iCore

Internet Connected Objects for Reconfigurable Ecosystem

Grant Agreement Nº 287708

D3.3 Virtual Object Concept Definition and Design Principles
WP3 Virtual Object Management

Version: 1.0
Due Date: 30.11.2013
Delivery Date: 9.5.2014
Nature: PU
Dissemination Level: CO
Lead partner: CREATE-NET
Authors: Matti Eteläperä (VTT),
Abdur Rahim (CREATE-NET),
Dimitris Kelaiodonis (UPRC),
Vasilis Foteinos (UPRC),
Michele Nati (UNIS),
Stylianos Georgoulas (UNIS),
Riccardo Pozza (UNIS),
Lucian Sasu (SIE),
Chayan Sarkar (TUD),
Ricardo Neisse (JRC),
Andrea Parodi (M3S)
Internal reviewers: R Venkatesha Prasad (TUD),
Matthias Fath (TNO)

www.iot-icore.eu

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

Deliverable Title:	Virtual Object Concept Definition and Design Principles
Deliverable Number:	D3.3
Keywords:	Virtual objects, semantic enrichment, Internet of Things

Executive summary:

The purpose of this document is to present specific Virtual Object level mechanisms and functionalities for implementing the VO level of the iCore architecture. This work is a continuation of the requirement gathering phase and the VO concept definition done earlier in the project. The final outcome of this work in the scope of WP3 will be reported later on in task 3.4 where a number of mechanisms described in this document will be implemented as proofs-of-concept.

Virtual Objects are abstracted, semantic descriptions of ICT objects and their functionalities and associated non-ICT objects. In this document we present the life-cycle of VOs: how VOs are created, deployed, used, updated and destroyed. In iCore the Composite Virtual Object level is the entity using VOs, so one of the key sections of the document discusses the alternatives for creating interfaces for the clients of VOs.

Along with these inputs, we also present novel ideas on how to include cognitive technologies even at the VO Containers – the run-time instances of Virtual Objects. These technologies include machine learning for sensor values and run-time re-configuration of the components.

Table of Contents

1 Introduction

1.1 iCore architecture in brief

The iCore architecture is presented in depth in work package 2 deliverables D2.2 and D2.3. Virtual Object (VO) level is the entity in iCore project responsible for serving real world information to upper levels in a semantically described format. The level is depicted at the bottom of Figure 1. Virtual object abstracts necessary real-world information and functionality for the use of the Composite Virtual Object (CVO) level seen in the middle of the figure. This level is responsible for acting upon the service execution requests coming from the Service Level (SL), and choosing the right VOs for implementing these requests. This level involves the term System Knowledge (SK), which is the status of the iCore system and its entities. On SL we have Real-World Knowledge, which is a database of "slowly" growing real-world facts, generated by testing hypotheses by data flowing from the lower levels. The main user interface (as seen on the top of the figure) to iCore is a natural language processing (NLP) module along with a graphical user interface. Authentication system cuts vertically across all levels and the implication for VOs are discussed later in the document.

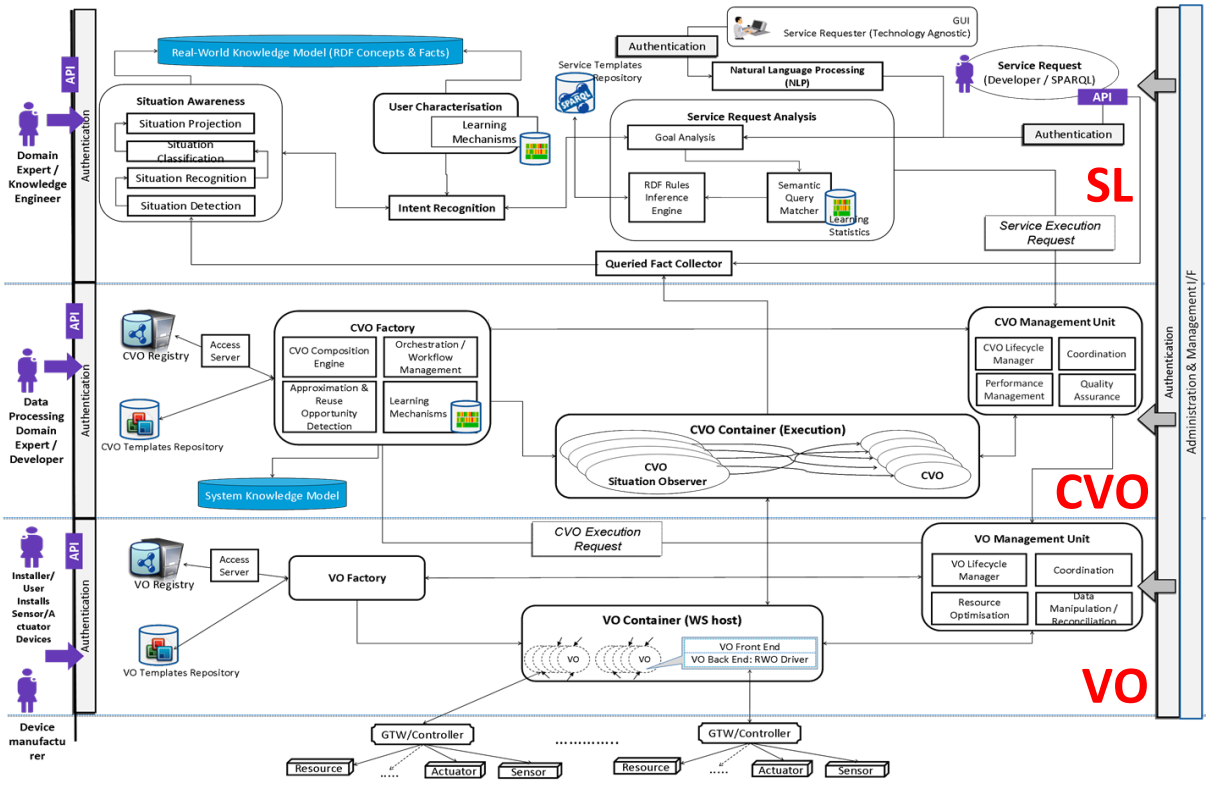

Figure 1. iCore architecture.

1.2 Purpose of this document

This document presents Virtual Object level components used for creating and managing associations to real objects. Essential mechanisms for exposing relevant real world and system information to upper iCore levels are presented. A key feature of the VO level is to semantically enrich data so that the context of it is relayed to the clients accessing and using it. These semantic descriptions of Virtual Objects are known as VO Descriptions, and they are

stored in VO registries for discovering ICT, non-ICT and Virtual Objects, their associations and available interfaces.

A suitable description for the VO has been agreed in Deliverable 3.2, but it is further refined and extended in this document. The aim is to promote opportunistic interoperability by describing the VOs semantically rather than explicitly choosing a fixed interface between the client and the VO. This also enables inherent extensibility of the type of real world information a VO may host. Being able to interact with VOs enables the creation of VO mash-ups, Composite Virtual Objects.

VO design, deployment and creation are also discussed in this document and relevant mechanisms to facilitate these functionalities are presented. Registration of VOs is a key mechanism in the design phase of VOs. In order to de-centralize the discovery repository in iCore we present insights on implementing VO registries in a distributed manner.

VOs may be actuators in addition to the typical sensor-VOs and this leads to additional requirements for access control. Access rights and policies for using information hosted by VOs are presented, as this is a very important aspect of the iCore platform and Internet of Things as a whole. We present a view on the life-cycle of VOs and VO states which are closely related to the concept.

Trust on the information provided by VOs can be evaluated at the client side by analysing time stamps generated by the VO back-ends.

Cognitive technologies in iCore are mostly suited for the upper levels (CVO and Service Level) of the platform, but we also present the means to add fine-grained cognition on the VO level itself. These include for example sensor value outlier detection through machine learning.

In addition, the following issues are addressed in this task:

- Identification, naming and self-advertisement capabilities of VOs
- How to make the virtual objects smart
- How to notify the change in VOs
- How someone can change and modify VOs
- How to generate domain knowledge and expose it through VOs
- Who can access and change VOs and how and what to share
- How to register or locate the VOs
- Learning events from VOs, event correlation and event prediction

The tangible outputs of this task are a set of mechanisms for virtual objects creation, semantic meaningful descriptions, as well as the development of algorithms for controlling and managing the VO.

1.3 Structure of this document

This document is structured in the following way. In Section 2 we present the key specification of the Virtual Object level. In section 3 similar specifications of VO Registries are presented, followed by the information about VO interfaces – especially towards the CVO level. Section 5 presents VO level privacy and access control mechanisms and section 6 discusses the necessary steps for VO deployment utilizing the VO factory. Section 7 presents the VO control and management unit functionalities and section 8 proposes a set of VO level cognitive and machine learning mechanisms. Finally section 9 wraps up the document.

Appendix A shows message sequence charts generated in WP2 which have relevance to WP3. Appendix B presents machine learning example code and Appendix C an example of a Web Service Description Language (WSDL) description for automatic generation of client-side interfaces for VOs.

2 Virtual Object Specifications

This section introduces the specifications of the VOs in the iCore system. A set of explicit requirements is defined through the VO specifications. These requirements should be satisfied by VOs, in order to be compatible with the system. The VO specifications focus on the semantic description of the VO and its development as a software agent.

2.1 VO Semantics and (meta-) data modelling

The semantics are introduced so as to support the semantic description of the available VOs in the iCore system. As already introduced, the description of the VOs is supported by the VO Information Model (i.e. VO model). The VO model (Figure 2) includes different types of meta-data (e.g.: VO Parameter, URI, Name, Feature, et cetera) that is used to specify and describe diverse types of properties that can be associated with a VO. The VO model will constitute part of the iCore Information model, which in its turn will include relevant information for the description of the different iCore entities (e.g. Service, CVO, et cetera).

The VO model, as it is defined in the deliverable 3.2, is structured by a set of different properties, which are defined as meta-data containers. Actually, the meta-data container constitutes the entity which includes different properties that describe a specific concept. For example the VO Parameter is a meta-data container, which includes properties such as the URI, the Name, et cetera, which constitute the meta-data that describe the VO Parameter. The meta-data containers can support the description of a wide range of possible devices, which can be part of the iCore system. The Semantic Web technologies constitute the key concept for the satisfaction of the VO model requirements. More specifically, through the exploitation of the ontologies it is possible to develop a generic VO model which is capable to include different types of meta-data for the description of the available VOs. The use of external ontologies facilitates the description of the heterogeneous features that can be associated with a VO, since it is possible to develop ontologies that contain specific or more generic and reusable meta-data.

Essentially, there is a generic structure for the VO description that is complemented by specific features defined by the external ontologies. Additionally, by using the Resource Description Framework (RDF) capabilities, it is possible to structure the different meta-data types, so as to

structure a specific meta-data model, which describes the corresponding VO. The specific model, which arises as the final result of the VO modelling, includes the VO semantics that in turn is stored in the VO Registry, making the VO discoverable in the iCore system by external entities.

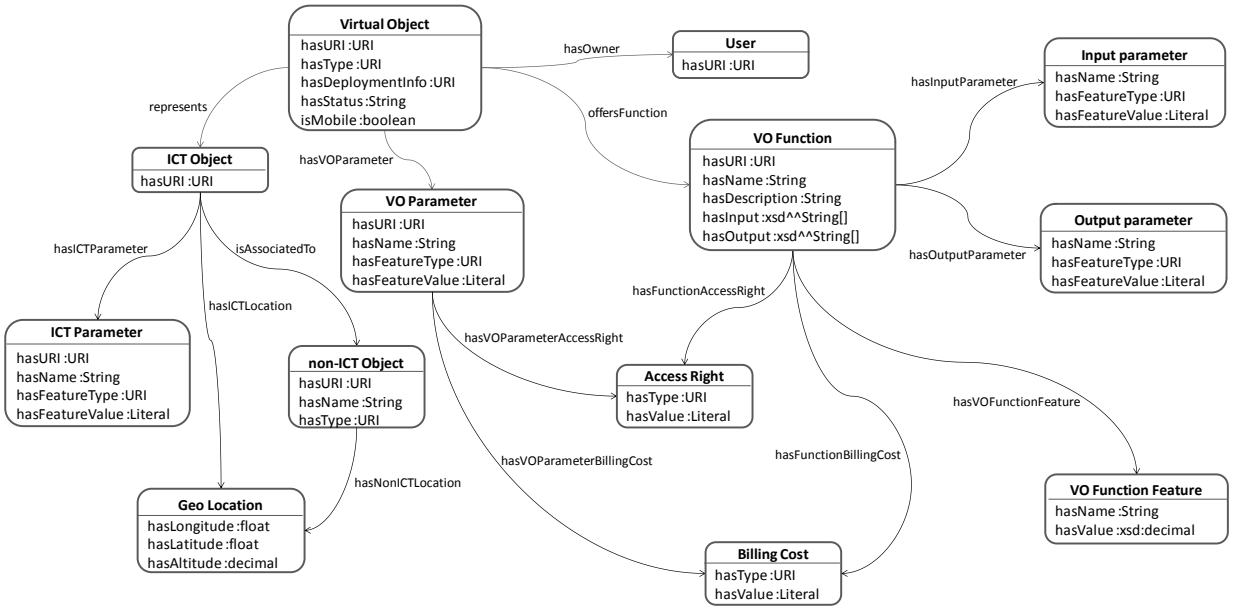

Figure 2. VO Model

Since the VO description is stored in the VO Registry, it can be used in different ways so as to allow various VO management processes, such as the VO discovery, selection, availability, accessibility, etc.

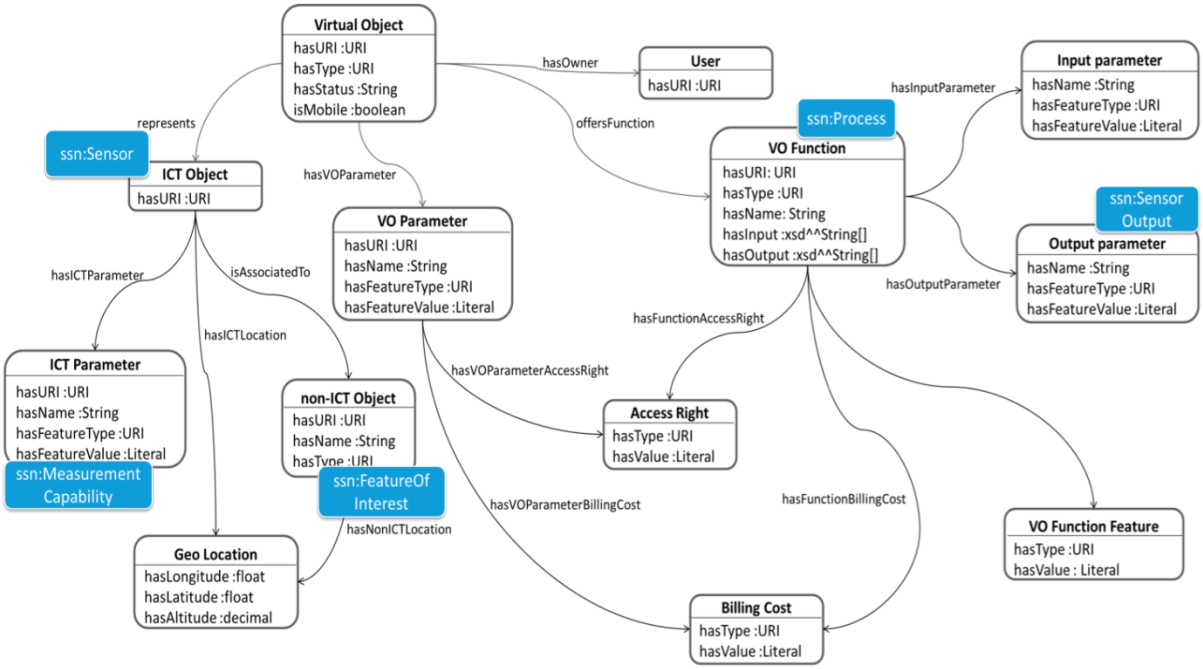

Figure 3. VO Model SSN Mapping

Figure 3 shows a mapping of the current VO model to existing ontologies, such as Semantic Sensor Network (SSN). As the above figure shows, for a specific VO instance, the ICT Parameter in the VO Model specifies the Measurement Capability of a Feature of Interest, represented by the non-ICT Object. However in order to support additional VO management process, such as discovering a VO based on the manufacturer information, type of available sensors and similar more "sub" ICT object parameters should be linked to the VO model.

