Applications without having to create physical or mathematical models (**straightforward problem statement/solving**) or care about how the best solutions are found.
- by offering a **social community** to the VEs, makes sure that, if a good solution is available, it can be found. Offering optimal available solution.
- by offering **feedback mechanisms** (from the user or the system itself) the available knowledge can be refined.

12. Conclusion

In this document we analyzed all the main mechanisms and tools that we have developed in order to provide a complete **coordination framework** for managing the network of Things considering different administrative rules, locations, reputation and trust patterns, as well as IoT application requirements and interactions between the Things. These mechanisms and tools allow Things to react in an **autonomous** and predictive way based on the information retrieved from other Things experiences, enable **runtime adaptability** of the network and **decentralized discovery** of various entities, and create a **social environment** for the Things. The requirements, architecture and components design for the Things Management have been analyzed and the several processes of the infrastructure and the interfaces between the components were described in depth.

By providing the opportunity to the developers to create cases, VEs become more knowledgeable and gain useful Experience that can be shared and make them evolve. To this direction, the social nature of the VEs is supported by exploiting and developing social monitoring and analysis mechanisms. With the introduction of a CBR Planner, COSMOS becomes able to support basic and complex Autonomous Management requirements. The more complex and sophisticated the Planner, the smarter the Things. Many opportunities for Decentralized Management are available by working more on services such as the discovery mechanisms. For example, the fact that the VEs are able to require cases from their Friends, instead of querying the Registry immediately, is a good step towards decentralized knowledge sharing.

Of major importance is the social approach that COSMOS introduces in the IoT domain. The COSMOS system can be characterized as a SIoT one since it defines, monitors and exploits social relations and interactions between Things and uses technologies from the domain of the social media. The social side of COSMOS improves the knowledge flow and the sharing of services between Things, a really important characteristic for the constant evolution of the IoT systems.

COSMOS goes beyond the state of the art by identifying and establishing social properties and relations between VEs in such a way that the resulting social network is effectively manageable, by describing a decentralized IoT architecture which supports the functionality required to form a social network following the SIoT paradigm and by creating a new Trust & Reputation model addressing most of the security threats that apply to distributed knowledge systems.

13. References

[1] IoT-A project: http://www.iot-a.eu/public

[2] "Autonomic Computing: IBM's Perspective on the State of Information Technology", last accessed in 24/04/2016

[3] “IBM Unveils New Autonomic Computing Deployment Model”, last accessed in 24/04/2016

[4] T. Toyry, “Self-management in Internet of Things”

[5] D.A. Menasce, J.O.Kephart, "Guest Editors' Introduction: Autonomic Computing", Internet Computing, IEEE , vol.11, no.1, pp.18-21, Jan.-Feb. 2007

[6] E. Manoel, M. J. Nielsen, A. Salahshour, S. Sampath K.V.L., S. Sudarshanan, "Problem Determination Using Self-Managing Autonomic Technology", IBM Redbooks, June 2005

[7] “The autonomic computing edge: Can you CHOP up autonomic computing?”, IBM, 2008

[8] N. Bicocchi, F. Zambonelli, "Autonomic communication learns from nature", Potentials, IEEE, vol.26, no.6, pp.42-46, Nov.-Dec. 2007

[9] M.G. Hinchey, R. Sterritt,"Self-managing software", Computer, vol.39, no.2, pp. 107-109, Feb. 2006

[10] B. Jennings, S. van der Meer, S. Balasubramaniam, D. Botvich, M.O. Foghlu, W. Donnelly, J. Strassner, "Towards autonomic management of communications networks", Communications Magazine, IEEE , vol.45, no.10, pp.112-121, October 2007

[11] “An architectural blueprint for autonomic computing”, IBM, Autonomic Computing White Paper, June 2005, Third Edition

[12] CISCO, “The Internet of Things, Infographic”, 2011

[13] J. Rowley, “The wisdom hierarchy: representations of the DIKW hierarchy”, Journal of Information Science 33(2, 2007, pp. 163-180

[14] A. Aamodt, E. Plaza, “Case-Based Reasoning: Foundational Issues, Methodological Variations and System Approaches”, Artificial Intelligence Communications, 1994, pp. 39-59

[15] S. Dutta, P. P Bonissone, “Integrating case- and rule-based reasoning”, International Journal of Approximate Reasoning, Vol. 8, Issue 3, May 1993, pp. 163–203, Elsevier

[16] Z. Budimac, V. Kurbalija, “Case Based Reasoning – a short overview”, Proceedings of the Second International Conference on Informatics and Information Technology, pp. 222-233

