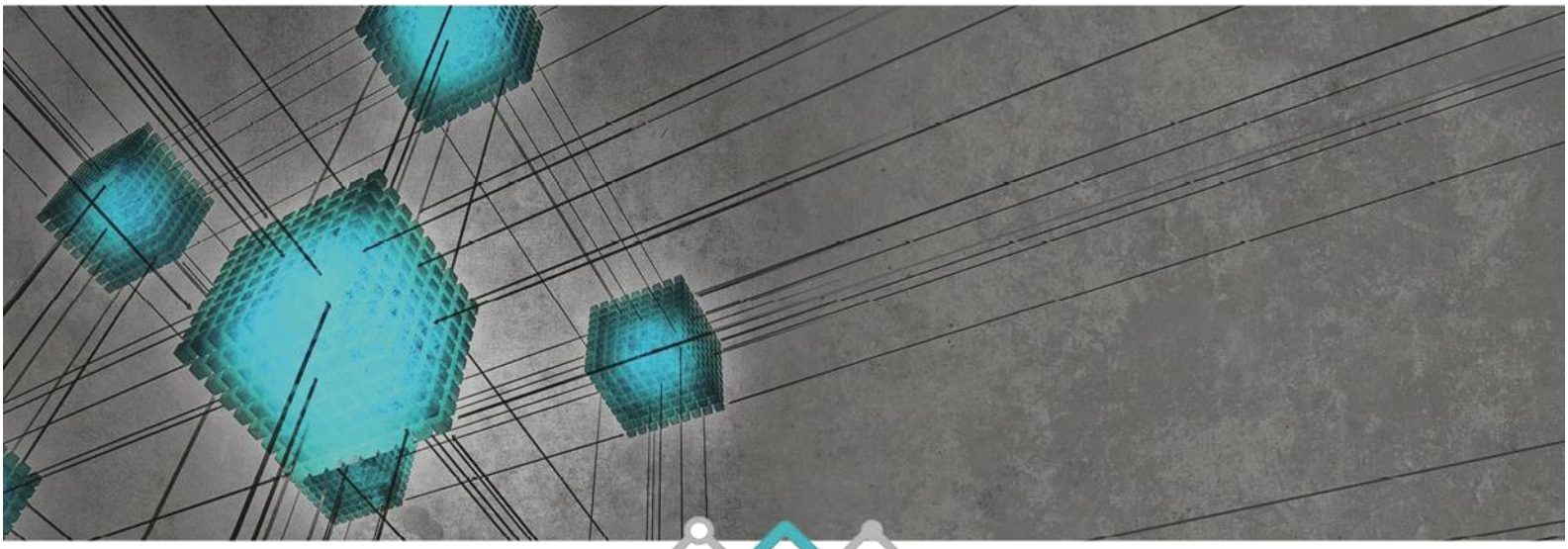


interiot

INTEROPERABILITY
OF HETEREOGENEUS
IOT PLATFORMS.

Deliverable D4.1

Initial Reference IoT Platform Meta-Architecture and Meta Data Model

15/01/2017

INTER-IoT

INTER-IoT aim is to design, implement and test a framework that will allow interoperability among different Internet of Things (IoT) platforms.

Most current existing IoT developments are based on “closed-loop” concepts, focusing on a specific purpose and being isolated from the rest of the world. Integration between heterogeneous elements is usually done at device or network level, and is just limited to data gathering. Our belief is that a multi-layered approach integrating different IoT devices, networks, platforms, services and applications will allow a global continuum of data, infrastructures and services that can will enable different IoT scenarios. As well, reuse and integration of existing and future IoT systems will be facilitated, creating a defacto global ecosystem of interoperable IoT platforms.

In the absence of global IoT standards, the INTER-IoT results will allow any company to design and develop new IoT devices or services, leveraging on the existing ecosystem, and bring get them to market quickly.

INTER-IoT has been financed by the Horizon 2020 initiative of the European Commission, contract 687283.

INTER-IoT

Initial Reference IoT Platform Meta-Architecture and Meta Data Model

Version: 1.0

Security: Public

12/01/2017

The INTER-IoT project has been financed by the Horizon 2020 initiative of the European Commission, contract 687283

Disclaimer

Executive Summary

The following document is the deliverable D4.1 Initial Reference IoT Platform Meta-Architecture and Meta Data Model of the INTER-IoT Project. It officially reports on the activity of the first six months in the Work Package 4 (Interoperability Framework) of the project. However, in practise, this document contains also knowledge and outcomes generated in Work Package 3 (Layer Interoperability) activities, due to the technical software engineering nature of the document.