In order to do this, the meta-data container provided by the ICT Object can be extended with external ontologies, by making use of its *hasURI* property. Using a linked data approach, the ICT Object properties can be specified through the URI pointing to the corresponding Device model. To this purpose, an adequate existing Device model has to be identified that can be used for inferring system knowledge by the VO management Unit and the CVO Management Unit. Currently the ICT Object can be naturally seen as *ssn:Sensor* class in the SSN ontology, however representing an ICT Object only as a concrete sensing object can be too limited, especially in the case when the considered sensing object is part of a more complex platform that can host different sensing objects, such as a Smartphones as envisioned in many iCore scenarios. To this purpose, additional knowledge about the system should be modelled and based on this it will naturally look more adequate to link the ICT Object to the *ssn:Platform* class.

While there have been already attempts in developing XML- and RDF-based device descriptions for sensor networks; however the results are still limited when considering the device heterogeneity envisioned by iCore. Examples of these efforts are the Sensor Model Language (SensorML) [5], the Semantic Sensor Network specification of W3C [6] and OntoSensor [7]. These efforts mainly focus on the sensing capabilities and observations measurement of a given device, therefore, their application remains limited on how to expose sensors as service endpoints in web service architectures.

In contrast, a recent effort by Nati et al. [8], instead of proposing an ontology model from scratch, proposed an extension of the recent W3C ontology on Semantic Sensor Network [6] by specifying further details pertinent to Internet of Things devices. The model has been originally designed to describe and support discovery of devices for IoT test beds. However, similarly to the case of experiments running on test bed devices (for which devices satisfying desired experimentation requirements need to be discovered) in the case of service provisioning VOs with specific ICT Object characteristics need to be identified in order to support the required services. In both the cases, a model for the same envisioned type of IoT devices is required.

The device model is showed in Figure 4. In order to leverage the increased machine processing capabilities of the growing eco-system of the semantic web, allowing an M2M communication without human interaction, one of the design decisions has been to semantically describe devices. Instead of reinventing ontology from scratch for the device model, existing state of the art ontologies in the sensor network and IoT domain have been carefully evaluated and the Semantic Sensor Network (SSN) ontology [13] has been selected as a starting point. SSN ontology has been extended with concepts that are pertinent to IoT and sensor network devices, including the most critical information that a test bed/service provider may require for an adequate selection of devices. Figure 4 provides an overview of the key concepts of SSN highlighted in grey while the extensions proposed by Nati et al. are represented in light grey. According to the model, the ICT Object that is part of specific VO instantiation can be seen as a specific instantiation of the enhanced *ssn:Platform* class. This makes the proposed device model suitable for extending the current VO model.

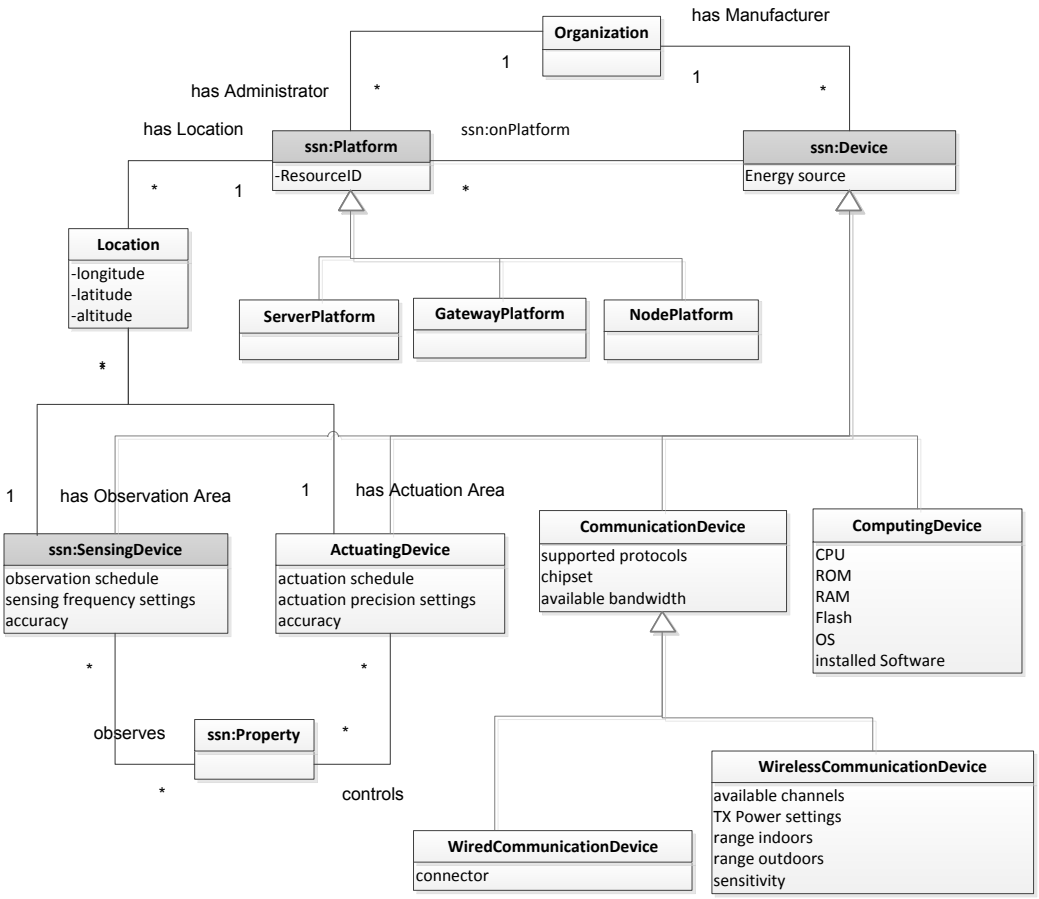

Figure 4. IoT-A device Model

An IoT device can be seen as *ssn:Platform* (class Platform of SSN ontology) featuring many *ssn:Device*. The proposed resource model takes into consideration the three possible device tiers that compose existing IoT architecture, by sub classing platforms for IoT node tier, gateway tier and server tier from the original SSN platform concept. A platform is deployed at a particular location, which has implication for most of the different devices that it hosts. Devices that are attached to a platform can be computing and communication devices and for the IoT node platforms often sensing and actuation devices. Communication devices can be wired or wireless. Sensing devices observe properties of their surrounding environment, which are constrained to a particular observation area. Likewise actuators may influence properties within their respective actuation area. This is an improvement over the pure SSN ontology that was only able to specify sensing devices.

For each of the concepts a variety of attributes have been defined that may be of relevance to the service/test bed provider for device selection. Specific service SW, e.g. a VO container, is often developed based on a particular operating system, e.g. Android and may have certain requirements on processing capabilities or available memory of the underlying node platform. It is therefore important to know whether there are devices supporting the execution of the required VO container. Some VOs may require specific sensing capabilities and/or cover specific geographic locations with these. In other cases VO selection may have a specific interest in communication devices available at IoT and GW nodes and what settings are possible for their configuration during execution, i.e. in order to make the VO container

accessible. It is important to note that this proposed device model is a starting point and could evolve as new hardware or software features become available or new service use cases will emerge. While some of the attributes are static and related to the associated ICT Object, some other can be updated at real-time and depend on specific device settings or the application of specific resource management procedure, i.e., available CPU, ROM, RAM, et cetera. Additionally other relevant attributes can be added to the existing properties already modelled that should comprise already all the most important properties of an ICT Object. Moreover, some redundant concepts existing in both the VO and device model can be identified, such as location that overlaps with the Geo Location concept and the *ssn:Property* representing the Features of Interest of a particular Sensing and Actuation Device. These concepts can be either referenced by the VO model or be instantiated when a specific VO instance is created.

2.2 VO Templates

VO Template is the main concept for the dynamic development of a VO, by following appropriate specification both for its semantic description, as well as for its implementation as a software agent. The VO Template is comprised by two different parts; (a) the VO Type and (b) the VO Code (Figure 5). The VO Type corresponds to the partial description of the VO base structure as it is provided by the ICT device manufactures (e.g. a factory which produces a specific type of sensor). The device manufacturer uses the VO model so as to define the main features that describe the ICT (e.g. ICT Parameters), which will be represented in the virtual world by the VO. In turn the VO Type can be used by the VO installer so as to be enriched with further information for the creation of the VO full description.

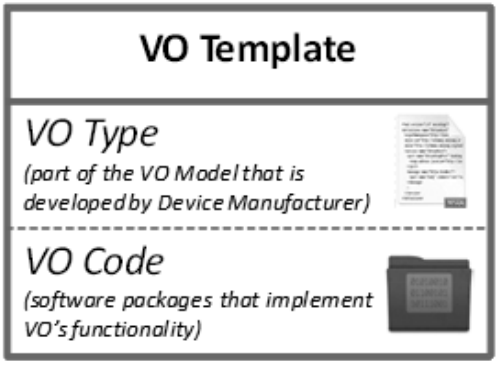

Figure 5. VO Template consists of VO Type and VO Code.

Having the VO description the next part of the implementation of the VO Template is the development of the VO Code. The VO Code includes the VO Back-end and the VO Front-end module(s). Specifically, as it is described, the VO Back-end constitutes the device software, which is provided by the device manufacturer during the device production, while the VO Front-end is a generic interface that is used to develop the module, which will be used to make the ICT compatible with the iCore system. The definition of the VO Front-end can be realized by providing a set of different requirements for its development, which constitute the VO Software specifications.

2.3 VO Template Repositories

The developed VO Templates are stored in the VO Templates repositories that are distributed in the iCore system. The VO Templates are stored as semantically query-able collections in the repositories by the device manufacturers. A VO Template repository is implemented as a software repository that includes the VO software resources and their meta-data, making

them discoverable by other external entities. In addition, some type of searching interface is provided to human or to other software systems. Consequently, it is possible to search and discover specific VO Templates that match the search criteria that are provided by the iCore external entities.

Defining a set of specific requirements for the VO Template Repository development, we describe the general specifications of it. More specifically, the VO Template repository should support the storing of the semantics that is included in the VO Type, as well as to store the software packages that implement the VO software agent modules and actually correspond to the VO Code part of the VO Template. Further to that, the VO Template Repository should be accessible from external entities and for this reason it should be defined a set of Requests and Responses that will be used to enable the interaction / communication with the repositories. Actually there is 1 type of request that corresponds to the request for download of a specific VO Template and includes a Uniform Resource Identifier (URI) [10] that is the VO Template URI. In addition, there are 2 different types of responses; (a) the Response from the VO Template Repository to its clients, which will include as payload the software packages that implement the VO Template and (b) the ACK response from a client to the VO Template Repository that actually confirms the success downloading of a VO Template.

Since the VO Template software packages are downloaded to the end-user side, they are able to develop additional functionalities and to install on to the VO containers, by registering in the same time the VO into the VO Registry. An indicative implementation based on the above specifications is depicted in Figure 6 where the GitHub API [9] is used to support the development of the Software repository that will store the VO Template Software packages. In this case, the end-user, having the URI of a specific VO Template can send a request to access the repository and download the corresponding software packages. Since VO Software module is developed, by using the VO Template, it can be deployed to the VO container so as to start its execution.

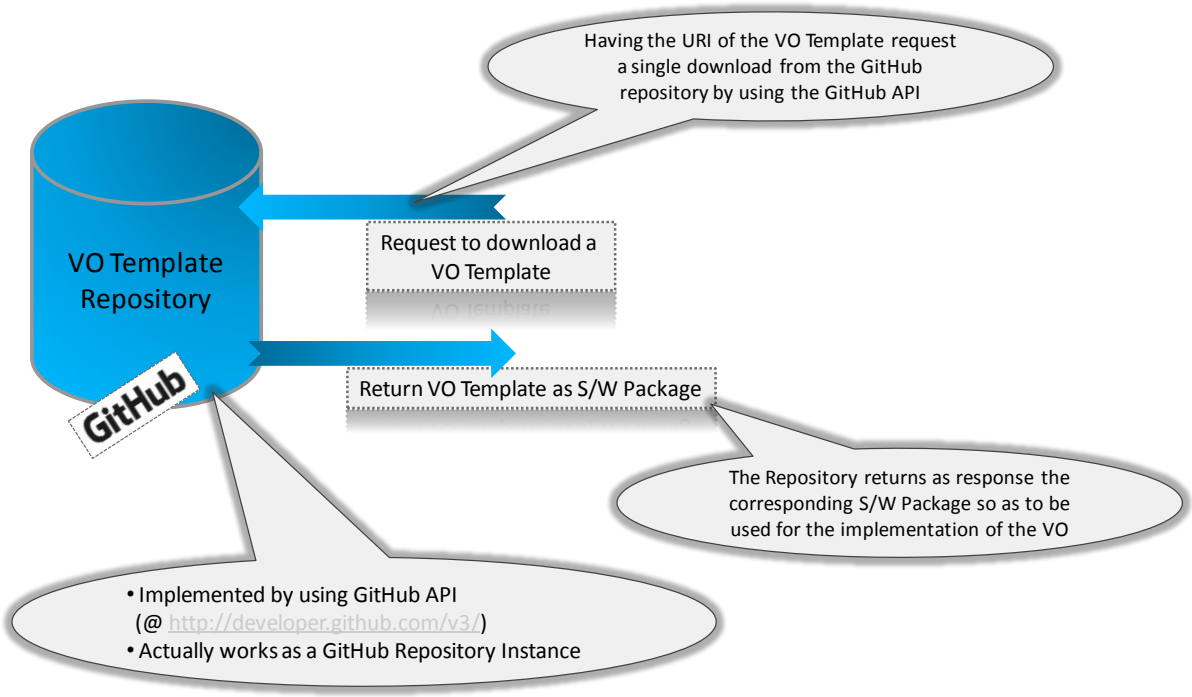

Figure 6. VO Template Repository implementation by using GitHub API

3 VO Registry Specifications

This section presents any relevant information regarding the design and the development of the VO Registry in the iCore system. The VO Registry constitutes the iCore entity, which will be responsible for the storing and management of the available VOs. It is flanked by a set of different mechanisms that support the implementation of different functionalities and therefore the execution of different processes related with the VOs and their semantics.

3.1 VO Registry design

As it is already described in the deliverable 3.2, the VO Registry is implemented as a Semantic Repository that stores structured data with its meta-data in a form of Graph Data Models. For the development of the Graph Models RDF [11] specifications are used, while the instantiated models are stored into RDF Databases (DBs), which constitute one of the sorts of the Semantic Repositories. For the implementation of the RDF DBs the Sesame Java Framework is used that provides both a Web Server to host the RDF DBs, along with a well-defined API to support the management of the repositories and their stored data.

In particular, the VO Registry is implemented as a Remote RDF DB, which is accessible over HTTP and it is hosted on the Sesame Web Server. Each RDF DB has its own SPARQL [12] endpoint through which the end users (either human or software agents) can access VO data. A SPARQL endpoint is accessible over the internet by using a specific URI [10]. Further details about the SPARQL endpoints as well as the requests towards the VO Registry, are given in the section 3.2.

Consequently, since the VO Registry can be accessible over the internet it is possible to develop different distributed registries that will be able to support the storing of the available VOs. The criteria for the distribution of the registries could be defined taking into account various and diverse factors such as the iCore compartment where the available VOs belong.

Therefore, by developing RDF DBs the distribution of the VO Registry into a cloud-based manner can be achieved, satisfying in the same time the requirements on scalability issues. Furthermore, the development of relevant ontologies that will describe different types of VOs based on their features, and/or the use of already existing standard, such as SensorML [13] and other OGC Standards, will allow the clustering of the VOs, which in turn could be one of the main criteria for the storing of the VOs into different registries (e.g. VO registry that stores only sensing devices, etc.). Further details regarding the distribution of the VO Registries are provided in the following section (**Error! Reference source not found.**).

3.2 VO Registry Endpoints and query/modification features

As introduced in section 3.1, the VO registry is complemented by its own SPARQL endpoint. The SPARQL endpoint takes over the execution of the SPARQL requests towards the VO registry. The SPARQL requests can be separated into two different types; (a) the SPARQL queries and (b) the SPARQL modification requests, which actually include the update and delete requests. Focusing on the structure of the SPARQL requests, these can be structured taking into account the information that is related with the VOs, so as to achieve the best possible result for the search and discovery of VOs.

Through the development of well-defined SPARQL queries, the search and discovery time is reduced since there are more specific data into the requests, which work as filter parameters.