[17] V. Supyuenyong, N. Islam, “Knowledge Management Architecture: Building Blocks and their Relationships”, Technology Management for the Global Future, PICMET 2006, Vol. 3

[18] R. Bergmann, J. Kolodner, E. Plaza, “Representation in case-based reasoning”, The Knowledge Engineering Review, Vol. 00:0, 2005, pp. 1–4, Cambridge University Press

[19] X. Wang, D. Zhang, T. Gu, H. Pung, “Ontology based context modeling and reasoning using OWL”, Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004., pp. 18-22, March 2004

[20] A. A. Assali, D. Lenne, B. Debray, “Case Retrieval in Ontology-Based CBR Systems”, 32nd Annual Conference on Artificial Intelligence (Ki 2009), Paderborn, 2009, pp. 564-571

[21] R. L. D. Mántaras, D. Mcsherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. L. Maher, M. T. Cox, K. Forbus, M. Keane, A. Aamodt, I. Watson, “Retrieval, reuse, revision, and retention in Case-Based Reasoning”, The Knowledge Engineering Review, Vol. 00:0, 1–2 Cambridge University Press, 2005

[22] I. Watson, “Case-based reasoning is a methodology not a technology”, Knowledge-Based Systems”, Vol. 12, 1999, pp. 303-308

[23] R. Davis, J. J. King, “The Origin of Rule-Based Systems in AI”

[24] E. L. Rissland, D. B. Skalak, “Combining Case-Based and Rule-Based Reasoning: Heuristic Approach”, pp. 534-530

[25] S. Brouninghaus, K. D. Ashley, “Combining Case Based and Model-Based Reasoning for Predicting the Outcome of Legal Case” (2003)

[26] J. Prentzas, I. Hatzilygeroudis, “Categorizing Approaches Combining Rule-Based and Case-Based Reasoning”

[27] C. Marling, E. Rissland, A. Aamodt, “Integrations with case-based reasoning”, The Knowledge Engineering Review, Vol. 00:0, Cambridge University Press, 2005, pp. 1–4

[28] J. Hage, ”A Low Level Integration of Rule-based Reasoning and Case-based Reasoning.”, pp. 30-39

[29] K. A. Kumar, Y. Singh, S. Sanyal, “Hybrid approach using case-based reasoning and rule-based reasoning for domain independent clinical decision support in ICU”, (2009) Expert Systems with Applications 36, pp. 65-71

[30] K. Khandelwal, D. P. Sharma, “Hybrid Reasoning Model for Strengthening the problem solving capability of Expert Systems”, International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 4, No. 10, 2013

[31] M.G.Walker, R.Kapadia, B.Sammuli, M.Venkatesh, “A Model-based Reasoning Framework For Condition Based Maintenance and Distance Support”

[32] S. Niwattanakul, J. Singthongchai, E. Naenudorn and S. Wanapu, “Using Jaccard Coefficient for Keywords Similarity”, Proceedings of the International MultiConference of Engineers and Computer Scientists 2013. Vol. 1, IMECS 2013.

[33] J. Jacobs, “Comparing Communities: Using β-diversity and similarity/dissimilarity indices to measure diversity across sites, communities, and landscapes”.

[34] S. Soltani, “Case-Based Reasoning for Diagnosis and Solution Planning”, Technical Report No. 2013-611, School of computing Queen’s university Kingston, Ontario, Canada October 2013.

[35] T. Osman, D. Dhavalkumar, D. Al-Dabass, “Semantic-Driven Matchmaking of Web Services Using Case-Based Reasoning”, IEEE International Conference on Web Services (ICWS'06), 2006, pp. 29-36

[36] S. Lajmi, C. Ghedira, K. Ghedira, D. Benslimane, “CBR Method for Web Service Composition. In Advanced Internet Based Systems and Applications”, Lecture Notes in Computer Science, Vol. 4879, 2009, pp. 314-326

[37] H. Fouad, A. Baghdad, “Dynamic Web Service Composition: Use of Case Based Reasoning and AI Planning”

[38] Node-RED: http://nodered.org/

[39] U.M. Borghoff, R, Pareschin, “Information Technology for Knowledge Management”, Springer, 1998.