D4.1 is the formal output of the WP4 tasks T4.1 Design of a Reference Meta-Architecture for Interoperable IoT Platforms and T4.2 Design of a Reference Meta-Data Model for Interoperable IoT Platforms. It also coincides with project milestone MS4 Initial Architecture Release, which is completely specified between documents D3.1 and D4.1.

This document is divided in 7 sections. Section 1 presents an initial introduction with the project purpose and background.

Section 2 describes the approach followed in the works related to this report, mainly, but not exclusively, based on the IOT-A proposed approach for creating reference architectures and following a similar approach to describe the reference model and other perspectives and views relevant in this stage of the project.

Section 3 defines the INTER-IoT Reference Model, which constitutes a novel proposal for the IoT Platforms Interoperability, reflecting not only architectural elements of the different kind of systems that can be found in the IoT domain but also canonical interoperability means frequently used to make heterogeneous systems work together. Thus, a model for the architectural components specified in the next sections is previously defined.

Section 4 contains the INTER-IoT Reference Architecture which, based on the previously proposed Reference Model, defines the different relevant architectural views for the INTER-IoT project and concerning the cases under study. This chapter is divided in 3 subsections: the functional view specification, which reviews the IOT-A definition of this view, analyses this perspective in 15 selected existing platforms and finally proposes a functional view for INTER-IoT, which is a base for the INTER-LAYER and INTER-FW specifications. Relation with solutions proposed in D3.1 is also discussed in this subsection. Additionally, it contains a subsection for other relevant view such as the information view. A third subsection is devoted for the different architectural perspectives, which will be better defined during the second year of the project and consequently reported in D4.2, the final version planned for this document.

The section 5 analyses and relates the resulting reference architecture with the INTER-LAYER solution, specified in D3.1. It shows the relation with the model and the actual mapping of the components with the functional components identified in previous sections.

Finally, sections 6 and 7 contains references and bibliography of the whole document and annexes to improve the understanding and to give further information in some specific areas.

The contents of this document set up a baseline for the works of tasks T4.3, T4,4 and T4.5 related to the design and implementation of the INTER-FW and its API.

The D4.1 has revision planned for month 24 (D4.2) which will revise and expand the contents of this document as well as report about the works performed in the second year particularly in the tasks mentioned in the previous paragraph.

List of Authors

Organisation	Authors
Universitat Politécnica de València	Jara Suarez, Andreu Belsa, Regel Usach, Eneko Olivares, Carlos Palau
Universita' Della Calabria	Raffaele Gravina, Giancarlo Fortino
Prodevelop	Miguel Montesinos, Miguel A. Llorente, Alberto Romeu
Tesnishe Universiteit Eindhoven	Geogios Exarchakos
Valencia Port Foundation	Pablo Gimenez, Miguel Llop
Rinicom	Eric David Carlson
AFT	Moncef Semichi
Noatum	Francisco Blanquer
XLAB	Flavio Fuart, Matevz Markovic
SRIPAS	Pawel Szmeja, Katarzyna Wasielewska-Michniewska, Marcin Paprzyck
ASL TO5	Anna Dott
ABC	Alessandro Bassi
Neways	Johan Schabbink

Change control datasheet

Version	Changes	Chapters	Pages
0.1	First full draft with main contributions from all partners	1-4 + Appendixes	Made in Google Doc (no page # valida)
0.2	Syntax and light content review (SRIPAS)	Id.	Id.
0.3	Content update for different sections after content review by PRO	Id.	Id.
0.4	Section 5 added	5	Id.
0.5	Consolidated MS Word format	All	Id.
0.6	First review for Word document from UNICAL	All	Id.
0.7	Fixes for sections and general review by PRO	All	Id.
0.8	Executive summary, Introduction	1	Chapter 1
1.0	Final fixes by UPV	All	All