Specifically, taking into account the information that is included in the VO model, it is possible to structure SPARQL queries that will be oriented to the discovery of VOs based on specific features, such as their functionality. Further to that, through the SPARQL Request as well as the SPARQL Endpoint capabilities, it is possible to support the control and access of data based on the Access Rights that are associated with the VOs [14]. In particular, Access Rights are represented as RDF triples in the VO Registry, as previous presented in the VO Model Figure 2. Every registered VO function or VO parameter goes along with a set of Access Rights. These Access Rights define whether function or parameter can be read or manipulated by a specific user or a specific user role. The advantage of this implementation is that it provides the possibility to individually assign read/write Access Rights to every VO function or VO parameter.

```
1. PREFIX vo: <http://IoT.com/virtualobject/>
2. PREFIX location: <http://IoT.com/locations/>
3.
4. SELECT ?VO
5.
6. WHERE {
7.  ?VO vo:Function "getTemperature".
8.  ?VO location:City "Amsterdam".
9. }
```

Figure 7. Query before parsing

Adding access rights properties as RDF triples to the VO Registry makes it is possible to exploit SPARQL Endpoints capabilities for finding a certain feature of a VO as well as use it to directly check whether a VO function or parameter is accessible for the querying user. To do so a query parser should be embedded in the SPARQL Endpoint. This parser decomposes the query and detects which parameters are requested by the user. Afterwards the parser manipulates the query by adding the access rights constraints to it. Figure 7 and Figure 8 present two indicative examples of the above functionality.

```
1. PREFIX rights: <http://IoT.com/accesrights/>
2. PREFIX vo: <http://IoT.com/virtualobject/>
3. PREFIX location: <http://IoT.com/locations/>
4.
5. SELECT ?VO
6.
7. WHERE
8.  ?VO vo:Function "getTemperature".
9.  ?VO location:City "Amsterdam".
10. ?VO rights:Read "UserB".
11. }
```

Figure 8. Query after parsing

4 VO Communication and Interface Design

This section describes the VO interface (north and south bound interface), the VO communication with CVO and VO communication with ICT and non-ICT objects.

4.1 VO and ICT and non-ICT object communication

VOs communicate with digital and physical objects via the south-bound interface of the VO Back-ends (VO BE). These components are drivers which can communicate with sensors and actuators. iCore does not provide requirements for this communication, but VO level has to be able to cope with the common communication patters present on these lower level interfaces.

4.2 CVO/VO communication

The communication between CVO and VO will be described in terms of:

- **Communication Patterns:** how data are exchanged, which communication model will be used (broker-mediated, publish-subscribe, synchronous rpc style), the technologies available to support these models and the way they will be used within iCore.
- **Communication Interfaces**: which "standard" interfaces a VO will have to expose and for which purpose (e.g. for VO control, for VO data access) and their details (parameters, etc.)
- **Implementation Options**: taking into account the needed patterns and interfaces specified at the previous points, the available options (protocols, data encoding, etc.) for implementing such patterns and interfaces.

Communication Patterns

A Communication Pattern is the message exchange that takes place when two parties exchange data. It can be described in terms of a simple UML sequence diagram. The following patterns will be supported by the iCore Platform in order to implement CVO/VO communication:

- Synchronous RPC Messages (Sync Request/Response)
- Asynchronous RPC Messages (Async Request/Response)
- Outbound (CVO->VO) Single Message
- Inbound (VO->CVO) Single Message
- Publish/Subscribe Messages
- Broker-Mediated Events

The communication patterns for CVO/VO interaction will be described in a way that is independent from the underlying wire protocol which will be used, so to make it possible to take into consideration different communication protocols, be it a RPC-style mechanism like SOAP/REST or a broker-based communication paradigm like MQTT. Different options for the communication implementation will be described.

4.2.1. Synchronous RPC

This is the most simple communication pattern which can be applied for CVO/VO communication. It allows the synchronous message exchange between CVO and VO. In the

Synchronous RPC communication pattern, the CVO typically blocks waiting for the inbound message from the VO.

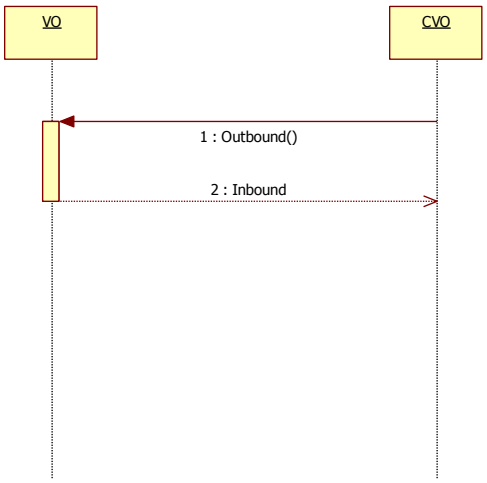

Figure 9. Synchronous RPC

4.2.2. **Asynchronous RPC**

In the Asynchronous RPC pattern, the CVO will not be blocked waiting for the inbound message, but it will instead provide to the VO a communication endpoint for sending back the inbound message, when available, in response to the outbound message.

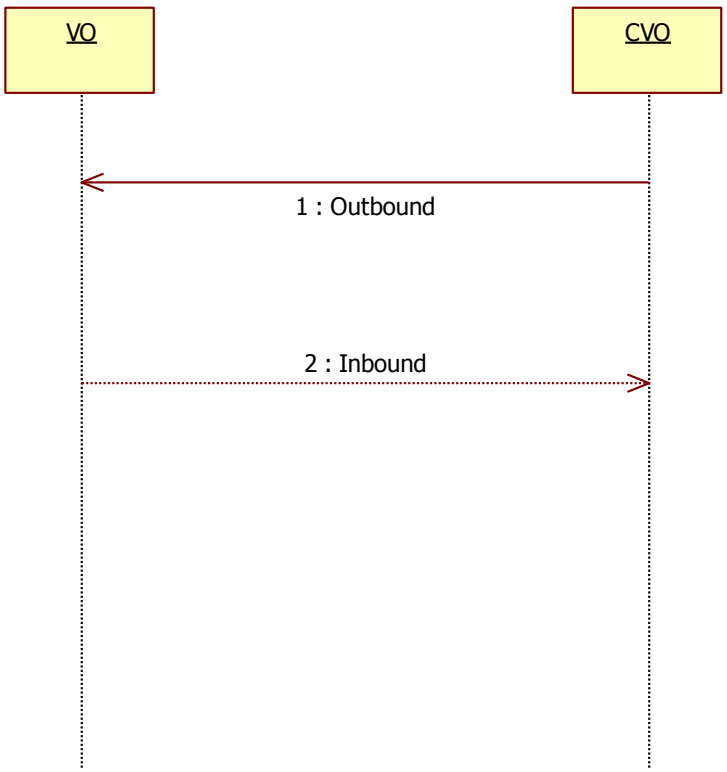

Figure 10. Asynchronous RPC

4.2.3. **Publish-Subscribe**

The Publish-Subscribe communication pattern is again an asynchronous pattern: the CVO "subscribes" with an outbound message for receiving inbound messages "published" from the VO, and the CVO will eventually "un-subscribe" when messages from the VO need not to be received. The "Subscribe" outbound message will likely contain the type of event that the CVO needs to receive, the frequency and the communication endpoint where the CVO will listen for inbound messages.

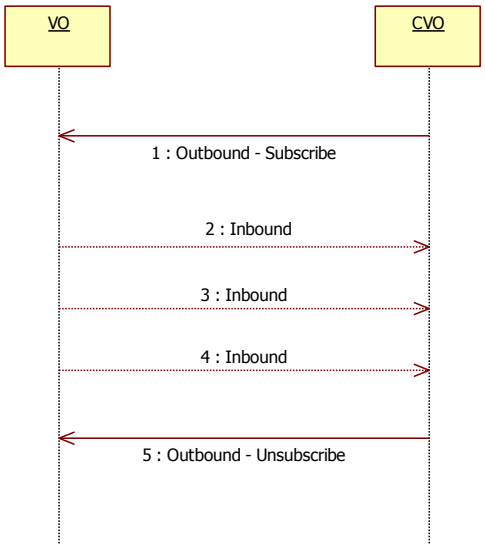

Figure 11. Publish-Subscribe

It must be noted that in this pattern the publish-subscribe pattern implementation is entire responsibility of the VO. From this point of view, this pattern provides a "simple" choice for implementing a publish-subscribe pattern, while a more sophisticated behaviour can be achieved relying on broker-mediated architectural choices.

4.2.4. Broker-Mediated Inbound Messages

This pattern defines a broker-mediated message exchange for communicating events from the VO to the CVO (inbound messages)

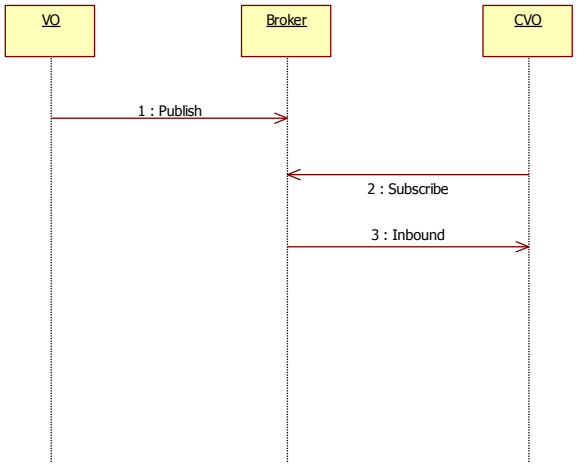

Figure 12. Broker-Mediated Inbound Messages

According to this pattern, the publish-subscribe mechanism implementation is a responsibility of the broker, which typically manages a set of "topics" for publishing and subscribing events. The VO will publish events to a specific VO topic, while the CVO will have to subscribe to the same "topic" for receiving the inbound messages.

4.2.5. Broker-Mediated Outbound Messages

This pattern defines a broker-mediated message exchange for outbound messages.

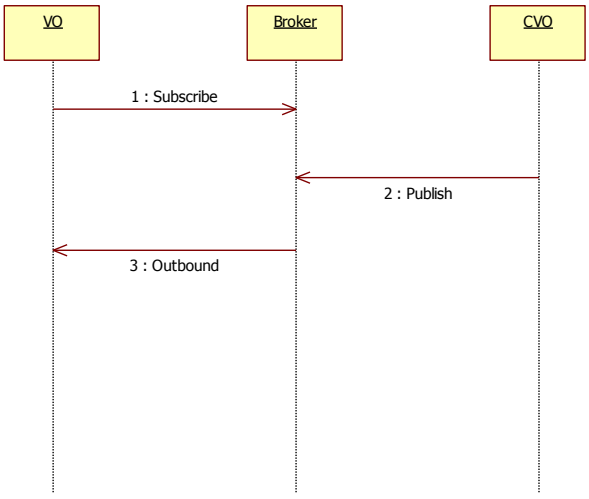

Figure 13. Broker Mediated Outbound Messages

In this case, the VO will subscribe to the VO-specific topic, while the CVO will have to publish the outbound message to the same topic to send the outbound message to the VO. In the case of correlated outbound/inbound messages (i.e. Broker-Mediated RPC Pattern) the messages will have to hold a specific ID.

4.3 General approaches for providing VO sensor data

VO Real-time sensor data are related to the measures' values for the sensor VOs. The set of measures will be sent from the VO to the CVO (inbound message), using different patterns according to the specific scenario or sensor technology. All the patterns will be suitable to be applied. The following figure gives an example of the specific interface (i.e. message content) for sensor data to be communicated from the VO to the CVO.

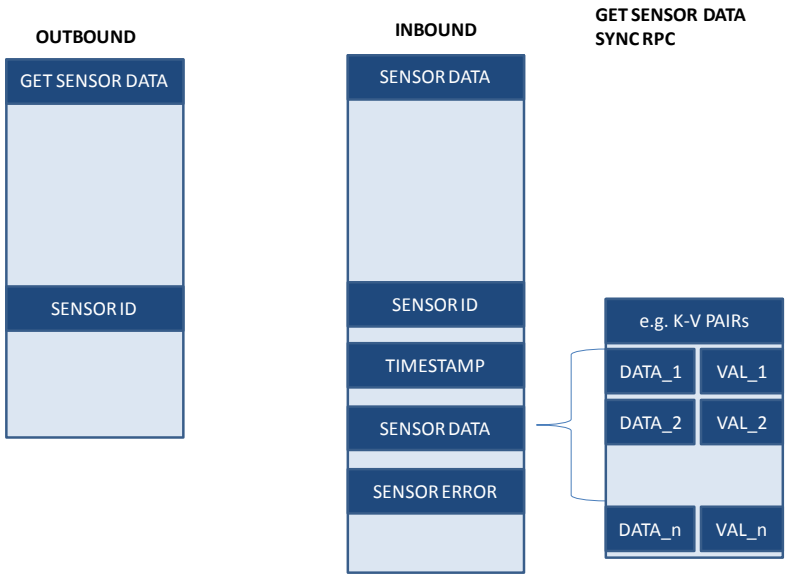

Figure 14. CVO-VO Interface for Sensor data – Synchronous RPC

In this example, the outbound interface from the CVO to the VO holds the sensor identifier, while the sensor data are packed within the inbound interface as a vector of Key-Value pair, each one holding a specific sensor data and its value, and the possible sensor error.

In the case of Asynchronous RPC communication pattern, the interface is slightly more complex since we need to accommodate some "header" information related to the specific pattern implementation. The following figure gives an example of such interface when used within a Asynchronous RPC communication pattern.

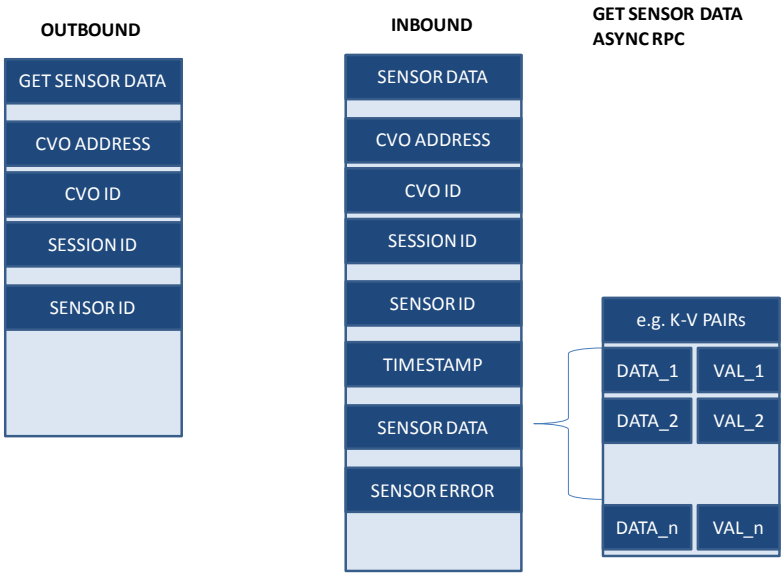

Figure 15. CVO VO Interface for Sensor Data - Async RPC

In this case, the outbound interface will have to convey as well information for the VO in order to send back the response towards the inbound interface (the CVO address and CVO id) and a correlation "Session ID" in order to let the CVO correlate the outbound and inbound messages.

In the case of the Publish-Subscribe Sensor Data communication pattern, the message exchange which takes place is slightly different, including also the subscribe/unsubscribe messages from the CVO to the VO.

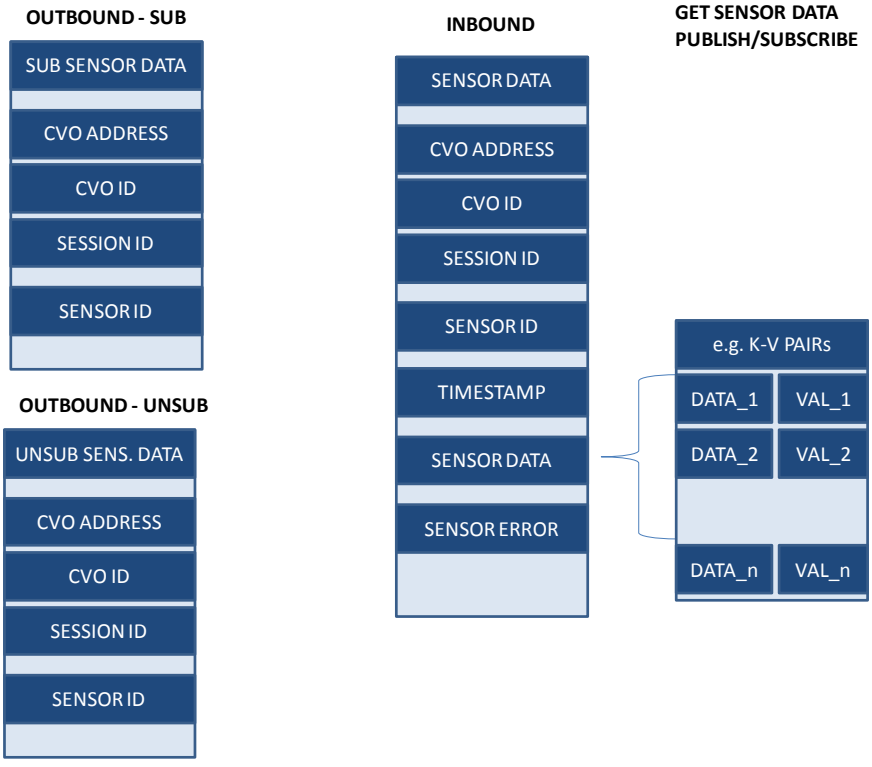

Figure 16. VO Interface for Sensor Data - Pub Sub

Finally, the Sensor Data interface can as well be implemented following a Broker-Mediated Inbound communication pattern.