[40] J. Guare, “Six Degrees of Separation: A Play”, Vintage Books, 1990

[41] Rodriguez, “RESTful Web Services: The basics”, Developer works page REST, 2008

[42] Raspberry Pi: https://www.raspberrypi.org/
[43] Raspbian: https://www.raspbian.org/
[44] Apache Jmeter®: http://jmeter.apache.org/
[45] Linked Data - Connect Distributed Data across the Web, http://linkeddata.org/
[46] L. Atzori, A. Iera, G. Morabito, M. Nitti, "The Social Internet of Things (SIoT) – When social networks meet the Internet of Things: Concept, architecture and network characterization", Computer Networks, Vol. 56, Issue 16, 14 Nov. 2012, pp. 3594-3608
[47] T. Falk, U. Fischbacher, "A theory of reciprocity," Gamesand Economic Behavior, Vol. 54, no. 2, 2006, pp. 293-315
[48] L. Page, S. Brin, R. Motwani, T. Winograd, "The pagerank citation ranking: Bringing order to the web", 1999
[49] K. S. Cook, R. M. Emerson, M. R. Gillmore, T. Yamagishi, "The Distribution of Power in Exchange Networks: Theory and Experimental Results", American Journal of Sociology, Vol. 89, No. 2 (Sep. 1983), pp. 275-305
[50] H. Al-Qaheri, S. Banerjee, G. Ghosh, "Evaluating the power of homophily and graph properties in Social Network: Measuring the flow of inspiring influence using evolutionary dynamics", Science and Information Conference (SAI), 2013, pp. 294-303
[51] H.Z. Asl, A. Iera, L. Atzori, G. Morabito, "How often social objects meet each other? Analysis of the properties of a social network of IoT devices based on real data", Global Communications Conference (GLOBECOM), 2013 IEEE, pp. 2804-2809
[52] D.A. Bader, K. Madduri, "Parallel Algorithms for Evaluating Centrality Indices in Real-world Networks", International Conference on Parallel Processing (ICPP), 2006, pp. 539-550
[53] A.R.M. Teutle, "Twitter: Network Properties Analysis", 20th International Conference on Electronics, Communications and Computer (CONIELECOMP), 2010, pp. 180-186
[54] R. S. Burt, "Structural holes and good ideas", American Journal of Sociology, Vol. 110, 2004, pp. 349–399
[55] E. Zhang, G. Wang, K.Gao, X. Zhao, Y. Zhang, "Generalized structural holes finding algorithm by bisection in social communities", Sixth International Conference on Genetic and Evolutionary Computing (ICGEC), 2012, pp. 276-279
[56] R. Albert, Al Barabasi, "Statistical mechanics of complex networks", Reviews Of Modern Physics, Vol. 74, 2002
[57] D. Knoke, "Social Network Analysis", Quantitative Applications in the Social Sciences series, SAGE Publiations, Second Edition, 2008
[58] M. Tsvetovat, J. Reminga, K. Carley, "DyNetML: A Robust Interchange Language for Rich Social Network Data", Institute for Software Research, International Carnegie Mellon University
[59] GraphML File Format: http://graphml.graphdrawing.org/
[60] L. Shi, C. Wang, Z. Wen, "Dynamic Network Visualization in 1.5D", Pacific Visualization Symposium (PacificVis), 2011 IEEE, pp. 179-186
[61] J. M. Huisman, M. A. J. van Duijn, "A reader's guide to SNA software", The SAGE Handbook of Social Network Analysis, J. Scott and P.J. Carrington Editions, 2011, pp. 578-600
[62] N. Akhtar, "Social Network Analysis Tools", Fourth International Conference on Communication Systems and Network Technologies (CSNT), 2014, pp. 388-392

[63] M.Bastian, S.Heymann, M.Jacomy, “Gephi: an open source software for exploring and manipulating networks”, International AAAI Conference on Weblogs and Social Media, 2009

[64] B. Guo, Z. Yu, X. Zhou, D. Zhang, “Opportunistic IoT: Exploring the social side of the internet of things”, Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference

[65] Y. Zhang, J. Wen, “An IoT Electric Business Model Based on the Protocol of Bitcon”, 18th International Conference on Intelligence in Next Generation Networks, pp. 184-191

[66] F. Almenarez, A. Marin, C. Campo, C. Garcia, “PTM: a pervasive trust management model for dynamic open environments”, First workshop on pervasive security and trust, Boston, USA; 2004

[67] M. Moloney, S. Weber, “A context-aware trust-based security system for ad hoc networks”, Workshop of the 1st International Conference on Security and Privacy for emerging areas in communication networks, Greece; 2005, pp. 153–60

[68] Boukerche, L. Xu and K. El-Khatib, “Trust-based security for wireless ad hoc and sensor networks”, Computer Communications 2007

[69] J. Sabater and C. Sierra C, “REGRET: reputation in gregarious societies”, Proceedings of the 5th International Conference on Autonomous Agents, Canada, 2001