Contents

List of Figures

List of Tables

Acronyms

6LoWPAN	IPv6 over Low Power Wireless Personal Area Networks
AAL	Ambient Assisted Living
AIOTI	Alliance for Internet of Things Innovation
AMQP	Advanced Messaging Queuing Protocol
API	Application Programming Interface
ARM	Architectural Reference Model
AS2AS	Application&Services-to-Application&Services
BO	Base Ontology
CEP	Complex Event Processing
CM	Communication Model
CoAP	Constrained Application Protocol
CRM	Customer Relationship Management
D2D	Device-to-Device
DDoS	Distributed Denial of Service
DERI	Digital Enterprise Research Institute
DM	Domain Model
DS2DS	Data&Semantics-to-Data&Semantics
DUL	DOLCE+DNS Ultralite
EPL	Eclipse Public License
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
EXI	Efficient XML Interchange
FC	Functional Component
FD	Functional Decomposition
FG	Functional Group
FPAI	Flexible Power Application Infrastructure
GE	Generic Enabler (FIWARE)
GSN	Global Sensor Network
HTTP	Hypertext Transfer Protocol
ICT	Information and Communication Technologies
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
INTER-FW	Inter Framework
INTER-FW	INTER-IoT Interoperable IoT Framework
INTER-LAYER	INTER-IoT Layer integration tools
IoT	Internet of Things
IOT-A	Internet of Things - Architecture

IPSM	Inter Platform Semantic Mediator
ISO	International Organization for Standarization
ITU	International Communications Union
JSON	JavaScript Object Notation
KNX	Konnex
LSM	Linked Stream Middleware
LWM2M	Lightweight M2M
M2M	Machine-to-Machine
MAC	Media Access Control address
MQTT	Message Queuing Telemetry Transport
MW2MW	Middleware-to-Middleware
N2N	Network-to-Network
NGSI	Next Generation Service Interface
OASIS	Organization for the Advancement of Structured Information Standards
OGC	Open Geospatial Consortium
OMA	Open Mobile Alliance
OPC	OLE for Process Control
OSGi	Open Services Gateway initiative
OSI	Open Systems Interconnection
OWL	Web Ontology Language
PHY	Physical OSI Layer
RA	Reference Architecture
RDF	Resource Description Framework
RM	Reference Model
SAML	Security Assertion Markup Language
SAREF	Smart Appliances Reference
SDK	Software Development Kit
SDN	Software Defined Networking
SDO	Standard Development Organisation
SDR	Software Defined Radio
SE	Specific enabler (FIWARE)
SIOC	Semantically Interlinked Online Communities
SOA	Service Oriented Architecture
SOTA	State of the Art
SSL	Secure Sockets Layer
SSN	Semantic Sensor Network
TLS	Transport Layer Security
UDP	User Datagram Protocol
W3C	World Wide Web Consortium
WGS84	World Geodetic System 84

X-GSN	Extended Global Sensor Network
XML	Extensible Markup Language
XSD	XML Schema Definition

1 Introduction

Interoperability is a big challenge identified recurrently by stakeholders concerning the IoT ecosystems and the future internet trends. The value of making software systems compatible and future proof is undeniable. However, the actual market shows that the number of solutions grows quickly, and none standard and/or generic platform seems to be dominant, stimulating the development of new systems in an, at least for now, unstoppable process. The reasons of the increasing number of IoT oriented solutions are not exclusively limited to the lack of predominance of standards or products, there are also other likeable reasons, such as the relative novelty of the IoT systems, especially for the consumer market and in some industrial environments; or simply the distributed and multi-domain nature of these systems, which allows a plethora of use cases and scenarios difficult to harmonize under a single specification.

On the other hand, the open IoT platforms are emerging slowly and most of those which are publicly available and usable, are close to the prototyping phases. This prevents to show a clear prevalence of solutions in the open segment and also discourages the massive adoption of open solutions in the IoT ecosystems.

INTER-IoT is initially intended to close the gap between open IoT platforms and make them fully interoperable at different levels. However, given the huge fragmentation in the existing market and being a project with real applications planned in a short term, the inclusion of commercial platforms was recommended since the very beginning, allowing to cover the theoretical and academic aspects of the solutions and, on the other hand, work with production systems, tying the results of the project to commercial solutions and to real scenarios with clear and tangible economic value. INTER-IoT, in other words, brings to the open IoT platforms scene the benefits of working with proven business models, and to the commercial/legacy systems the possibility of being flexible or leveraging the open source communities to innovate and improve the current applications.