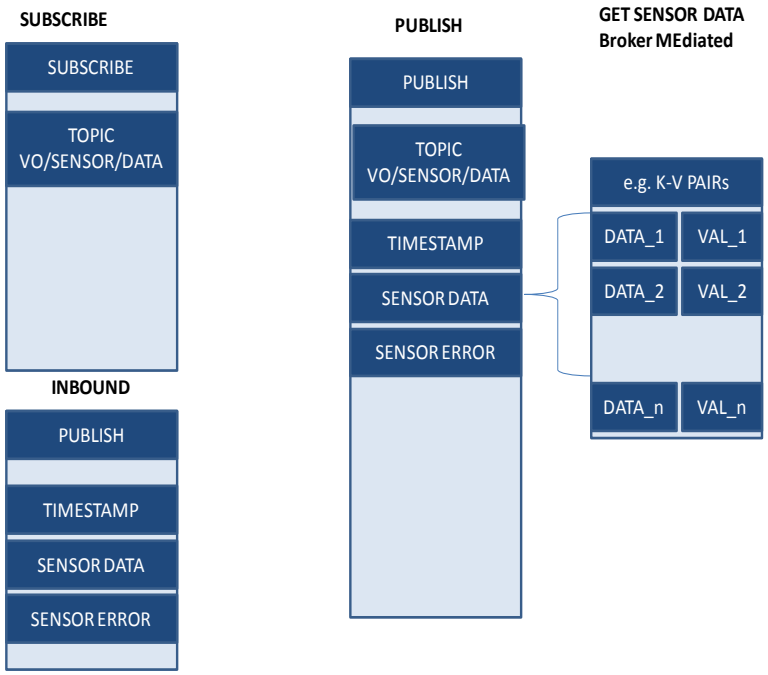

Figure 17. VO Interface for Sensor Data - Broker Mediated

4.4 VO North-bound Interface Implementation Options

Here we show the baseline choices for CVO-to-VO communication, according to the previous analysis and the available technologies. In particular, we will have to define 1) the protocol of choice (MQTT, RESTful HTTP, SOAP, CoAP) and 2) the payload content. Table 1 shows a comparison of the interface options for northbound interfacing from the VO level. RESTful interfaces and SOAP are similar as both host a server in the VO container, whereas MQTT interface is a client which pushes data to an event bus (MQTT broker between VO and CVO level). CoAP is a contemporary RESTful machine to machine interface, which we have also added to the comparison.

Table 1. VO level interface types for data exchange.

Interface type	**RESTful HTTP**	**SOAP**	**MQTT**	**CoAP**
data location (from CVO point of view)	VO container	VO container	Event bus	VO container
VO container north-bound interace	Server (synchronous)	server supporting asynchronous operations	client (pushes data to an external event bus)	server supporting asynchronous operations

performance under multiple clients	highly dependent on VO network resources	highly dependent on VO network resources	best, event bus supports multiple clients	highly dependent on VO network resources
typical user	human or machine	machine	machine	Machine
client-server coupling	Loose	Tight	loose	Loose
communication pattern	request-response	request-response	publish-subscribe	request-response and publish subscribe (Observe function)
security	Transport level security (TLS)	Transport level security	Transport level security	Datagram transport level security (DTLS)
native access control	HTTPauth (session/ cookie based access control not allowed)	multiple choices	MQTT based access control per topics (equivalent to HTTPauth)	Access control lists at servers for authorizing DTLS clients
VO registry role (eg. discovery)	CVO discovers VO_BASE_URI or IF description (WSDL/WADSL/etc…)	CVO discovers specific IF description	CVO discovers IF description	CVO discovers VO_BASE_URI or IF description (WSDL/WADSL/etc…)
built-in semantic descriptions	yes (using HTTP link header relation parameter for any content type, or by using hypermedia content types)	no (described in VO registry)	no (described in VO registry)	CoRE Link Format provides semantic descriptions of resources similar to RESTful HTTP
notes	Entire service discoverable from a VO_HOST_URI, if HTTP link headers or hypermedia as content types are used			CoRE link format includes a specification for resource directory enabling CoAP resource discovery on VO container

Table 2 shows content types recommended to be used at the VO north bound interfaces. The choices for these media types stems

Table 2. VO level content types.

Content type	**HTML**	**JSON**	**XML**	**CSV**
IANA Media Type	text/html	application/json	application/xml	application/Csv
apply to interface type	HTTP REST	HTTP REST, SOAP, MQTT	HTTP REST, SOAP, MQTT	HTTP REST, SOAP, MQTT
typical user	human	machine	machine	machine
supports context description	yes	no	yes	no
built-in hypermedia links	yes	no	no	no
iCore usability	VO API descriptions for developers	Data delivery when content is described in VO registry	Data delivery when content is described in VO registry	Data delivery when content is described in VO registry

4.5 VO level and Linked Data

In order to simplify the link between the ICT Object in the VO model (presented in section 2) and its description and by accounting for the heterogeneity and similarity of available ICT object, a linked data approach could be exploited. To this purpose, a device repository, accessible thought a front-end, containing the model for each considered device and its static property should be built, according to the following linked data specific rules:

- Use URIs as names for things;
- Use HTTP URIs so that people can look up those names;
- When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL);
- Include links to other URIs, so that they can discover more things.

By following such approach it will be easy to extend the current implementation of the VO Registry API to add the possibility to query also for external VO features, stored in a *device repository,* using combined SPARQL queries.

In order to build a repository, a set of tools can be provided to allow developers to model their platform, while a set of *stubs* could be implemented for different OS (such as Android and TinyOS) in order to extract model from dedicated hardware platform and to push the

extracted description to the device repository. The device repository should be able to maintain only unique descriptions. According to this and to the provided searchable feature of the device repository, the bootstrap procedure to create a given VO would be able to include the correct description for the given ICT Object, by filling the VO Type information (see Section 2.2). Additionally the VO management unit could be used to update the dynamic attributes of each particular VO instance, by interacting with the VO Container management provided by the VO Container (see Figure 1**Error! Reference source not found.**) to extract and transmit the required information, such as battery level (*ssn:Device:EnergySource*), location, and other info belonging to the *ComputingDevice* class.

4.6 Design for RESTful Web Services (Linked data style)

This subsection presents the mechanisms and functionalities required to create RESTful Linked Data style front-end for VO front-ends. These mechanisms include general REST requirements along with resource discovery mechanisms which should complement the VO registry style of discovery.

4.6.1. Fulfilling the HATEOAS requirement in VO REST FE

Hypermedia as the engine of application state (HATEOAS) is an important constraint of the representational state transfer (REST) architectural style. In iCore it is only related to the HTTP driven VO REST FE in the VO Container. A REST client does not need any prior information when accessing an URI as the client can discover new resources by following links the HATEOAS-compliant server hosts. Within the iCore scope this would mean that information present in the VO Descriptions are duplicated as resources hosted on the VO REST FE on the VO container.

HATEOAS requires either hypermedia content types such as HTML, JSON-LD or the usage of HTTP link headers to be able to create links between resources. In iCore the choice of HTTP link headers for linking is natural, because it means that the data created at the VO Back-ends in JSON, XML or CSV content types can be passed to the front ends as it is. Otherwise we would have to use a content-type such as JSON-LD or HAL for enabling links in JSON payloads.

4.6.2. Adding context to data through VO REST FE

In iCore the context of the data accessed by upper levels is present in the VO Registry as VO descriptions. If we use pure RESTful front-ends to access the information, the context has to be embedded in the resources or their representations. We argue that as with creating links between resources, HTTP link headers are a useful way to attach the semantics to data. In practise this means that we can attach a description of the data in JSON-LD format to a resource with a JSON representation. An example [38] of such a HTTP header could be:

```
HTTP/1.1 200 OK
...
Content-Type: application/json
Link: <http://json-ld.org/contexts/person.jsonld>;
rel="http://www.w3.org/ns/json-ld#context";
type="application/ld+json"
```

Here the *Link* points to a .jsonld file describing the data and the relation tag *rel* holds the type of the URI. JSON-LD is a multi-faceted language, but it can be viewed as a RDF serialization. This means that RDF based VO Descriptions can be linked to VO REST FE resources as their

context. Thus the information present in the JSON-LD schema is also present in the VO Registry.

4.6.3. Web Discovery of VO resources

As an optional alternative we may use web discovery using HTTP [39] and web linking [40] to advertise changes in the resource structure of a VO Container. The VO_BASE_URI shall host a list of links to all available resources on the VO Container and update this description during run-time. Unlike in HTML, we use HTTP header property *Link:URI* to assign a link or a list of links between resources. This feature enables us to use non-hypertext content types such as JSON or CSV in VO resources and still benefit from client-driven states. The link relation parameter *rel=URI* of these links must point to the VO Model class the resources belong to. Thus, we do not use predefined link relation, but link the resource to an ontology (here: the iCore VO model). Semantic descriptions of the VO model can be used by the client to infer more information about the resource. All resources hosted by the VO Container other than the VO_BASE_URI must include a link back to the VO_BASE_URI.

4.6.4. CoRE discovery of VO resources

A tertiary and optional alternative for realizing self-advertisement is the VO container local resource directory for VO REST front-end. A resource directory enables querying of all resources hosted by a VO with a single RESTful query. This type of functionality can be used with VO HTTP REST and VO MQTT front-ends, but not with SOAP style front-ends. One possible alternative would be to use the resource directory from Constrained RESTful Environments (CoRE) Link Format [41]. Although CoRE is usually used with Constrain Application Protocol (CoAP), it can also be used to advertise resources of other RESTful protocols such as HTTP. As we are not using CoAP as a VO front-end protocol, we keep CoRE only as an alternative for implementation.

4.6.5. Historical data storage at VO level

Historical data storage on the VO level is an optional feature for VO REST Front-ends to implement. This data is stored at the VO Container. The presence of such functionality should be noted in the VO Description as relevant VO Functions. Although we do not enforce a single API for historical data storage access, we highly recommend using a single API in iCore scope because of practical issues. The API is a simplified version of the Xively Historical Data storage API:

Table 3. VO Historical API (HTTP REST).

Example URL	http://VO_BASE_URI/vo/vo_function/vof_id**?range**
HTTP method	GET

The type of historical data query can be changed according to the “?range” query string at the end of the URL as seen in Table 3. **range** is one of the following:

- start=**timestamp**, end=**timestamp**, start=**timestamp**&end=**timestamp**
- start=**timestamp**&duration=**time_unit**
- **timestamp** is an ISO 8601 formatted date
- **time_unit** is either seconds, minutes, hours, days, weeks, months, years.

4.7 Design over SOAP Web Services

One implementation available for the introduced interfaces and patterns is through standard SOAP Web Services. In such implementation, a VO will have to provide a SOAP Web Service interface, described by WSDL, to be accessed by the CVO. An example of the WDSL description can be found in

Appendix C. The SOAP Web Service interface is suitable for implementing an Asynchronous communication pattern or a Pub-Sub communication pattern, provided that CVO endpoint for sending the inbound message is sent together with the request. This can be commonly achieved either by including the SOAP CVO endpoint within the SOAP request parameters or by exploiting the WS-Addressing standard SOAP headers.

4.8 VO Performance and Monitoring

VO performance and monitoring are done externally. The ICT objects may expose resources which can be used to transmit such information, but this is not a mandatory feature. However, here we describe a set of tests we can perform to evaluate the performance of Virtual Object implementations:

VO level performance metrics are an important issue for the scalability of the iCore project. In Table 4 we present the metrics which will be evaluated during the final project year. The table shows the main types of metric to be estimated. The exact measurements will be defined and reported in D3.4.

Table 4. VO performance metrics to be collected

Operation class	Metric	Rationale
ICTo to CVO data (REST front-end)	Latency [t]	Measure the latency of accessing data from the CVO level using the VO REST FE
ICTo to CVO data (MQTT front-end)	Latency [t]	Measure the latency of accessing data from the CVO level using the VO MQTT FE
VO installation to first CVO data access (REST)	Latency [t]	Measures how quickly data hosted by a VO can be accessed after the installation using VO REST FE
VO installation to first CVO data access (MQTT)	Latency [t]	Measures how quickly data hosted by a VO can be accessed after the installation using VO MQTT FE
VO registry – discover VO	Latency [t]	Measure VO discovery latencies
VO registry – install VO	Latency [t]	Measure VO installation latencies
VO registry – update VO	Latency [t]	Measure VO update latencies
VO registry – discover VOs	Latency versus requests and VOs installed	Measure VO registry scalability as latency of request versus concurrent requests and VOs installed

4.9 VO Naming

This subsection will constitute the extension of the corresponding section into the D3.2 document by adding further information for the naming & addressing issues as well as

describing potential mechanisms that would be developed with respect to the requirements of this field.

4.9.1. **PURL**

Persistent URL can be used in iCore to dereference any Virtual Object and thus also gather information about the associated ICT and non-ICT objects. PURLs are not usable with non-iCT objects due to the fact that the association between the entities with Internet connections and the non-ICT objects they are associated with may change over time. The PURL of a VO is tightly coupled with the VO_BASE_URI.

We aim to use a PURL REST API (https://code.google.com/p/persistenturls/) in the VO Factory related components.

4.9.2. **uID**

A possible scheme for identifying non-ICT objects is the use of uID technology. A 128 bit globally unique code can be assigned to physical objects and Internet based information may be accessed based on this via a HTTP based interface. With uID it may be possible to perform a reverse mapping between non-ICT objects and the Virtual Objects they are associated to, e.g. it may be possible to discover a VO_BASE_URI of a VO and thus access the resources it hosts by for example reading a ucode tag attached to a non-ICT object.

5 VO Privacy

In iCore we adopt a Model-based Security Toolkit (SecKit) that supports the specification and enforcement of privacy, security, and trust policies at all the layers of the iCore infrastructure: VOs, CVOs and Services. The framework is based on a collection of meta-models, which provide the foundation for security engineering tooling, add-ons, runtime components, and extensions to address requirements of governance, security and privacy. A detailed description of the policy support and how the different security and privacy requirements are addressed by the SecKit in iCore is available in deliverable D2.4 [42].

5.1 Security Toolkit at the VO level

In the SecKit, the modelling of the IoT system for security specification purposes is done using a conceptual language inspired by an existing generic design language to represent the architecture of a distributed system across application domains and levels of abstraction called Interaction System Design Language (ISDL). In the SecKit meta-models the system design is divided into two domains named entity domain and behaviour domain, with an assignment relation between entities and behaviours. In the entity domain the designer specifies the entities and interaction points between entities representing communication mechanisms. In the behaviour domain the behaviour of each of the entities is detailed including actions, interactions, causality relations, and information attributes.

The following figure shows the screenshot of the Behaviour model section of the SecKit Graphical User Interface (GUI). In this example the Smart Home behaviour, assigned to a VO Container entity (see status bar in the figure) is highlighted with contained behaviour instantiations. Using the SecKit it is possible to specify, in addition to the system behaviour model the data, identity, context, trust, role, structure, risk, and security rules model. The purpose of the detailed specification of these set of meta-models is to have a reference for the specification of security, trust, and privacy elements.

