Empowering IoT through Cognitive Technologies

iCore

Internet Connected Objects for Reconfigurable Ecosystem

Grant Agreement Nº 287708

D2.5 Final architecture reference Model

WP2 Cognitive management and control framework for IoT

Version:	1
Due Date:	M36
Delivery Date:	02.11.2014
Nature:	Report
Dissemination Level:	Public
Lead partner:	TCS
Authors:	TCS, ALU, UPRC, SIEMENS, VTT, CNET, M3S, SAG, AMBIENT, UNIS, ATOS
Internal reviewers:	VTT, SIEMENS, ALU, CRF, ZIZPOS, ATOS, AMBIENT

www.iot-icore.eu

SEVENTH FRAMEWORK PROGRAMME

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 287708

Version Control

Revision	Date	Author(s)	Author's Organization	Description
0.1	18/08/2014	Stéphane Ménoret	Thales Communications and security (TCS)	First ToC and calls for requirements
0.2	19/09/2014	Stéphane Ménoret	Thales Communications and security (TCS)	Last ToC update following PMT meeting on 18/09/2014; now features centric
0.4	25/09/2014	Stéphane Ménoret	Thales Communications and security (TCS)	Updates for section 2
0.5	30/09/2014	Stéphane Ménoret	Thales Communications and security (TCS)	Updates for sections 3,4,5
1.0	24/10/2014	Stéphane Ménoret	Thales Communications and security (TCS)	Updates for lessons learnt and section 6; internal review release candidate
1.0 final	02/11/2014	Stéphane Ménoret	Thales Communications and security (TCS)	Final release with internal review comments addressed

Deliverable Title **iCore final iCore architecture reference model**

Deliverable Number **D2.5**

Keywords: IoT, architecture, Virtual Object Level, Composite Virtual Object Level, Service Level, Real World Knowledge, System Knowledge, Policy Decision Point, Policy Enforcement Point

Executive Summary:

Initially reported in deliverable D2.3 the iCore architecture reference model has been refined according to implementation work and validation feedbacks from other technical Work Packages 3, 4, 5, 6 and 9. This document illustrates the resulting final iCore architecture reference model through lessons learnt and selected examples taken from iCore use cases and trials.

Contributors

First Name	Last Name	Affiliation	Email
Stephane	Menoret	Thales TCS	stephane.menoret@thalesgroup.com
Marc	Roelands	ALUB	marc.roelands@alcatel-lucent.com
Raffaele	Giaffreda	CNET	raffaele.giaffreda@create-net.org
Swaytha	Sasidharan	CNET	swaytha.sasidharan@create-net.org
Vera	Stavroulaki	UPRC	veras@unipi.gr
Minerva	Roberto	TI	roberto.minerva@telecomitalia.it
Panagiotis	Vlacheas	UPRC	panvlah@unipi.gr
Vassilis	Foteinos	UPRC	vfotein@unipi.gr
Antonis	Moustakos	UPRC	amoustak@unipi.gr
Matti	Etelapera	VTT	Matti.Etelapera@vtt.fi
Ricardo	Neisse	JRC	Ricardo.Neisse@jrc.ec.europa.eu
Gianmarco	Baldini	JRC	gianmarco.baldini@jrc.ec.europa.eu
Andrea	Parodi	M3S	andreaparod@gmail.com
Stylianos	Georgoulas	UniS	s.georgoulas@surrey.ac.uk
Septimiu	Nechifor	Siemens	septimiu.nechifor@siemens.com
Lucian-Mircea	Sasu	Siemens	lucian.sasu@siemens.com
Jorge	Pereira Carlos	ATOS	jorge.pereirac@atos.net
Dimitris	Kelaidonis	UPRC	kelaidonis@gmail.com
Lodewijk	van Hoesel	Ambient	lodewijk.vanhoesel@ambient-systems.net
Dimitrios	Karvounas	UPRC	dkarvoyn@unipi.gr
Aristi	Galani	UPRC	agalani@unipi.gr
Alexandros	Antzoulatos	UPRC	alexant@unipi.gr
Joachim	Teutsch	Software AG	Joachim.Teutsch@softwareag.com
Akshay	Uttama Nambi,	TU Delft	Akshay.Narashiman@tudelft.nl
Aristotelis	Margaris	UPRC	amargaris@unipi.gr
Aimilia	Bantouna	UPRC	abantoun@unipi.gr
Andreas	Georgakopoulos	UPRC	andgeorg@unipi.gr
Stefan	Meissner	UniS	s.meissner@surrey.ac.uk
Nikolaos	Loumis	UniS	n.loumis@surrey.ac.uk
Miguel	Rodriguez Fuentes	ATOS	miguel.rodriguezf@atos.net
Panagiotis	Demestichas	UPRC	pdemest@unipi.gr
Evangelia	Tzifa	UPRC	etzifa@unipi.gr
Georgios	Poulios	UPRC	gpoulios@unipi.gr
Vassileios	Foteinos	UPRC	vfotein@unipi.gr
Konstantinos	Tsagkaris	UPRC	ktsagk@unipi.gr
Yiouli	Kritikou	UPRC	kritikou@unipi.gr
Marios	Logothetis	UPRC	mlogothe@unipi.gr

Table of Acronyms

Acronym	Meaning
API	Application Programming Interface
CCTV	Close circuit television
CVO(L)	Composite Virtual Object (Level)
CAPEX	Capital expenditures
CoAP	Constraint Application Protocol
DoW	Description of Work
(G)UI	(Graphical) User Interface
HVAC	Heating, Ventilation and Air-Conditioning
ICT	Information and Communications Technologies
IERC	European Research Cluster on the Internet of Things
IoT	Internet of Things
IoT-A	Internet of Things Architecture
OPEX	Operating Expense
OWL	Web Ontology Language
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PoC	Proof Of Concept
RWO	Real World Object
RDF	Resource Description Framework
RTSP	Real-Time Streaming Protoco
SA	Situational Awareness
SER	Service Execution Request
SRA	Service Request Analysis
SL	Service Level
SLA	Service Level Agreement
ST	Service Template
VO(L)	Virtual Object (Level)
WP	Work Package
w.r.t	With Respect To
RWO	Real World Objects
RWK	Real World Knowledge

SK	System Knowledge
OGC	Open Geospatial Consortium
MQTT	Message Queue Telemetry Transport
WSN	Wireless Sensors Network

Table of Contents

List of Figures

List of Tables

1. Introduction

This document represents the new architectural achievements of the iCore Project. It is based upon results presented in the deliverable D2.3[1] (iCore Architecture Reference Model), and extends those by highlighting the improvements and refinements or amendments to the architecture needed for encompassing the requirements posed by use cases and trials. This gives a unique consolidated view of the iCore architecture strengthened by technical implementations and new architecture refinements achieved during the whole project. The goal is not to rewrite entirely the deliverable D2.3 but rather to complete with relevant points of view the huge amount of work that represents the deliverable D2.3. In other words this document consolidates the architecture proposed in D2.3 by considering the outcomes and return of experience of several use cases. In particular technical architecture updates and feedbacks coming from WP6 and WP9 are explained and justified against the Proofs of Concepts concerns whatever they come from operational constraints, business constraints or purely technical constraints. Possibly iCore functional architecture is revisited since not all the features of iCore architecture have been used extensively in the same measure across all the use cases, though there is a wide coverage of features that has been achieved collectively between all use-cases and trials.

The goal of the document is to provide advancements and conclusions that explain **why** and **how** use cases and trials have used and benefited from the iCore architecture **and vice versa**.

In chapter 2 we remind the overall iCore architecture principles with rationales, benefits and main features supported. We organized the rest of the chapter with short dedicated sections for each of the use cases and trials (from WP6 and WP9), i.e., smart home, smart office, smart city transportation, smart urban security, smart business, medical asset management and smart tour in the city.
Each section introduces the overall business domain context and what role the iCore platform plays in this context as an element of a more global ICT system. A description is provided for each business domain scenario driving the use cases and trials demonstration describing typically interactions between stakeholders. The goal here is to clarify to the readers the purpose and the added value of an iCore platform, implementing iCore architecture for a given demonstrator and business domain context with emphasis on iCore main features overviewed in following chapters.

Chapter 3, 4 and 5 are about the technical architecture feedbacks from WP6 and WP9 all related to the three functionalities of iCore architecture, namely “Virtual Object” (VO) level, “Composite Virtual Object” (CVO) level and “Services Logic” (SL) level, and the possible improvements based on lessons learnt. Each chapter starts with a summary of the level purpose; then, main features of each architecture level are described with their rationales and findings about cognition and its concrete benefits; how levels are interrelated is highlighted as well. An overall picture showing how the features are supported by iCore components is provided whenever there is the need to show relationships between components at the specific abstraction level.
In addition, concrete examples extracted from use case and trial scenario explain how the features are used and how technical standards are used (e.g. at VO level, how sensors discovery is used, how MQTT supports this).
These chapters refer to deliverables D3.4[2], D4.4[3] and D5.4[6] related to Proof Of Concepts (PoC) validation since these ones provide also feedbacks from use cases but with a validation point of view.
A last but not least subsection summarizes the main elements described in each chapter in order to recap and highlight lessons learnt, open issues and possible updates of iCore architecture.

Finally chapter 6 is dedicated to a set of overall statements covering the whole iCore architecture and its use for the different vertical domains addressed; in particular its applicability/generalization to these domains is discussed.

2. ICore architecture for business domains

This chapter provides an overview of the overall iCore architecture comprising its principles with rationales, benefits and main features supported. The chapter comprises also short dedicated sections for each of the use cases and trials (from WP6 and WP9), so smart home, smart office, smart city transportation, smart urban security, smart business (supply management), medical asset management and smart tour in the city.

2.1 ICore architecture overview and reminders

iCore architecture is a solution designed and developed for speeding up the creation, deployment, execution and back drawing of new IoT services and applications. These applications have also the advantage to exploit and leverage the adaptation capabilities offered by the platform in terms of virtualization, aggregation and abstraction properties. Application that have strong requirements regarding adaptation to changing situations (self-x properties and context awareness) can largely benefit from the iCore cognitive capabilities in order to execute and achieve their tasks. iCore architecture defines a programmable framework with three levels of abstraction and virtualization that application can use to control and make use of real world objects and internal platform functionalities. These levels are from top to bottom:

- A way to **invoke or query for Services** programmable in the platform, as specific to supported Application Domains automatically taking into account corresponding **Real World Knowledge (RWK)** towards the platform-internal service instance specification.
- A run-time management and execution environment that **efficiently manages using System Knowledge (SK) and executes the requested pool of service instances** as a composition of so-called **Composite Virtual Objects (CVOs)**, connected to an
- **Abstraction of Real World Object data** (via sensors and actuators) with functional enrichment, though so-called **Virtual Objects (VOs)**.

Next figure represents the logical architectural separation within the iCore architecture and relates the execution of iCore components derived by the designed objects on the left, with a set of repository that do represent the real world knowledge about a specific problem domain.

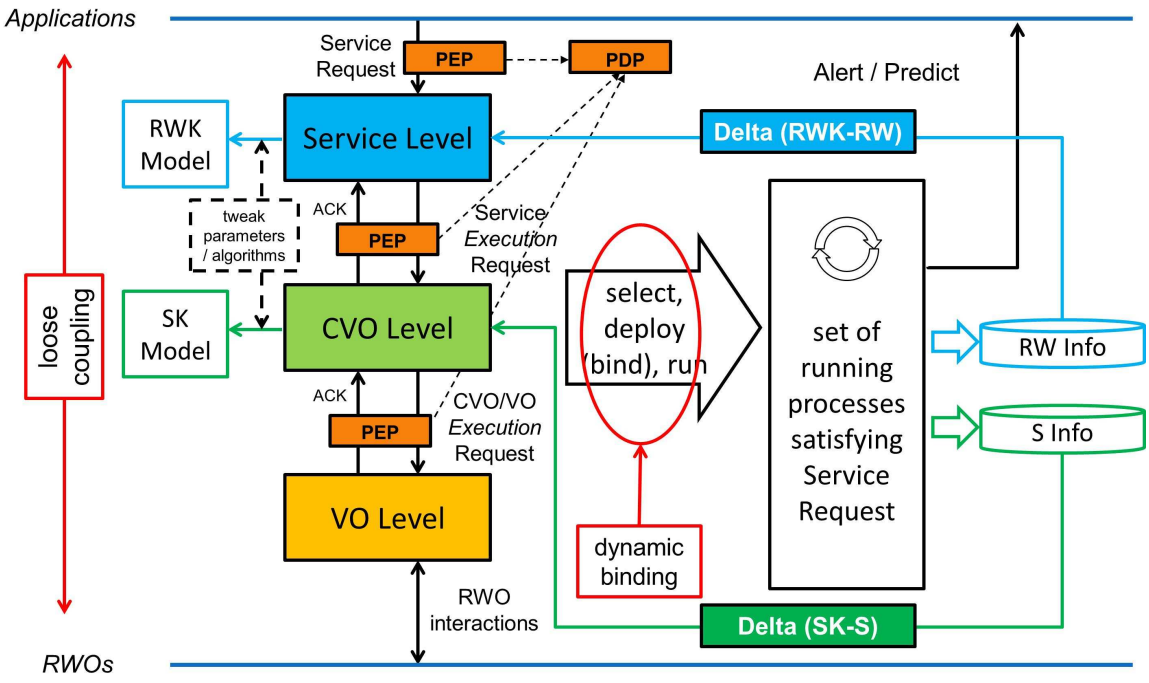

Figure 1: iCore architecture main principles.

The three levels' purposes are respectively reminded in next chapter 3, 4 and 5. We highlight hereafter main rationales and benefits of this architecture level by level from top to bottom.

The main cognitive innovation targeted by the iCore **Service Level** (SL) relates to the leveraging of so-called **Real-World Knowledge** (RWK), and the growing of that as a platform asset, for improving the execution efficiency, the effectiveness and situation awareness of services. The Service Level is the "brain" of the architecture planning and understanding what requested services have as an effect/goal in the Real World. As such, it determines the strictly needed service instance specification/"contract"/Service Level Agreement (SLA) that the platform is to deliver/execute by means of its CVO (and VO) Level, by considering constraints/redundancies (RWK) that exists in the Real World in which it acts.

The **CVO Level** (CVOL) **executes/fulfils** the service specifications derived by the Service Level, and **adds system resource concerns** to the equation as captured in **System Knowledge** management functions. If from the point of view of adding intelligence Service Level represents the brain towards the outer world, CVO Level should expose same type of intelligence towards the inner machinery, the one coordinating optimal placement, deployment and execution of CVO and it's subcomponents in CVO containers. CVO Level maps at best Service Execution Requests in executable Composite Virtual Objects – coherent functional grouping of VO Services and "smart glue code" based on RWK models and situational data. From case to case iCore implementations showed various types of distribution for the components of cognitive cycle (with its specific autonomic characteristics) regarding sensing and monitoring, reasoning, execution planning and adaptation. Most common consumed approach used the distribution of roles between Service Level and CVO Level leaving to the Service Level the coordination role of sensing and "situation understanding" and concentrating in CVO Level executional capabilities, including situation observation implemented as CVOs. Another approach may suppose an increased decoupling between Service Level and CVO Level placing Situation Observation at Service Level.

Within the IoT context, according to the huge heterogeneity of connected equipment, sensors and actuators, abstraction is a means that makes them usable in a uniform way by a multitude of

applications and services across a wide set of application domains and with varying requirements. This makes the VO Level the third important element to the platform. **VO level** (VOL) approaches the vast heterogeneity of the sensed physical world with semantic technologies. This allows sensors and actuators from multiple vendors to interact in iCore context, enabling the creation of device mash-ups at a higher abstraction level. VO level hosts components that allow registration and discovery of VOs and a set of interfaces towards upper iCore levels. VO discovery is performed by CVO level entities possibly using powerful semantic queries. In iCore use cases we have implemented protocols using both publish-subscribe and request-response communication patterns for accessing information resources hosted by VOs. Although cognition is mainly present at the Service Level in iCore, various aggregation algorithms can be implemented at the very edges of the network in order to provide enriched data.

Finally, in all levels **Policy Enforcement Point** (PEP) components may be deployed to intercept Service, CVO, and VO execution requests. The interception of these request generate events that are signalled to a **Policy Decision Point** (PDP) component, that evaluates security policies and return enforcement actions to the PEPs. The result of a policy evaluation may allow, deny, modify, or delay the execution requests in case policies controlling the respective request are deployed. Section 2.6 presents a concrete example where PEPs are used at the VO level to prevent the monitored information from being accessed by unauthorized entities. Section 5.5 shows examples of security policies and the Model-based Security Toolkit (SecKit) that is used for enforcement and evaluation of policies.

The iCore functional architecture is depicted in next figure.

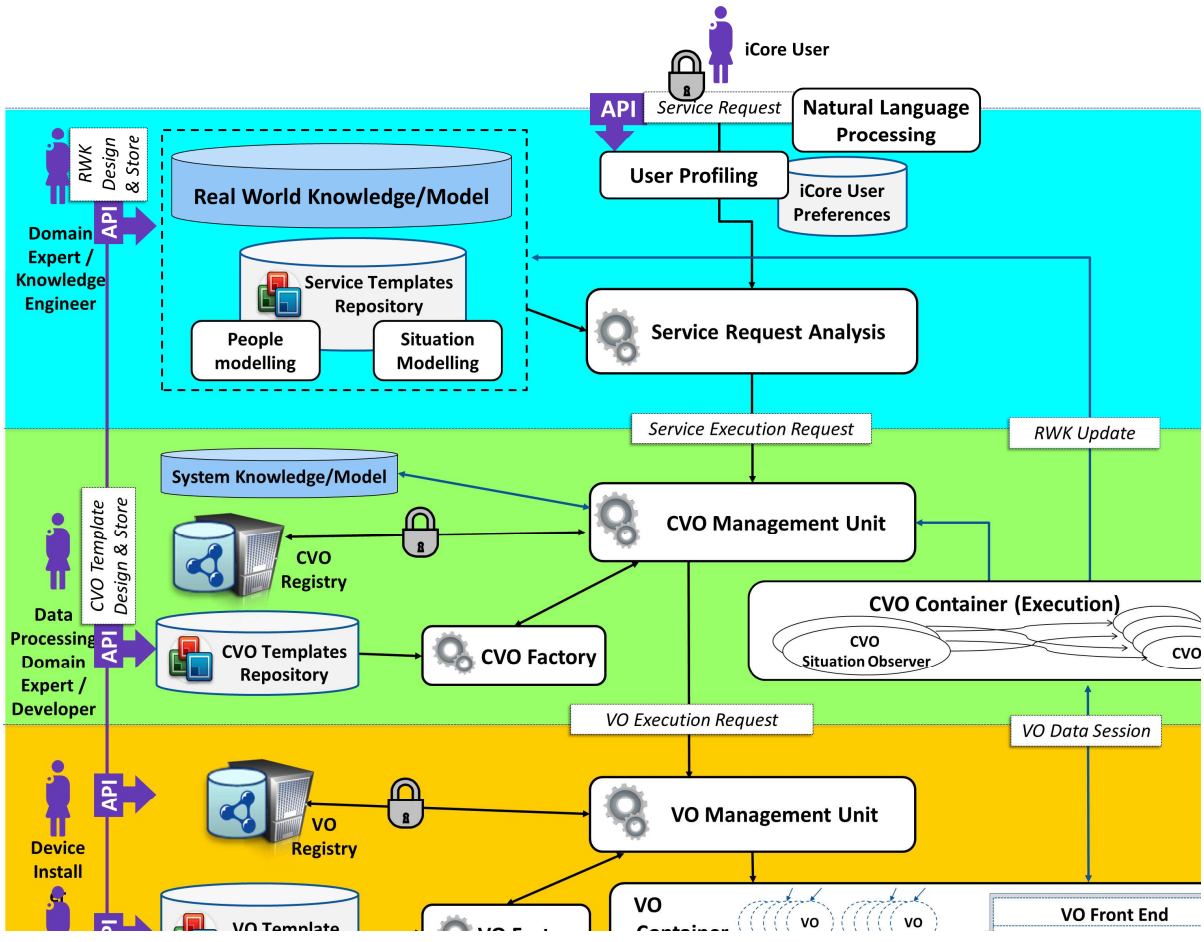

Figure 2: iCore 3 levels functional architecture

Two examples of sequence diagrams are provided in appendix (chapter 7) to illustrate iCore components interfaces and relationships.

2.2 Smart Home: Living Assistant

2.2.1 Business context – requirements and services

The 'smart home living assistant' use case (see D6.2 [8]) refers to the PoC prototype for the validation and performance evaluation of the iCore framework, both in terms of technical and business perspective. In particular the scenario of this PoC comprises two different (business) domains; (a) a smart home, where an elderly woman (Sarah) who has opted for an assisted living service lives in, and (b) a medical centre, where doctors monitor Sarah's environmental conditions and health status remotely, using the smart objects that exist in smart home. Next figure introduces the concept:

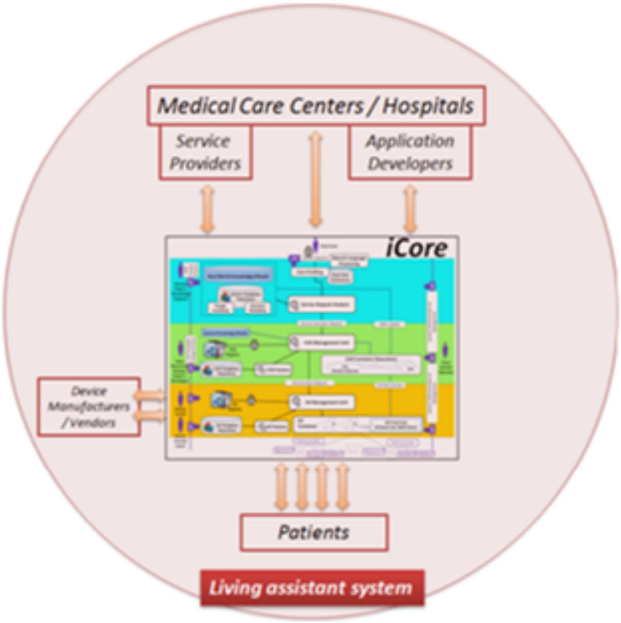

Figure 3: iCore platform positioning in a living assistant system

In this particular use case, the iCore platform is introduced to support and enhance the operation of such integrated system, allowing in the same time the facilitation of new innovative capabilities for the participants in the PoC.

2.2.2 Contribution to architecture and major components used

Initially from the technical point of view, iCore platform is used in the following way. Since the platform is running, the doctor provides a set of application requirements to the service level through an appropriate user interface. The Natural Language Processing (NLP) and Service Request Analysis (SRA) components take over to analyse doctor's request and extract a set of request and situation parameters that are forwarded to the CVO Level and particularly to the Approximation and Reuse Opportunity Detection (AROD) mechanism, which in turn, search in the CVO registry for a previously created CVO that could fulfil the requested application. If an appropriate CVO is not found, the provided parameters are sent to the CVO Composition Engine (CCE), which will select the most appropriate VOs to satisfy the requirements and policies in the best possible way, and will trigger the creation of a new CVO. The newly created CVO is registered in the CVO registry together with the situation parameters under which it was requested for future reference by the Request and Situation Matching entity. Finally, the doctor, or member of the medical staff, can use the

dynamically created CVO to monitor the medical status of the home inhabitants, as well as the environmental conditions in the smart home. Meanwhile, feedback regarding the operation of the CVO can be provided from the end-users, whereas this feedback will be encountered by Machine Learning mechanisms and will be exploited for future potential exploitation in the dynamic CVO composition and any other relevant operation of the platform.

2.2.3 Benefits from the architecture

Figure 4 presents an overview of the overall 'valuable' iCore architectural aspects that are exploited in the realization of the 'smart home living assistant' use case.