[70] S. Marti and H. Garcia-Molina, “Taxonomy of trust: categorizing P2P reputation systems”, Computer Networks 2006

[71] F. Almenarez, A. Marin, D. Diaz, J. Sanchez, “Developing a model for trust management in pervasive devices”, Proceedings of the 4th annual IEEE International Conference on Pervasive Computing and Communications Workshops, IEEE Computer Society; 2006. p. 267

[72] J. Carbo, J. Molina and J. Davila, “Trust management through fuzzy reputation”, International Journal of Cooperative Information Systems, 2003

[73] S. Songsiri, “MTrust: a reputation-based trust model for a mobile agent system”, Autonomic and Trusted Computing, 3rd international conference, vol. 4158, 2006, pp. 374–385

[74] F. G. Marmol and G. M. Perez, “Providing trust in wireless sensor networks using a bio-inspired technique”, Proceedings of the networking and electronic commerce research conference, NAEC’08, Italy; 2008

[75] W. Wang, G. Zeng and L. Yuan, “Ant-based reputation evidence distribution in P2P networks”, 5th International Conference on Grid and Cooperative Computing, IEEE Computer Society; 2006, pp. 129–132

[76] F. G. Marmol, G. M. Perez and A.G. Skarmeta, “TACS, a trust model for P2P networks”, Wireless personal communications, special issue on ‘‘Information Security and Data Protection in future generation communication and networking’’, 2008

[77] S. Kamvar, M. Schlosser and H.Garcia-Molina, “The EigenTrust algorithm for reputation management in P2P networks”, 2003

[78] L. Xiong, L. Liu, “PeerTrust: supporting reputation-based trust in peer-to-peer communities”, IEEE Transactions on Knowledge and Data Engineering, 2004, pp. 843–857

[79] OASIS: https://www.oasis-open.org/committees/orms

[80] B. Can, B. Bhargava, “SORT: A Self-Organizing Trust Model for Peer-to-Peer Systems”, IEEE Transactions on Dependable and Secure Computing, Vol. 10, 2013.

[81] F. M. Gomez, G. M. Perez, “Security threats scenarios in trust and reputation models for distributed systems”, Computers & Security, 2009; pp. 545–556

[82] R. Zhou, K. Hwang, “PowerTrust: a robust and scalable reputation system for trusted peer-to-peer computing”, Transactions on Parallel and Distributed Systems, 2007
[83] J. Douceur and J. Donath, “The Sybil attack”, Proceedings for the 1st International Workshop on P2P systems (IPTPS ’02); 2002. pp. 251–260
[84] S. Lam, J. Riedl, “Shilling recommender systems for fun and profit”, WWW ’04: Proceedings of the 13th International Conference on World Wide Web; 2004
[85] J. Girao, A. Sarma, R. Aguiar, “Virtual identities – a cross layer approach to identity and identity management”, Proceedings for the 17th wireless world research forum, Heidelberg, 2006
[86] F. G. Marmol, G. M. Perez, “TRMSim-WSN, Trust and Reputation Models Simulator for Wireless Sensor Networks”
[87] F. G. Marmol, “Implementing and Integrating a new Trust and/or Reputation Model in TRMSim-WSN”
[88] Chlipala, J. Hui, G. Tolle, “Deluge: Data Dissemination for Network Reprogramming at Scale”, CS262/CS294-1, Fall 2003 Class Project
[89] J. Rico, J. Sancho et al, “Trusted computing for embedded systems”, Chapter 5, Springer January 2015
[90] Xen Server overview: https://www.citrix.com/products/xenserver/overview.html
[91] Kernel-based Virtual Machine: https://openvirtualizationalliance.org/what-kvm
[92] VMware VSphere virtualization: https://www.vmware.com/products/vsphere/
[93] Docker website: https://www.docker.com/what-docker
[94] Introduction to LXC: https://linuxcontainers.org/lxc/introduction/
[95] OpenVZ Virtuozzo Containers: https://openvz.org/Virtuozzo
[96] Solaris Zones: http://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html
[97] FreeBSD Jails handbook: https://www.freebsd.org/doc/handbook/jails.html
[98] Docker Hub Community: https://hub.docker.com/
[99] Main CoreOS website: https://coreos.com/docs/
[100] CoreOS rkt website: https://coreos.com/rkt/
[101] S. Wasserman and K. Faust, “Social Network Analysis: Methods and Applications”, Structural Analysis in the Social Sciences, Cambridge University Press.
[102] EnergyHive: http://www.energyhive.com/
[103] HDD: http://en.wikipedia.org/wiki/Heating_degree_day