In such as fragmented ecosystem, modelling is a valuable mechanism to abstract commonalities of existing platforms, to extract the main features that define the IoT domain and to build general approaches to face the interoperability in a universal way.

For this reason, INTER-IoT has defined a Reference Model (or “meta-model”) for IoT Platforms Interoperability, a Reference Architecture defined based on this model and a complete interoperability system.

A reference model is, according to OASIS[1] definition:

> A reference model is an abstract framework for understanding significant relationships among the entities of some environment. It enables the development of specific reference or concrete architectures using consistent standards or specifications supporting that environment. A reference model consists of a minimal set of unifying concepts, axioms and relationships within a particular problem domain, and is independent of specific standards, technologies, implementations, or other concrete details. A reference model may be used as a basis for education and explaining standards to non-specialists.

[1] http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

While, an architectural reference model is a term extensively used in IOT-A and defined in this document as an architectural pattern in [2]:

> Architectural reference model is a description of elements and relation types together with a set of constraints on how they may be used.

Finally, the definition of reference architecture used in this document is the following (also taken from [2]):

> A reference architecture is a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them.

In this document, a full description of the Reference Model (RM) and the Reference Architecture (RA) is given, in its initial model[2]. In addition, several relevant concepts for the architectural definition, such as the domain model or the information model, are also specified.

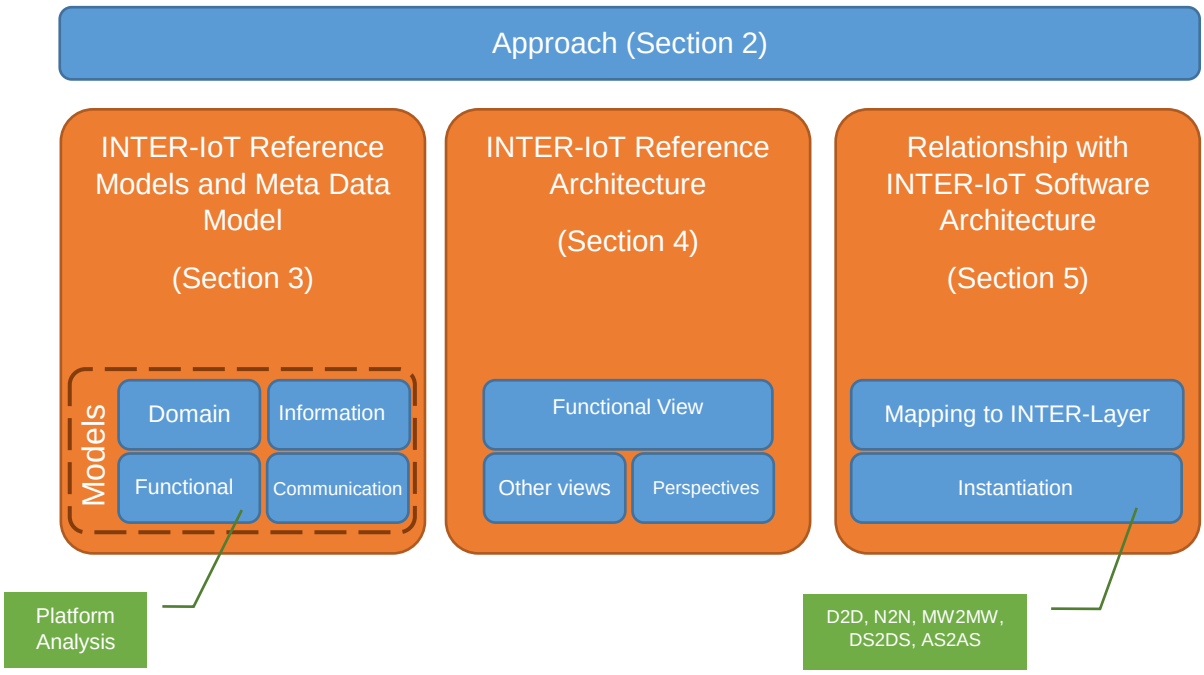


Figure 1 Structure of the document

[2] Please note that a final versión of this deliverable is planned for the end of 2017.

Other relevant perspectives and views are slightly defined or even omitted due to the initial phase of execution of the project. This is the case, for example, of the deployment view which will be defined near to the pilots' deployment, by the end of the second year of the project execution.