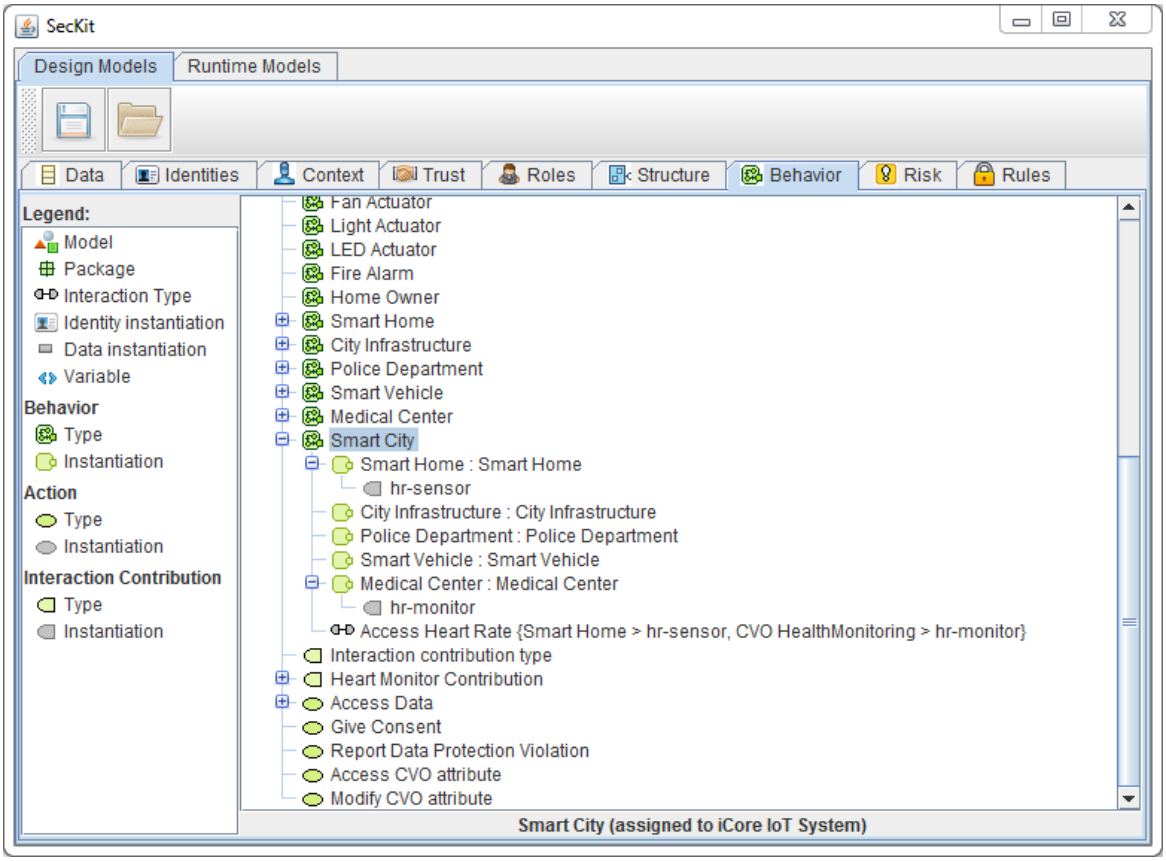

Figure 18. Behaviour design model

The SecKit itself allows the specification and evaluation of policies for any technology choice of a real system, however, in order to be useful in concrete implementation scenarios, the SecKit must be extended with technology specific runtime monitoring and enforcement components. In the iCore project we provide one extension to support monitoring and enforcement of policies for a MQTT broker, which is the technology, adopted by most of the project partners to support communication between VOs and CVOs. In the MQTT architecture clients can connect to a broker to publish or to subscribe to messages in topics.

The reference MQTT architecture behaviour extended by us to implement enforcement of policies is presented in Figure 19. In Figure 19 the interactions between the Client and Broker behaviour represent the MQTT standard messages, which are intercepted at the broker side by a Policy Enforcement Component (PEP). The PEP is a technology specific extension that generates events to a Policy Decision Point (PDP) and receives enforcement actions as a response. The PDP component is accessible through HTTP and is integrated with the SecKit GUI interface that allows the specification of design and runtime models.

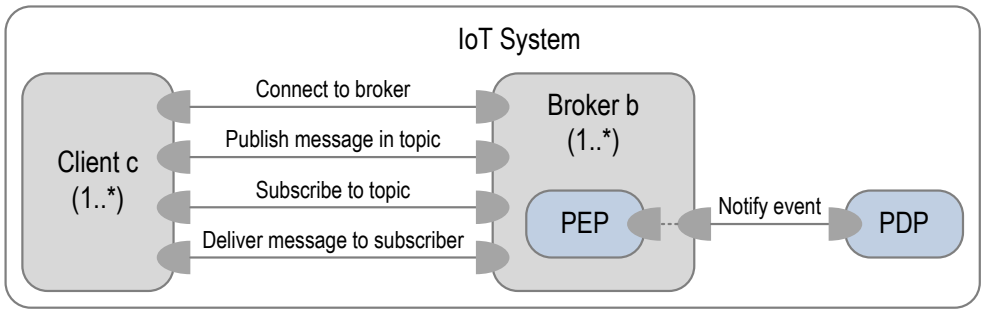

Figure 19. MQTT architecture

Figure 20 shows an example policy specified using the SecKit GUI, which was specified for the hospital scenario defined in WP9. The enforcement target of this policy is the MQTT broker PEP, and the objective is to allow a visitor access to the patient location in a hospital. This policy illustrates some aspects of the SecKit enforcement language, including the support for parameterized enforcement with variables, role-based access control, integration with external information sources through the "is Visitor Of" invocation, and matching of event attributes using regular expressions.

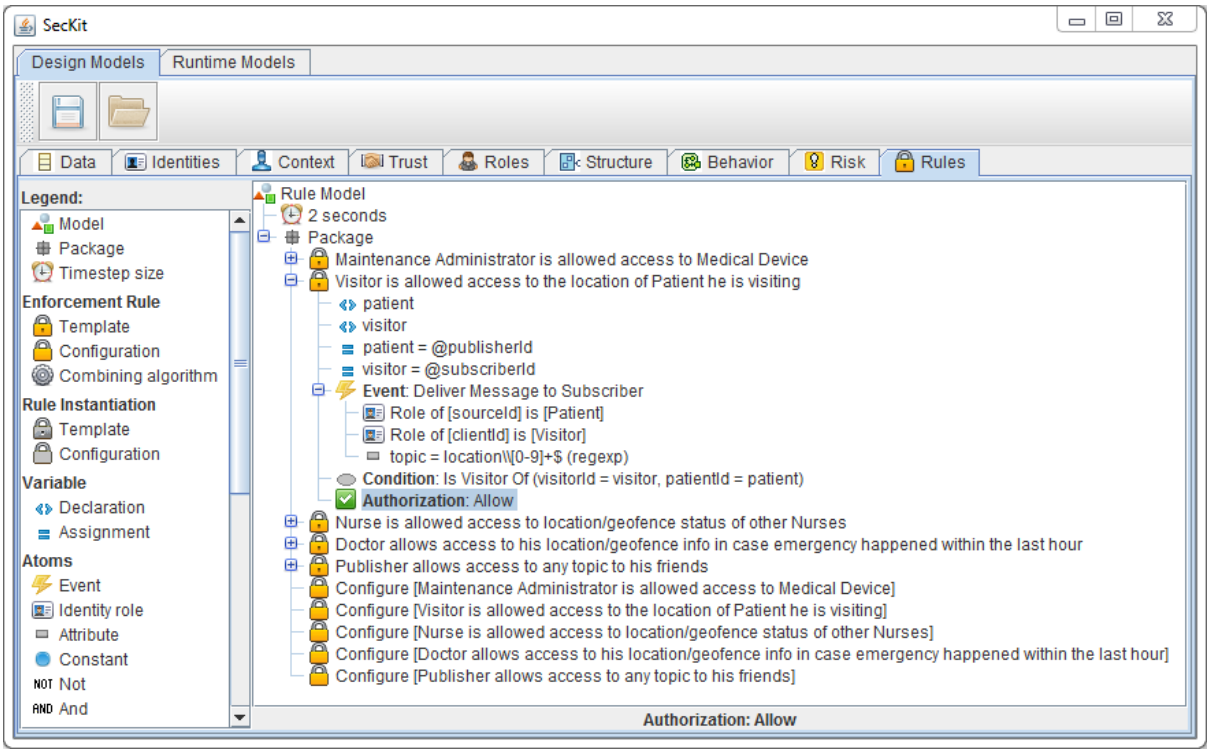

Figure 20. Policy to allow a visitor access to the patient location

The specified policies are monitored in SecKit using a rule engine that receives events notifications from the PEP components. Figure 21 shows the runtime interface of the rule engine that instantiates the specified policy rules and receives events generated by extended

MQTT broker for the hospital trial. Our PEP extension is a connector that intercepts the messages exchanged in the broker and notifies these events to the SecKit interface.

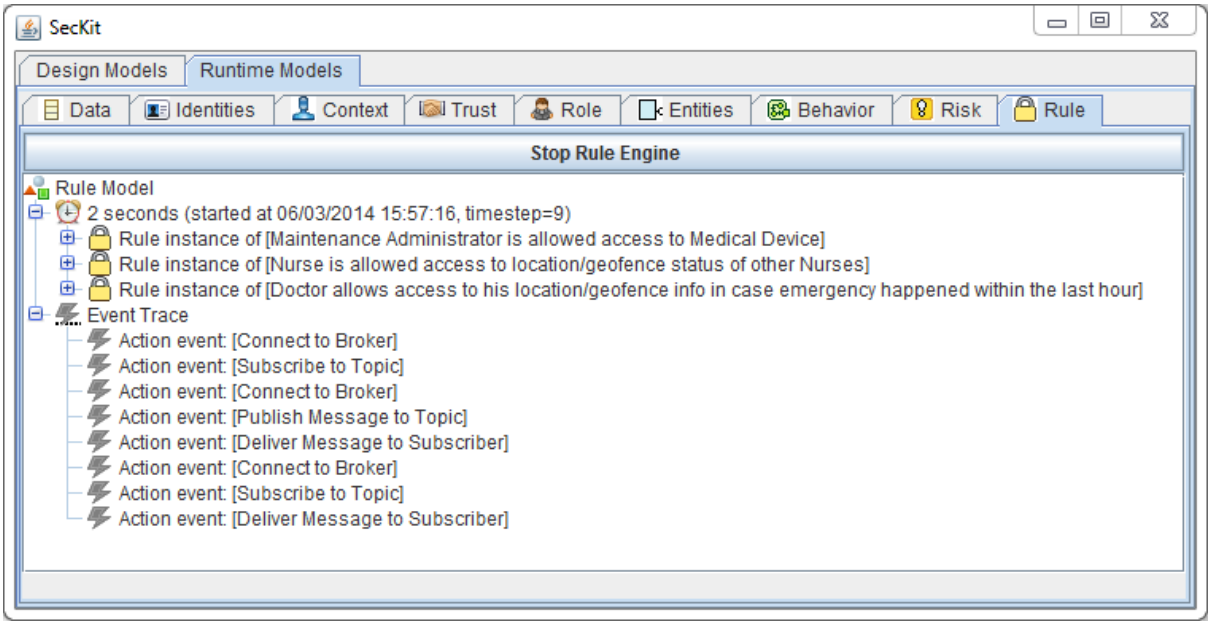

Figure 21 MQTT events received by SecKit.

6 VO Factory and associated mechanisms

In general this section and its subsections will provide the description of the VO Registry API mechanisms, by extending the work that has been done in D3.2 through relevant updates. Requests from VO Life-Cycle Manager will instruct VO Factory to instantiate particular VO based on templates. VO Factory also search/discover in the VO registry already instantiated VOs and change (them) their associations based on trigger received by the VO Life-Cycle Manager.

6.1 VO Creation Tool – Integration of VO Creation / Registration and Modification Mechanisms

The VO Creation Tool constitutes a powerful tool that includes creation and registration mechanisms so as to allow the end-user to perform the creation and the registration of the VOs in an easy way. In addition the tool allows the user to manipulate the VO related data stored in the VO Template Repositories and the VO Registries. Figure 22, presents the overview of the VO Creation Tool functionality in a high-level approach. In particular, the tool offers a user friendly Graphical User Interface (GUI) that helps the end-user to add the description of VOs in a traditional way by completing data forms. The background functionality of the VO Creation Tool takes over the dynamic and automatic transformation of the information into meta-data properties, based on the VO Model (Figure 2), constructing thus the VO Semantics that in turn is stored in the VO Registry. In order for somebody to use the VO Creation Tool he should have specific credentials that are associated with a specific user role and a specific unique API-Key. Depending on the user role, the end user has different and diverse rights in the system. Moreover, the system uses the API-Key so as to enable various

algorithmic functionalities for the performance of different actions in the system such as the VO Registration.

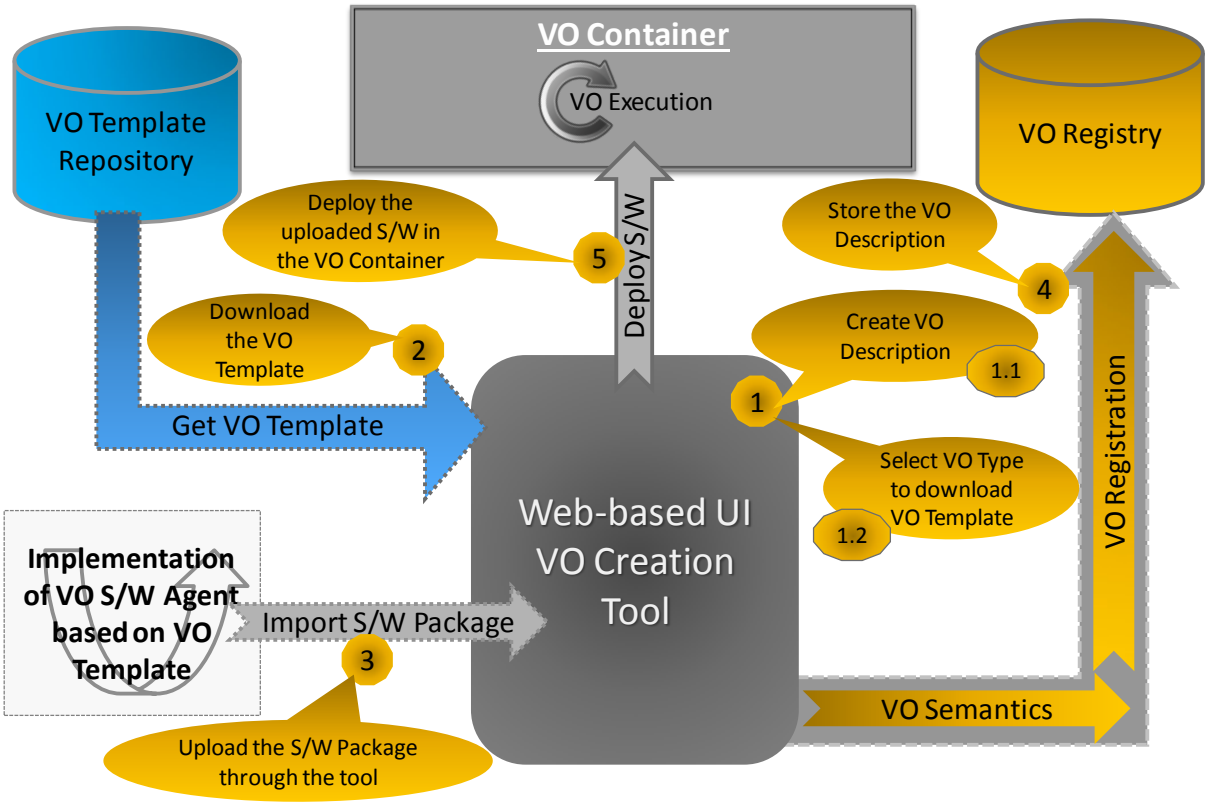

Figure 22: VO Creation Tool and associated processes

Further to the above functionality, the tool allows the end-user to interact with the available VO Template Repositories, by performing request to download VO Templates, while when the VO Software module is developed, the end-user can deploy it in the preferred VO container just following some simple steps in the tool's GUI. Having deployed the VO in the VO Container and registered it in the VO Registry, the VO Creation Tool allows further interaction with the VO data through its modification, by updating or deleting them through the GUI easy and fast, by following traditional techniques, such as the completion of data forms.

At this point it should be highlighted that the current tool implements the VO Registry API, making an abstraction of the offered functionality, as well as by hiding the complexity from end-users. Consequently, the above described functionality constitutes part of the VO Factory functionality, while the offered GUI makes the use and the exploitation of these mechanisms easier.