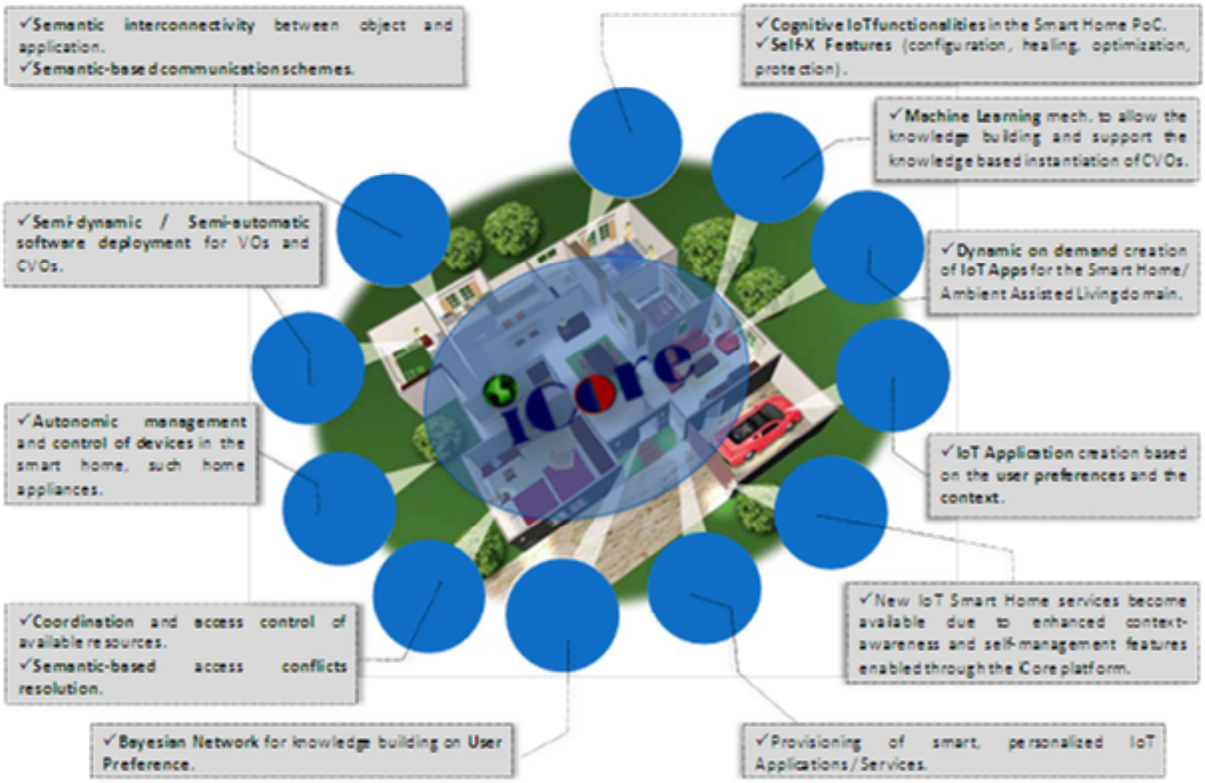

Figure 4: "valuable" iCore architectural aspects exploited in the 'Smart Home living assistant'

We can here answer the question "which are the key concepts of the iCore added value for Smart Home PoC?" as follow. The iCore framework allows / support in the Smart Home PoC: (a) the **dynamic provisioning of autonomic Smart Home services**, (b) the **dynamic deployment of Hardware Components** (RWOs), (c) the **semantic interoperability between RWOs**, (d) the **Semantic-based management and control of Smart Home RWOs**, (e) the **self-healing and self-configuration of applications** and (f) the **control of access and use of objects and services towards a secure and reliable Smart Home environment**.

From the business perspective, iCore provides a set of benefits and advantages by its introduction in this PoC. In particular, the benefits include: (a) the creation and provisioning of new added value services for disabled individuals through iCore platform, (b) the simplification of management of IoT infrastructure for smart home/assisted living reducing the need for human intervention (lower OPEX), (c) the enhancement of the interoperability, re-usability of objects that leads to the reduction of the CAPEX, whereas the decreased OPEX and CAPEX can drive down the cost for end users.

2.3 Smart Office: Easy meeting

2.3.1 Business context – requirements and services

The Smart Office environment aims to offer a seamless meeting experience to meeting participants and the meeting organizers with added capabilities of control. The iCore components highlighted in the following figure support a set of features of value during the whole meeting lifecycle, from its organization to its execution and aftermath.

These features have been described in detail in the iCore D6.3[7] deliverable and are, as such, mentioned very briefly here just to ease the understanding of the merit the implemented iCore components brings to the realization of these features.

- Smart Meeting Organization; inviting meeting participants and sending out QR codes to be used for Guidance
- Smart Meeting Guidance; guidance of participants through QR-code scanning at Smart Panels to the meeting venue
- Smart Meeting Break; identifying the need for a break based on monitoring of the attention span or at will as the meeting organizer sees fit
- Smart Meeting Recording; record the meeting using Smart devices the participants bring with them, removing the need of dedicated recording equipment
- Smart Meeting Wrap-up; gathering the recording to a designated web area for further processing and offline availability to participants

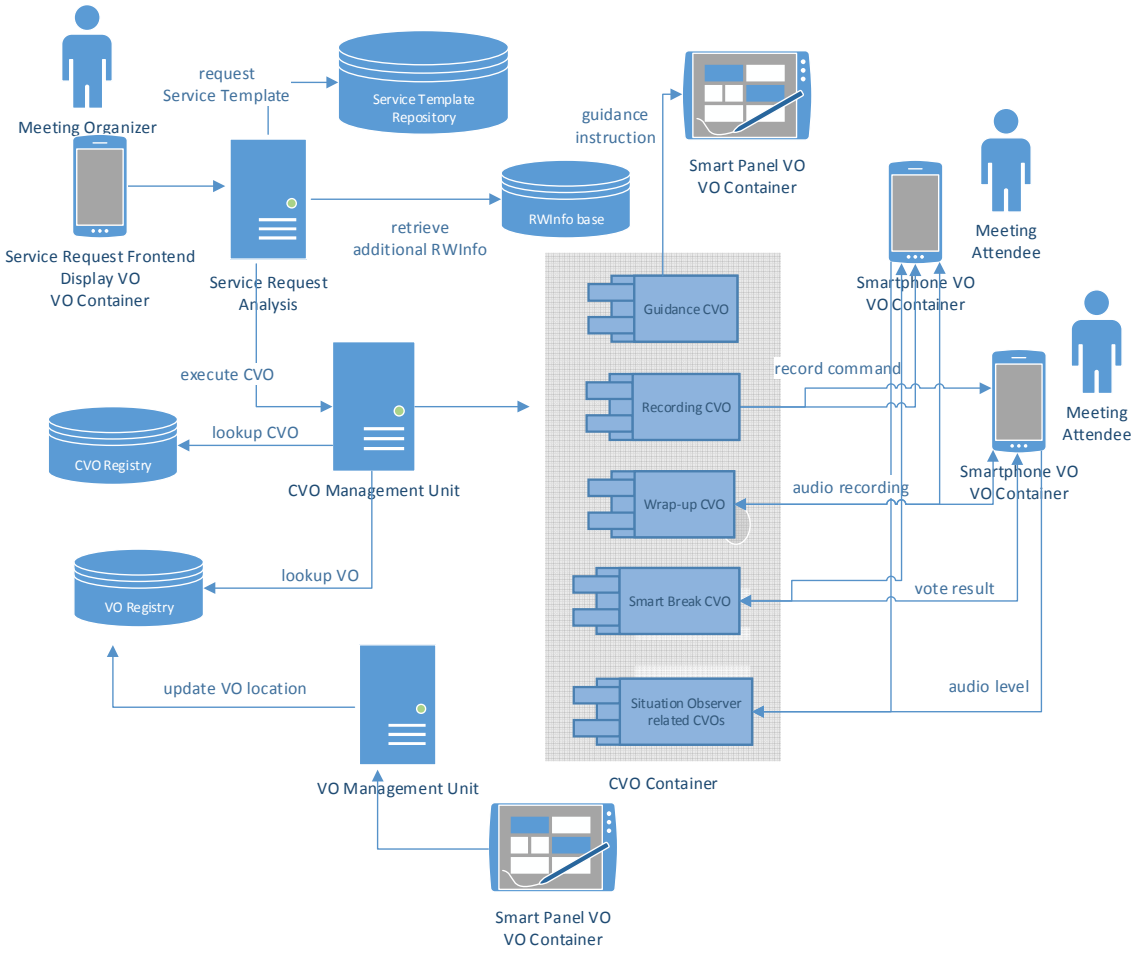

Figure 5 : iCore architecture instantiation for the Smart Office: Easy Meeting use case.

2.3.2 Contribution to architecture and major components used

In the rest of this section, we will briefly describe the involvement of the highlighted components and the value they bring in the Smart Office Realization:

- **Service Request Front-End** is used by the meeting organizer to issue the aforementioned service requests. While not offering some special intelligence or functionality it allows the service requests from the meeting organizer to be issued in a very user-friendly way.
- **Service Request Analysis**; The implementation and use of the SRA component allows the easy matching of a service request to a service template appropriate for fulfilling the needs of the service request. The use of this component allows the easy and automated look-up of the appropriate template based on semantic matching and hides the lower level details from the organizer. Furthermore, if a better way to fulfil a service request can be identified in the future, simply replacing the template at hand is enough for implementing this better way.
- **Service Template Repository**; The use of this component, which stores the service templates, allows easy fulfilment of the same service request in different meeting scenarios. Instead of "hardcoding" a service request to specific resources (software and hardware) it allows the soft binding of requests to their needs in terms of functionalities and features, allowing the lower layers to then perform the best match based on the situation at hand. In addition it allows for the easy "re-mapping" of a service request to needed features.
- **Real World Knowledge Model**; this model allows filling in missing or incomplete information in the service templates before the passing of the Service Execution Request from the SRA component to the CVO Management Unit. By capitalizing on pre-existing Real World Knowledge, service requests that could have not been fulfilled due to lack of information can instead be fulfilled by querying and retrieving information from the RWK database.
- **CVO Management Unit**; This component play a key role in the realization of the use case. It allows the mapping of desired/needed features to actually available resources to carry out a service request. This is done by consulting the CVO registry and VO registry identifying already up and running processing logic functionalities. By doing so the topological and other induced restrictions are resolved (i.e. for the Smart Recording service request, only devices of interest present in the meeting are selected and used)
- **CVO Registry**; This component allows retrieving and re-using CVOs and their processing logic, which are already installed in the iCore system;
- **CVO container**; the use of Java-based CVOs allows a great variety of devices to host the Smart Meeting required processing functionalities. Many of the CVOs have built-in decision making capabilities. E.g., for the Smart Break Recommender Situation Observer, several CVOs are deployed to the CVO container realising the three parts of Situation Awareness: situation detection, situation classification and situation projection. Those CVOs are decision making components that implement cognition functions. Same also holds for the Smart Meeting Recording CVO that intelligently selects the appropriate recording device each time according to resource optimization objectives.
- **VO registry**; The use of this component allows (when seen together with the VO Management functionality) the proper mapping of VO desired features to actually suitable VOs; suitable both in terms of capabilities but equally important, suitable in terms of location induced restrictions. From that point of view, the VO Registry can be seen as contributing some kind of System Knowledge to the CVO level when VOs are searched for being linked with specific CVO engines.
- **VO container**; Implementing a VO container for android devices allows interoperability with a vast number of devices (android phones, tablets, laptops) maximizing the chances of availability of devices (at the VO level) suitable for use in Smart Office scenarios.

- **VO Management functionality**; within VO container, this component also plays a key role in the realisation of the use case. It allows updating the VO registry so that it depicts at all times the correct properties of the VOs. In the Smart Meeting scenario where the service requests pose location restrictions and the involved VOs are largely mobile (i.e. smart phones that the meeting participants carry with them), maintaining the geo-location property of such VOs up-to-date is paramount for the meaningful and successful fulfilment of a service request.

2.3.3 Benefits from the architecture

From a business point of view the following key benefits are provided through an iCore compliant implementation of the use case, on top of the specific features mentioned above that by themselves bring value to a meeting environment.

- User-friendly way to issue meeting related service requests
- Re-use of heterogeneous resources to undertake a variety of tasks
- Resource optimization according to pre-defined objectives
- Decreased development time and easy introduction of new smart meeting features through the use of templates
- Improvement of service fulfilment rate and quality through knowledge exploitation
- Reduction of manual reconfigurations needed in order to "setup" similar services in different contexts.

2.4 Smart City: Transportation

2.4.1 Business context – requirements and services

The Smart City Transportation (see D6.4 [9]) storyline aims at demonstrating the exploitation of the iCore Architecture in the context of the automotive and Smart Mobility scenarios. The Smart City Transportation Proof-of-Concept implements services for the delivery of on-trip information to the car driver through the Car On-board Unit. The services are based on data coming from road infrastructure services (open data from city Parking Systems), on data coming from the Car itself (the GPS position) and based on Real World Knowledge such as User Behaviour Modelling in terms of recurrent User trips and destinations.
In particular, the Smart City Transportation PoC showcases the following scenarios:

- The driver is on her trip towards a usual destination, she uses the iCore "Parking Around Me" Service in order to receive information for the available parking sites around the position where the car is located at the moment, the Service sends the information to the Car On-Board Unit to be displayed to the Driver
- The driver is on her trip towards a usual destination, the iCore "Parking at Destination" service becomes available to the driver when the Situation Detection module (based on User Modeling/Real World Knowledge) discover the location of the recurrent destination based on the first position received from the On-board Unit. The Driver activates the Service "Parking at Destination" in order to receive information on available parking slots at destination.
- The driver is on her trip in another city, she uses the iCore "Parking Around Me" Service in order to receive information on available parking slots, the VO Registry is exploited in order to detect a different Parking System to be used for getting the information (Service Adaptation)

The following figure shows the prototype implementation for the Smart City Transportation Proof-of-Concept with a given car connected through IT backend and related parking applications through a 3G connection.

Figure 6: Smart City Transportation Prototype Implementation

2.4.2 Contribution to architecture and major components used

The Smart City Transportation PoC implements or integrates several iCore Architecture Components such as:

- **User Behavior Modeling** implemented by the User Recurrent Destination learning and prediction module provided by CRF.
- **CVO Execution/CVO Container**, implemented by the Workflow based composition engine provided by M3S. The CVO Container hosts the two CVOs which are available to be executed based on the prediction of recurrent destination by the User Behavior Modeling.
- **VO Container** – On-Board Unit, a VO Container hosting the Virtual Object providing the interface towards the Car, that is the GPS Sensor interface and the On-Board Device Display interface
- **VO Container** – Parking System, a VO Container hosting the Virtual Object providing interface towards the Parking System information platform.
- **VO Registry Integration**, which has been exploited in order to allow the Service Adaptation, selecting the most appropriate Virtual Object to be used by the CVO according to the Car position.

2.4.3 Benefits from the architecture

The Smart City Transportation iCore PoC showcases the following iCore Architecture benefits:

- **Object Virtualization**: the Car On-Board Unit and GPS device have been virtualized so that different devices and different hardware can seamlessly integrate into the PoC without any change to the Service. At the same time, a "Parking System VO Template" has been defined, so to allow the easy integration of other Parking System platforms in different domains/cities.

- **Service Adaptation to Context**: the Smart City Transportation CVOs are not bound to use a specific Parking System Virtual Object but, thanks to the VO Registry integration and to VO Templates, a Service can be deployed without knowing in advance the details of the VO that will be used. The effective VO will be selected at runtime querying the VO Registry for the most appropriate VO
- **Service Discovery based on User Modeling**: thanks to the Recurrent Destination and Trip Learning and Detection, provided by CRF, as Real World Knowledge/User Behavior Modeling component, the available services can be discovered and selected based on the available knowledge on recurrent destination.
- **CVO and VO Composition**: the CVO Execution iCore Architecture component is implemented by the M3S workflow-based composition engine, which provides a high scalable event-based workflow engine suitable for processing high volumes of data.

Growing and use of Real World Knowledge: the Real World Knowledge is generated and used by the Smart City Transportation PoC and used. The generated Real world Knowledge include the User behaviour Modelling, Recurrent Trip and Destination Learning and Prediction, which is used both to empower the iCore Services and to support offline decision making procedures and planning by the City Managers.

2.5 Smart Urban Security

2.5.1 Business context – requirements and services

The demonstration scenario of the smart city – urban security PoC (see D6.4 [9]) is about VIP visiting a huge exhibition site and its protection and evacuation based on a deployed surveillance system supporting accurate situation awareness. So we have on the field the VIP with his close protection team, several undercover policemen from local security forces deployed near the VIP visit and a remote mobile Control and Command (C2) within a truck outside the exhibition site; this C2 is operated by local police. The VIP team use several wearable technologies such as headsets and GPS, while the undercover policemen have small chemical sensors and video cameras. All are connected wirelessly with an ad-hoc network to coordinate the protection and potential evacuation but also because all sensors information contributes to a shared COP (Common Operational Picture). The C2 is also controlling a subset of the already deployed exhibition site CCTV system through a dedicated connection and the wireless network.
The VIP visit is predefined and formalized according to operational plan and procedures by the local security forces and the VIP team. The operational plan here takes into account the potential threats and manageable exits from the exhibition site. In case of severe unfriendly event, the VIP visit is interrupted immediately and safe evacuation is organized. The evacuation is fully managed by the remote C2 that delivers instructions about the exit to take to the VIP team. In our scenario, a powerful “dirty” IED (Improvised Explosive Device) explodes prematurely close to the entrance of a building a few minutes before the VIP comes to the building; in addition to the strong blast that generates wave of panic, deadly neurotoxic gas dispersal begins killing people around.
Because all critical situations and events can’t be anticipated, security missions require a strong **adaptability** and **survivability** during execution. Here three pillars come into play: first is preparation and training before missions, second is adaptability during the missions to unplanned situation last is debriefing and update of operational procedures and security systems to take into account past experiences and leanings.

What we demonstrate with the smart security proof of Concept illustrates somehow the first and the second pillars but not only.

Indeed, the PoC we provide integrates built-in capability supporting validation of the main surveillance system components behaviour that is the *wireless network* and the iCore platform implementing iCore architecture. A validation of the integrated system is achieved by connecting these two components with the Thales SE-Star application, simulating full demonstration scenario introduced above. The simulation runs VIP team, policemen, crowd, IED explosion and all the sensors deployed providing raw situation awareness with video streaming, chemical detections, VIP location that are sent to the C2 through the wireless network and iCore platform. This validation is made clearly during system development with a focus on interface testing, checking correct behaviour of the system components and so on.

Regarding *mission preparation*, end-users can reuse the complete system in a similar way to validate many operational plans and procedures. In our PoC, VIP evacuation in case of unfriendly event is the main operational procedure we demonstrate; this evacuation is managed with the support of the iCore platform that provides some *real-time* suggestions about the best path for the VIP to follow to escape quickly and safely the exhibition site.

Finally, regarding *adaptability during mission execution*, iCore platform achieve also important enhancement that really matter to the C2 operators. iCore platform performs here a *real-time* selection of a minimal set of video cameras streaming that ensure the most relevant situation awareness according to VIP location, tracking and *predictions* about Improvised Explosive Device (IED) generating toxic gas dispersal and moving crowds; Above this, reduction of cognitive load of the operators is a top benefit. The selection of video streaming takes into account several kind of information from the operators themselves; for instance through Service Requests priority the operators are able to provide relative importance of tracked objects or people then to video streaming; Also this priority is useful as one ingredient for an efficient management of the network bandwidth dropping less important data flows if the network is overloaded.

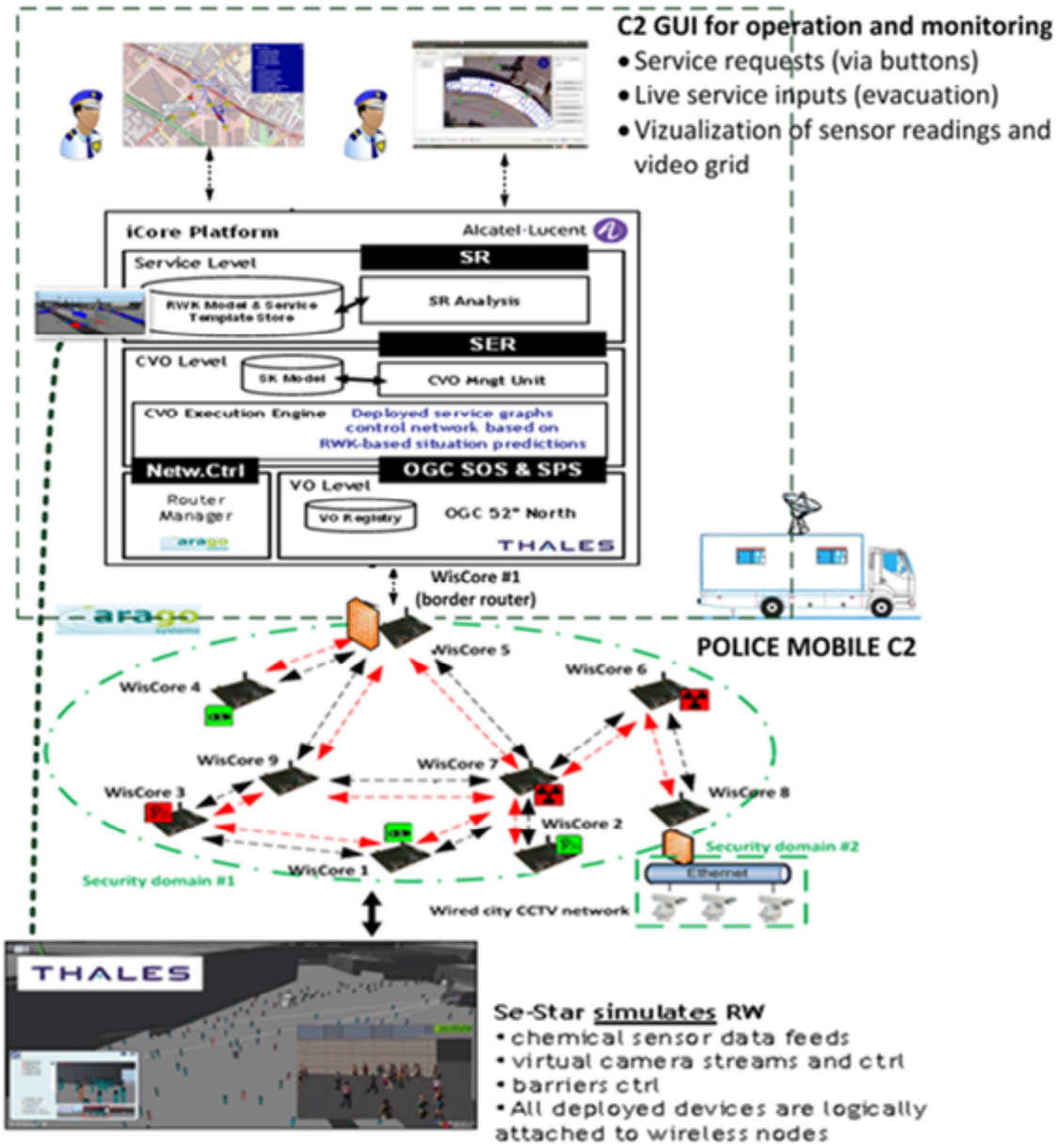

Figure 7: iCore architecture implementation overview within Smart Urban Security setup

2.5.2 Contribution to architecture and major components used

As illustrated in Figure 7, the Urban Security case offers a realistic context for validating the iCore Service, CVO and VO Level concepts. The case provided even an opportunity to mature many of the iCore architectural concepts, such as (going from Service Level down to VO Level) for this use case:

- the **RWK Model**, for the domain as inspired on surveillance operational knowledge and Se-Star city behavioral models,
- a realistic and useful set of **Service Request expressions,** and (in this case, one-to-one) corresponding Service Templates,
- **real infrastructure SK**, reflecting properties and constraints of the real WSN network infrastructure used to transport the streams for the services,
- matured insights on the **architectural SER boundary (CVO Level API)**, before which RWK-based constraints/behavior is added to the Service Goal expressed in the SR, and after which SK-based system concerns are added, in particular including in SERs: **Situation Observers**

(and RWK-based predictors) for the tracked phenomena (VIP, crowds, chemical cloud) and connection of those to service-specific relevance selection as well as Flow priority reservation control towards the network,

- a number of **CVO Level infrastructure functions** as well as generic **CVO Types** parameterized for the service instances at hand, with reuse of joint sub-graphs across service instances (Situation Observers) and aggregating or arbitrating joint CVO instances e.g. determining overall stream priority and needed quality prediction, and
- VO registration, querying and data stream control is abstracted behind Open Geospatial Consortium **OGC (SOS and VOD SOS) standard interfaces**. Pan-Tilt-Zoom (PTZ) control of cameras and other forms of **actuation** are done through **OGC SPS**.

2.5.3 Benefits from the architecture

As such, two key benefits are obtained through iCore:

1) ***Cognitive overload for the service user is reduced*** to the most relevant, over-seeable information, as an optimized decision support, and
2) ***Network resources are devoted to the highest priority, high-quality-requiring streams***, with best effort support for additional streams (as far as battery autonomy requirements allow).