The use of a RM and an ARM to create a RA to instantiate a software architecture in the domain of IoT is described in IOT-A [30] and appears previously in [2]. Section 5 (See Figure 95) describes how this process has been put into practice in INTER-IoT.

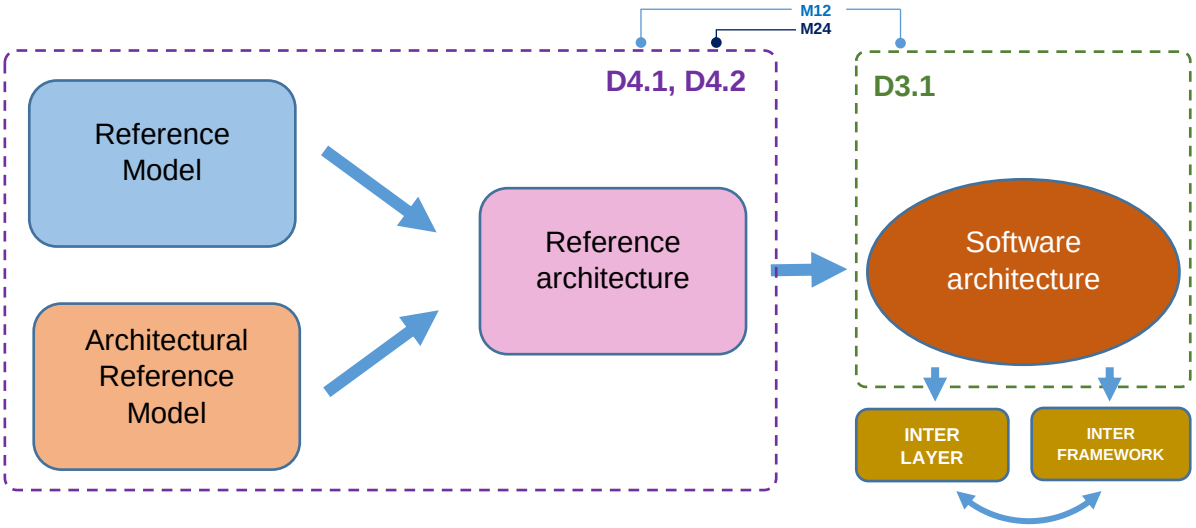


Figure 2 Relation with other documents and artefacts

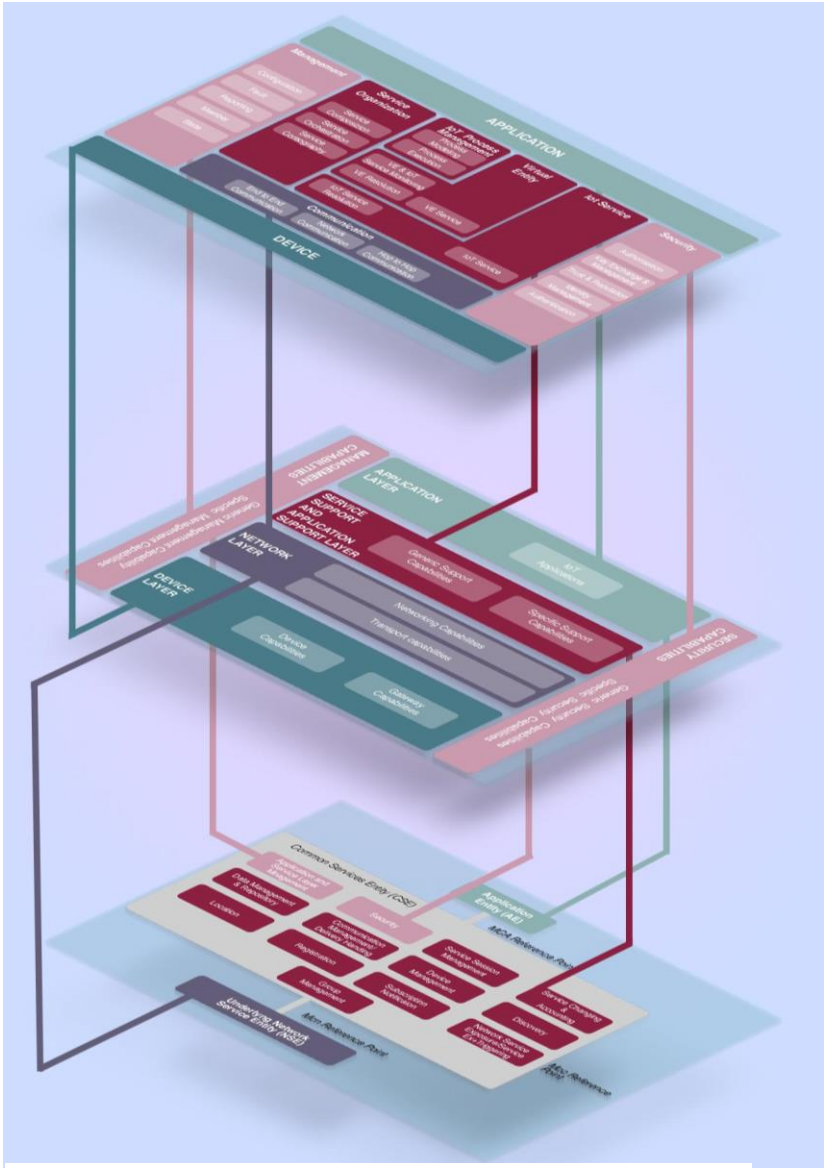
The interoperability system (INTER-Layer) is thoroughly described in INTER-IoT Deliverable D3.1 and its specification is, therefore, out of the scope of this document. The design and specification of the INTER-Framework will be started after the submission of this deliverable, and is also out of the scope of this document.