7 VO Control and Management Unit and Associated Mechanisms

7.1 VO Life cycle manager

The VO lifecycle manager constitutes the iCore entity, which is responsible for the monitoring and management of the VO life cycle that essentially is defined by different VO states. Specifically, the VO during its lifecycle may have different states that present its current status. The various states can be defined in an ontology, enhancing in such way, the functionality of the VO Life cycle manager using semantics. Following the approach, which refers to the ontologies, an appropriate data model should be developed, in order to present the included states and the associations with each other.

In particular, the VO States model will comprise the different states in which a VO can be, while it will present the associations between the different states. A VO can go from one state to another one, under specific conditions / situations and the VO life cycle manager component monitors these transitions and updates the VO current state. Figure 23 depicts the VO States model that constitutes the base for the development of the ontology, which will contain the semantics (in terms of meta-data) for the characterizations of the VO states. The VO life cycle manager will take into account the semantics for the VO states so as to manage the transitions between them, while will take over to update dynamically and continuously the current state for the available VOs. The current state for each VO will be stored into the VO registry, by exploiting the VO Model property, which is called as *VO Status.*

The VO States model could be developed by applying semantic web technologies such as OWL in order to describe the concepts and RDF in order to structure the model as it is presented in Figure 23. Furthermore, the lifecycle manager can be developed so as to support the observation and the update of the VO State in an automatic way by using SOARQL query language. In particular, by using SPARQL it will be able to acquire the current VO State from the VO Registry, while in the same time it will be able to perform SPARQL update requests to update the status of each VO, depending on the situation.

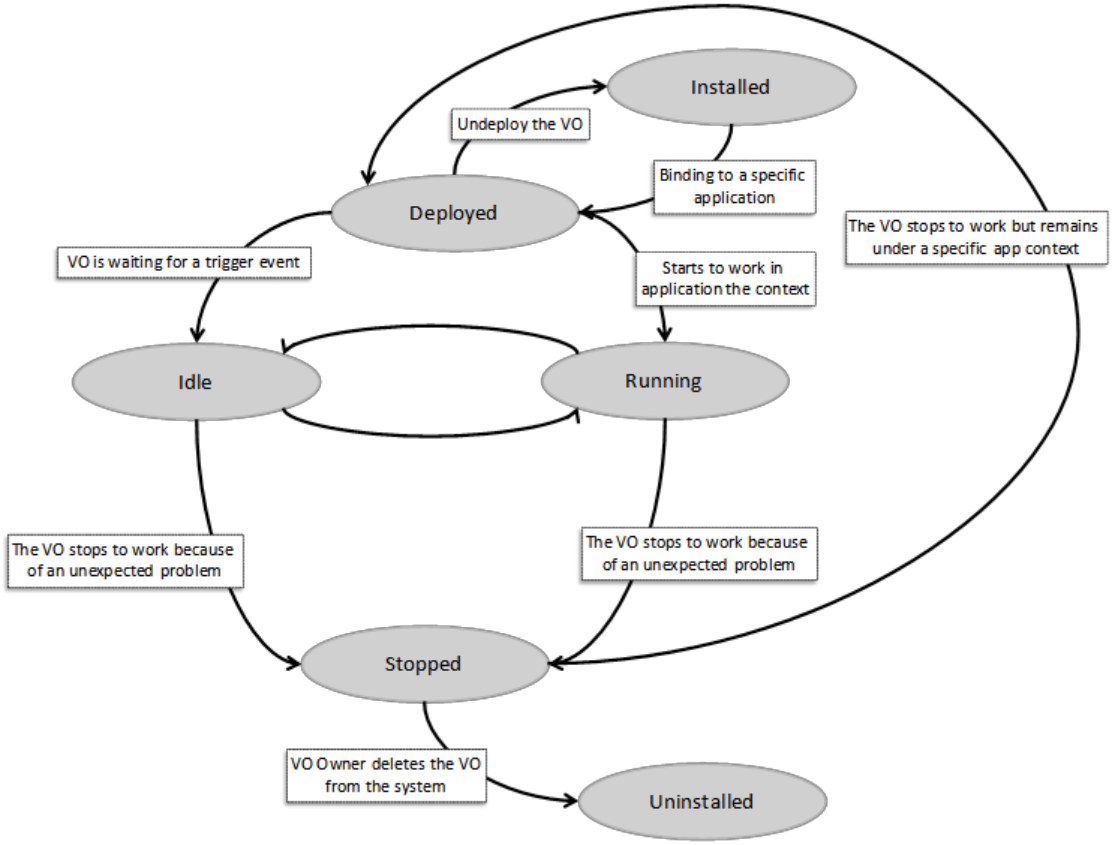

Figure 23. VO Possible States

7.2 VO Registry Access Control mechanisms

VO (Container) related privacy and access control were discussed in Section 5 with the introduction of Security Toolkit for VO level, so this subsection presents mechanisms for VO Registry access control.

As it is already described in the section 3.2, the VO Registry is complemented by SPARQL Endpoints that are able to support the access control to the VO Registry by using semantics to define access rights, associated with specific user roles (

). Each access right is associated with one or more user roles that are authorized to perform the respective action, which is defined by the access right. The user roles are distributed to each user during the registration process on the iCore platform and as already mentioned, they are associated with specific credentials.

In addition to that, as it is presented in Figure 2, the VO Model includes the VO Owner for each VO. The cardinality for the association between a VO and a User, as VO Owner, is 1-1 that means that each VO may have only one user as its owner. Each user is identified by a unique URI that indicates to the description of the user as a resource in the IoT world. The description of the user is based on a User Model that includes specific properties that describe the user entity in terms of user preferences, user characteristics, user roles, et cetera. Furthermore, the VO Functions are governed by specific rules that are expressed as Access Rights.

Table 5. User Roles

User Role
VO Owner
iCore Simple User
iCore Premium User

At this point it should be highlighted that each user that is the owner of a VO takes by default the role "VO Owner" for the respective VOs while depending on their user account in the iCore platform takes the respective additional user roles for the rest of the available VOs.

7.3 Management of Multiple Access Control and Conflict Resolution

If any two entities (CVOs) try to access the same resource in VO at the same time a conflict may arise. Now the obvious question is who will get the access? On what basis the access will be given? Can one CVO claim the access right by pre-empting another CVO? All these questions are answered by a cognitive module at the VO level, called the conflict resolver. There can be two different possibilities of ownership, which might lead to conflict.

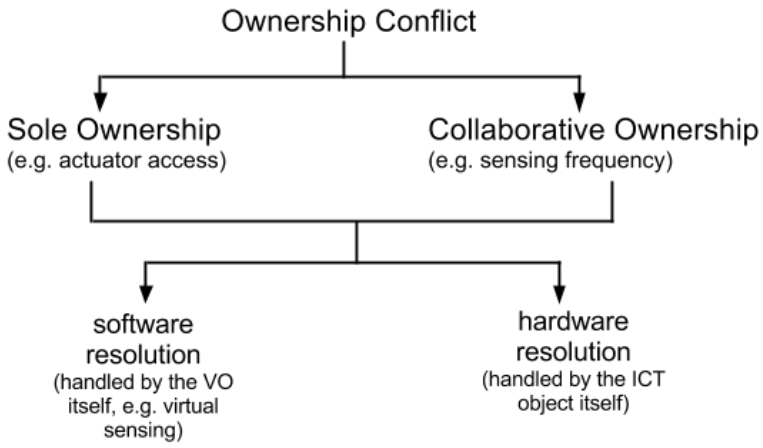

Figure 24. Resource ownership and conflicts

The ownership of some resources (at the ICT object) can be shared. In those cases, a single CVO can have the ownership of the resource. This type of resources is referred as "Sole Owning" resource, e.g. access to an actuator (Figure 24). On the other hand, some resources can be simultaneously accessed by multiple CVO, and they referred as "Collaborative Owning" resource, e.g. voltage level in a sensor. When multiple CVOs try to access the same resource in an ICT object through its corresponding VO, a conflict might arise based on the type of access requested by the requesting CVOs. The responsibility of the Conflict Resolver is to first identify such conflict and then find a way to resolve it. Some of these conflicts can be handled by the ICT object itself (hardware resolution), while some can be handled at the VO level (software resolution).

The conflict resolver not only resolves the possible conflicts, but it also decides (and eventually provide access) how many simultaneous requests from the CVOs can be handled. Let's say there are N simultaneous requests are received at certain point of time. Out of these, the ICT object can handle n simultaneous requests. Additional m requests can be served by software solution. The remaining p requests (p=N-n-m) are queued to be served in future (Figure 25). The number m is not fixed and it can vary based on the type of requests.

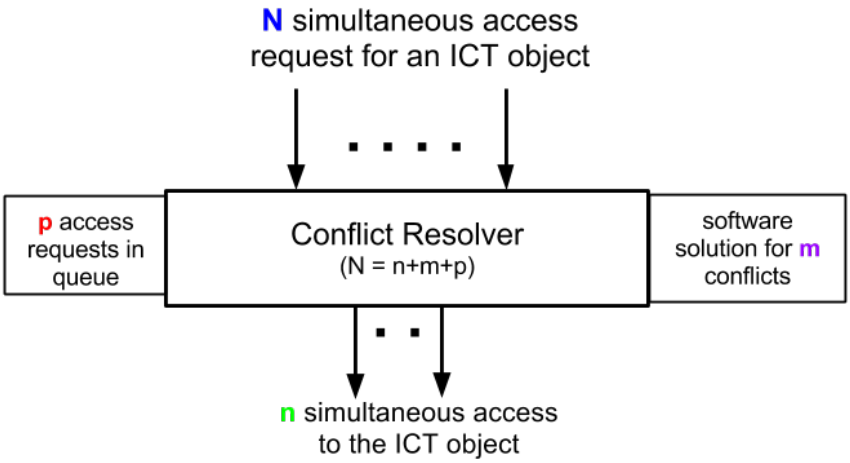

Figure 25. Simultaneous access request handling by the conflict resolver

8 Virtual Object Level Cognition and Machine Learning

This section describes the cognitive and machine learning techniques for VO level. We present some statistical and machine learning (ML) based approaches which may be used at the VO level. We propose some ML algorithms for some conceptually simple issues that might be solved at the VO level. The ML algorithms and data they act upon may be hosted by the VO backend, and thus the VO model is kept simple for its consumers.

A set of cognitive capabilities are required at the VO level to address the following aspects:

- VO and RWO communication maintenance
  - Identification and triggering of link failure
  - Link reestablishment
- Management of multiple access of a VO
  - Handling simultaneous access request
  - Identify and resolve conflict in access request
- VO resource optimization
  - Energy management at the ICT object
  - Handling ICT object mobility
  - Fair-share among competing CVOs for a VO resource
- VO self-advertisement mechanism
- Semantic community among VOs
  - Group formation among VOs based on geographical collocation and similar functionality

8.1 Energy efficient and timely resource discovery through cognition

The extensive consideration of mobile and wireless ICT devices for service provisioning in IoT environments makes the energy efficient discovery of VO available resources a daunting task of great importance for various reasons. By achieving energy conservation at the ICT device level, thus extending the useful lifetime of such battery constraint devices, saving of additional device resources at discovery time will make them eligible for participating in service provisioning for much longer, once discovered.

Contrary to the fixed world where the resources available to support service requests are usually assumed to be known *a priori* and are also assumed to be in place for the whole duration of relatively long-lived services (e.g. provisioning of a Noise-Level detection service in public space takes for granted the availability and location of a dedicated audio hardware for the whole duration of the service, unless of course the occurrence of a hardware fault or some other abnormal/unexpected condition takes place) the same cannot be said for the IoT world where service requests can be much more short-lived, thus requiring efficient and timely *on the fly* use and release of resources. Additionally devices can move, disconnect, reconnect etc. making the timely and correct discovery of their resources a very important task, upon which the service requirements very heavily depend. The ICT/non-ICT association in such cases is

predominantly not static but changes as the ICT object moves and even disconnects/reconnects with the iCore system, becoming unavailable temporarily (or even permanently) and then available again at a different location.

A straightforward way for the VO Management Unit (VO Lifecycle Manager) to maintain the entries of such VOs in the registry up-to-date would require ICT objects to continuously probe the environment for discovering "*associations*" for the benefit of the iCore platform. This process however, apart from imposing considerable strain on the battery life of the mobile devices (battery life that could be used for other useful operations) may not result in an efficient utilization of the device after all, if e.g. such frequent attempts for discovering and updating associations do not offer any added benefit. In such cases continuous probing would eventually have the same exact effect as no probing but at a much higher energy cost, thus resulting in detrimental performance for the ICT object that will affect the ICT object user experience, especially in the case of ICT object being bound to a user smartphone. Side effect of this will be a sense of distrust towards the overall iCore system stemming from the overall reduced availability of the ICT object.

A way to overcome this would be through the incorporation of learning mechanisms at the VO/ICT object level that would allow the mobile VO to learn to probe only when it expects that it is within communication range of a location/infrastructure that will create associations beneficial for the iCore platform and also only when it expects to move considerably. That is, as long as the VO stays within the bounds of a certain area it does not have to declare its presence continuously, as long as this does not change substantially. Assuming as an example a reinforcement learning (RL) approach incorporated in this discovery/probing process, the mobile VOs would initially probe frequently and even at wrong times but eventually based on the rewards/penalties imposed by the RL approach to correct/wrong probing actions, they would eventually learn when and how frequently to probe to declare their presence and also maximize their availability to the iCore platform. Incorporation of learning mechanism as such can offer energy efficiency at the ICT device level.

In addition, the learning mechanisms can offer the benefit of timely and prolonged availability of the mobile VO resources to the iCore platform. Prolonged availability both because of the reduced strain on the battery which can increase the ICT object useful lifetime, but also because by establishing association beneficial to the iCore platform as soon as they are within communication range of a location/infrastructure, the opportunities of the VO resources being used by the CVO instances they are bound to are maximized.

If the knowledge built this way is taken into account from a VO Management Unit perspective, it can offer further benefits to the provisioning of the services themselves. This can be illustrated using the following example; assuming the VO Management Unit gets to know the mobility patterns of mobile VOs it can (*Resource Optimization*) take advantage of this information to ensure as much as possible disruption-free and quality assured services.

For example, based on the mobility pattern, the Lifecycle Manager can insert information in the VO registry about the availability or not of VOs, indicating as well factors such as "expected remaining available time" and "expected to become available time". As such, when service requests are issued, this information can be taken into account and lead to the selection of VO instances corresponding to ICT objects that are expected to be more stationary during the specific service request lifetime (if that assists in fulfilling the service requirements) and are also less prone to disconnects and reconnects that can lead to service disruption. Viewing this one step ahead, one may even consider the possibility of binding a VO instance that will be needed at a later point or within a given acceptable delay from a service request at a certain

location if there is no other VO at that location currently, but there is high probability -*based on the mobility pattern*- that the VO in question will be at that location when actually needed by the service request. That is, instead of rejecting the service request, the iCore system can grant it with certain degrees of confidence if that "satisfaction degree" is something that the service request tolerates.

The way that such knowledge can be communicated and taken advantage of from a VO Management Unit perspective depends on the nature of the learning mechanism themselves. For example, if Q-learning (one of the most common RL approaches) is used by the ICT objects to learn when and for how long they need to probe to declare their presence and maximize the availability of their resources to the upper layers, the information included in the Q-tables can be used to indirectly derive the current remaining available time and the subsequent available times and their durations. So if for example an ICT object has determined as the best action to take to find a location/infrastructure beneficial to the iCore platform being the action "*probe after 15mins and for 30secs since the last time in contact with the infrastructure*", the VO Management Unit should expect the VO bound to the ICT object to exist after 15mins and for 30secs.

Alternatively the iCore "association points" can run similar mobile ICT movement prediction mechanisms, either in conjunction with the learning mechanisms operating at the mobile ICT level or standalone, further reducing the overall energy budget associated with the iCore platform operations and building the knowledge needed for updating the VO registry entries appropriately without requiring the additional communication (for the needs of knowledge building) between the mobile VOs and the iCore platform.

8.2 Anomaly detection for sensed data

Anomaly (outlier) detection refers to reporting data which are outside a presumed model. A definition of outlier is provided by Hawkins in [16]: "An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism".

Outlier detection is subject of long time research; see for example [15] for a recent review of this work. Some motivating examples are analysis of financial markets, system diagnosis, user-action sequences, and biological data. In iCore, the sensor network may benefit from detecting anomalous reading values, compared to the recorded time-ordered sequences. The crux is to formulate hypothesis on temporal patterns and detect the events that does not conform to them.