2.6 Smart Business: Supply Chain Management and Logistics

2.6.1 Business context – requirements and services

In the context of smart business PoC (see D6.5[10]) goods are transported from suppliers to retailer via a "mesh" of warehouses with road/air/sea transport in between. The real end-user issue is the lack of insight in the storage and transport conditions of goods between suppliers and consumer. To accomplish this, a fine-grained ICT monitoring system (e.g. a wireless sensor network) is applied. For example, a retailer wants to know if he can accept a shipment of temperature sensitive medicines, a transport operator wants to know if it can avert a claim of spoiled goods, since it thinks it kept the goods within the specified temperature tolerances and suppliers/retailers wants to know when to expect a delivery of goods. In case of violation of storage and transportation conditions, parties responsible for the goods want to be able to act as soon as possible to reduce product spoilage (and associated claims). All stakeholders require access control to the data collected by the monitoring system.

The Smart Business use case demonstrates the following:

- Acceptance reports: the system provides the end-user of perishable goods (e.g. a retailer or pharmacist) a report at delivery of the products showing its conditions during transport and storage. Based on these reports, the end-user is able to decide whether to accept the shipment or not; the report can only be accessed after the shipment is delivered.
- Real-time alerts/early warnings: the system provides (early) warnings to transporters when the products are outside their optimum storage conditions. The system provides these warning to guarantee the quality of products throughout the supply chain and enables transporters to act before damage occurs; Access to the monitoring system at the warehouses is only allowed to the device owners and to the logistic companies responsible for the shipments being monitored.
- Automatic control of storage and transport conditions: Since the iCore system has knowledge of what conditions are required for products within a warehouse/truck. Therefore, it can provide functionality to automatically control environmental conditions based on the requirements of products stored in (compartments of) warehouses or trucks by directly controlling coolers/HVAC systems.

Their positioning is shown in next figure.

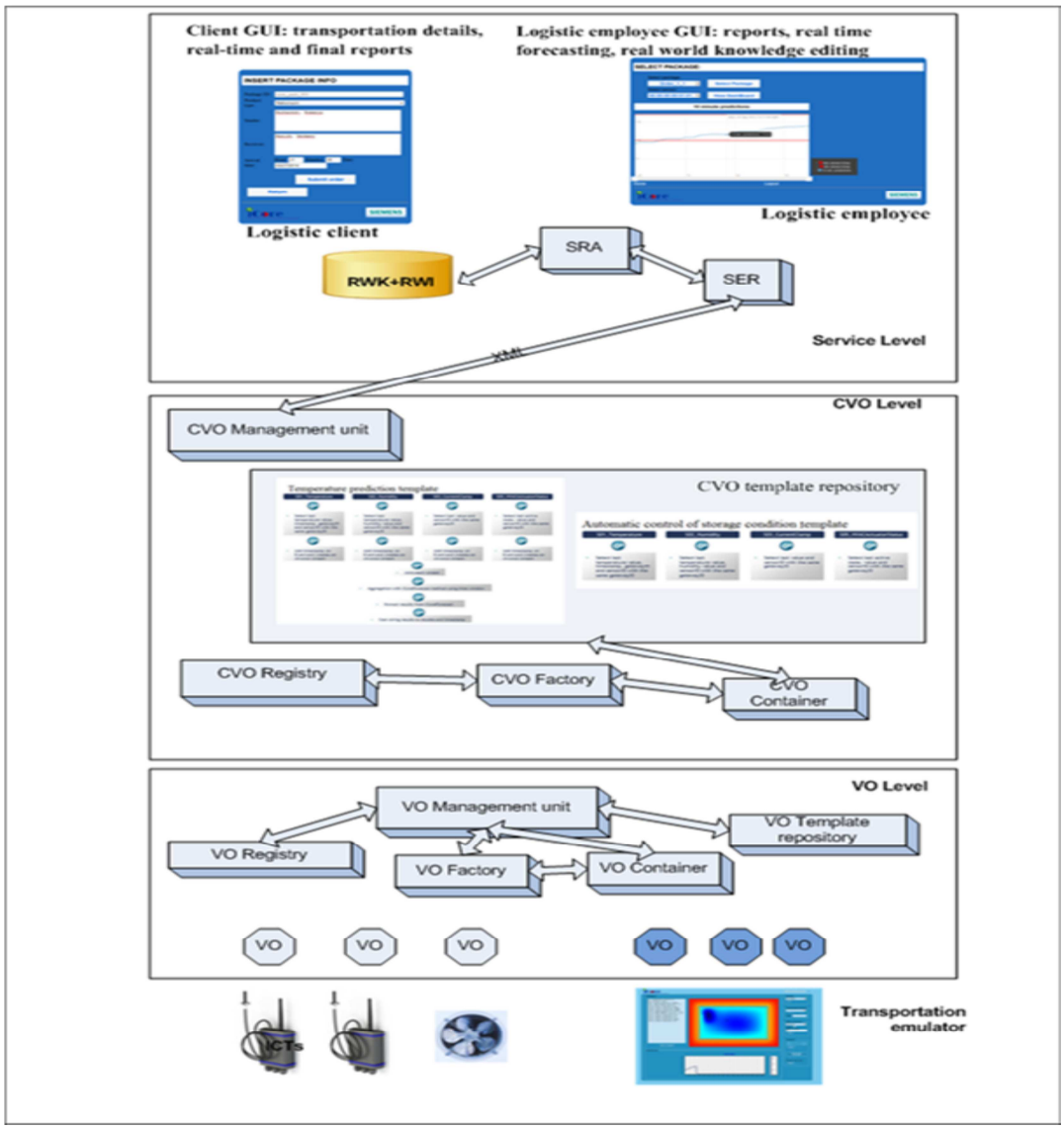

Figure 8: iCore architecture implementation overview within Smart Business setup

2.6.2 Contribution to architecture and major components used

All three levels of iCore architecture are involved in accomplishment of the functionalities introduced above; typically:

- Service level: real world information, service request analyser, situation modelling;
- CVO level: CVO management unit, CVO container, CVO registry, CVO factory, CVO templates and template repository;
- VO level: VO registry, VO Factory, VO Template Repository, VO Management functionality, VO Container, VO Back-end, VO Front-end.

For example, the sensor data (the so-called real world information, RWI) communicated through VO level is further used to derive a predictive model exploited in the early warnings scenario. The predictive model is trained and maintained (i.e. retrained) at the Service level, and deployed and used by the CVO level, embedded in a CEP engine as a user defined function.

The enforcement of security policy rules is done using the Model-based Security Toolkit (SecKit) components, which is applied at the VO level to prevent the monitored information from being accessed by unauthorized entities. The SecKit intercepts transparently the messages exchanged with VOs at technology specific implementation layer (MQTT) and enforce the specified security policies by the iCore system operator**.** Details about the SecKit support for specification and enforcement of security policies are described in Deliverable D2.4 [11] and D3.4 [2].

Besides the component implementation and deployment, different user roles are supported: the domain expert specifies the acceptance criteria for the products (e.g. min/max temperature and humidity), stored by real word knowledge (RWK).

The same RWK serializes time-series forecasting models for multivariate data provided by RWI. The appropriate predictive model and corresponding acceptance criteria are selected by the Service Request Analyser and passed to the CVO level. The CEP engine aggregates and aligns data from various VOs (which expose internal and external temperature, humidity, current clamp, and parcel temperature sensors) and provides data appropriate for the training stage. Later on, the predictive model is fed with real-time data by VO artefacts, also pre-processed through CEP statements. Besides feeding the predictive model with appropriately-formatted data, customized CVO templates are used for real-time checking of whether temperature fulfils acceptance criteria or GPS sensor outage.

2.6.3 Benefits from the architecture

As sketched above, the general iCore architecture provides enough support for hosting various mechanisms; the prototype benefits from the architecture mainly due to the enforced separation of concerns and a clear workflow. More specifically, the use case benefits from the iCore architecture are due to the following:

- Ability to integrate ICTs or virtualized forms of ICTs through appropriate VOs;
- Support for integration of security policies for VOs and VO data;
- Integrating data from multiple VOs and correlating sensors and actuators activities;
- Deploying and using a knowledge model fed with real-time data;
- Support for domain expert and machine learning specialist;
- Detailed reports for logistic client based on real time ICT-provided data.
- Enforcement of security policies at the VO level.

2.7 Smart tour in the city

2.7.1 Business context – requirements and services

The ‘Smart tour in the city’ trial aims at the validation of the worthiness of the iCore platform, both in terms of technical and business perspective. In particular, the trial comprises three different (business) domains as shown in next figure; (a) IoT application developers, which exploit the functionalities and benefits of this platform and develop IoT applications, (b) Travel agencies, which are the clients of the IoT application developers and buy these applications in order to offer them to their customers, and (c) Tourists, which are the end-users of the developed applications. Device manufacturers/vendors are also considered in this trial as they are involved through the real world objects that are used.

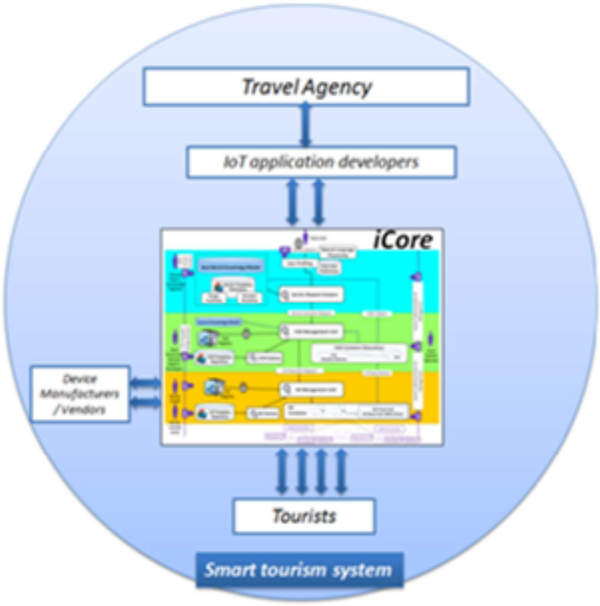

Figure 9: iCore platform positioning in a smart tourism system

2.7.2 Contribution to architecture and major components used

In this trial, a travel agency (namely CLIPPER travel) requests an IoT application (e.g. "I want a Smart Tourism application") from WINGS ICT Solutions (IoT application developers). WINGS ICT Solutions develops the application (hereafter called "Smart Tourism CVO") using the iCore platform and the defined Service Templates that are applicable to the request. For this case, a set of indicative functions that the CVO should offer include: "Fastest route to destination", "Scenic route to destination" and "Avoid traffic". The application is sold to the CLIPPER travel, which further distributes it to its customers (tourists).

Let's assume that a tourist arrives at Athens (Greece) or Santander (Spain) and starts using the application to help him navigate/explore the city. Firstly, he is required to complete a simple registration in order to start using this application (an e-mail will be sent to confirm the registration). Upon first use of the application his credentials will be requested (from the registration process mentioned above). Thereafter, he will be allowed to request the service he needs from a predefined set or insert his own request in natural language, as well as a destination point to be guided to. For this purpose, the destination address is entered to the Smart Tourism CVO, which calculates all possible routes from the current location to the destination.

According to the requested service(s) the next steps of the process may vary. If the user specified that he wants to avoid traffic, the Smart Tourism CVO will look up the traffic conditions for every street in the routes that have been calculated. In particular, the Traffic Analyser CVO is responsible for the selection of the routes, based on collected traffic statistics. This CVO receives a set of routes and the country code and performs the required operations to return the route with the least traffic. The country codes supported for the trial are "GR" and "ES" (Greece and Spain respectively). Depending on the country code received by the device, the CVO will perform a different operation (as the traffic information originates from different sources). For example, in the case of Spain the Smart Santander CVO is used to retrieve the traffic information from the Smart Santander facilities. Additionally, historical data are exploited using the *Hadoop* platform in order to provide a more accurate prediction of the traffic conditions of each route. For both cases (Greece and Spain), a score needs to be calculated for each route (the lower the traffic the lower the score).

Afterwards, the route with the lowest value assigned and therefore the smallest amount of traffic is presented to the tourist (or tourist group). If the user specifies that the "fastest route to destination" is needed then, as described above, the routes/traffic will be calculated. In this case however, the Smart Tourism CVO will take an additional factor under consideration; the route distance. It will find the best combination of low traffic and short distance and present it to the tourist. In case a "Scenic route to destination" is requested, the Smart Tourism CVO will load the sights available in the circular area covering the tourist's location and the destination that has been set. It will then draft a route of "sensible" distance that covers as many of the sights found as possible.

Furthermore, in all the cases, weather information as well as "points of interest" (i.e. shops, gas stations, restaurants, etc.) will be displayed to the tourist according to his location and profile preferences. The Smart Tourism CVO also receives traffic jam, accident information in real time (from other users, authorities or web sites) and recalculates the selected route accordingly.

As long as the Smart Tourism CVO is operating, it sends data (indicatively: requested services, often visited points of interest, preferred means of transport) to the User Preferences and Profiling mechanism in order to be able to further parameterize the service offered to the user. Finally, the users are asked to provide feedback (rating) concerning their experience with the application upon closing it.

Mechanisms in all layers of the iCore architecture are involved in the fulfilment of the aforementioned functionalities:

- Service level: Natural Language Processing, Service Templates Repository, User Preferences, Service Request Analysis
- CVO level: CVOs, CVO Templates Repository, CVO Registry, Approximation & Reuse Opportunity Detection, Optimization-based Decision Making, Dynamic CVO logic/workflow composition
- VO level: VOs, VO Template Repository, VO Registry

2.7.3 Benefits from the architecture

A number of features can be identified in the above brief storyline of this trial: i) Service request in natural language, ii) Identification of user preferences, iii) Dynamic creation of personalized services, iv) Knowledge generation & situation prediction, v) Traffic diagnosis & prediction, vi) Abstraction of heterogeneity and complexity, vii) Interoperability of objects, viii) Knowledge-based instantiation of services, and ix) Exploitation of user feedback.

These functionalities are realized and facilitated from a set of architectural aspects that are exploited in this trial:

- Dynamic (on demand) creation of personalized services.
- Knowledge-based instantiation of services.
- Self-x capabilities (configuration, optimization, healing).
- Knowledge generation & situation prediction.
- System performance & SLA management.
- Autonomous discovery, control and interoperability of virtual objects corresponding to heterogeneous sources of information.

2.8 Medical Asset Management

2.8.1 Business context – requirements and services

The trial on "Medical Asset Management" has been executed at the Hospital of Trento "Santa Chiara" within the Neonatology unit part of the Paediatric Department. The basic need of this particular stakeholder is to keep track of location and usage of objects, more specifically medical devices, to minimise overheads in finding objects, maintaining objects, managing their lifecycle and supporting with evidence any planning related to purchasing, using, dismissing various items.

To enable doctors and nurses concentrate mostly on their core duties, iCore has been used to implement a solution where objects become more active and are enabled to generate data and notifications, which are then automatically collected to support the development of various applications to suit several needs.

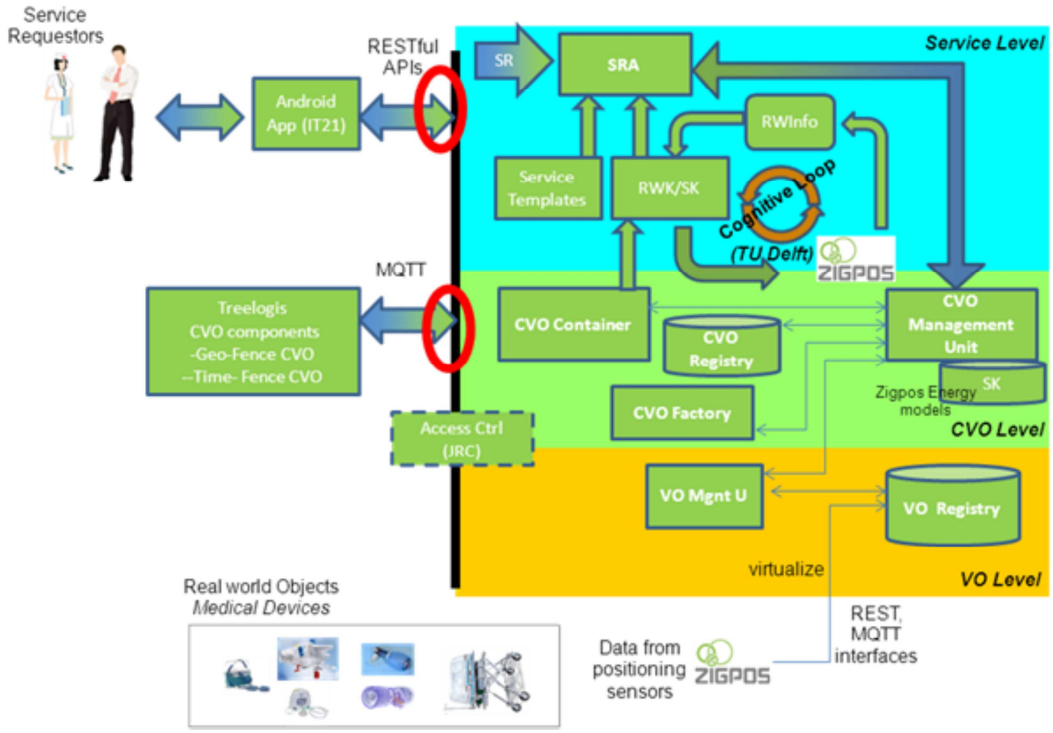

Figure 10: iCore platform within the Smart Medical Asset Management trial

2.8.2 Contribution to architecture and major components used

To achieve this purpose, a number of iCore architectural components have been used as illustrated in figure above. The need to create a virtual counterpart of a real object capable of collecting relevant data for the envisaged purposes was obtained here simply through tagging medical objects with positioning tags. As opposed to bespoke solutions tailored around the specifics of the tagging technology used this feature clearly enables re-use of a large part of the software developed to extract knowledge from location of objects, as this can work with any positioning technology which might be part of legacy or IT infrastructure procurement choices in different hospitals for example.

This created distinct Virtual Objects that would be logged in a Virtual Object Registry as soon as these are connected to the positioning system. Moreover some extra data has been associated with each virtual object: this would be the coordinates of a "geo-fence" associated with the real-world object

(to enable notifications of “out-of-fence” besides change of location), as well as “time-fence” for all purposes related to managing maintenance, objects out of place for too long etc.

As far as fostering interoperability is concerned, the communication layer used by the Virtual Objects has been implemented using a publish/subscribe bus (the MQTT broker). This is used for all notifications such as change location and out-of-fence (worth noticing here that this is all agnostic of the positioning technology used and hence largely reusable). Upper layer queries are on the other hand answered using a REST-based client/server communication with the VO Registry. In the specifics of this trial, the flexibility given by the VO Registry consisted in accommodating a data model where each Virtual Object has a location, as well as a fencing perimeter and a time-fence for notification of various messages related to object (medical device) status.

With regards to the use of cognitive technologies and the upper layers of the iCore architecture, the trial had main targets: one to support the end users with knowledge extracted out of behaviour over time of tracked objects (Real World Knowledge (RWK) modelling), the other to support system operators with knowledge that helps them reducing energy consumption (System Knowledge (SK) modelling). These two targets have been achieved through the analysis of the recorded locations, which enabled the creation of predictive models for these two purposes. In the RWK Modelling case, the location and usage of objects has been used to sort out objects that are hardly ever used, objects that go out of fence often, predict trends of usage and location; in the SK Modelling case, predictions about clustering of objects had been used to reduce its overall energy consumption (the logged data about object locations is used to assess where objects cluster more likely and achieve energy reduction by reducing the number of anchors needed to keep track of objects).

Besides the Service Level and VO Level functionality, the trial also implemented part of the CVO level functionality, namely the selection of the appropriate CVO after a Service Request Analysis to locate an object or to predict the location of an object based on a trained model. Here the CVO Management Unit looks up the registries and the CVO factory. The CVOs are then deployed in the CVO container. The container is a workflow based execution engine, which also has complex event processing features. More details about how each of the iCore architectural components was used in this trial can be found in the later dedicated sections.

2.8.3 Benefits from the architecture

The value of the iCore architecture, especially in the RWK case, can be summarised in the ability to separate concerns between a “mainstream” software development activity meant to create various services based on the assessment of the location of objects and the machine learning part, which is meant to gather data and as this data keeps coming in, build and train models that can be used to predict various situations, enhancing therefore the “mainstream” service proposition.

3. iCore Service Level

As explored and discussed in the WP5 deliverable documents D5.2 [4] and D5.3 [5], and with WP5 PoCs discussed more extensively in D5.4 [6]. as positioned and leveraged in the WP6/9 cases, the main cognitive innovation targeted by the iCore Service Level relates to the leveraging of so-called **Real-World Knowledge (RWK)**, and the growing of that as a platform asset, for improving the execution efficiency, the effectiveness and situation awareness of services. The implied Service Level architecture elements include an RWK Model Store, also related to the **Service Template (ST) Repository**, and a **Service Request Analysis (SRA)** function that matches incoming **Service Requests (SR)** to the corresponding Service Templates, weaving in behavioural constraints as expressed by the relevant RWK, eventually handing off the resulting service specification as a **Service Execution Request (SER)** to the CVO Level via the CVO Level API. In this respect, the SERs contain all service specification data that can be determined a priori to the execution of the service, i.e. all that does not require processing of observed data, such data processing execution being performed at the CVO Level upon the reception of SERs. For readers' convenience, we remind the positioning of these main Service Level functions and interfaces in figure below.

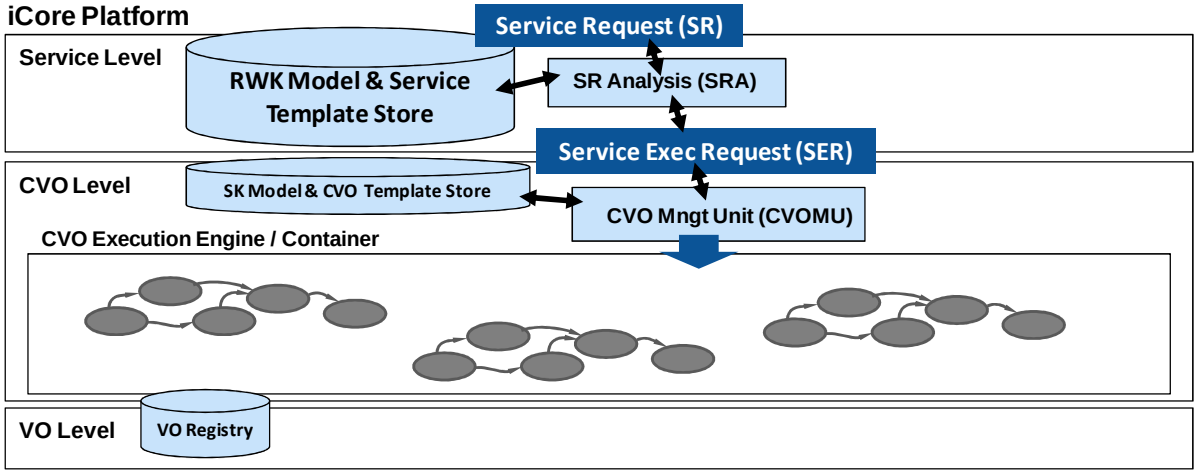

Figure 11: Overview of main Service Level functions and iCore architecture entities they interface with

We distinguish the modelling of the *'Real World'* from the modelling of the iCore *System*. While the 'System' concerns the *organisational units that use computational and communication resources at its disposal for realising a particular impact outside itself*, the 'Real World' is the *collection of anything outside the System on which the System aims to have impact*. In practice, a System is limited in scope as to the subset of the otherwise infinite 'Real World' it is aimed to be capable of having impact on. In the iCore architecture we foresee **modelling of relevant *universal, constant behavioural properties* of such scoped subset of the Real World on a system-wide, cross service basis**, under the term Real World Knowledge. Because of its universal, constant nature to the scope an iCore platform takes, RWK can reduce a drastic part of the complexity of service specification and execution *before* services are actually executed. We therefore scope the Service Level to analyse Service Requests taking into account Service Templates and RWK, to determine what is actually requested to be executed (as a SER). [Note that, before going to the actual execution phase, the CVO Level can also take into account system behavioural properties (System Knowledge, SK) to determine what is the best way to execute, given the collection of SERs.]

As a consequence of RWK definition and design implications, it is of crucial importance to the effective working of an iCore platform to consider an *RWK world scope definition as needed by the use case domain*, including the definition of all needed types of observable RWInfo 'variables' that are processed as live sensor data by the CVO Level. [Note that the RW meaning of data in the iCore

system is typically derived from the semantic annotation assigned to VO types and VO instances, but can also be a derivative of that according to the RWK/Service Template declared RW semantics.]

In the next sections, we discuss the main architectural aspects of the iCore Service Level:

- ***The way RWK is leveraged in the platform*** (section 3.1),
- ***The way service instances are issued on the platform and how the services are modeled*** (section 3.2), and
- ***How a service description for execution by the CVO Level is derived for each service instance*** (section 3.3).