This document has a strong basis on the works done in IOT-A EU Project and a deep analysis of 16 heterogeneous IoT platforms carried out in the INTER-IoT project. The results of the latter are also reported in this document under the *Functional View* section and the actual data gathered is available in *Annexes*.

2 Approach

2.1 Introduction

There is currently a plethora of organizations trying to develop the ultimate reference models for IoT systems. It's possible to enumerate several efforts such as IoT-A[3], IEEE P2413[4], ITU-T[5], IIC[6], oneM2M[7], just to name a few.



It must be noticed, though, that despite a very diverse vocabulary, most concepts are more or less the same. For instance, in figure 1 it is possible to notice the fact that IoT-A functional model, ITU-T reference architecture and oneM2M functional architecture are quite equivalent.

The IoT-A Architectural Reference Model has been chosen as a reference for the work performed in INTER-IoT. Reasons for this choice are:

1) IoT-A is a complete and mature solution that allows to go from a use case and a number of requirements to a concrete architecture, taking into considerations different factors such as communication between devices, security, information flow, ...
2) As all architectural approaches are somehow similar, it's not time-consuming or complex to map different efforts into the IoT-A concepts.
3) As IoT-A is used as a base by most EU projects, it provides a common ground with other results in the EPI cluster.

[3] http://www.iot-a.eu/public
[4] https://standards.ieee.org/develop/project/2413.html
[5] http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx
[6] http://www.iiconsortium.org/
[7] http://www.onem2m.org/

The IoT-A provides a complete methodology for creating IoT platforms based on a reference architecture and using the characteristics of the use cases and requirements of a determined application. INTER-IoT, however, is intended to provide tools and mechanisms to achieve interoperability among existing IoT platforms. This INTER-IoT interoperability must be built as much generic as possible to provide global solutions regardless the technologies or instances (platform independency) or the field of application (domain independency). Then, while the problem domain (IoT) and a lot of concepts of IoT-A are valid and applicable to INTER-IoT, the approach is different and new meta-models must be developed. Accordingly, this document follows the terminology and the general methodology approach described in IoT-A, to define an architecture reference model and a reference architecture to create interoperability mechanisms among IoT platforms. INTER-IoT will use this framework to instantiate several mechanisms (so-called Layer Interoperability Infrastructures, LIIs) and thus validate the RA developed. However, as for its generic approach, the RA and/or the ARM could be used to generate new solutions for different interoperability problems in the future.

In the subsequent sections, the approach used to generate each of the models and views that eventually define the INTER-IoT ARM and RA is detailed.

2.2 The IoT-A[8] Architectural Reference Model

2.2.1 IoT-A Background

IoT-A [25][30] was a lighthouse EU-funded project that established an Architectural Reference Model for the Internet of Things domain. The project ran from 2010 until 2013, and can be considered the foundation for all the EU efforts done in this area since then.