At least two questions arise in context of using outlier detection in context of iCore:

1. Which is the best approach for outlier detection?
2. How should one react to outliers?

For the former question there are at least two answers. One of them is encompassed into the "There is no free lunch" results, in search, optimization and machine learning [22-24]: it is theoretically impossible to find a single approach which works best for any problem. However, for a specific problem subset (e.g. predicting temperature evolution for some transportation services and flagging anomalous values), one might come with appropriate models.

The second answer is articulated by Mitchell in form of "inductive bias" [25]: the task of learning a model, without additional assumptions, cannot be solved exactly. The kind of

necessary assumptions about the nature of the target function are subsumed in the phrase inductive bias. Here, the domain expert might favour or disapprove certain models, and different iCore tasks are expected to make implicit or explicit assumptions on the models to be used.

The second question is a more challenging one. An outlier might signal a malfunction of a sensor attached to an iCore instance – and this value may be skipped or further reported with a "dubious" flag attached. However, if other related or closed sensors are also providing values which are seen as outliers, this might be a signal that something unusual really happens; it is more rational to assume such an unexpected local context instead of simultaneous malfunction of a set of sensors. The CVO which receives such outlier signal may decide to further perform an extended request towards the closest/related sensors, in order to have a broader view and to decide what action should be followed.

In context of sensor data, a hand-crafted approach for outlier detection is to specify the set – e.g. an interval - of allowable values. A more elaborated approach is to detect the data abnormality by comparing the current values with the recorded history. This path is a more elaborated one, and requires statistical methods or adaptive (learning) abilities. Note that in this case one may declare some values as being anomalous, even they are not outside a range. This is accomplished based on the current context, automatically deduced through data aggregation or machine learning. In this section we present both statistical and ML-based approaches.

The data may be analysed for outlier detection either by taking into account the temporal feature (i.e. the chronological order of data) or by neglecting it. The later approach starts from a bulk of data which are considered as being representative for a "normal" behaviour.

This later approach is the simplest one: the current recording is compared with some historical values. One may use statistical based approaches (either parametric models, e.g. Gaussian-based, or non-parametric models, e.g. histogram and kernel functions), nearest neighbours based approaches, clustering and classification based approaches, spectral decomposition based methods (mainly starting from Principal Component Analysis) [17].

From these techniques, those performing online adaptation, without supervision, with low computational demands, and with auto configuration ability are the most preferred ones; other desiderata can be found in [17].

A simple statistical based approach is based on interquartile range (IQR). An input feature (measurement) *x* which fulfils:

$$Q_3 + OF \cdot IQR < x \leq Q_3 + EVF \cdot IQR$$

or

$$Q_1 - EVF \cdot IQR \leq x < Q_1 + OF \cdot IQR$$

triggers flagging as outlier the whole record to whom it belongs to. Q_1 and Q_3 are 25% and 75% percent quartile, respectively, computed based on a previously recorded dataset. *IQR* is computed as $IQR = Q_3 - Q_1$, and OF and EVF are outlier and extreme values factors (coefficients), respectively ($OF \leq EVF$) . An implementation based on the Weka library is provided in Appendix 0.

A ML based approach for outlier detection uses the so-called one-class classification technique [26]. Unlike binary or multiclass classification tasks, here the model is provided with data considered as "normal"; later patterns are confronted with the derived model and labelled as outlier or normal data. One popular approach is presented by Schölkopf in [19] starting from

Support Vector Machines with kernel methods. It basically separates all the data points from the origin and maximizes the distance from a hyper plane to the origin. The model learns a binary function which labels as positive examples the training data and those patterns which are closed to it (that is, a small region around the training dataset), and negative cases elsewhere.

Weka code using LibSVM library is provided in Appendix 0. Alternative approaches can be obtained by using the `weka.classifiers.meta.OneClassClassifier` meta-classifier, described in [19].

For temporal data there is some recent work surveyed in [21]. Data type facets (time series, data streams, distributed, spatio-temporal, and network data) are the first feature taken into account and the outlier detection algorithms are developed accordingly. The common target – both for statistically rooted methods and for non-parametric methods - to create a more consistent context against which any potential outlier is judged. The increased consistency comes from considering the time dimension, for periodic or recent past based events.

Note that models for outlier detection are continuously developed. For example, the recent work in [20] describes an outlier isolation strategy which does not employ any distance or density measure, and [21] introduces an algorithm which estimates the degree of outlierness. Different data domains in outlier analysis typically require dedicated techniques of different types. In iCore context, this variance is supported by deploying customized outlier detection mechanisms into VO back-ends.

8.3 Missing value estimation

While gathering data from ICTs, it is reasonable to expect episodic missing values, e.g. due to sensor malfunction, or temporarily broken link between ICT and iCore instance. This issue might be solved by iCore in at least two ways:

- Let the CVO level to request the VO registry to find a different VO, whose corresponding ICT provides the values of interest
- Allow the VO to make some estimation of the missing values and forward these estimations to the consumer(s).

For the later approach, this section sketches some missing value estimation (imputation) methods. Note that, as in case of anomaly detection, the algorithms may be hosted by the backend component of the VO. As described here, this functionality is accomplished individually, independently of other VO instances.

The literature devoted to this subject is impressive; see for example [28-30]. In [28] and [29] statistically rooted methods are detailed. Alternatively, multiple ML-based frameworks were proposed: the Bayesian framework and direct function approximation are among them.

For example, the Bayesian framework allows one to specify a model as family of probability distributions and assigning prior probability to every family member. Once the current sample with missing value(s) *y* is received, the posterior predictive distribution is computed and its result is used as an estimation of *y*. The probability distributions can be data-driven (deduced based on the previously recorded data) or influenced by the domain expert, which can dictate the prior probability. The maximum a posteriori predicts the missing value based on the distribution with the highest posterior probability, from a set of candidate probability distributions. Finally, the direct function approximation framework works by trying to directly estimate the missing value, based on some data-driven derived functions. Artificial Neural Networks are some popular choices [31-33].

In the following we briefly describe the expectation maximization (EM) approach for missing value estimation.

For missing value estimation, the expectation maximization (EM) approach was often successfully used [34-37]. EM was originally conceived as an estimation method for the parameters of a statistical model. It loops over a pair of steps which alternately infers an approximation function for the expectation of the log-likelihood function; then it runs a maximization step which adjusts the parameters for the inferred function. Appendix B contains code snippet based on the Weka library.

8.4 Virtual Sensing

With more number of embedded devices deployed every day, a large chunk of virtual objects will comprise of sensor nodes. For easy installation and management, these sensors are mostly wireless. With wireless benefit, wireless sensor nodes come with the limitations of limited resources. By resource, we mean memory, processing power and energy. As a wireless sensor node operates on battery power, energy is a very precious resource. Changing battery of a device every other day is quite annoying and sometimes not easily feasible. As a result, lot of efforts have made to reduce energy consumption for a wireless sensor device.

“Virtual Sensing” (Sarkar, Rao, & Prasad, 2014) is technique that aims to reduce energy consumption is a sensor node. These are inherent spatial and temporal correlation among the data produced by various sensors deployed in a region. Spatial correlation refers to the correlation in data produced by two sensors located at two different locations (possible near-by). On the other hand, temporal correlation refers to the correlation in the data produced by the same sensor over a period.

“Virtual Sensing Framework” creates “Virtual Sensor” for each physical sensor (wireless sensor node) at the VO level. A virtual sensor can predict the sensed value of physical sensor without actually fetching the value from it. The virtual sensor exploits the spatio-temporal correlation of sensor data from the deployed region to predict the sensed data accurately. As the virtual sensor and its virtually sensed data is available at the VO level, the CVOs can access the data easily. On the other hand, the physical sensor can reduce its activity in collaboration with its corresponding virtual sensor to reduce its energy consumption. The virtual sensor framework, decides which virtual sensors can accurately (or with certain error bound) predict the sensed data. The corresponding physical sensors are put to dormant state (low-power sleep mode). By putting the physical sensors in dormant state, the virtual sensing framework can conserve energy in sensor nodes. In the meantime, if any CVO tries to access the sensed data, the VO provides the virtually sensed data.

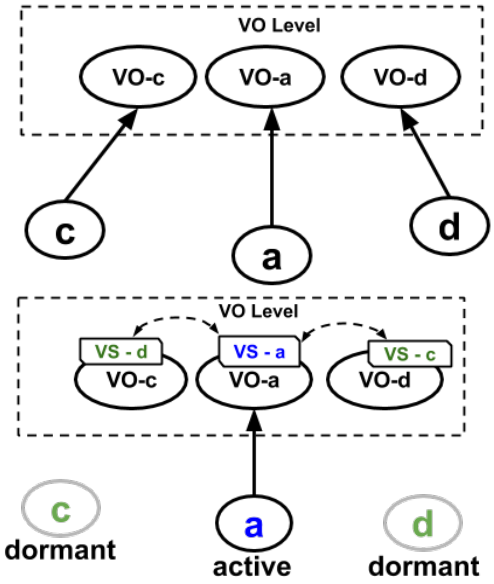

Figure 26. Data collection scenario with Virtual Sensing Framework

An example of how virtual sensing framework works is shown in Figure 26. The sensed data from sensor a, d & c is exposed through their corresponding VO at the VO level. The framework creates virtual sensors against each physical sensor at the VO level. It then decides sensor c & d (for example) can be kept dormant at this moment. By keeping them dormant, a lot of energy can be saved. Their sensed values are predicted by the corresponding virtual sensor with the help of another spatially correlated sensor (sensor a) and their own past values (temporally correlated data). To balance the energy consumption, the role of active and dormant sensor is swapped after sometime. Please note that, if the data produced by the sensors shows very high correlation, the predicted data can be more accurate. At the same time, more number of sensors can be kept dormant; thus a higher energy savings can be achieved. On the other hand if there is very lower data correlation among various sensor nodes, limited number of sensors can be kept dormant (also for smaller duration). As a result, energy saving can be less.

Virtual sensing framework optimize (reduce) the energy usage at the physical sensor nodes without affecting the upper level components of the system (iCore components). This transparency helps to decouple the application development from energy optimization. The Virtual sensing framework is provided as an optimization tool-box for the VO level. It requires data streams (physically sensed data over a period) from the sensors in a region. Then, it creates virtual sensors corresponding to each physical sensor. It automatically decides which physical sensor can be put to dormant state and when. Then, it predicts the sensed values for the dormant sensors. Additionally, the toolbox provides method to predict the missing sensor value of a sensor. A detailed description of the virtual sensing technique can be found in (Sarkar, Rao, & Prasad, 2014).

9 Conclusions

In this deliverable we have described and presented the key mechanisms related to the iCore VO level. At the core of this deliverable we presented VO level specifications, VO registry specifications, VO factory and life-cycle descriptions and VO interface descriptions. The work related to generating these outputs builds on the architecture work done earlier and in the specific requirements for VOs. Along with these key components we presented special features for enabling cognitive management on VO level by using machine learning. The important north-bound interface definition towards the CVO level was also presented with options for RESTful HTTP, SOAP, MQTT and CoAP protocols.

A number of these mechanisms will be demonstrated as proof-of-concept which will be reported in Deliverable D3.4.

10 References

[1] J. W. Walewski, Ed., *D1.2 – Initial Architectural Reference Model for IoT,* IoT-A Public Deliverable, 2011.

[2] GeoNames. (2011), GeoNames ontology. Available: http://www.geonames.org/ontology/documentation.html

[3] W3C. (2011), W3C SSN Incubator Group Report. Available: http://www.w3.org/2005/Incubator/ssn/wiki/Incubator_Report

[4] OMG SySML. Library for Quantity Kinds and Units: schema, based on QUDV model. *1.2*. Available: http://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu

[5] Botts, M., Robin, A.: OpenGIS Sensor Model Language (SensorML) Implementation Specification

[6] Semantic Sensor Network, http://www.w3.org/2005/Incubator/ssn/charter

[7] Russomanno, D.J., Kothari, C.R., Thomas, O.A.: Building a Sensor Ontology: A Practical Approach Leveraging ISO and OGC. In: The 2005 International Confer- ence on Artificial Intelligence, Las Vegas, NV (2005)

[8] Nati, Michele, Gluhak, Alexander, Abangar, Hamidreza, Meissner, Stefan and Tafazolli, Rahim. "A Framework for Resource Selection in Internet of Things Testbeds.." Paper presented at the meeting of the TRIDENTCOM, 2012

[9] GitHub Developers. GitHub API v3. [Online]: http://developer.github.com/v3/ . Accessed: March, 2013.

[10] W3C Naming and Addressing : Uniform Resource Identifier (URI). [Online]: http://www.w3.org/Addressing/ . Accessed: March, 2013.

[11] Resource Description Framework (RDF). [Online]: http://www.w3.org/RDF/. Accessed: March, 2013.

[12] SPARQL. [Online]: http://www.w3.org/TR/rdf-sparql-query/. Accessed: March, 2013.

[13] OGC SensorML. http://www.opengeospatial.org/standards/sensorml . Accessed: March, 2013.

[14] Tilanus, P., Ran, B., Faeth, M., Kelaidonis, D., & Stavroulaki, V. (2013, June). Virtual object access rights to enable multi-party use of sensors. In World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a (pp. 1-7). IEEE.

[15] Manish Gupta, Jing Gao, Charu C. Aggarwal, Jiawei Han, "Outlier Detection for Temporal Data: A Survey", IEEE Transactions on Knowledge and Data Engineering, VOL. 25, NO. 1, JANUARY 2013

[16] Hawkins, D., Identification of Outliers, Chapman and Hall, 1980

[17] Yang Zhang, Nirvana Meratnia, and Paul Havinga, "Outlier Detection Techniques for Wireless Sensor Networks: A Survey", IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 12, NO. 2, SECOND QUARTER 2010

[18]Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, John Platt, "Support vector method for novelty detection", Advances in neural information processing systems, Vol 12, pp. 582-588, 2000.

[19]Kathryn Hempstalk, Eibe Frank, Ian H. Witten, "One-Class Classification by Combining Density and Class Probability Estimation", Proceedings of the 12th European Conference on Principles and Practice of Knowledge Discovery in Databases and 19th European Conference on Machine Learning, ECMLPKDD2008, Berlin, 505--519, 2008.

[20]Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou, "Isolation-Based Anomaly Detection", ACM Trans. Knowl. Discov. Data 6(1), (March 2012), 39 pages.

[21]Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander "LOF: identifying density-based local outliers", in Proceedings of the 2000 ACM SIGMOD international conference on Management of data (SIGMOD '00). ACM, New York, NY, USA, 93-104, 2000.

[22]Wolpert, D.H., Macready, W.G., "No Free Lunch Theorems for Search", Technical Report SFI-TR-95-02-010 (Santa Fe Institute), 1995.

[23]Wolpert, D.H., Macready, W.G. "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation,* 1997.

[24]Wolpert, David "The Lack of A Priori Distinctions between Learning Algorithms," *Neural Computation*, pp. 1341–1390, 1996.

[25]Mitchell, T. M., "The need for biases in learning generalizations", CBM-TR 5-110, Rutgers University, New Brunswick, New Jersey, 1980.

[26]Tax, D, "One-class classification: Concept-learning in the absence of counter-examples", Doctoral Dissertation, University of Delft, The Netherlands, 2001, http://homepage.tudelft.nl/n9d04/thesis.pdf

[27]Manish Gupta, Jing Gao, Charu C. Aggarwal, and Jiawei Han, "Outlier Detection for Temporal Data: A Survey", IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 25, NO. 1, JANUARY 2013

[28]Donald B. Rubin, Roderick J. A. Little, "Statistical analysis with missing data", 2nd ed., New York: Wiley. ISBN 0-471-18386-5, 2002.