For each of these aspects, we highlight relevant mechanisms and components elaborated in iCore platform implementations as leveraged in particular WP6/9 use cases, thereby illustrating how the iCore platform is beneficial from the perspective of this aspect.

One important feature of the iCore platform that can be conducted based on the Service Level modeling, and fully exploits the potential benefits of the iCore Service Level, the so-called **RWK Cognitive Loop**, was studied theoretically as the main topic of deliverable D5.3 [5], but has no full implementations in any of the use cases yet. Partial implementations where done, indicated in the sub-sections.

We wrap up the chapter by discussing the achievements, alternative options and open issues requiring further research.

3.1 Leveraging Real World Knowledge

As introduced, the iCore platform provides the possibility to model ***universal, constant behavioural properties*** of (a relevantly scoped subset of) the Real World **on a system-wide, cross service basis** as **Real World Knowledge** (RWK), thereby allowing

1. To reduce complexity of service specification at the SR stage, while
2. Making services Situation Aware and exploiting RW stream redundancies at the SER-stage service descriptions, i.e. all before services are actually executed at the CVO Level.

Exploited RWK typically will relate to the behavior or behavioral constraints of physical entities (objects) or phenomena (directly or indirectly) observable the real world, such as the geographical trajectory of people, crowds or vehicles, the physics laws dictating the evolution of properties like temperature, e.g. of goods, people or buildings, or still more abstract phenomena such as the weather or city traffic patterns.
Service instance declarations via SRs do not need to include all related RWK that is related to the goal expressed by the SR, hence holding the potential of being expressed more purely as a service goal.
In contrast, live observation of RW objects or phenomena as declared by the RWK identifies *Situation Observers* (that will run as connected CVOs) including a situation prediction capability based on the RWK model of the concerned object/phenomenon behavior. Due to this translation to Situation Observers, the RWK can also be leveraged to do *dynamic stream selection*, i.e. section VOs and connected processing.

3.1.1 Examples from use cases and trials

In the remainder of this section we illustrate **Real World Knowledge** with a range of examples for the iCore use cases.

The real world knowledge model used in the **Smart Home: Living Assistant** PoC, is shown in figure below. Specifically, this includes Sensors and Actuators used in the context of the Smart Home,

Persons (e.g. Sarah, the elderly inhabitant of the house), rooms and outdoor places inside the area of the house as well as the services offered (see the figure for more details).

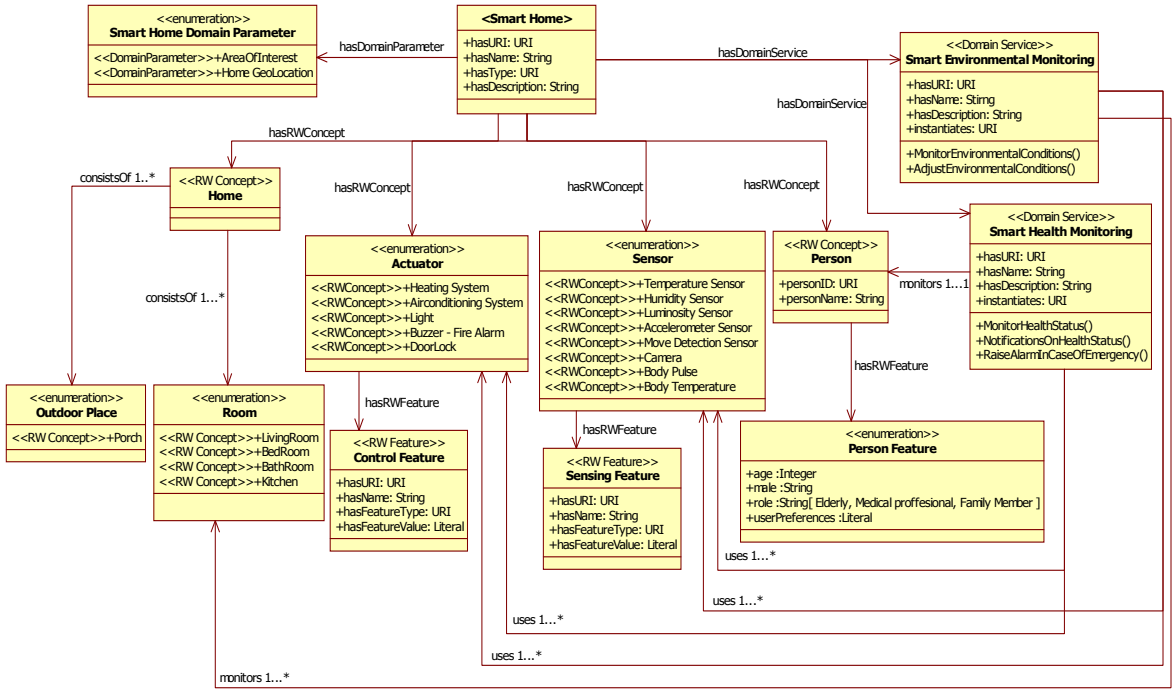

Figure 12: Smart Home RWK model

In the **Smart Office** use case the RWK concept 'Meeting' is in the focus of attention, together with adjacent properties, that are essential for organizing, managing and executing meetings in an iCore-compliant manner. In particular following RWI can be captured during the use case:

- **Where** and **when** is the meeting going to take place? For this, the meeting location as well as the date and time are specified as semantic by linking to reusable RDF resources.
- **Who** is the meeting organizer, and who is invited to the meeting? The people involved in the meeting organization are represented by semantic information accessible through the RWK knowledge base.
- **Who** has accepted or declined the meeting invitation and is therefore expected or not expected to attend the meeting?
- **Where** is the current position of the meeting attendee when entered the building in which the meeting takes place? During Smart Meeting Guidance, this information is used to guide the meeting attendee through the building from the entrance to the meeting room.
- **When** is the best time for having a break in an ongoing meeting? Based on RWK and the noise level over a period of time in the meeting room the Situation Observer recommends a time when to make a break.
- **What devices** are brought to the meeting by the attendees that can be used as audio-capture VOs for the Smart Meeting Recording feature?

The RWK model specified for this use case is used to allow a smooth integration of different features and ensure service request fulfillment through exploitation of real world knowledge and information. Information gathered through the Meeting Organization service is stored and later used by the other services.

As an example, for the Smart Meeting Recording service, suitable audio-capturing devices need to be discovered that are currently located in the meeting location. During the Service Request Analysis (SRA) for a Smart Recording service request, the Service Template (ST) for this service demands the

meeting organizer to be given as parameter through the CVO level API to trigger the 'START recording'-command. Instead of the meeting organizer, the meeting identifier was given in the service request though. Although the SRA component cannot find the 'meetingID'-parameter in the ST, it is still able to ask for more information about the meeting of the particular identifier in the RWInfo base. The domain-specific knowledge that meetings have an organizer is modeled in the RWK knowledge base accessible by the SRA component.
In a similar way, geo-location is required to be passed through the CVO level API; during the requests only the meeting location as a room/building number is given. By querying though the RWK base, the specific geo-location of that room can be retrieved and be passed to the CVO level.

In the **Smart City Transportation** case, Real World Knowledge is involved in the context of several aspects of the implementation:

- the knowledge of the User Behavior, in terms of the identification and prediction of Driver's recurrent trips based on the data received from the Car GPS Virtual Object, is involved in the discovery of the most suitable CVO to be executed: whenever a recurrent trip and destination are identified, the CVO Management Unit can suggest the execution of the "Parking at Destination" Composite Virtual Object. The identified destination is as well passed to the CVO as execution parameter, together with the current position, so that the CVO will be able to exploit this information to i) search for parking suggestions near the destination coordinates and ii) detect alternatives based on the knowledge of current position AND destination (so for example to avoid traffic congestions)
- the execution of the CVO helps to grow the Real World Knowledge, thanks to the collection of many events (Real World Information) during the execution: the location of the Driver at the time when the Parking suggestion has been sent and the location of the Driver when he/she reached the Parking place (and the notion whether the parking place has been reached or not) are collected by the CVO, stored and aggregated so that this information can be used as input to (for example) traffic modeling and prediction systems.

In the **Urban Security** case, a specific city region, a particular person, crowds and chemical clouds are observed. (In fact, in this case this is at the same time the goal of potentially issued service instances, see next section.) This implies that RWK can be leveraged concerning:

- **What are important areas of the city region** (*areas of interest*, as geo-coordinate-based polygons) according to the experts of the application domain (in this case, police public safety experts familiar with the city region), allowing to **distinguish** these areas of interest as distinct observable subjects (each for which the observation means, in this case camera views, can be **selected** at runtime),
- **Movement properties of a person** (in the example a VIP making a city visit), who is e.g. observed via GPS localization, allowing to *predict the trajectory* of that person (the live location data stream allows to extrapolate with a simple linear model as RWK), complemented with RWK on the **known course waypoints** this person is planned to follow (a VIP is planned to visit particular venues in a city in a particular order), which allows to extend the trajectory model with a probabilistic weighting with the planned waypoints, allowing to **predict the trajectory** in a less trivial way (not taken into account in the actual iCore platform implementation, but a further RWK example that can enhance such prediction as available from the Se-Star RW simulator used in the demonstration, is the knowledge on what are pedestrian areas in the city, and the fact that people can only enter/cross building through entries in walls),
- **Emergence and movement of crowds**, for which also a simple cluster speed extrapolation model can be specified as RWK (again, next to potentially many other crowd city movement

properties, not exploited in the actual demonstrator), as well as an heuristic as RWK for crowd detection via people density estimates in a video image, allowing to **predict in which city areas crowds are likely to occur next**, based on a current status, and

- **The dispersion of a toxic cloud** (after a bomb explosion in or at a building), for which a simple (circular) cloud dispersion model is expressed as RWK (again here, more complex models could be considered, e.g. taking into account observable wind speeds and building walls), allowing the **prediction of where toxic danger will occur**.

The Urban Security demonstrator assumes RWK models for each of these RW behaviors; it needs to be noted that although strongly simplified models were used in the implementation, the **effective benefits of RWK in the platform are apparent** in the demonstration (as elaborated in WP1/6 documents, a reduced cognitive load of observing humans and a dynamic network resource prioritization to the most critical decision support at all times is achieved).
The iCore architecture's possibility of using an **RWK Cognitive Loop** to train RWK Models in-situ of service execution, and extend models by means of Hypothesis Tests, could also be applied in practice for the Urban Security domain, adding systematically new parameter tunings and extended models concerning e.g. crowd behavior or toxic cloud dispersion into the system for ever smarter mission support behavior. Because of the availability of a city simulator, this model training and extension does not have to happen on the actual real world (as real terrorist attacks, crowd panic and human casualties obviously cannot be experimented with) but can be an advanced form of *transfer learning* [14] replacing more expensive individual person/particle models used in the full-fledged city simulator by assumed/validated equally effective online prediction models in the service execution context.

For the **Smart business** use case, the domain expert includes in the RWK the acceptance criteria for different types of products, which can be formalized as follows:

- The recommended value of a parameter (e.g. minimum and maximum temperature, humidity, etc.)
- The peak value of a parameter: the netromycin can be stored in the temperature interval 15-18ºC for a time interval shorter than 30 min;
- The minimum and the maximum value of a parameter: the setting temperature for netromycin has to be greater than 5ºC and smaller than 18ºC.

The structure of the RWK database is shown in figure below.

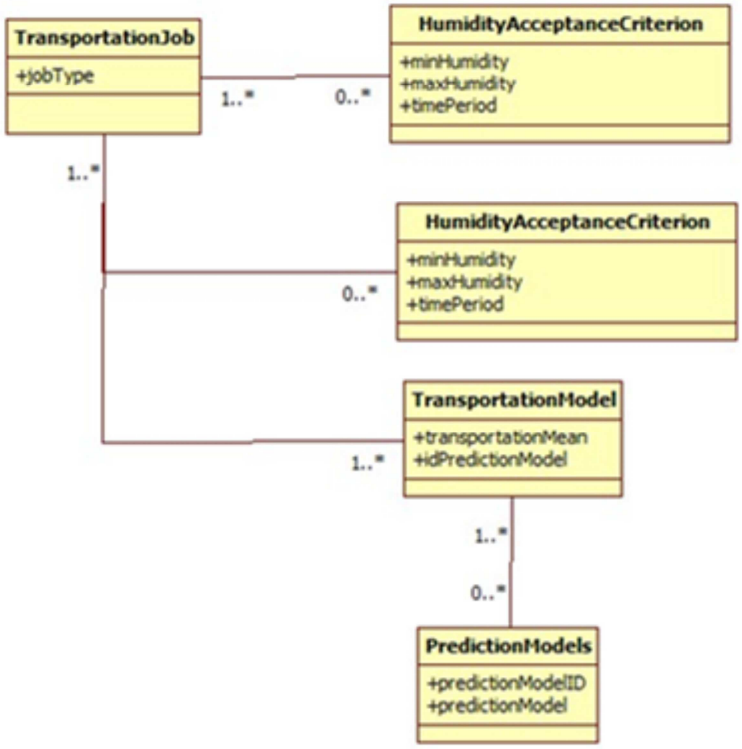

Figure 13: RWK structure for Smart Business PoC

These thresholds are used for post-factum acceptance report generation, for real time alerts, and for early warnings generation. The last two cases require passing the aforementioned thresholds values down to the CVO level by the service request analyser. The RWK part also contains time series forecasting models. A machine learning expert, supported by a domain expert is allowed to add or select candidate predictive models. For this use case, the candidate models include multiple linear regression, multilayer perceptron, and regression trees. The Java binary serialized form is passed down to the CVO level and used during the early warning scenario. The quality of the prediction model is monitored and compared to actual data (stored as real world information) and if the predictions are far from the expected values, a retraining of the model – and thus an update of real world knowledge – is performed.

In the **smart medical asset management** trial, the role of the RWK models is twofold i.e. to ensure the service continuity and to enhance the energy savings of the indoor positioning system. The hospital scenario is characterized by device movement patterns. This information is logged and used to derive prediction models that are used to provide location estimates when the positioning system is unavailable/energy efficient mode and enables the recommendation CVO to provide energy efficient plans in accordance with ZIGPOS energy models to improve the energy efficiency of the positioning system.

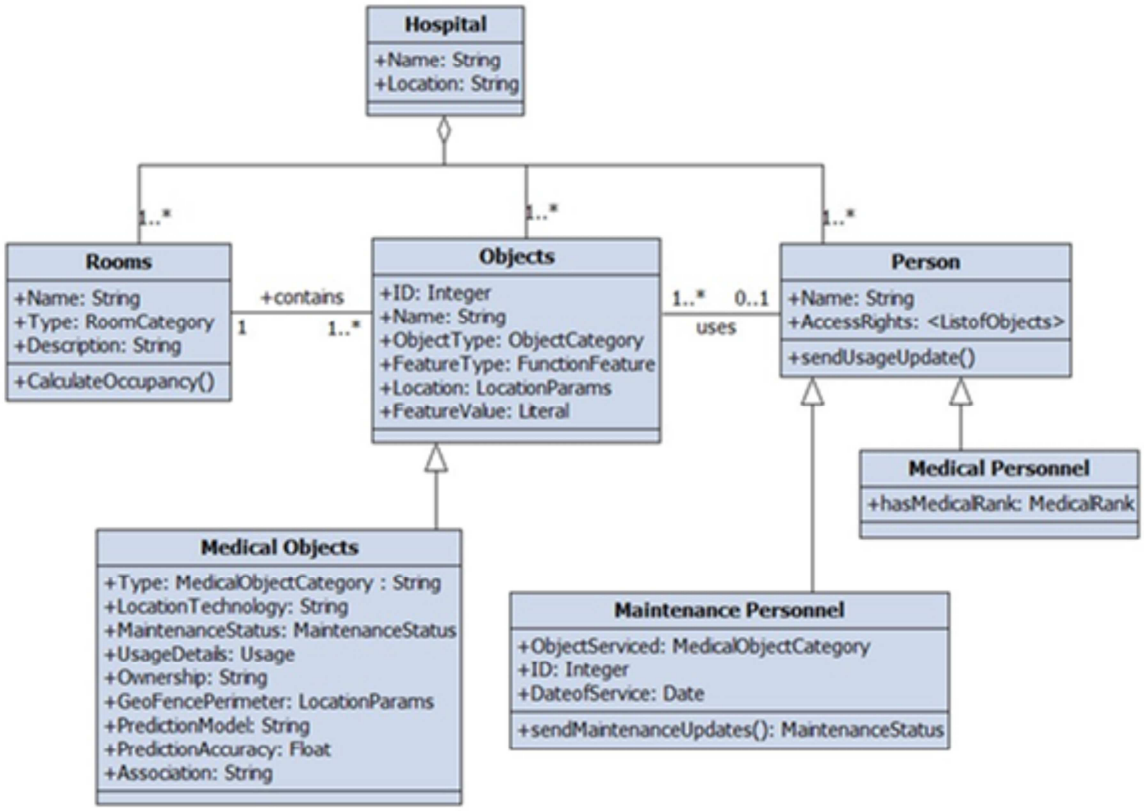

Figure 14: Real World Knowledge model for the Medical Asset Management trial

This picture illustrates the overall RWK model that describes parameters and relationships of the devices and actors in the generic context of a hospital owning IT asset. The parameters and relationships are captured in a relational database, which is implemented as a MySQL database. The database resides in a server and is updated with the changes observed in the real world. While the model represented above captures all the relationships, the trial implementation does not extend to the personnel details due to restrictions in tracking personnel within the hospital premises. Within this overall context a more detailed RWK model was generated for prediction purposes. This is where the iCore architectural component RWK Model appears making use of machine learning algorithms that underpin the model with predictions based on past observations. For an environment where the location and/or usage of objects is tracked, there might be preconfigured models that have already been successfully used and therefore become available as CVO Templates to help predict the location of an object based on a particular training set.

The RWK building has been realized in collaboration with TU Delft leveraging their expertise in the field of statistical analysis and machine learning. During the development phase, a test bed was set up in the premises of the iCore partner, Trilogis where the indoor positioning system was deployed. The people within the office carried the positioning tags during the day. This was done to realize a system representative of the actual scenario, i.e. simulate movement patterns of the position tags. Due to the initial restrictions in the access to the hospital, this alternate arrangement was agreed upon. After the initial tests, data from the real deployment will be logged for a long period to study the movement patterns and the developed model (on the basis of the test set up) will be adapted to the actual movement patterns.

3.2 Service Programmability and Modelling

Assuming a horizontal platform paradigm, an iCore platform supports an arbitrary number of service types and instances in one or more given application domains. To introduce a new (suite of) service type, a domain expert/programmer adds (a suite of) **Service Templates** to the platform (as is done with the RWK which is typically referenced from these templates). In this way, the platform is enabled to support a potentially wide range of Service Requests that, when issued, express goals iCore Users ask the platform to fulfil. As elaborated in D5.2 [4], the issuing of a Service Request result in the selection and instantiation of a (hierarchy of) Service Template(s) according to the Service Request-provided parameters. In this way, a Service Template holds the code to combine RWK and goal-specific service logic into a service description, anticipating in this logic all relevantly known RW behaviour and goal-induced effects, the so-called **Service Execution Request** (SER) passed into the CVO Level API, expressed in terms of the available executable CVO and VO types. The main Service Level function, called **Service Request Analysis** (SRA), is performing the template instantiation and produced SERs out of it.

3.2.1 Examples from use cases and trials

In the remainder of this section we illustrate **Service Request** and **Service Templates** by means of a range of examples from the iCore use cases. (The actual logical descriptions contained in SERs are the subject of the next section.)

In the **Smart Home** and the **Smart tour in the city cases**, a set of service requests are supported by implemented Service Templates. Indicatively, a few of them are:

- "I want an ambient living application." (Smart Home: Living Assistant)
- "I want to check the health of Sarah. Moreover, control the conditions of the room, please." (Smart Home: Living Assistant)
- "I need help moving through the city. I also need to get to my destination fast and see the sights." (Smart Tour in the city)

In the **Smart Office** use case the service requests supported correspond to the features mentioned in Section 2.3. The Service Templates for fulfilling these service requests are stored in the Service Template Repository in XML format and are queried by the Service Request Analysis component. They include a number of service parameters and features that need to be met by VOs and CVO logic. For the Smart Recording Service request, for example, the corresponding Service Template specifies that a *recording* feature must be supported by a CVO, and additionally the VOs connected to the CVO must provide the *recording* feature too. The Service Template depicted in figure below specifies service parameters *"recordOptimization"* for configuring the recording mode (power saver or high quality), *record Duration* (short, medium or high) and the *geo-location* which specifies the geo-position where the recording is to take place.

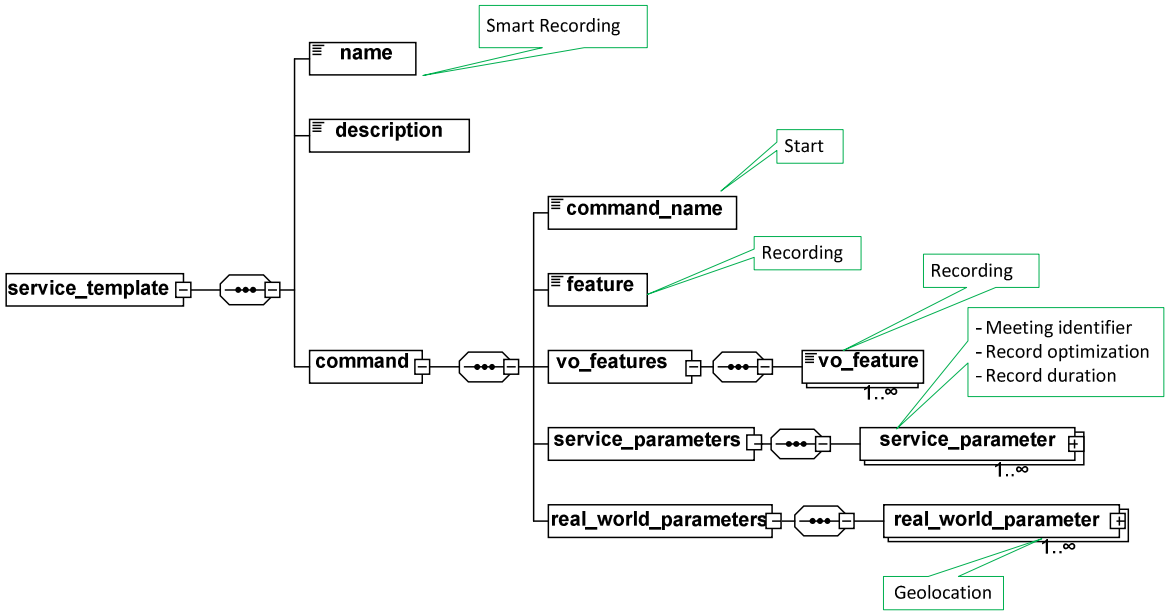

Figure 15: Service Template for the Smart Recording service

For the latter, as described in section 3.1.1, the SRA queries the RWK and RWInfo bases to resolve and to obtain information in order to fill the required parameters in the template. After all required parameters were filled, the SRA then passes the CVO Execution Request specified in XML to the CVO Management Unit via the CVO level API.

In the **Smart City Transportation** case, the Service Requests are managed by the CVO Management Unit. The CVO Management Unit exploits the knowledge available from the Recurrent Trip and Destination detection unit in order to discover the CVO available to be executed. The available CVOs are stored in the CVO Execution Engine XML format (see below, CVO level chapter) and the Management Unit can discover the potential CVO exploiting the Execution Engine APIs and invoke an execution request as well passing the needed parameters together with the request. The available Service Execution requests which are handled by the implementation are the "**Find a Parking Around Me**" and "**Find a Parking at Destination**". Both Service Execution Requests are proposed to the Driver (when available) and triggered by the driver through the appropriate device, the on-board *uConnect* Infotainment System.

In the **Urban Security** case, the following SRs are supported by implemented STs (and referenced RWK), and can be activated from a dedicated application GUI front-end for the Urban Security case (a so-called Control & Command console):

- "overview of area", showing a 'less redundant' selection of relevant camera views on a given geo-area (issued typically during whole police surveillance operation),
- "VIP monitor & control", showing camera views (PTZ controlled) where the person occurs and is expected to occur a short time ahead, and an interactive possibility to dictate live waypoints for evacuation (issued typically at the start of a VIP visit, until the end of it, with interactive part starting in case of evacuation),
- "crowd monitor & control", showing camera views where crowds occurs and are expected to occur a short time ahead (issued in case there are crowds or crowd mobility expected)
- "CBRNE cloud monitor, people effect video", showing camera views where people are hit by (and fall down due to) toxic cloud (issued typically when an explosion incident occurs)

- "CBRNE cloud monitor, CBRNE measurements", showing sensor data of CBRNE sensors on map (permanently monitored during whole mission, for immediate situation-aware action in case of an incident)

The corresponding STs are expressing logical graphs of (qualified) CVO and VO types (and are foreseen to be stored in an extended DOT-based format). As part of the front-end GUI, Service Requests are also labelled with an overall request priority, the observing policemen indicate for each SR compared to other SRs that are simultaneously active. These user preferences will ripple through into the SER expressions, for the system to take into account during execution of the simultaneous service instances. Although the Urban Security case does not offer a typical "horizontalization" (there is only a single application domain addressed here), the fact that the iCore platform is flexibly programmable concerning the service goals (while leveraging a common base domain knowledge model), which is essential for e.g. police mission management, where every mission is bringing essentially different requirements where 'one size does not fit all'.