IoT-A main goal was to promote a high level of interoperability between different IoT systems. This interoperability had to be developed from the communication level as well as at the service and knowledge levels across different platforms established on a common grounding. The IoT-A project developed common tools and methodologies to achieve this. While existing literature like [3] provide methodologies for dealing with system architectures (hereafter called concrete architectures) based on Views and Perspectives for instance, establishing a reference architecture is a quite different business, at least as far as describing Views and Perspectives is concerned.

An Architectural Reference Model (ARM) can be visualised therefore as the matrix that eventually derives into a large set of concrete IoT architectures. For establishing such a matrix, based on a strong and exhaustive analysis of the state of the art (SOTA), a super-set of all possible functionalities, mechanisms and protocols that can be used for building concrete architectures must be identified. Providing such a technical foundation along with a set of design choices, based on the characteristics of the targeted system based on different dimensions like distribution, security, or response time, we can then select the baseline technologies, such as protocols, functional components, or architectural options, that we need to build our INTER-LAYER / INTER-FW solutions. A usual representation of IoT-A is the "famous" IoT-A tree.

[8] http://www.iot-a.eu/public

Figure 4 The IOT-A tree

The basic concept of the picture is that IoT-A connects several baseline technologies, such as communication protocols (6lowpan, ZigBee, IPv6…) and device technologies (sensors, actuators, tags…) with an almost infinite number of application and services. The trunk of the tree represents the Architectural Reference Model, composed by the Reference Model and the Reference Architectures: the set of models, guidelines, views, perspectives, and design choices that can be used for building fully interoperable concrete domain-specific IoT architectures (and therefore systems).

2.2.2 Basic Usage of the IoT-A ARM

The ARM can be used for several purposes, from more abstract ones to more concrete developments.

2.2.2.1 Cognitive aid

At a more abstract level, such as product conception and development, an ARM can be used for different purposes.

First, it provides a roadmap for discussions, since it defines a clear language and grammar that everyone involved in the creative process can use, and which is intimately linked to the architecture, the system, the usage domain. As well, the high-level view provided in such a model is of high

educational value, since it provides a comprehensive and at the same time abstract view of the domain, helping non-technical people (or simply, people new to the IoT field) in understanding the particularities and intricacies of IoT.

Furthermore, the ARM can assist IoT project leaders in planning the work and organizing the teams needed. For instance, the Functionality Groups identified in the Functional View of the IoT system can also be a list of independent teams working on an IoT system implementation.

The ARM provides a clear guidance as well in identifying independent building blocks for IoT systems. This constitutes very valuable information when dealing with questions like system modularity, processor architectures, third-vendor options, or re-use of already developed components.

All these points show that establishing a common ground for any field is not an easy task. In this field, a common ground would encompass the definition of IoT entities and the description of their basic interactions and relationships with each other, which is the main objective of the IoT-A effort.

2.2.2.2 Generation of architectures

A major benefit of the IoT-A ARM is the capability of generating architectures for specific systems. This architecture generation is done by providing best practices and guidance for helping translating the ARM into concrete architectures. The benefit of such a generation scheme for IoT architectures is not only a certain degree of automatism of this process, and thus the saved R&D efforts, but also that the decisions made follow a clear, documented pattern.

2.2.2.3 Identifying differences in derived architectures

When using the IoT ARM-guided architecture process any differences in the derived architectures can be attributed to the particularities of the pertinent use case and the thereto related design choices [4]. When applying the IoT ARM, a list of system function blocks and data models, together with predictions of system complexity, can be derived for the generated architecture. Furthermore, the IoT ARM defines a set of tactics and design choices for meeting qualitative system requirements. All these facts can be used to predict whether two derived architectures will differ and where.

The IoT ARM can also be used in a "reverse mapping" fashion. System architectures can be cast in the IoT-A ARM language; this is what we will do in this document, analysing the different platforms and translating the different system architectures into a common language and mapping.

2.2.2.4 Achieving interoperability

While developing a concrete architecture, fulfilling a set of qualitative requirements inevitably leads to design challenges. Since there is usually more than one solution for each of the design challenges (we refer to these solutions as design choices), the IoT-A ARM cannot guarantee interoperability between any two concrete architectures a priori, even if they have been derived from the same requirement set. Nevertheless, the IoT-A ARM is an important tool in helping to achieve interoperability between IoT systems. This is facilitated by the "design choice" process itself. During this process, it's possible to identify the design choices made; comparing two different architectures, it should be clear what architecture measures must be taken to achieve interoperability and at which point in the respective systems this can best be done. Interoperability might be achieved *a posteriori* by integrating one IoT system as subsystem in the other system, or by building a bridge through which key functionalities of the respective other IoT system can be used. Notice though that these workarounds often fall short of achieving full interoperability. Nevertheless, building bridges between

such systems is typically much more straightforward than completely re-designing either system; usually doing so, a fair level of interoperability can be achieved.