[29]Craig K. Enders, "Applied Missing Data Analysis", New York: Guildford Press. ISBN 978-1-60623-639-0, 2010

[30]Benjamin M. Marlin, "Missing Data Problems in Machine Learning", PhD Thesis, http://www.cs.ubc.ca/~bmarlin/research/phd_thesis/marlin-phd-thesis.pdf, 2008

[31]Jaisheel Mistry, Fulufhelo V. Nelwamondo, Tshilidzi Marwala, "Missing Data Estimation using Principle Component Analysis and Autoassociative Neural Networks", Journal of Systemics, Cybernetics & Informatics, vol. 7 Issue 3 p. 72-79, 2009

[32]Bahrami, J.; Kavianpour, M. R.; Abdi, M. S.; Telvari, A.; Abbaspour, K.; Rouzkhash, B, "A comparison between artificial neural network method and nonlinear regression method to estimate the missing hydrometric data", Journal of Hydroinformatics, vol. 13 Issue 2 p. 245-254, 2011

[33]Gupta, Amit; Lam, Monica S., " Estimating missing values using neural networks", Journal of the Operational Research Society, vol. 47 issue 2 p. 229-238, 1996

[34]Karmaker, A.; Kwek, S.; , "Incorporating an EM-approach for handling missing attribute-values in decision tree induction," Fifth International Conference on Hybrid Intelligent Systems (HIS'05)

[35]Deng, Jing; Huang, Biao, "Bayesian method for identification of constrained nonlinear processes with missing output data", Proceedings of the American Control Conference p. 96-101, 2011

[36]Karmaker, A.; Salinas, E.A.; Kwek, S.; ed. Arabnia, H.R.; Quoc-Nam Tran, " EMMA: an EM-based Imputation Technique for Handling Missing Sample-values in Microarray Expression Profiles", Proceedings of the 2011 International Conference on Bioinformatics & Computational Biology. BIOCOMP 2011.

[37]Kumar, R.Kavitha; Chadrasekar, R. M., " MISSING DATA IMPUTATION IN CARDIAC DATA SET (SURVIVAL PROGNOSIS)", International Journal on Computer Science & Engineering, 2010

[38]JSON-LD 1.0, retrieved 7th of March 2014: http://www.w3.org/TR/json-ld/#interpreting-json-as-json-ld

[39]RFC2616 "Hypertext Transfer Protocol -- HTTP/1.1", retrieved on 7th of March 2014: http://tools.ietf.org/html/rfc2616

[40]RFC5988 "Web Linking", retrieved 7th of March 2014: http://tools.ietf.org/html/rfc5988

[41]RFC6690 "Constrained RESTful Environments (CoRE) Link Format", retrieved on 7th of March 2014: http://tools.ietf.org/html/rfc6690

[42]iCore Deliverable D2.4 "iCore System Governance Model".

11 List of Acronyms and Abbreviations

API	Application Programming Interface
CoAP	Constrained Application Protocol
CVO	Composite Virtual Object
DoW	Description of Work
FP7	Seventh Framework Programme
HTTP	Hypertext transfer protocol
ICT	Information and Communications Technologies
IoT	Internet of Things
IoT-A	Internet of Things Architecture
ML	Machine Learning
MQTT	Message Queue Telemetry Transport
NBIF	North-bound interface
OWL	Web Ontology Language
RDF	Resource Description Framework
REST	Representational State Transfer
RWO	Real World Object
SecKit	Security Toolkit
VE	Virtual Entity
VO	Virtual Object
WP	Work Package

Appendix A

Two message sequence charts regarding the VO level are presented in Figure 27 and Figure 28. These charts show the components and actions related to installing a VO and accessing data of a VO.

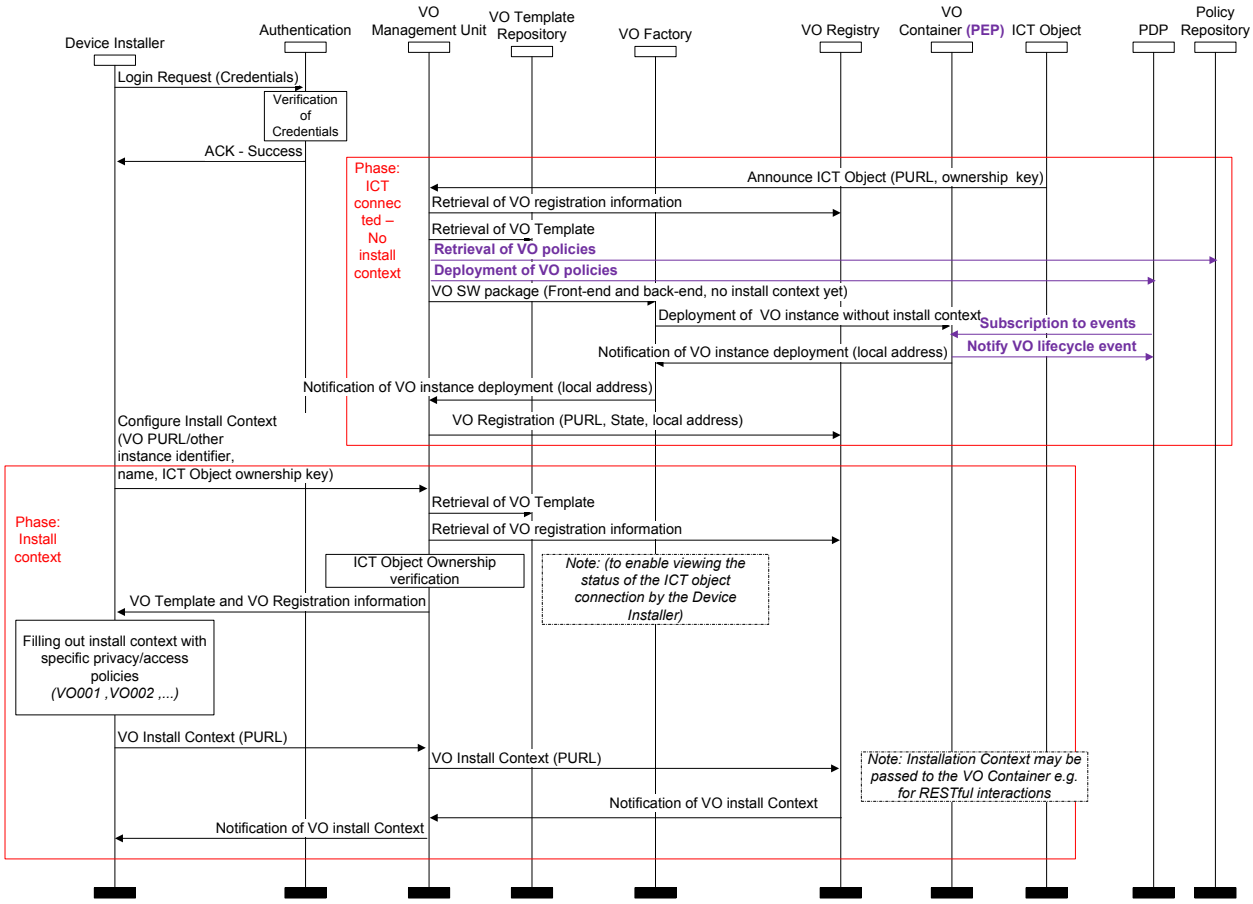

Figure 27. VO installation procedure message sequence chart

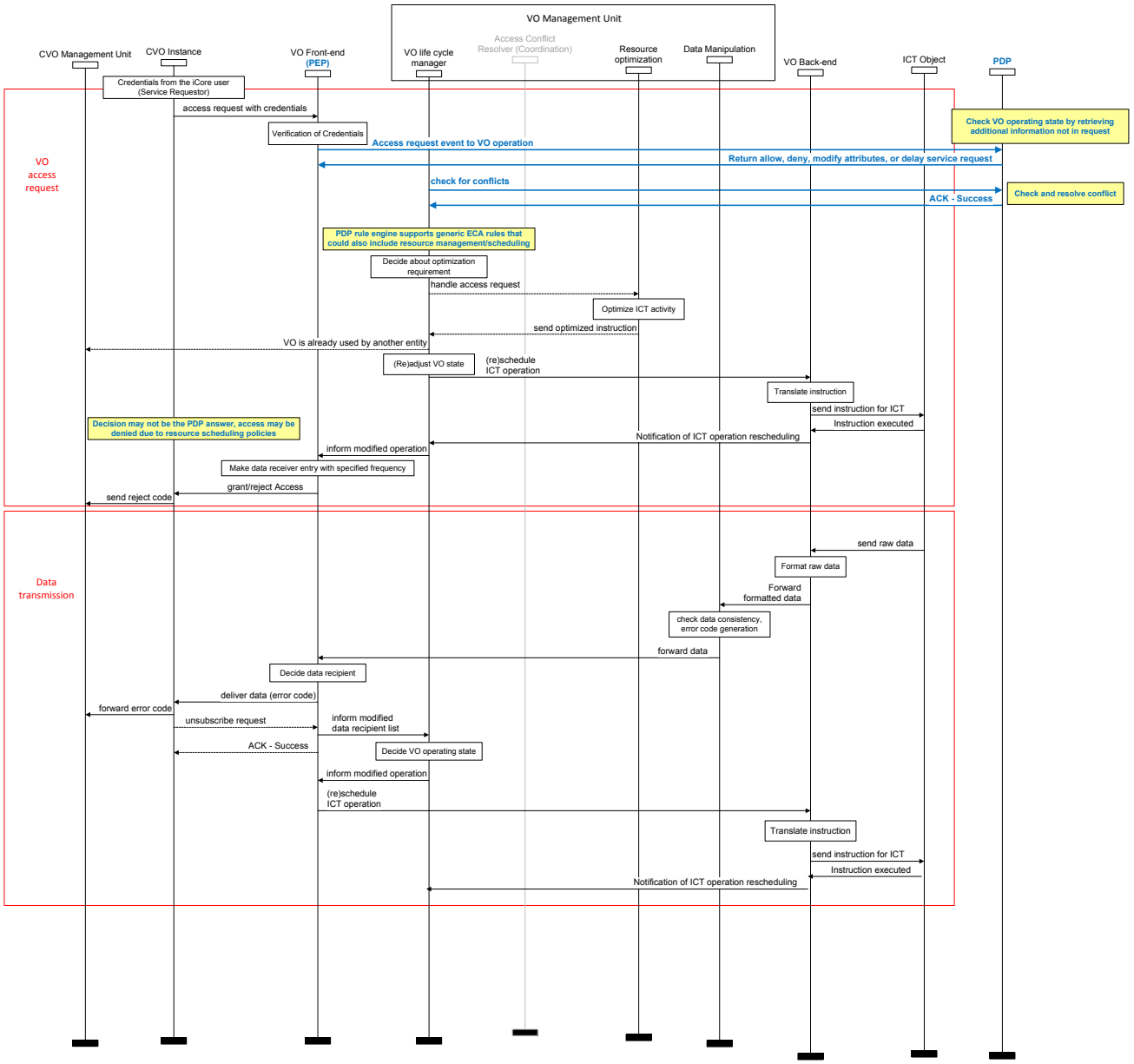

Figure 28. VO data flow message sequence chart

Appendix B

Three examples of statistical and ML-based algorithms for VO level proof of concepts.

Interquartile range-based outlier detection

The code below uses Weka's InterquartileRange filter to detect values which fulfils $Q_3 + OF \cdot IQR < x \leq Q_3 + EVF \cdot IQR$ or $Q_1 - EVF \cdot IQR \leq x < Q_1 + OF \cdot IQ$ for at least one feature *x*. Q_1 and Q_3 are 25% and 75% percent quartile, respectively, $IQR = Q_3 - Q_1$, OF and EVF are outlier and extreme values factors, respectively. In the code below, any extreme factor is also considered an outlier.

```
package eu.iot_icore.outlierDetection;


import weka.core.Instances;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.InterquartileRange;


public class IQROutlierDetection {
	/**
	 * @param instances the data to be processed
	 * @param outlierFactor threshold
	 * @param extremeValueFactor  threshold
	 * @return the original dataset + two new columns: "Outlier" (yes/no), "Extreme value" (yes/no); extreme values are also seen as outliers
	 * @throws Exception if the inputFormat can't be set successfully or the interquantile filter can't be used successfully
	 */
	public Instances markOutliers(Instances instances, double outlierFactor, double extremeValueFactor) throws Exception {
		InterquartileRange filterInterquantileRange = new InterquartileRange();
		filterInterquantileRange.setAttributeIndices("first-last");
		//a whole pattern is marked as outlier, if needed
		filterInterquantileRange.setDetectionPerAttribute(false);
		//also consider extreme values as outliers
		filterInterquantileRange.setExtremeValuesAsOutliers(true);
		filterInterquantileRange.setOutlierFactor(outlierFactor);

	filterInterquantileRange.setExtremeValuesFactor(extremeValueFactor);
		filterInterquantileRange.setInputFormat(instances);
		Instances newInstances = Filter.useFilter(instances, filterInterquantileRange);
		return newInstances;
	}
}
```

One class classification with SVM

The code below uses Weka's SVM to produce a classifier which learn from provided data and is further able to label data as "normal" or "outlier".

```
package eu.iot_icore.outlierDetection;


import weka.classifiers.AbstractClassifier;
import weka.classifiers.functions.LibSVM;
import weka.core.Instances;


public class SVMOutlierDetection {


	/**
	 * @param allOptions SVM specific options
	 * @param data the target data
	 * @return a clasifier which can label new patterns as "normal" or
"outlier"
	 * @throws Exception if parsing of the options fails or
	 */
	public AbstractClassifier getClassifier(String allOptions, Instances
data) throws Exception
	{
	  LibSVM classifier = new LibSVM();
	  String[] options = allOptions.split(" +");
	  classifier.setOptions(options);
	  classifier.buildClassifier(data);
	  return classifier;
	}
}
```

Missing value imputation through Expectation Maximization

```
package eu.iot_icore.missingValueImputation;

import weka.core.Instances;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.EMImputation;

public class EMMissingValueImputation {

	/**
	 * @param input data with missing values
	 * @return original data + estimations for missing values
	 * @throws Exception if the input format can't be set successfully for
the EM filter
	 */
	public Instances imputeMissingValues(Instances data) throws Exception
	{
		EMImputation filterMissingValueImputation = new EMImputation();
	filterMissingValueImputation.setInputFormat(data);
	Instances result = Filter.useFilter(data, filterMissingValueImputation);
	return result;
	}
}
```

Appendix C

WSDL example for SOAP style interactions.

```
<wsdl:types>
      <xs:schema xmlns:ns="http://www.iot-icore.eu/framework/cvo/eventGateway/1.0"
                 attributeFormDefault="qualified" elementFormDefault="qualified"
                 targetNamespace="http://www.iot-icore.eu/framework/cvo/eventGateway/1.0">

         <xs:element name="notifyEvent">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="eventName" nillable="true" type="xs:string"/>
                  <xs:element minOccurs="0" name="baseServiceID" nillable="true" type="xs:string"/>
                  <xs:elementminOccurs="0" name="baseServiceInstanceID" nillable="true" type="xs:string"/>
                  <xs:element minOccurs="0" name="compositionSessionID" nillable="true" type="xs:string"/>
                  <xs:element maxOccurs="unbounded" minOccurs="0" name="sessionProperties" nillable="true"
                           type="ns1:SessionProperty"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>

         <xs:element name="notifyEventResponse">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="return" type="xs:int"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
        <xs:schema xmlns:ax211="http://orchestrator.m3s.icore/xsd"
                  attributeFormDefault="qualified" elementFormDefault="qualified"
                  targetNamespace="http://orchestrator.m3s.icore/xsd">
         <xs:complexType name="SessionProperty">
            <xs:sequence>
               <xs:element minOccurs="0" name="propertyName" nillable="true" type="xs:string"/>
               <xs:element minOccurs="0" name="propertyValue" nillable="true" type="xs:string"/>
            </xs:sequence>
         </xs:complexType>
      </xs:schema>
   </wsdl:types>

   <wsdl:message name="notifyEventRequest">
      <wsdl:part name="parameters" element="ns0:notifyEvent"/>
   </wsdl:message>
   <wsdl:message name="notifyEventResponse">
      <wsdl:part name="parameters" element="ns0:notifyEventResponse"/>
   </wsdl:message>
   <wsdl:portType name="CVOPortType">
      <wsdl:operation name="notifyEvent">
         <wsdl:input message="ns0:notifyEventRequest" wsaw:Action="urn:notifyEvent"/>
         <wsdl:output message="ns0:notifyEventResponse" wsaw:Action="urn:notifyEventResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="CVOHttpBinding" type="ns0:CVOPortType">
      <http:binding verb="POST"/>
      <wsdl:operation name="notifyEvent">
         <http:operation location="CVO/notifyEvent"/>
         <wsdl:input>
            <mime:content type="text/xml" part="notifyEvent"/>
         </wsdl:input>
         <wsdl:output>
            <mime:content type="text/xml" part="notifyEvent"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="CVO">
      <wsdl:port name="CVOHttpport" binding="ns0:CVOHttpBinding">
         <http:address location="http://localhost:8080/axis2/services/Orchestrator"/>
      </wsdl:port>


   </wsdl:service>
```