In case of **Smart business** case, a service request GUI is used by the transport company clients for generating the transportation requests (see next figure). The client has to specify the product type, the addresses of the sender/receiver and the arrival time. The acceptance criteria for the products are included by the domain expert in the RWK database.

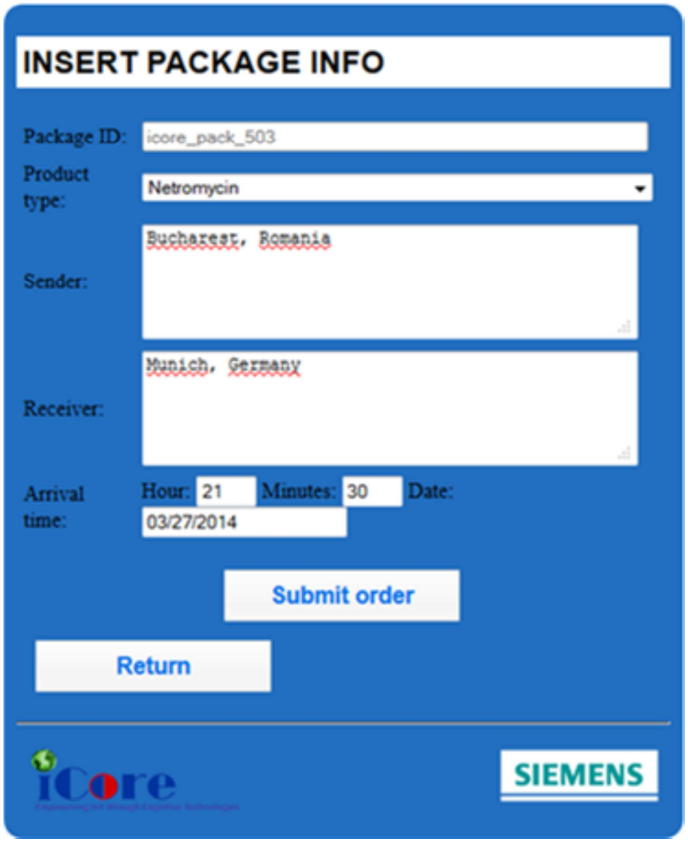

Figure 16: The GUI used by the transport company employees for registering a transportation request

In the **Smart Medical Asset Management** Trial, the Service Templates defines the required VOs and CVOs to fulfil a Service Request. The Service Templates are implemented as a workflow based model using the open source business integration platform, *Drools*. The pictured below shows a graphical representation of a simple service template as depicted in the graphical interface of the Drools workflow editor.

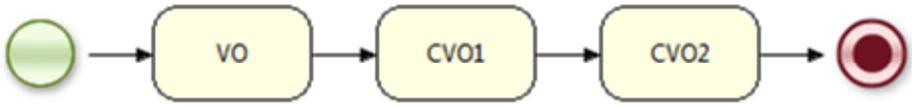

Figure 17: graphical representation of a simple service template

The underlying model is an XML file that gets stored in the repository together with metadata describing the features of the model to help assess suitability in satisfying a particular Service Request. Thus through the ST, a logical map of the interconnected components can be derived. Every SR has an associated ST. Four main services are implemented in the trial (i) **Locate medical objects** – returns the location of the medical object to the user (ii) **Trace medical objects** – monitors the object for movements and generates events when the objects enters/exits a defined geo/time-fence (iii) **maintenance services** - provides information on the maintenance status of objects (iv) **recommendation of energy efficient power plans**. For each of these services corresponding service templates defines the logical flow of execution to fulfil the service request.

3.3 Generated Service Execution

As mentioned, the eventual output of the iCore Service Level are the generated **Service Execution Requests**, **by means of a logical service description instructing the CVO Level which service instances to construct, deploy and execute**.

As the design approach taken in iCore considers the CVO Level API as the **boundary between the fulfilment of RW and System concerns in the service creation process**, the service description is expressed in terms of CVO types/names, VO descriptions or query expressions, and the data connections or orchestration among them. The SER should therefore hold **all options and constraints from an RW perspective**, thus identifying everything that is **known before execution from an SR and RWK model perspective** concerning *when which* (VO-sourced and sinked) data needs to be processed according to which generic functions (CVOs) in the running service. A clear advantage of the iCore platform making this split, lies in the facts that

1. static RW-dependencies are pre-analyzed before spending expensive real-time execution resources, and
2. RW models (RWK) and System models (SK) are not mixed, allowing for better model reuse and better human designability (**domain expert knows domain, system designer knows system**).

While this is a highly non-trivial design problem, the iCore research has identified early approaches to the specification and the automatic derivation of SERs.

3.3.1 Examples from use cases and trials

In the remainder of this section we illustrate these challenges by discussing the **specific service logic contained in SERs** as used in the different WP6/9 use cases.

The Service Execution Request is a collection of system requirements and information that is used to instantiate a CVO. In the **Smart Home and the Smart tour in the city** cases, typical system requirements are:

- The date the request was made.
- The type of device the CVO will be instantiated on (Desktop, Mobile, and Tablet).
- The role of the service requestor.
- The area the service (CVO) will cover; “Indoor” (Sarah’s house) or “Outdoor” (in this case a Google Map view is used).

- The policies selected by the service requestor; 1) “Low Network Costs”, 2) “Low Expenditure”, 3) “High Performance”, 4) “High Quality”, 5) “Energy Efficiency” and 6) “Security”.
- The CVO/VO types required.

For the **Smart Office** use case, the Service Execution Request consists of the XML Service Template document with all its fields filled in based on the interactions of the SRA with the RWK and RWInfo databases. The XML still describes features and location restrictions for fulfilling the service request, without instructing the binding to specific VOs and CVOs. It is the role of the CVO Management Unit to identify which are the most appropriate CVOs and VOs (in case multiple that provide the desired features exist) to be selected to fulfil the specific service request.

In the **Smart City Transportation** case, the generated Service Execution Request is issued by the CVO Management Unit, based on i) the Driver’s desires ii) the available knowledge on the Driver’s recurrent destination and iii) the availability of a CVO to fulfil the request. This is in turn translated into a CVO Execution Request towards the CVO Execution Engine passing i) the identified CVO to be executed and ii) the identified parameters. The CVO Execution Request is a SOAP call to the CVO Execution Engine.

As listed in 3.2.1 for the **Urban Security** case, the services considered in the Urban Security make a live selection of video streams (from a range of video cameras installed in the city) as relevant to a range of prioritized monitoring requests. For this type of services, one or more **Situation Observers** (expressed as possible composed CVO Types) are indicated in the SER service descriptions (and may be reused at the CVO Level driven by System processing gains), feeding data to **CVOs containing RWK Models of the observed entities, predicting** future observation opportunities. As such, the service description contains the logical formulation of what video streams to select in real time and which video streams will become relevant in the near future (upon which the service logic may also entail the output of these ‘to pay attention to soon’ live streams to a monitoring screen). For each SR, but also across the consolidated collection of issued SRs, the domain users indicate the maximum number of video streams that are acceptable as a cognitive load of the observation team. Although a simple set of integer values, this can also be considered RWK, and so should be incorporated in the issued SER expressions, together with an SR- (or RWK-) imposed priority rating. This is expressed in the SER by including a (cross-SR shared) CVO (a *Stream Priority Prediction CVO*) which determines the stream selection that is desired as an output for the SR-collection. Therefore, because of rather simple RWK being taken into account, ***additional logic*** needs to be generated from the RWK Model combined with the service-goal-specific logic expressed in the Service Templates. For the Urban Security iCore platform implementation, as well as in earlier experiments done in iCore, such as the Race Track WP4/5 PoC, we have assumed a **logical graph of parameter-decorated CVO Types** as the format to express the service logic in the SER towards the CVO Level. (Note that, typically, we expect this graph of CVOs to be *altered* at the CVO Level, by adding System-criteria and constraints to it, while also instantiating it for actual deployment and execution; see chapter 4 for a discussion of that.)

In the **Smart Business** use case, for practical reasons Service Execution Requests is sent towards CVO level in the form of an XML document, packaging all needed information including the prediction model to be used during transportation process. The SER is parsed at the CVO level to select and fill in the CVO templates for the task execution.

In the **Smart Medical Asset management** Trial, the iCore framework is triggered based on an incoming Service Request (SR). In this scenario, the service requestor i.e. user/hospital authorities

issues the SR. This interaction is facilitated by means of an *Android* mobile application. This is done in collaboration with the iCore partner Innotec-21. The SR request is communicated to the iCore framework, which resides on a server hosted at Create-Net. A REST based client-server communication model has been designed for the interactions. The open source *Jersey* libraries are used to realize the RESTful web services in Java.
The SR formulation extracts the tokens from the service request to create the SER. It has NLP functionalities based on the *OpenNLP* library integration. In the implementation it is currently restricted to extracting tokens and based on the presence of the "functional" tokens (i.e. track, trace, maintenance) on the basis of which, the subsequent sequence of operations are carried out. This is a reasonably standalone piece of functionality that can be improved at will without changing anything else. The format of the SER in the implementation is a hash map of parameters containing all the information of the required VOs and CVOs.
The SR analysis retrieves the required VO and CVOs in order to perform the SR execution. This is done based on the service templates, which are configured to reflect the functional token names and in co-ordination with the RWK models which plays an important role in determining the choice of the CVOs and thereby in the execution path. This component is what actually enables factoring out from the service execution chain the concerns and competence of "machine learning" experts that will look at the data set gathered and come up with a suitable model that can represent how the locations of objects are evolving over time, if there are any patterns and how these can be exploited.

3.4 Lessons learnt

In the previous subchapters, the main architectural features of the iCore Service Level have been illustrated by their application in the use cases where they are most relevantly applied. This has shown that there are clear, and to some extend also unique, benefits to the iCore approach reported first by deliverable D2.3 [1]. At the same time, it is also clear that further research is needed to refine the ambitious architectural design goals to the next maturity level, leveraging upon the valuable lessons learnt from the iCore experiments.
In this paragraph, we go over the identified benefits and values established and point out future research-orienting lessons learnt.

In e.g. the Urban Security case and the Smart Home case, the flexible horizontal programmability of the iCore platform by means of ***Service Request*** that can be issued at will by iCore Users is apparent. While supported by previous provisioning by a domain expert / programmer under the form of ***RWK*** and ***Service Templates***, the ad-hoc launching (during platform operation) of new Service Requests is clearly required and valued by the Control & Command police team in the Urban Security case, and the diversity of service needs that may occur in Smart Home scenarios.
Various expression formats for Service Requests and corresponding (hierarchies of) Service Templates (and RWK) were explored across consecutive iterations of the work and across use cases, and generic properties were identified.
A lesson learnt is however that it is hard to find a universally applicable expression for SR and STs based on the work of a single use case or technology base. It is therefore recommended to reiterate the architectural design in this respect over multiple domain-related (or totally unrelated) use cases, as such further maturing the architecture beyond the current iCore findings. A related element is also the actual production (programming) of the ST (and RWK) hierarchy for a particular application domain. This was not in focus for the iCore project (templates and RWK were most often manually added to the platform), but for a commercial platform product a proper creation environment and design methodology for the templates should be in place.

Another vital feature in the iCore platform is the support for ***RWK-based situation awareness***, which can functionally drive a service as an inherent part of it, e.g. when an observed phenomenon is monitored as an inherent goal part of certain SRs. Examples of this were shown in most use cases, e.g. the Urban Security case (services that track a person, crowd or chemical cloud), the Smart Meeting case (various meeting conditions), the Smart City Transportation case (behaviour of the driver / service user with respect to reaching a parking spot), the Smart Business case (temperature and humidity conditions of transported goods), the Smart Medical Asset Management trial (medical device location patterns).
A lesson learnt here, from the many examples explored, is in fact that a universal formalism for Situation Awareness, as a further refinement of RWK-based Situation Observer definition in the service platform context (and thus going beyond traditional Context Engine or Location-Based Services) would be beneficial for IoT Service Platforms. The current iCore Service Level approach of declaring RW concepts, behaviour and observable properties in an RWK model and then programming CVO compositions for observing the phenomena via VOs, as such making services and service execution situation-aware, has shown to be effective in this respect, but e.g. does not fully automatically support for (CVO Level decisions taken to do) aggregation of Situation Observers across service types or service instances yet. From a Service Level perspective, this may again come down to proper design of the ST and RWK hierarchy (and so fixing a methodology for that, and implying automation 'hooks' for the CVO Level to act upon), but further research is needed to explore further dependencies and requirements for such fully automated support.

Beyond situation-aware behaviour of the service instances themselves, RWK-based situation awareness can also be leveraged for ***optimization of RW or System resource spending***.
A typical RW resource is the human service user for whom the cognitive load should be kept below a human-manageable threshold (e.g. the observing policemen being presented only with the most relevant live sensor and video data at any given time in the Urban Security case, or the context-aware service interactions assumed in the Smart Meeting or City Transportation cases).
Several use cases have shown the benefit of optimizing resource spending based on RWK for System resources, typically ***anticipating dynamic resource needs by RWK Model-based predictions***. This is e.g. demonstrated in the Urban Security case (network video stream loading prioritized and flows/QoS dynamically pre-assigned for the streams the highest critical RWK-based relevance), and the Racetrack WP4/5 PoC (video processing reduction based on predicted RWK-based relevancy).
A lesson learnt in this perspective is around the challenge of separating the Real World and System concerns properly for being able to achieve an optimal effect: the RWK-based 'relevance profiling' expressed in the SER should not only be expressed in a generic way for the CVO Level to stay RW-agnostic, but also should allow 'weaving' in System concerns (as System Knowledge) that determine how exactly this should be leveraged to get optimal system gains. In the cases explored in iCore this was only achieved up to the extent of including simple number constants as SK, e.g. knowing the maximum amount of streams that fit within the overall maximum/ideal capacity of a given network, as an upper bound to the total of service instances running on the system. For this, a matured expression for Service Execution Requests towards the CVO Level needs to be addressed in future platform designs, in such a way as to express fully the flexibility and constraints inherent to the RW-realm of a service while allowing for the transformation of service instances according to SK/policy-expressed execution behaviour optimization goals.

Ultimately, ***training RWK models during service execution***, as a partial implementation of the RWK Cognitive Loop, allows true platform cognition, in the sense that during the operation of a collection of services, the service platform can refine or adapt its RWK Models, hence improving on the behaviour of services and the platform itself in the RW context. This was most articulate in the Smart Medical Asset Management trial (preserving energy in the location tracking system based on a learnt

RWK Model) and the Racetrack WP4/5 PoC (trained HMM for car camera position prediction). The RWK Cognitive Loop concept, as was discussed more extensively in deliverable D5.3 [5], has many more extensive variant possibilities, which e.g. include machine learning in both an 'online constrained' part of the loop inside the real-time control loop of service instances and a 'more relaxed offline' part of the loop outside the control loop of service instances, allowing for aggregated observation over time or many service instances, followed by an RWK adaptation strategy. This more extensive mechanism was not validated in any of the use cases and requires further research.

Nevertheless overall, the use cases have demonstrated the above discussed benefits to using the iCore Service Level mechanisms and concepts, and thus provides an encouraging basis for further maturation into commercial value propositions (as identified by partner exploitation plans indeed) as well as specifically targeted research refinements.

4. iCore Composite Virtual Object Level

Since Service Level represents northbound interface of an iCore system to world, having as key mission to identify real world knowledge and Service Requests, CVO Level is to be in charge to execute them using contextual information and best match of Service Execution Requests on available or generated as instance CVO. Of course, this step needs to be executed following an established procedure and using at execution policy level the System Knowledge – continuously acquired regarding resources from inner managed space.
Independent from the way how knowledge is acquired, CVO Level identifies a set of global functional feature groups involved in the management of level. **CVO Management Unit** as an entry gate from the Service Level direction coordinates the process of CVO lifecycle and keeps track of specific conditions fulfilment such as SLA. It can be considered from this point of via as autonomic core of CVO Level. **CVO Factory Unit** groups the "smart" features the CVO Level with the composition engine for CVOs. This grouping is supported by the logic that CVO Level smartness is relevant from the functional optimal matching of Service Execution Requests over the existing CVO level infrastructure that use System Knowledge Model as reference. **CVO Container Unit** represents the envelope who groups the various containers who can execute CVOs or CVO orchestrations based on their implementation technology. Since a CVO is not a single request /single match product, **CVO Registry Unit** is used to record on hold/running/executed CVOs with their corresponding package of Situation data and Service Execution Requests. This component aims to incrementally improve the performance and coverage of the system. **CVO Template Repository** is set-up since an iCore system needs to have a fundamental set of construction templates wiring domain of functional capabilities.

In the next sections we discuss the main architectural aspects of the iCore Service Level:

- The way the iCore platform executes **services as CVO compositions** (section 4.1),
- The **way specific SK is added and leveraged** in the CVO Level (section 4.2), and
- How the CVO Level **manages, deploys and shapes service execution**, given its resource overview, its suite of CVO templates and running CVO instances (section 4.3).

For each of these aspects, we highlight relevant mechanisms and components elaborated in iCore platform implementations as leveraged in particular WP6/9 use cases, thereby illustrating how the iCore platform is beneficial from the perspective of this aspect.
We wrap up the chapter by discussing the achievements, alternative options and open issues requiring further research.

4.1 Execution Engines and Distributed Execution

As explained above, iCore does not limit the wealth of IOT processes and information and limit the perspective to just a single formal model such as event processing networks or workflows, well established in IT solutions space. Considered CVO artifacts encode a solution correlating content (in different formats), time, and order and geospatial dependencies. Globally speaking a CVO as Service Execution Request represents a correlation of various VO and CVO glued together with "smart plumbing" code. This way, a CVO may contain a combination of various processing units hosted in specialized containers managed and orchestrated by CVO Container Unit.
A CVO Container Unit is in charge with a set of different functions:

- Manage the state of allocated containers including security aspects and associated SLA (level of load, speed of processing, pondering between execution time and precision, etc.)
- Identify, allocate and deliver CVO in appropriate execution container and CVO execution result towards the Service Level (as Service Execution Requestor).
- Deliver data to collect, analyze and formalize towards the CVO factory in order to maintain System Knowledge.

- Use and update regarding the fitted mapping between Real World Knowledge and System Knowledge aiming best use of resources according the communicated SLA parameters in Service Execution Request. This step represents a local cognitive optimization loop, since system is performing a self-optimization based on real world results performance measurement.

4.1.1 Examples from use cases and trials

In the remainder of this section we illustrate with a range of examples from iCore use cases different execution, **execution environments** (CVO Containers) that are supported by the iCore platform framework (workflow engines, CEP engines, etc.) and the way CVO instances can be deployed, connected by data streams and executed as part of one (or more) service instances based on CVO Template.

In the **Smart Home: Living Assistant** PoC and the **Smart Tour in the City** trial the CVOs are instantiated through a CVO Container supported by a collection of CVO Templates. The Container has been implemented as a RESTFul web server that is capable of incorporating and instantiating any Java based CVO (the CVO can also be a Restful web server). The logic that links the selected CVOs is decided by a dynamic workflow composition engine.

In the **Smart Office** use case, all CVOs are implemented in Java to be deployed on Java virtual machines running on regular personal computers or on smart phones running Android-based operating systems. For the Data Management CVO used in the Smart Break Recommender Situation Observer the open source Complex Event Processing '*Esper*' has been used for aggregating the noise level values over a period of time. The statistical calculations inside the Situation Detection and the Situation Classification CVOs have been implemented with the *Apache Commons Math library*.

The CVO Management Unit has been implemented as a Java Standalone program being able to run on standard hardware. The SRA information arrives to the CVO Management Unit via MQTT protocol; to understand MQTT messages, the application uses the Paho Java Client. The main MQTT operations are subscribing to MQTT-topics and publishing messages events. The CVO Registry implemented for the Smart Meeting use case provides a RESTful HTTP interface on which lookup requests can be specified as SPARQL queries. The CVO registry is realised as RDF triple store by using open source software *OpenRDF Sesame*. For the VO Registry the iCore reference implementation developed by partner UPRC has been reused that can be accessed through a well-defined HTTP API.

In the **Smart City Transportation** case, the CVO Execution is managed by the *M3S workflow-like composition engine*, providing composition logic based on event/action paradigm. The Composite Virtual Objects are stored in an XML format in the Execution Engine Repository and they are made available to the other architecture levels (CVO Management Unit) through an appropriate API, which allows the execution of a specific CVO together with the needed parameters. The Virtual Object composition, created by means of the appropriate Composition Authoring tool, in terms of the iCore Architecture is properly a CVO Template which is stored in a local CVO Template Repository. The CVO Template can be deployed and executed by the CVO Execution Engine, and the running Composition instance is, in terms of iCore Architecture, a CVO. The Execution Engine provides appropriate API in order to i) list the available CVO Templates ii) deploy a CVO Template to the Execution Engine and iii) execute a CVO Template.

In the **Urban Security** case, CVO instances are executed in different **execution engines**, depending on the computational nature of their processing:

- an *Esper* engine is used for most CVOs doing stream handling logic;
- A video processing environment (*Mosami*) is used for CVOs that perform video stream content processing;
- A dedicated GIS engine is used for CVOs doing specific geospatial calculations.

CVO Templates thus respectively capture stream-parametrizable Esper code, video processing pipeline declarations, depending on what engine they were designed for. (Note that it is the CVO Management Unit that decides on the actual CVO template/implementation to instantiate; the Service Level, for producing the SER expressions, used abstract CVO Type names which are engine-agnostic, effectively hiding any processing concerns from the Service Level.)

In **Smart Business** use case the CVO execution container is implemented using complex event processing networks with WebMethods CEP engine. In order to demonstrate online learning capabilities those WebMethods statements are making use of java function call-outs. Such call outs may contain user defined functions and there is the place where machine learning algorithms from *Weka* package are called. In this manner, online learning features are available and CEP statements are not used just to detect certain situations but also to use and improve prediction models themselves embedded in dedicated CVOs as user-defined functions.

In the **Smart Medical Asset Management** Trial, the CVO container is used to deploy the selected CVOs. It is a workflow based execution engine. The implementation is based on *Drools*, the open source business execution platform. It largely consists of rules that define the execution flow, with features of complex event processing. The CVO container is also linked to the *MySQL* database to store the outcome of the executions, which can then be utilized by the subsequent service requests as knowledge to provide better services. The CVO templates are executed as defined by the execution flow in the service templates, enhanced with the information from the RWK. The rule based format of the *Drools* engine also facilitates easy integration to calling other services for example the geo-location CVOs which reside in the server of the Trilogis partner. This interaction with external CVOs is achieved via MQTT connection. On reception of the SER, the CVO management unit has to find the required CVOs. The CVO registry holds information of which CVOs are available to satisfy the Service Request (Locate, Predict and Association CVO in this particular context); this is highlighted by figure below.

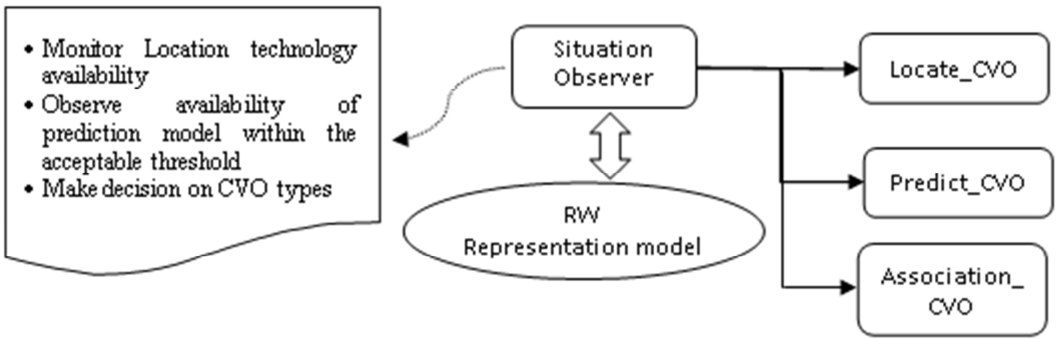

Figure 18: CVO Management Unit decision process

The situation observer in this case is the discriminator, which helps the iCore platform select automatically the most suitable CVO to fulfil the request based on context (Indoor positioning technology availability, predictive model already trained or association info). The decision making is based on the representation model of the real world, which consists of information of the current real word including information on the existence of prediction models and their accuracy.

4.2 Leveraging System Knowledge

In iCore, we distinguish the modelling of the 'Real World' from the modelling of the iCore System. The 'System' concerns the organisational units that use computational and communication resources at its disposal of realising a particular impact outside itself (while the 'Real World' is the collection of anything outside the System on which the System aims to have impact, which is in focus for the iCore Service Level, see chapter 3. System Knowledge (SK) as assumed in iCore at the CVO Level (like RWK at the Service Level) expresses behavioural properties of the iCore platform such as resource addressability / availability, resource usage costs and interdependencies / constraints. In contrast to RWK (of which the universal, static nature is leveraged by the platform), SK handling does not always benefit from being rigorously split from (run-time) system information such as the occupancy status of the system's execution or communication resources.

Deriving from SK and system information, the iCore CVO Level management functions essentially will decide on

- Which exact CVO Types to instantiate (may be a subtype of an abstract class as indicated in a SER).
- Where to deploy (place) these CVO instances (i.e. in one out different types of execution engine types corresponding on the implementation-specific CVO Template chosen, or across multiple distributed CVO containers, in any case taking into account current occupancy and service data connection dependencies, CVO to CVO or CVO to VO), and ultimately even
- How to shape/transform the logical CVO-composed service description into an effective deployment issued in a SER by means of SK-based dynamic behavior for improving execution or communication efficiency.

In this way, SK (and system status info) will determine how the platform spends resources to fulfil services as issued through SERs (potentially also taking into account SLA type of constraints expressed in the SER).

4.2.1 Examples from use cases and trials

In the remainder of this section we illustrate **System Knowledge** modelling with examples of SK as leveraged in particular iCore use cases.

In the **Smart Home: Living Assistant** PoC and the **Smart Tour in the City** trial the System Knowledge is comprised by information stored in the CVO registry by the Approximation and Reuse Opportunity Detection (AROD) mechanism at the time of the Dynamic CVO Creation. This System Knowledge is later exploited for the **knowledge-based instantiation** (re-use) of a CVO. More specifically, in the two cases, the System Knowledge information stored is mainly related to network cost and overall system responsiveness.

In the **Urban Security** case, SK is used for keeping **resource spending in the Wireless Sensor Network** under control. It is combined with RWK-based Situation Awareness, which is leveraged beyond the user experience improvement (discussed in chapter 3) by using it for determining the current and short-term predicted video streaming requirements for the network (see mechanism details in section 4.3). SK used in this perspective could concern the maximum allowable loading under battery-autonomy conditions, as a total time a WSN node can perform wireless streaming (under assumed distance and topology variability), and as a maximum possible bandwidth for each WSN node. For the actual implementation, it was chosen to simplify this to an overall value as a heuristic for the overall network. The design choice to have the iCore platform control the network via the *flow* abstraction (as in *OpenFlow* and Software Defined Networks in general) indeed implies that individual nodes and routes are not 'seen' by the iCore platform. Rather, useful SK in the Urban Security case is about the **total bandwidth that the network is capable to provide through its border**

router (gateway node), being a best-case upper limit to the battery-powered routing options through the entire WSN. Based on this number, the CVO Level can decide (as said, in combination with the dynamic RWK-based situation awareness and prediction) how many 'HQ' video streams can be requested simultaneously given that all other needed streams are requested in 'LQ' (meaning by 'HQ' and 'LQ' two distinguished quality levels). The two quality levels correspond to a **high quality and a low quality video viewing requirement**, for which the individual bandwidth requirements are two additional SK quantities. Given these SK numbers, a dedicated CVO instance can thus be programmed to oversee that at all times the number of requested HQ and LQ streams do not add up to a total bandwidth above the best-case achievable one. (Within that SK-based limit, an RWK-based flow prioritization with QoS reservation is performed dynamically, see further under section 4.3.)
Additionally, SK is also used concerning the **worst-case propagation time needed for router table updates throughout the whole WSN** to obtain Flow QoS reservations upon a flow priority update. This SK number determines how much time the live CVO processing needs to predict ahead the RWK-based situation evolution, for triggering the flow prioritization well in time (see again further under section 4.3).

In the **Medical Asset Management** Trial, the System Knowledge Model is representative of the infrastructure that enables the virtualization of the objects, the indoor positioning system. The energy model of the ZIGPOS positioning system has been evaluated and the SK model is able to assess what energy-saving actions are to be enforced to minimize energy consumption based on the actual logged object location.
Through a dedicated interface between the model and the indoor positioning system, the former can provide indications to the latter as to what choices will bring benefit from an energy budget point of view.

4.3 Execution Management Intelligence

iCore involves the use of cognitive like functionalities in both responding to services requests and in the maintenance of execution infrastructure. For CVO Level Management those aspects are reflected along all large functional groupings: CVO Management Unit, CVO Execution Unit and CVO Factory. Each unit is using available functions on different purposes. **CVO Management Unit** is independent from domain specific issues and will be able to keep the containers for example at right level of computing performance; System Knowledge in this case acts as policy support for the self-management functions implementation. The topic presents a relevant future perspective due to intersection with similar evolutions on Cloud Computing hypervisors. Key purpose of **CVO Factory** is to secure the delivery of "best matching" CVO as response to Service Execution Requests. Therefore, System Knowledge is usually relevant in planning and deploying concurrent CVO instances in one or more execution engines. CVO Factory contains also learning function collecting from CVO Execution Units field data in order to apply long term analytics. Finally **CVO Execution Unit** optimizes the load of individual CVO execution containers and aiming the realization of relevant KPI per CVO and iCore instance.

4.3.1 Examples from use cases and trials

In the remainder of this section we illustrate CVO Management Unit and other functions of the CVO Level with examples of SK as leveraged in particular iCore use cases.

In the context of the **Smart City Transportation** Use Case, the CVO Engine has been exploited in order to provide the composition logic for executing the logic which is requested by the Service Level. The composition logic is created by means of the specific Composition Authoring Tool, and in particular two different CVO have been provided for the purpose of this Use Case: Parking around me

CVO and Parking at Destination CVO, both described as CVO Template. The CVO Engine provides specific API for i) discovering which CVO Template are available for execution and ii) which CVO Template must be executed. The following figure describes the composition used for the "Parking around Me" CVO.

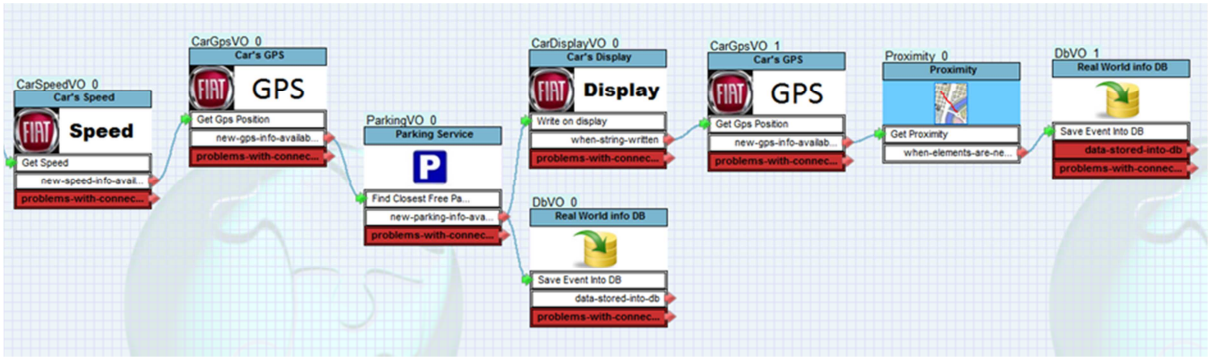

Figure 19: The Parking around Me Composition Logic

The event-based data flow orchestrator is able to process events incoming from the Virtual Objects and invoke actions towards the Virtual Objects, exploiting the SOAP Web Services interface defined for the Virtual Object level.

In the **Urban Security** case, the CVO Level leverages the RWK-based behaviour captured in the SER expressions it receives to obtain not only the desired fulfilment of the requested services, but also to **spend the scarce WSN resources intelligently and dynamically**. The situation prediction (realised through Situation Observer/Predictor CVO instances of type *Trajectory Forecast CVO*, *Circular dispersion prediction CVO* or *Clustered velocity prediction CVO*) are each feeding into a *Stream Priority Prediction CVO* for each service instance, which keeps the dynamic overview of which video streams, at what quality, should get the priority within the SK-determined time for that service. A **Flow (Priority) Controller**, functionally part of the CVO Management Unit, consolidates these per SER predicted priorities (taking also into account the overall priority of each SER) and issues *flow priority* updates to the (SDN-like) Router Manager of the WSN. The general principle is illustrated in next figure. By this mechanism, the network is **provisioned just in time to reserve the Quality of Service for particular flows** (corresponding to their Camera VO source in the use case) according to the predicted priority requested streams will have in all service instances. As such, the executing CVO graph uses RWK and SK, logic from Service Templates as well as additional CVO Level logic captured in CVO Templates, to get to utilisation of the network resources in line to the application requirements; ***the iCore platform programs the network in an application-aware way***.
Another notable (more generic) CVO Level functionality supported in the Urban Security iCore platform implementation is the **automated reuse of CVO instances** that are working on the same data for different service instances. The CVO Management Unit detects this in the consecutive SER descriptions (CVO Type-based graphs in this case) and reuses data streams already resulting from running service instances rather than doubling the processing with another CVO instance. This is typically the case for Situation Observer structures, but a similar reuse is enforced for CVOs performing actuations towards particular VO instances. Indeed, when multiple service instances require the control of one particular VO instance, such as e.g. PTZ control of camera orientation in the Urban Security case, they all need to share a CVO instance that arbitrates which control commands make most sense.

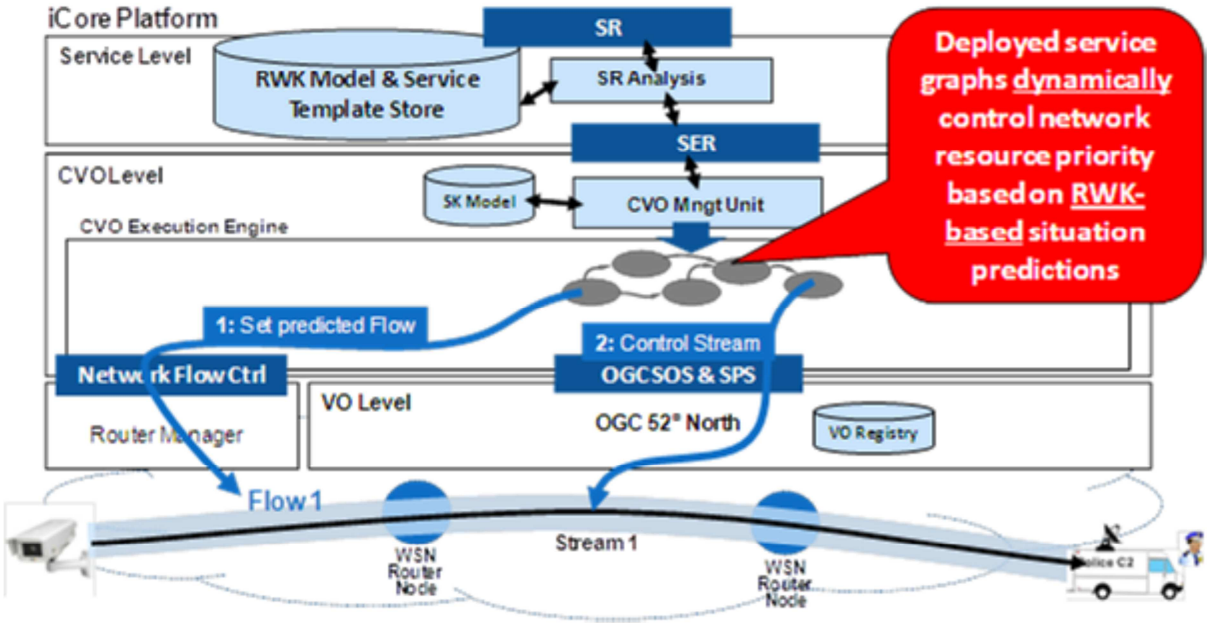

Figure 20: Network flow priority control leveraging RWK-based situation prediction in Service instances

Agnostic from the specifics of the use case **Smart Business** implemented a Service Requests Analyzer (SRA). For the effective use case details, this component is used for generating the appropriate service execution request, depending on the job type specified by the transport company client. SRA identifies from the real world knowledge model the transportation criteria for the product and selects a list of prediction models which can be used for estimating the transportation condition of a package. The list of prediction models contains various prediction models for the available transportation means (by truck, by minivan, by plane, etc.)
Regarding the Smart Business use case the RWK contains some predictive models which are in charge for triggering early warnings, based on previous experience and on the current input. For the current SRA a collection of candidate predictive models are handled to the CVO Management Unit. The concrete CVOs which are executed influence the choice of the predictive model.
The predictive model of choice is de-serialized and wrapped by a CVO which further connects to appropriate datasets. The embedding CVO may include the workflow traditionally found in a ML process: selection of the data, preprocessing (other than CEP, it might include here some customized data conversion, e.g. discretization or casting from categorical to numerical data).

In the **Smart Medical Asset Management** Trial, the CVO Management Unit carries the functionality of lookup of the CVOs as specified in the SER. The CVOs reside as templates that can be instantiated. There exists a generic CVO template interface which must be implemented by all the CVOs of the system. These are then executed in the CVO container which is a workflow based Drools engine with control over the sequence of execution. There also exists the Situation Observer which continuously monitors the real world based on the information obtained from the RWK to make decision on the type of CVO. For example, the observer continuously looks for the status of the positioning system, i.e. if it available or in the sleep mode. If it is in the sleep mode, then the availability of prediction models are verified along with the performance check, i.e. if the accuracy of the model is above a specified threshold.
Base on the availability of the prediction model, the estimated locations of the devices are determined. All this information is extracted and is represented as part of the RWK model. In addition to the RWK, the system information such as power modes, battery levels of the ZIGPOS positioning system along with the energy model of the deployed infrastructure constitutes the system knowledge. The "**recommendation CVO**" uses the estimated locations from the RWK and

uses the system knowledge in order to derive efficient power plan recommendations that will be provided to the ZIGPOS positioning system. The communication of the recommendation is handled via the REST interfaces. Thus, RWK and SK together facilitate the efficient use of resources in addition to fulfilling the requirements of the SR and providing a reliable service.

4.4 Lessons learnt

Similarly to the structure of the previous chapter we focus in this paragraph on the identified benefits and values established by iCore approach (see D2.3[1]) and its application then we point out future research-orienting lessons learnt.

The existing implementations fully exploit the architectural building blocks. Beside the executive role, the CVO components are actively involved in the cognitive cycle: the Service Execution Requests carry appropriately embedded RWK and cleverly orchestrates the available VOs.
In this regard, the architectural thinking identified two large ways to approach **end to end cognition** inside an iCore platform:

- A **global cognitive loop** involving both Service Level and CVO Level, separating the roles of them as "command and control" for SL and execution for CVOL. This way has the advantage that all complex computation regarding state sensing is isolated in specialized CVO instances called Situation Observers. Major drawback is the coupling and dependency generated between those conceptual levels.
- **Separated level Situation observing capability each one focused on different aspects of knowledge space**: SL on Real World and CVOL on System World. This approach promotes specialization and functional decoupling but requires increased processing capabilities on deployment infrastructure.

Each of those approaches expose advantages on the use cases level depending on level of distribution considered. Both of them secure the acquisition of knowledge and the use of it on needed place.

It is worth emphasizing some of the choices made for implementing CVO functionalities. In the Smart office, Urban Security and Smart Business use cases, Complex Event Processing engines such as *Esper* are in charge of CVO instances execution. In Smart Medical Asset Management trial the *Drools* supports calling of appropriate services. Using such standardized third party tools are the premises for adapting and extending the current implementations to other use cases.

Regarding **SLA management** (level of load, speed of processing, pondering between execution time and precision, etc.) by CVO container and adaptation of mapping between Real World Knowledge and System Knowledge (aiming best use of resources according the communicated SLA parameters in Service Execution Request), few progresses have been made in practice and this is a topic for next research.

The **System knowledge** is somewhat underrepresented in the use cases, and we see this as a result of the difficulty of creating truly autonomic systems. Because global cognitive loop has been the approach used by iCore, another potential issue is that implementation of the CVO level is effectively tightly bounded to the corresponding SL one; thus, their services might be not reusable. A challenging question is whether one may define a unique API or at least cluster the existing implementations based on shared functionalities. Thus, service externalization might be possible. Also because of this monolithic perspective of 1-to-1 construction of Service Level and CVO Level, the intelligence for RWK and SK growing is specialized by domain and leads to the need to re-train/re-model both if the architecture model change to more decoupled levels. Furthermore, the mapping

between RWK and SK can consistently influence system performance, if the system design is not properly splitting the concerns.

The CVOs (recommendation and prediction) have been implemented according to pre-configured **CVO templates** created by a domain expert (resulting in workflow-based implementation within the CVO Container). This is a part where there is clearly scope for further research: availability of more general purpose CVO templates that become easily adaptable in similar situations is what will move us closer towards automated execution of service requests.

In a nutshell, the architecture at the CVO level has efficiently covered and supported the implementation of different use cases and trials. The building blocks are sufficient to host the aimed functionalities, although **more standardization and service externalization** may boost implementation and integration at this level.

5. iCore Virtual Object Level

iCore Virtual Object level provides an abstraction of physical objects linked to the iCore system. This includes interfaces towards the CVO level and semantic descriptions of ICT objects associated (and associated non-ICT objects). **Discovery** of Virtual Objects is performed through the VO Registry module, which enables CVOs to discover VOs using semantic queries. VO level also includes limited cognitive functionalities. The general functional VO level architecture is presented in figure below. The figure shows the core components of the VO level which are reflected also in the use cases and pilots to a high degree.

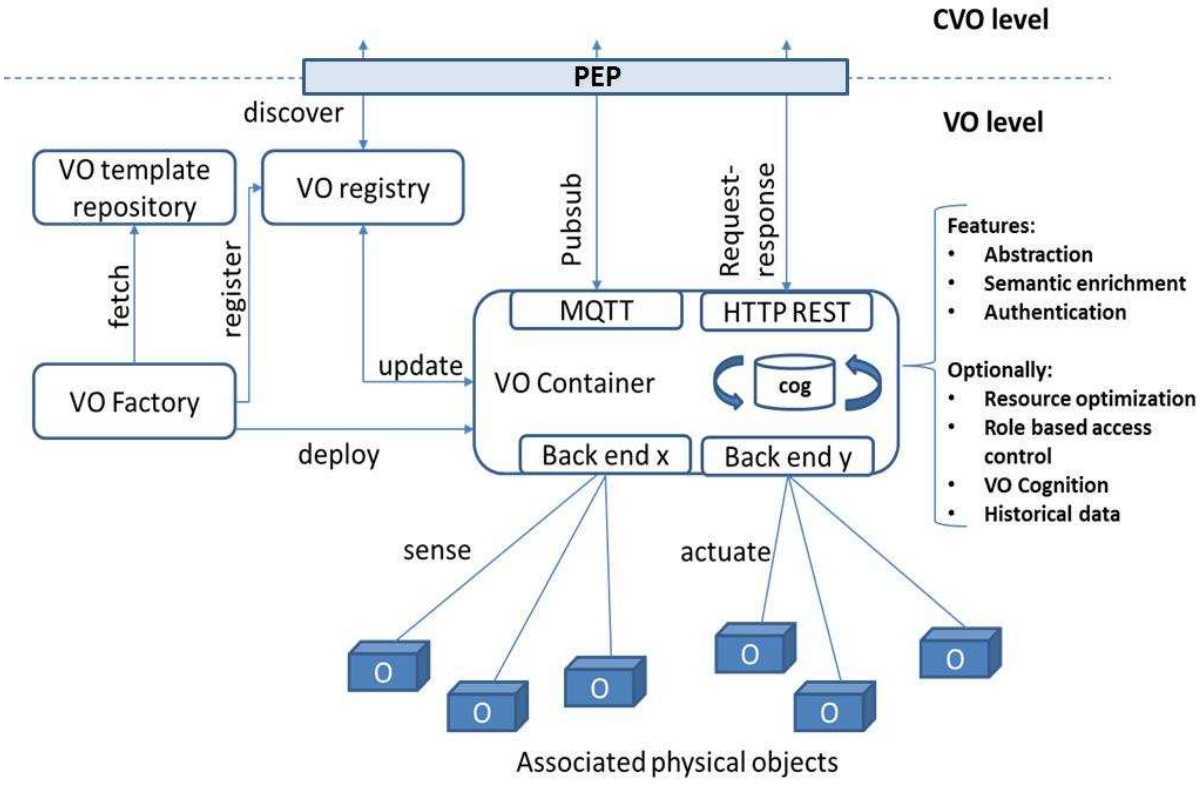

Figure 21: VO level functional architecture

At the core of this level is the VO information model, which describes the VOs. The VO **information model** has previously been mapped to the IoT-A architecture reference model and SSN sensor ontology (in deliverables 3.2 and 3.3 respectively [17][18]) and it can be categorized as a general semantic model for IoT actuators and sensors. **Naming and addressing** of VO is solved using approaches similar to the World Wide Web, namely Universal Resource Identifiers, URIs. At the heart of the VO **execution** is the VO Container, which hosts VO front-ends (namely MQTT and HTTP REST) and various ICT device specific back-ends, depending on the use cases and trials. **Security** interceptors and associated policies are in place with Policy Enforcement Point to prevent the ICT devices from being accessed by unauthorized entities (e.g. CVO). VO Container also hosts various default and optional functionalities. In the following sections, we present core concerns of the VO level in more detail and show how key components are reflected in use cases and pilots.

5.1 Information model

The vast amount of the devices that are and will be part of the IoT imposes the need of the development of an abstract information model for the description of heterogeneous devices. Consequently, the VO Information Model was developed as a generic model that is used for the description of a wide range of IoT devices that are applied on simple infrastructures such as sensors

and actuators, as well as on more complex technological infrastructures like computers and smartphones. The exploitation of semantics constitutes a key concept with added value for the VO Information Model. The definition of appropriate metadata into ontologies gives the ability to create semantically enriched virtual representations, which represents in the virtual world the specification and the capabilities of heterogeneous ICT Objects. In addition, the need to describe non-ICT Object that is associated with ICT Objects necessitates the definition of relevant metadata that will describe the features of these objects. Thus, the VO Information Model includes a set of metadata that is used to describe both ICT and non-ICT object, in terms of their properties and associations, in one common data structure, which is called VO description. In this direction an information model has been designed and developed using OWL for the definition of the metadata as well as using RDF for creation of data structures that composes the metadata in order to present a stable overall description of a virtual object. The model is presented in figure below.

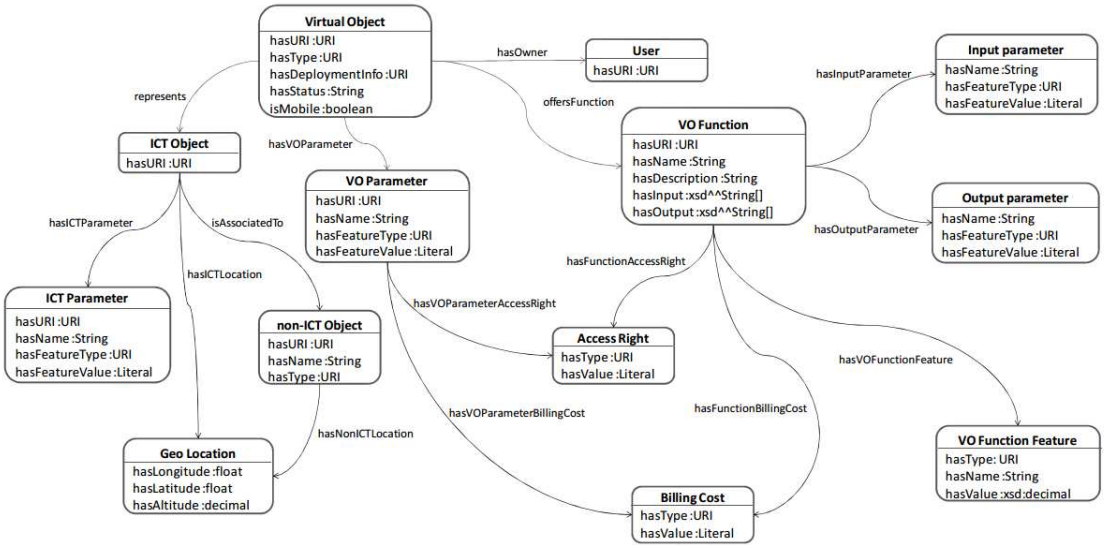

Figure 22: Virtual Object Information Model

5.1.1 Examples from use cases and trials

In the **Smart Home: Living Assistant** PoC and the **Smart Tour in the City** trial, the VOs used can vary greatly depending on their functionality and therefore the entirety of the VO information model (see above) has been used to describe/register each VO in the VO Registry. In both cases, the VOs used can be divided in sensors (e.g. traffic, temperature, luminosity, accelerometer sensors) and actuators (e.g. lamps, fans, heaters, buzzers) and they are used in conjunction with the VO Container in order to provide CVO functionality such as room conditions control for the Smart Home CVO or traffic monitoring for the Santander CVO.

In the **Smart Office** use case the VOs represent android devices involved in the features. These include as ICT parts Tablet and Smartphone devices used as Smart Displays and for guidance, collection of information for the break, recording and wrap-up features as well as for displaying relevant notifications to the participants and the meeting organizer. The VOs are specified according to the Virtual Object information model presented in previous section. Each Virtual Object is represented as a series of features and functionalities. In the Smart Office scenario, the geo-location of an ICT object and its owner is really important and is included in the VO registry. Later in this document, more details will be presented concerning the role of the geo-location in the discovery and exploitation of the features provided by any given VO. As far as the aforementioned features are concerned, they are all presented based on an XML VO template, similar to the one used in the

Urban security case. The XML template encapsulates information on the hardware sensors, on the mobile devices, functionalities supported by the VO, and protocols that can be used for communication purposes.

In the **Smart City Transportation** case the information handled by the Virtual Objects is related to the following entities:

- The Parking System VO provides information on the number of parking places available at the closest parking site, based on the provided location.
- The Car GPS VO provides information on the current Driver position, extracted by the GPS devices
- The Car Infotainment VO is an actuator, and it handles the Information Display Requests incoming from the CVO Execution Engine.

The following table describes these VOs through examples of properties, here functions "invoke Action" and "notify Event" used by respectively CVO to VO and VO to CVO data flows:

Virtual Object	**invokeAction**	**notifyEvent**
CARDisplayVO	String row1 String row2 String row3 String row4 /* each row is a row message in car display */	String infoState; /* info displayed OK, or error message */
CARGpsVO	String sendPosition	Latitude Longitude Speed
ParkingVO	Latitude Longitude	ParkingName ParkingLatitude ParkingLongitude TotalSlots FreeSlots

Table 1: example of VO models for Smart City - Transportation use case

In the **Urban security** case, information models at VO level cover two main points: first is related to discovery process, second is related to sensors readings and devices control in general. We focus here only on information models related to northbound interface and sensors readings and devices control in general. These information models are linked with southbound interface information model specifically defined for urban security wireless sensors network purpose. Shortly, sensors readings modelling include phenomenon measurements and related metadata such as detection time, period of validity, quality of the measure, events that generated the measurements. Model describing devices command includes list of available commands, parameters for each with range, etc. XML based **VO templates** are also used that support standardized and generic description of a simple sensor, a group of sensor or even a sophisticated detection process with geo-location, types of observations, inputs and outputs of the detection process, detection coverage area, available network interfaces, etc.
Because of their Open Geospatial Consortium (OGC) origin, these XML based information models are kept extremely flexible with a high emphasis on geomatics concerns, which is critical for urban security and surveillance missions where C2 relies heavily on cartography for situation awareness.

In the **Smart Business** use case, the VO semantic annotation describes the ICT abilities. For example, in case of temperature sensor, the associated description includes sensor's abilities, range of

operation, whether it is bound to a parcel, last update time, associated parameters, etc. See figure below from below for an excerpt of semantic description.

```
:101200cc2534  vo:identifier  "10:12:00:cc:25:34" ;
        vo:lastUpdate      "Thu Feb 20 23:37:23 CET 2014"^^xsd:dateTime ;
        vo:manufacturer    "Ambient Systems b.v." ;
        vo:non_ict_object  [ as:shipment  [ as:shipmentActive      false ;
                                            as:shipmentIdentifier  ""
                                          ] ] ;
        vo:owner           "iCore" ;
        vo:parameters      [ as:function  [ as:accuracy            0.5 ;
                                            as:functionMQTTTopic
"iCore/Temperature/TMP102/Temperature/*/10:12:00:cc:25:34" ;
                                            as:functionName        "Temperature" ;
                                            as:functionProvides    "Temperature" ;
                                            as:functionSensorType  "TMP102" ;
                                            as:functionUnit        "°C" ;
                                            as:maxValue            120.0 ;
                                            as:minValue            -40.0 ;
                                            as:resolution          0.0625
                                          ] ] ;
        vo:parameters      [ as:function  [ as:functionMQTTTopic
"iCore/Temperature/PT100/Temperature/*/10:12:00:cc:25:34" ;
                                            as:functionName        "Temperature" ;
                                            as:functionProvides    "Temperature" ;
                                            as:functionSensorType  "PT100" ;
                                            as:functionUnit        "°C" ;
                                            as:maxValue            120.0 ;
                                            as:minValue            -100.0 ;
                                            as:resolution          0.015625
                                          ] ] ;
        vo:parameters      [ as:communicationProtocol  "MQTT" ;
                             as:communicationUrl       "tcp://vps38114.public.cloudvps.com:1883"
                           ] ;
```

Figure 23: Example VO Registry entry for a temperature sensor with measurement to be inserted in perishable produce

In the **Smart Medical Asset Management** Trial, the VOs represent medical devices that are tagged with the positioning sensors. It consists of the sensor information i.e. the location information of the objects, the name of the medical device, unique address, description, custom name, timestamp, movement information (information from an embedded accelerometer) and the accuracy of the readings. The semantic enrichment of the VOs i.e. adding additional contextual details is also performed e.g. adding the geo/time fence perimeters which are required by the Geo-Fence CVO in order to define the boundaries where the entry/exit events will be triggered. While the iCore defined VO model is used along with the VO registry implementation of UPRC, these additional details are also captured in the information model of the VOs specific to the trial. The RWInfo coming from these VOs are logged in a database. It is utilized by the RWK models for building learning models. The communication to the VOs as mentioned in the preceding section is possible through both REST interfaces, to extract information of all the available VOs as well through MQTT in order to obtain only the updates of the location of objects. The MQTT interfaces are structured to receive all the location updates if the wildcard is utilized on the topic or one can listen for the updates of only a particular VO. For example /assetPosition/+ provides the location updates of all the devices whereas /assetPosition/assetId provides updates only the specific device. The information model is tailored to reflect and include the features of the trial deployment scenario, i.e. a hospital and taking into account the kind of information received from the various virtual objects.

5.2 Naming and addressing

At runtime, each iCore system interacts with a large number of ICT and non-ICT objects, which are virtualized by VOs. Each iCore instance has to be uniquely named; furthermore, each iCore instance behaves as a naming authority for the entities it contains. Each VO is addressable from within any iCore instance. Entity names are unique and give an informative description of the referred entity. The iCore instance represents an umbrella under which the VO entities are stored. Each iCore

platform contains a naming schema that defines the rules for naming the resources. Because iCore instances and VOs are Web Resources that are available in the IoT, the URLs can serve both as names and addresses for these resources. The address of an iCore instance must contain the domain name registered in the DNS (Domain Name System) server. The addresses of all VOs stored in an iCore instance must contain the address of the instance and *several naming schemas can be used*. For example the REST style URL of a temperature sensor can be:
www.example.com/iCore/building/room/sensors/temperature/17.

The URIs can be used both for the naming and for addressing (N/A) of the VOs as web resources. Thus the VO N/A scheme can be used both for the naming and for addressing of VOs, since it allows the dynamic creation of unique URIs. Each VO will have a unique base URI (VO_BASE_URI) that will constitute in the same time its own name and address in the Web. For instance, if we assume that a VO has the address www.example.com/iCore (as it is presented above) this address is considered as the VO_BASE_URI. The **VO N/A Scheme it is based on the VO information Model** and through its use it is possible to support the dynamic naming and addressing of the available VOs, exploiting the functionality of the VO Registry mechanisms and more specifically the functionality of the VO Registration Mechanism. The VO Registration mechanism adopts the N/A scheme and creates dynamically the URIs for the resources that are associated with the VO (e.g.: VO functions, VO parameters, etc.), which is already identified by the VO_BASE_URI. Consequently, the scheme allows the dynamic creation of names and addresses for the VO related resources, requiring only the VO_BASE_URI. For instance if a VO has two different functions with id 'funct_1' and 'funct_2', respectively, then the URIs, following the VO N/A scheme will be formed as follow:

- funct_1: VO_BASE_URI/vo_functions/funct_1/
- funct_2: VO_BASE_URI/vo_functions/funct_2/

In order to face a dynamic and changing Web infrastructure, we can use the Persistent URLs (PURLs). Instead of resolving directly to Web resources, PURLs provide a level of indirection that allows the underlying Web addresses of resources to change over time without negatively affecting systems that depend on them.

5.2.1 Examples from use cases and trials

In the **Smart Home: Living Assistant** PoC and **Smart Tour in the City** trial, approximately 150 VOs have been registered for use. The registration process is done through the VO Creation Tool (VO Factory). Each VO can be accessed through a unique URI, e.g. "*http://83.212.238.249:8040/urn_smartsantander_testbed_3310*" (as specified in section 5.2). These VOs are discovered during the CVO creation time (as described in section 5.3) by the AROD ("Approximation and Reuse Opportunity Detection") mechanism. Afterwards, they can be used by each individual CVO through the VO Container.

For the **Smart Office** use case, the involved devices when firstly introduced in the iCore system subscribe to an MQTT broker at a known IP address and their ids (unique *Android* device id) are stored in the VO registry.
The communication afterwards is performed using MQTT with the device id being encapsulated in the MQTT message payload. The reason for relying on MQTT communication is the highly mobile nature of the VOs involved in this use case, which would have made the use of static URIs inefficient.

In the **Smart City – Transportation** case the Virtual Object Level interface towards the CVO Engine is provided by the Virtual Objects as SOAP Web Services, which are invoked by the CVO Engine through a specific event/action API. The northbound interface towards the CVO Engine provides specific

method calls ("invoke Action") which asynchronously returns the control to the CVO Engine, while the Virtual Objects fire back events to the CVO Engine exploiting a specific call-back method provided by the CVO Engine SOAP Web Services ("notify Event").
The Smart City – Transportation PoC has implemented the following Virtual Objects:

- The Parking System VO is the Virtual Object towards the Parking System infrastructure information system: it gets geographic coordinates as input data (sent by the "invoke Action" method) and it returns the nearest parking place together with the available parking slots, based on these coordinates.
- the Car GPS Sensor VO is the Virtual Object firing events to the northbound interface with the information on the actual position of the car
- the Car Infotainment Display VO is the Virtual Object providing the interface towards the Car Infotainment System, receiving as a "display" action the message to show to the driver

In the cases of the Car Infotainment System and the Car GPS Sensor, the Virtual Object address used in the context of the specific CVO execution is defined in terms of a SOAP HTTP Web Service address, like described below:
http://<CAR_VO_CONTAINER_ADDRESS>:8080/axis2/services/<DISPLAY_VO>
http://<CAR_VO_CONTAINER_ADDRESS>:8080/axis2/services/<GPS_VO>

In the **Urban security** case VO level can be viewed through its southbound and northbound interfaces both supported by **technical standards**.
Southbound interface relies on several URL scheme[1] and convention defined by IETF CoAP, AXIS VAPIX, IETF RTSP, while northbound interface is based on Open Geospatial Consortium URL.
OGC standards are used for northbound interface with CVO level because it defines a **unified sensors interface** that support abstraction of any kind of sensors.
Because the surveillance system uses heavily video camera, VAPIX / RTSP URL are specialized to this and a reasonable choice while CoAP is specifically dedicated to the others sensors with lower bandwidth requirements (chemical sensors, VIP GPS, etc.)
For example, a request from CVO level to get chemical sensors detections will use addressing and naming like:

```
http://localhost:8080/52n-sos-webapp/sos/kvp?service=SOS&version=2.0.0
&request=GetObservation&procedure=urn:ogc:procedure:Thales:Chemical1
```

This request is related to CoAP notification from CoAP resources such as *coap://[aaaa::200:0:0:2]:5683/R2="l=120"h=1406213839* over the sensors network.
A request from CVO level to steer the video camera to the required position will use *http://localhost:8080/SPS/SPS* that will be further translated into *http://172.16.105.11/axis-cgi/com/ptz.cgi?zoom=5400* controlling the given video camera.

Architecture of the VO level integrates required mapping between physical (or virtual in our case) deployed devices with URN at VO northbound interface; URN suffix are preconfigured for each sensor and is set-up in VO templates through the discovery process (described in next section).
We don't use other URL mechanisms such as PURL (Persistent URL). During a given mission when URL and resources behind (typically sensors) are unavailable this is because of network failure (wired/wireless communication module of a wired/wireless sensor), or because of detectors failure (detector module of a wired/wireless sensor). Both kinds of events are reported as soon as possible

[1] All references may be retrieved in deliverable D6.6 [12].

through alert messages to C2 operators and the system enters downgraded mode decreasing probably situation awareness quality. Possible fallbacks are either manually or automatically set-up; ultimately some data may not be available. If a network (communication module of a sensor) failure happens, possibly network re-routing is achieved while if a detector failure happens nothing can be done before restarting or replacing physically the device: in these cases the issues here are not about underlying resource moving from one location to another since a URL maps into a given sensor; furthermore another sensor can't replace the one in failure; each sensor provides a unique set of data that is not equivalent to another set of data from another sensor; this is also because sensors coverage that are overlapping during deployment is not the preferred option at least for same type of sensors due generally to small number of sensors available.

For the **Smart Business** use case, the combination VO Back-end/Front-end transforms Ambient Systems specific messages (via socket communication e.g. received from gateways in wireless sensor networks) into XML-formatted messages (based on XML templates) and publishes those to an MQTT broker. MQTT topics under which messages are published are created according to the following scheme: *owner/net_id/class/sub_class/function/type/device_id*
"Owner" represents the owner of the specific device; this is derived from a database maintained by the device installer or manufacturer. The "net_id" represents the serial identification of the WSN Gateway through which the sensor information was received. The "class"/"sub_class"/"function" identifies the data content e.g. *Temperature/TMP102/Temperature* indicates a temperature reading from a specific sensor type. The "type" indicates the type of the message e.g. "Alert" tells that the sensor device itself considers the reading to be alerting. And finally, "device_id" contains the serial identifier of the sensor device itself. The topic is designed in such fashion that it is easy to create content specific MQTT subscriptions. The VO Registry provides means to compose MQTT topics based on VO function names in addition to the URL of the MQTT broker

In the **Smart Medical Asset Management** Trial the VO Registry stores the last-known location of all the objects. It can also hold additional information such as a perimeter, which the object is not allowed to cross without a notification being generated and a time-fence (for maintenance notification purposes). The VO Registry implements REST APIs and can be queried in a client-server communication mode. Sample REST interfaces to get the information of the VOs are:

https://ziqpos.com:9082/rest/
https://ziqpos.com:9082/rest/devices [this URL provides a list of all the VOs registered in the system.]
https://ziqpos.com:9082/rest/devices/deviceaddress [this URL provides information of a single VO.]
https://ziqpos.com:9082/rest/positions/deviceaddress [this URL provides information of the position of VOs.]

Updates associated with a VO come directly from that VO and these can be dealt with using an MQTT broker (pub/sub communication). As already mentioned, the value of having this architectural feature consists in the flexibility given by the VO Registry which consisted in accommodating a data model where each Virtual Object has a location, as well as a fencing perimeter and a time-fence for notification of various messages related to object (medical device) status. This flexibility feature enables the creation of different services based on different metadata stored. A public MQTT broker implementation based on *Mosquitto* MQTT broker is configured on the iCore server. Sample MQTT interfaces defined along with the payload format is shown in the Table 1. The asset refers to the medical device.

Description	Topic	Response
Request format of the location information	/assetPosition	{ "AssetId":123432432

		}

Response format of the location information	/assetPosition/{assetID}	<pre>{
		"success": true,
		"error": null,
		"AssetId":123432432,
		"Position":{
		"x":10.0,
		"y":45.0,
		"z":1.0,
		"accuracy":0.0},
		"ShowAccuracy":false
		}</pre>

Table 2: MQTT Interfaces to communicate with the VO for smart medical asset management

The creation and instantiation of Virtual Objects happen in the bootstrapping phase when Real World objects are tagged and the tag becomes active. This generates a corresponding Virtual Object to be created, with details stored in the VO Registry. The VO management unit has been designed only to have the role of providing recommendations to the system that deals with the indoor positioning technology. This particular feature however was not core part of the actual trial that was implemented.

5.3 Discovery

VO discovery mechanisms enable the search and discovery of the available VOs in the system. An external entity uses a VO Registry Client to perform discovery requests to the VO Registry. The discovery requests can be structured by a set of specific search constraints that will be taken into account by the discovery mechanisms, making the discovery process faster and more accurate. Next to search constraints, the discovery mechanisms have to consider the Access Rights regarding the VO and the VO Registry client that performs the discovery request. Specifically the discovery request is expressed on form of SPARQL query that includes the search constraints and it is complemented by the User Role and its Access Rights (e.g. read, write, etc.), which respect to the VO Registry client. Since the VO registry client sent the SPARQL request, it is executed by the SPARQL Endpoints that are available in the server side and constitutes part of the VO discovery mechanisms.

With specific VO Container types, the hosted resources can be discovered using only the VO_BASE_URI (which may be queried from the VO registry). This optional type of VO discovery uses the features of Hypertext Transfer Protocol (HTTP) headers to inform the CVO about available resources. This functionality corresponds to Web Linking RFC 5988 [13]. MQTT front-ends can be discovered through the same method by indicating with a custom HTTP Header “X-MQTT-FE: [true|URI]” the existence of a stream for the given resource for streams on local (hosted on VO Containers) or external brokers.

5.3.1 Examples from use cases and trials

The Discovery of the VOs is the cornerstone of the **Smart Office’s** privacy policy. As mentioned in other documents, only the VOs that represent ICT objects that have physical existence in the meeting venue can and should be participating in the iCore system.

The VO registry captures all the information of an ICT object that participates to a VO in our system. Details about the geo-location, features, and ownership can be stored, updated, and retrieved in a quick and robust way. Keeping a VO registry updated enables the system to select the most efficient device to handle requests. Furthermore, the geo-location information enhances the security and

privacy; a VO can be discovered and exploited only when the represented ICT object is located in a specific place and has granted the proper access rights. The VO registry is updated in a seamless manner through the VO Management Unit, which more specifically updates the geo-location property of VOs during the Smart Meeting Guidance.
As such, when subsequent service requests, once a meeting starts, that require a strict geo-location value (e.g. Smart Recording at a specific meeting venue), the last updated geo-location value, which should correspond to the meeting location upon a participant arrival, can be used as some form of System Knowledge by the CVO Management Unit for the proper binding of "location suitable VOs" to a corresponding CVO logic engine.

In the **Smart City Transportation** case, the discovery of the most suitable VO to be used by the CVO is performed through the integration of the VO Registry API: the Parking VO is discovered at runtime based on the location of the Driver (SPARQL query) and the address is provided to the CVO Execution Engine. This allows to different Parking Systems to be used, potentially provided by different City Infrastructure managers, and selected to be used according to the Driver's context (the city the Driver is actually located).

In the **Urban security** case**,** automated discovery of sensors and actuators over the network (aka devices Plug n Play) and their corresponding VOs is of outmost importance because surveillance system deployment has to be as quick and user friendly as possible without complicated configuration.
As introduced in section 5.2.1 above for urban security case, we have two abstraction layers at VO level with southbound and northbound interfaces. *The discovery process is a multiple steps process from sensors and actuators up to CVO level.* Typically, a day before the VIP visit surveillance system set-up is started with deployment of a few wireless communication nodes that will be used as network backbone; location of the wireless communication nodes depends of the VIP visit path and must provide enough coverage for the entire mission. Then connection is achieved with wired sensors already deployed within exhibition area (chemicals sensors and CCTV network); for this, one dedicated wireless node is plugged into a security forces restricted and hidden plug; thus the wireless node serves as a gateway between wired and wireless network. The resulting network is connected to a Control and Command (C2) police truck standing in the area. During these first steps, each new wireless communication node entering the network generates a notification up to the VO layer that initiates discovery process. First a request is sent to the communication node that provides back some meta-data about its connected sensors (if any); these information further described in next sections are description gathering low level network information (IP addresses), logical name (URN), specific data (e.g. area coverage for video camera), GPS location and capabilities such as commands supported. Information is formalized as a new VO instance executed by the VO container and registered in VO registry to make it available to CVO level for further request about available devices.
The day of the VIP visit, few undercover policemen bringing wireless video camera and chemical sensors as well VIP bringing wireless GPS initiate the same discovery process. Last step is the VO discovery initiated by the CVO management unit to set-up or reuse CVO instances. The discovery uses two OGC operations; first operation supports pure sensors discovery to provide sensors geo-location, observation properties, type of observations, etc.; second operation enables sensors *readings discovery based on semantic filtering* reusing previous sensors and observations related criteria.

For the **Smart business** use case, the VO Registry is implemented as a SPARQL-endpoint and keeps track of the relation between sensors and actuators (ICT objects), trucks and other facilities (non-ICT objects), and their virtualization within the iCore framework (VOs). Each entry in the VO Registry contains a subset of the VO model as described in deliverable D3.2 [17]. This allows other

components to search for specific non-ICT or ICT objects e.g. based on a specific location. *Semantic discovery is used in this use case to establish a relation between - for instance - trucks and the location of goods.*

The VO Registry also contains information on how to obtain results from VO functions e.g. how to obtain temperature readings or how to control a HVAC in a warehouse.

In the **Smart Medical Asset Management** Trial**,** the VOs are represented by the medical devices which are tagged with the positioning sensors from ZIGPOS. In addition there are the anchor nodes which are deployed along the perimeter of the area where the devices are being monitored. These anchor nodes are responsible for triangulating the position of the medical devices. All ICT-objects are registered on the system at the time of deployment. The VO discovery process is twofold. The first is the discovery of all the available VOs registered. This information of the list of devices can be retrieved by RESTful commands. Also the system information e.g. power modes of the devices can be retrieved via REST. The APIs thus define the URIs, the payload formats and the security certificates required to connect and extract information from the VO. In addition, the information from the positioning system i.e. the location updates of the devices are provided through MQTT. Here again, the address of the MQTT broker and the topic and payload formats are configured. The second part of the discovery process is when the CVO management unit looks-up for the required VOs. All the information regarding the VOs is made available in the VO registry and in the local database facilitating easy SQL based queries. Using the device address and the APIs/topics, all the required information from the VO can be retrieved. The initial deployment thus exposes the interfaces/ topics which allows the search and discovery of the VOs, as and when required by the CVO management unit. In the trial, the access to VOs is thus limited to the interactions via REST and MQTT.

5.4 Execution

Execution is achieved in the run-time environment of VOs, including deployment and installation aspects. VO Container is the core component, acting as the abstraction component between the heterogeneous physical world and the (more) homogeneous iCore CVO level. The key interfaces of VO Container are the VO front-ends, which provide the semantically enriched data from the VO back-ends in either a “future-proof” request-response HTTP REST interface, or a publish-subscribe MQTT interface. VO Containers may incorporate VO management functionalities for updating information such as positioning etc. in the VO registry during run-time to enable accurate metadata based discovery of VOs.

5.4.1 Examples from use cases and trials

The VO Container is introduced in the **Smart Home** PoC, aiming to facilitate the easy and dynamic, automated deployment of heterogeneous VOs in the system. Each VO includes a software part that implements its functionality and it should be hosted by a relevant infrastructure to be executed. This role is assigned to the VO Container, which take over to host and realize the execution of the available VOs. Through this powerful component, we achieve the integration of the heterogeneous objects, and enable their interconnectivity, under a CVO composition, without the need to create customized solutions or domain-oriented software modules that should be updated each time to support a new VO with different specifications. During the development of Smart Home PoC the MQTT front-end and cloud based VO deployment (VO Front-ends hosted at Amazon Web Services [19] and back-ends via a SSH pipe on local laptops and sensors) have been tested with success.

Even if technological detail, it should be highlighted that the VO Container can host any REST-based S/W module and for this reason except the Smart Home PoC, it has been applied in the **Smart Tourism** Trail, so as to host the involved VOs in this use case.

The Java based VO container for android devices used in the **Smart Office** use case, offers certain advantages. First of all, being implemented using open-source platforms makes it suitable for a wide range of devices of all budgets. Furthermore, taking under consideration the spectrum of potential host-devices, it offers a responsive UI that enables interactions. All the available functionalities are presented in a very clear yet discreet way without compromising the user experience. It also:

- Serves as a platform to inform the VO owner by using the notifications of the ICT device.
- Allows the system to exploit the hardware resources in a scalable and efficient manner
- Since the communication relies upon lightweight protocols (MQTT) in terms of energy demands it makes it suitable for the battery constrained nature of the mobile devices.

5.5 Security

For specification and enforcement of the security policies at the different layers of the iCore architecture, we designed and implemented the Model-based Security Toolkit (SecKit). The SecKit foundation is a collection of meta-models that provides the basis for security engineering tooling, add-ons, runtime components and extensions to address security, data protection and privacy requirements. The SecKit meta-models support the integrated modelling of data, time, identities, roles, context, structure, behaviour, trust, risks, and security policy rules. At VO level, the specification of security policy rules is done using parameterized rule templates, which can be used to specify authorizations and obligations. These rules templates follow an Event-Condition-Action (ECA) structure, when the event (E) is observed, and the condition (C) evaluates to true, the action (A) is performed. The event part is an event pattern, which may be VO activity (action or interaction), context information and situation events, and lifecycle events of VOs and data instances. The condition part of a rule template consists of event pattern matching, propositional, temporal, and cardinality operators.

5.5.1 Examples from use cases and trials

Amongst all current iCore use cases and trials deployed, **smart business** case is the one that report the most significant and mature experience about security at VO level. In that case, SecKit technologies have been integrated as an extension to MQTT broker (one of the two possible front-ends for VO container). The following figure shows the design of the MQTT and smart business security behaviour model using the SecKit GUI. In this model, we specify among others the behaviour types, their respective actions and interactions. We model the interactions between the MQTT components and the specific actions required by the scenario, for example, **actions that evaluate who is the company responsible for a particular shipment**. The behaviour model is one of the required models used in SecKit to support the design of the security policies.

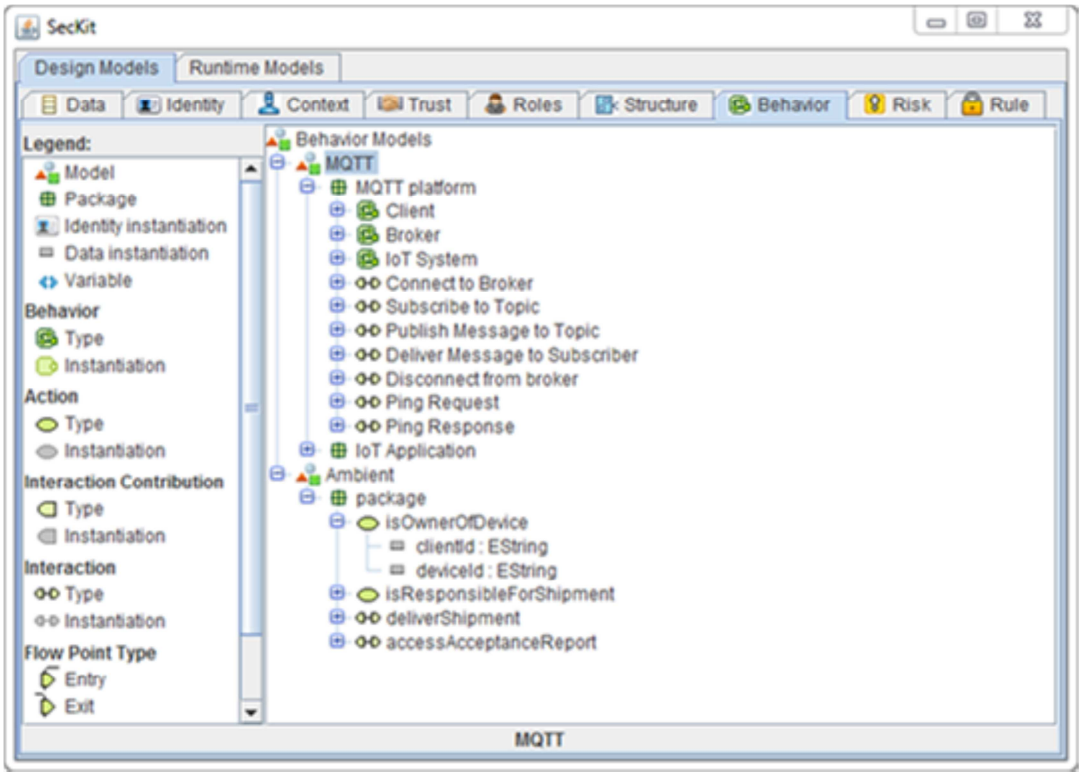

Figure 24: Behaviour model specification in SecKit for smart business scenario

The following figure shows the design of the security policies specified for the smart business scenario in the SecKit GUI following the Event-Condition-Action structure. The first policy specified states that an MQTT client can read the data of a device if the client is the device owner. Reading data is captured by the event generated by the MQTT broker when a message is delivered to a subscriber, and the check of ownership is done in the condition of the rule. The specified policies are enforced at the MQTT broker, and details about this technology developed by us including a performance evaluation of the policy evaluation and enforcement engine are provided in the following publications [15][16].

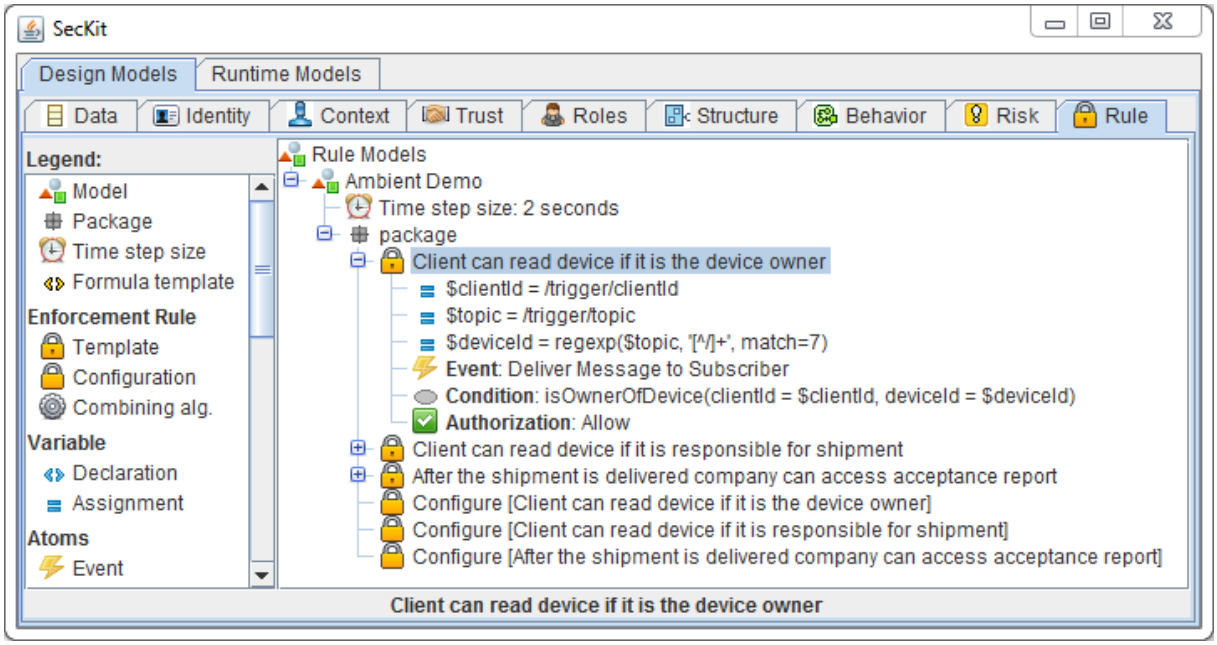

Figure 25: Security policy rule model

5.6 Lessons learnt

iCore VO level is responsible for abstracting the highly heterogeneous physical world using semantic technologies. This approach is a contemporary way of solving interoperability issues close to physical levels. At the start of the project, we faced a dilemma with a large number of communication technologies to choose from and use as the baseline for **VO implementations**. During the course of the project, several alternative solutions to addressing the VO level interfaces were proposed. It was quickly realized that a classical HTTP REST based request-response type interface was not optimal for representing all kinds of interactions at the VO level. This lead to a search for a "publish-subscribe" protocol which could also supports actuator and stream based interactions. The most feasible protocol was MQTT(-S), which enables low latency event based communication without the need for polling. An important lesson learned was that SOAP-style interfaces are not very feasible for Internet of Things usage and modern approaches almost always claim to be built on "future-proof" RESTful interfaces and additional publish-subscribe protocols for handling event streams.
Although it was not an explicit goal of the project, we get quite close to inter-use case interoperability of VOs. Full out-of-the-box interoperability is not always reached due to the fact that the CVO level would have to support the various types of VOs, but approaches related to semantic interoperability and translation between ontologies were researched in the first year of the project (and reported to IERC)[20].

The concept of **VO factory** and **the deployment of VOs** are described in general in D3.3 [18]. A general proof-of-concept for VO Deployment was not reached due to the large number of different programming environments in use, but the design principles of the VO Factory have been followed by various partners in use cases and pilots.

VO discovery was another research topic at the middle of the project. **VO registry API**, **VO Information Model** and **VO Naming and Addressing** scheme for VOs are the key for solving interoperability issues at the VO level. These features promote component modularity and separation of concerns, both very important aspect of distributed architecture design. Two compatible styles of VO discovery were created, both relying on description registered in the VO Registry. The "fine grained" method uses SPARQL to discover specific VO functions from the VO registry, whereas the web based approach may use the VO registry to discover the VO base address and further discover resources via these addresses. Both of these approaches were tested successfully during the project.

In the design and development of the SecKit we observed that it is of fundamental importance to associate the design models of the system with the security design models specifying the policy rules. This association made it evident the selection of the **Policy Enforcement Point** (PEP) locations and it allowed us to have a clear view of the meaning of the specified policies. A simple analysis of the policy rules and associated behavior model explicitly specifies which execution request is the target of enforcement. Furthermore, using our implementation of the SecKit runtime components we learned that from a performance point of view it is feasible to implement fine grain security policy rules. Both our policy rule engine implemented in the **Policy Decision Point** (PDP) [15] and MQTT PEP [16] showed the capability of handling a big number of policy rules, event subscriptions, and enforcement interception with a very low additional delay in the delivery of the messages to the VOs.

In conclusion, this level was proved less focused on cognition than initially considered (e.g. VO is not dynamically created as existed in DoW, but provided by an iCore actor). However, efforts have been required to be done in other directions, such as the VO design itself (VO front end and VO back end)

with appropriate naming/addressing scheme, the VO information modelling, the VO execution environments and the VO registries. Since many IoT projects reside in this level, it may enable many synergies with other projects.

6. Conclusions

In this document we have reported the final iCore architecture reference model initially set-up in deliverable D2.3. The model has been refined according to implementation work and validation feedbacks from other technical Work Packages 3, 4, 5, 6 and 9. So, this document illustrates the resulting final iCore architecture reference model through lessons learnt and selected examples taken from iCore use cases and trials.

In chapter 2, we have reminded first (section 2.1) the ***main aim of the iCore architecture as a framework which endows ICTs with cognitive mechanisms***, making a leap from the pure Internet of Things domain.
Most of the times the cognitive mechanisms described in academic papers are detached from a real world environment: one imposes high abstractions, and the environment is often conveniently represented or removed at all. The most popular assumptions is that the data which are used for machine learning training and model choice are ready to be used for learning and prediction (e.g. data are already time-aligned and multiple data sources are conveniently joined); additionally, the normal application life cycle – as accepted in software engineering - is often overlooked for Machine Learning model deployment and maintenance. To make real the iCore proposals and to confirm that the architecture allows *cognitive IoT* with mashing of ICTs and customized cognitive models for several different business domains, we have implemented this architecture through a set of use cases and trials.
The remainder of section 2 highlights the **main iCore architecture results** with an introduction to the use cases and trials, the contributions they pushed to the architecture and the benefits they pulled from the architecture.

Next to the chapter 2, chapters 3, 4 and 5 make concrete the iCore architecture results with selected examples from use cases and trials, respectively focusing on main features at Service level, CVO level and VO level. For each level, feedback and lessons learnt are provided identifying improvement of the iCore architecture and further research towards the field of cognitive IoT.

First cognition innovation related to architecture is supported by the iCore ***Service Level*** leveraging **Real World Knowledge (RWK) modeling** and use of these domain specific models at run-time; the goal is about making the system aware of Real World situations to support reasoning and various domain problem predictions, so the *effective benefits of RWK in the platform are apparent through system effectiveness and efficiency improvement in all use cases and trials***.**
Certainly further research should be concentrated on ***RWK models training*** during execution time as a step towards truly autonomously self-enhancing systems.

The ***Composite Virtual Object Level*** is involved in the ***cognitive cycle*** in the sense that all complex computation regarding state sensing and predictions, both for Real World and for System World, are all achieved at this level by specialized Situation Observers. If system performance is in general higher, major drawback of this end-to-end cognition approach is a cross-layers coupling between CVO level (RW situations Observers) and Service level (RWK) that reduce independent reuse opportunities. Ultimate system effectiveness and efficiency may only be reached through the ***combination of RWK and SK situation awareness*** (e.g. smart medical asset management for energy saving and smart urban security for WSN optimal use); SK providing additional “guidelines and constraints” to the cognitive cycle. Although several cases have been studied in the project, this has not been sufficiently matured yet in iCore and requires further feedback from specific business domains as well as from further software architecture iterations to ensure fully horizontal applicability.

During the project, it was clear that the ***Virtual Object Level*** focused more on ***interoperability*** concerns than cognition. Efforts have been developed to select, adapt and integrate a minimal set of existing proven solutions, typically:

- For VO ***naming and addressing***, approaches similar to the World Wide Web, namely Universal Resource Identifiers, URIs have been used with several naming schemes to accommodate business domains requirements.
- A VO ***Information model for IoT actuators and sensors*** has been experimented and mapped to the IoT-A architecture reference model and SSN sensor ontology; variants such as OGC based semantic models have also been used. Main point here is that formal compatibility of all models is still not there; iCore would benefit with further research related to automated models transformations bridging them all since it seems the most reasonable short and maybe mid-term solution.
- The VO northbound interface supports ***HTTP-Rest and Publish-Subscribe communication protocols***; these have been adopted by all use cases.

Regarding ***security***, successful integration and efficient fine-grained use of authentication at VO northbound interface demonstrates clearly the benefit but much more work is required to confirm that iCore architecture offers true and global security by design; so additional work:

- Towards CVO level and Service level, that takes into account cognitive loop and possible distribution of the VO containers.
- Towards the VO back ends to secure connections between an iCore platform and IoT devices networks.

Whatever the architecture level, two lessons learnt are noticeable: first lesson is that availability of iCore templates (VO, CVO and Service) as a "market of templates" may be beneficial either dedicated to a particular domain or more generally publically available. Such public (or domain expert community) exposure would accelerate adoption and boost iCore architecture maturity. Second lesson is that implementation and execution of several use cases and trials dedicated to specific business domains have shown that universally applicable languages for expressing RWK/SK, ST/SR and other 'content' are difficult to specify universally in practice; one possible option for future research therefore can be automated models transformation bridging the gaps between the different languages used, for which some methods are known.

Looking to the way iCore architecture would benefit from ***Cloud Computing*** and highly distributed processing landscape, ***externalization of (some of) the cognitive mechanisms as well as diffusion knowledge*** (both RWK and SK) is relevant.
At the time of writing, we witness an increasing market of "ML as a platform": Google Prediction API[21], Azure ML[22], BigML[23], and IBM Watson ecosystem[24], all being cloud-based. The natural result is that the architecture should include more 'interworking' gateways between iCore systems. It is however expected that specialized cognitive services to be implemented and offered by specialized vendors and in this scenario a particular iCore instance might be open to consume them.
iCore effort started in 2011 together with the booming advent of what is known today as analytics. In term of instruments, the vision of cognitive technologies is confirmed by Big Data wave empowered with Data Mining and Machine Learning techniques. Technologies founded currently in Data Centers migrate now to mobile cloud offering intelligence close to the process. In this view what we consider Real World Knowledge and System Knowledge are used and maintained in the IoT systems edge, and communicated along distributed nodes on demand. There is an opportunity here to generalize the iCore concepts, more explicitly into this distributed cloud setting.

Finally, **all iCore use-cases and trials have made clear use and demonstrated the value of the iCore components and architecture**, even if not in the same measure because of their own business

requirements and constraints and the diversity of these; so the positive generalization of iCore architecture is a reality.

7. Appendix: iCore architecture components sequence diagrams

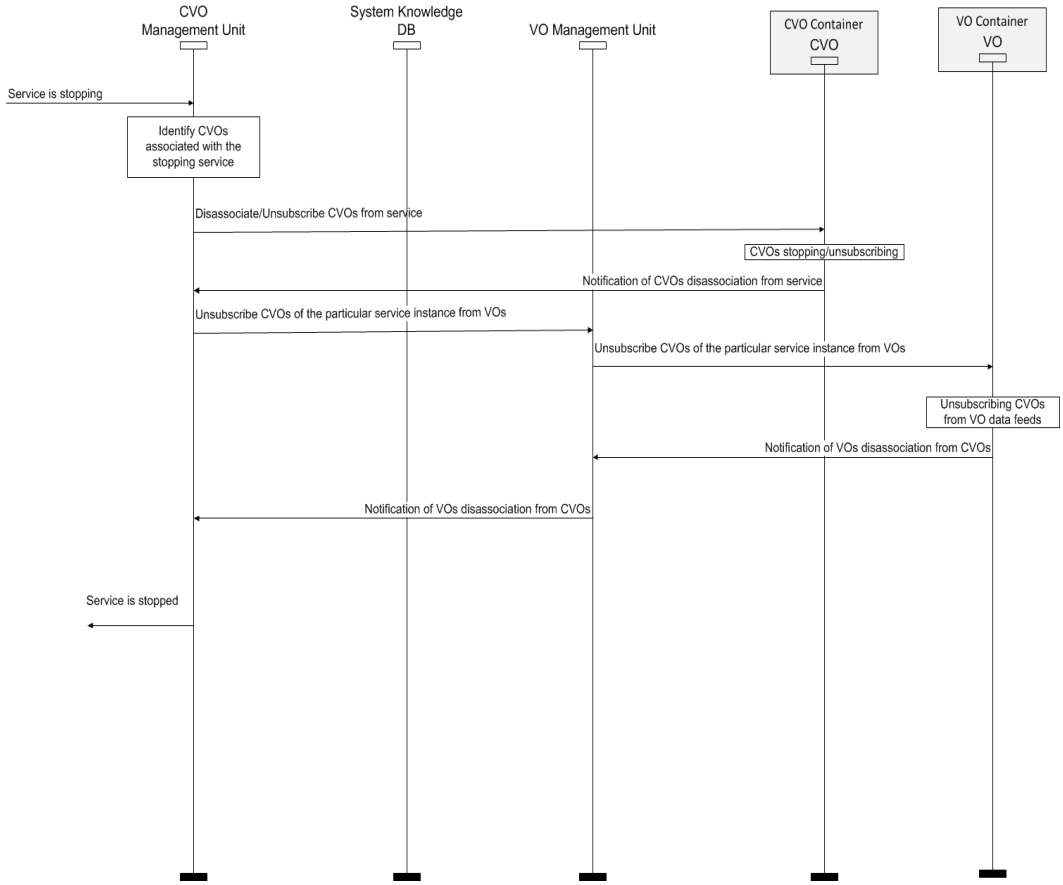

Figure 26: CVO dynamic creation

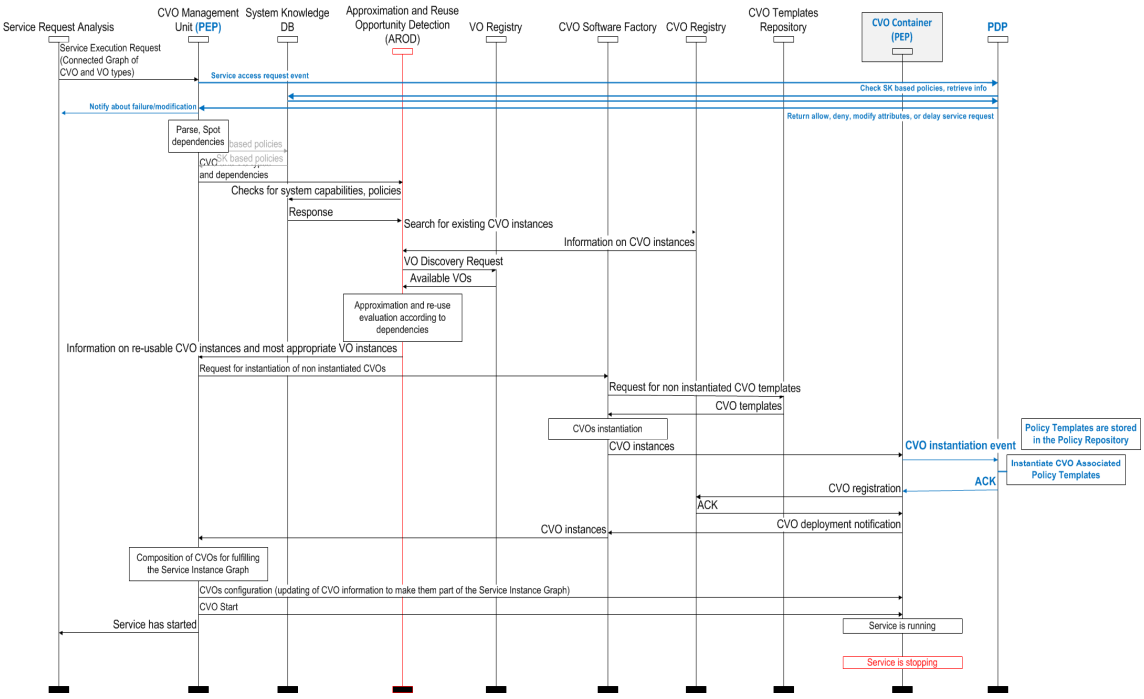

Figure 27: CVO execution reuse and access enforcement

8. References

[1] iCore Deliverable 2.3 (D2.3), Architecture Reference Model, June 2013.
[2] iCore Deliverable 3.4 (D3.4), Virtual Object proof of concept, August 2014
[3] iCore Deliverable 4.4 (D4.4), Composite Virtual Object management, August 2014
[4] iCore Deliverable 5.2 (D5.2), Final version of the "Application knowledge inference toolkit", February 2013
[5] iCore Deliverable 5.3 (D5.3), Full specification of iCore Service Level cognitive management and control mechanisms, November 2013
[6] iCore Deliverable 5.4 (D5.4), iCore Service Level cognitive management and control mechanisms proof of concept and validation, June 2014
[7] iCore Deliverable 6.3 (D6.3), Final proof of concept prototype for Smart Office: easy meeting, May 2014
[8] iCore Deliverable 6.2 (D6.2), Final proof of concept prototype for Smart Home: Living Assistant, May 2014
[9] iCore Deliverable 6.4 (D6.4), Final Proof of Concept prototypes for Smart City – Transportation and Urban Surveillance (security) Use Cases, May 2014
[10] iCore Deliverable 6.5 (D6.5), Final proof of concept prototype for Smart Business, May 2014
[11] iCore Deliverable 2.4 (D2.4), iCore System Governance Model, March 2014
[12] iCore Deliverable 6.6 (D6.6), Report on the validation of the final iCore prototypes, September 2014
[13] IETF RFC 5988 Web Linking, October 2010
[14] [4-TL] Sinno Jialin Pan and Qiang Yang, A Survey on Transfer Learning, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 22, NO. 10, October 2010
[15] Neisse, R.; Steri, G.; Baldini, G. Enforcement of Security Policy Rules for the Internet of Things. 3rd International Workshop on Internet of Things Communications and Technologies (IoT-CT), in conjunction with The 10th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2014;
[16] Neisse, R.; Nai Fovino, I.; Baldini, G.; et al. A Model-based Security Toolkit for the Internet of Things. International Conference on Availability, Reliability and Security (ARES), University of Fribourg, Switzerland, 2014;
[17] iCore Deliverable 3.2 (D3.2), Real Object Awareness and Association, May 2013
[18] iCore Deliverable 3.3 (D3.3), Virtual Object Concept Definition and Design Principles, May 2014
[19] http://aws.amazon.com/fr/
[20] IoT Semantic Interoperability: Research Challenges, Best Practices, Solutions and Next Steps- IERC AC4 Manifesto - "Present and Future"
[21] https://cloud.google.com/prediction/
[22] http://azure.microsoft.com/en-us/documentation/services/machine-learning/
[23] https://bigml.com/
[24] http://www.ibm.com/smarterplanet/us/en/ibmwatson/ecosystem.html