2.2.3 Architecture concepts

Architectural views provide a standardized way for structuring architectural descriptions [3]. As demonstrated by [4], views can also be used for structuring reference architecture descriptions. Choosing architectural views for the development of a coherent IoT Reference Architecture showed to be instrumental for the success of the IoT ARM, as they provide an intuitive delineation of each addressed aspect.

There is not a single, commonly accepted list of architectural views; the chosen ones for this work are:

- Context view;
- Functional view;
- Information view;
- Deployment view;

As discussed in detail in [3], views do address technical aspects, while stakeholder requirements are often formulated as qualitative requirements. Their solution to this issue, which we adopt in this document, is to introduce architectural perspectives. These perspectives cut across the views. In other words, they do not replace views but provide an abstraction layer above the views.

The table summarises how the models in the IoT Reference Model relate to the views and perspectives featured in the IoT Reference Architecture.

IoT Reference Model	IoT Reference Architecture
IoT Domain Model	
IoT Information Model	Information view
IoT Functional Model	Functional View
IoT Communication Model	Communication Functionality Group (part of the functional view)

Table 1 Relation between the IoT-A Reference Model and Reference Architecture

2.3 Domain Model

The Domain Model(DM) is the first step in the creation of the reference model. In the IoT realm, the creation of a Domain Model is carried out starting from the analysis of the concepts used in the Internet of Things, like devices, things, services and so on. IoT-A's DM is a good example of this.

INTER-IoT, aims at building a reference model on the foundations defined by IoT-A, leveraging terminology and representation methodology to solve the problem of interoperability in existing IoT platforms. The reason is that traditional IoT models focus on designing a system around the Internet of Things concept, whereas in INTER-IoT we deal with making a set of IoT Platforms interoperable. This means that, although all the IoT concepts and models are valid, we need to extend them to consider the existence of different platforms.

As a matter of fact, this could be seen as a system of systems approach, with multiple platforms and an upper actor configuring an overall system, but also as a mesh of platforms that need to interchange content through a multilayer mediator. INTER-IoT allows both approaches.

Taking all that into account, we have analysed the IoT-A's Domain Model, and have checked it against the most common sensor ontologies (W3C SSN[9], IETF SAREF[10], One M2M[11], OGC Sensor Things[12]…) to check its validity.

[9] https://www.w3.org/2005/Incubator/ssn/ssnx/ssn

[10] https://sites.google.com/site/smartappliancesproject/ontologies/reference-ontology

[11] http://www.onem2m.org/

[12] http://ogc-iot.github.io/ogc-iot-api/

Once we have ensured that the IoT-A's Domain Model is valid for the INTER-IoT objectives, we have extended it to include new concepts necessary for the INTER-IoT, mainly related to its multi-platform approach.

For achieving this, we have performed several steps. First, we have reviewed the requirements of the project. Next, we have made an analysis of various IoT platforms. We have been collaborating in parallel with several IoT platform analysis tasks that have been conducted in the project. The result is a Domain Model aimed at the interoperability of IoT Platforms, suited to INTER-IoT goals.

2.4 Information Model and Meta Data Model

2.4.1 Introduction

IoT-A defines a generic model of information that passes through any IoT system. The central element of this model is a *VirtualEntity* (see Figure 5 IoT Domain Model) that has some *Attributes* with *MetaData* attached. An IoT-A *VirtualEntity* needs to have two special data elements that describe it: an *identifier*, and a type (*entityType*). Additionally, Virtual Entities may have multiple attributes, each with a name, type, and annotated values. The description of an entity, in this model, allows for multiple values of attributes, each of which may be annotated with meta-data. The annotations may go deeper, with meta-data about meta-data and so on. This description is realized in the IoT-A information model through a *ValueContainer*, an instance of which combines an attribute value and its meta-data annotations. Additionally, IoT-A defines generic classes for *Service*, *Resource* and *Device* descriptions.

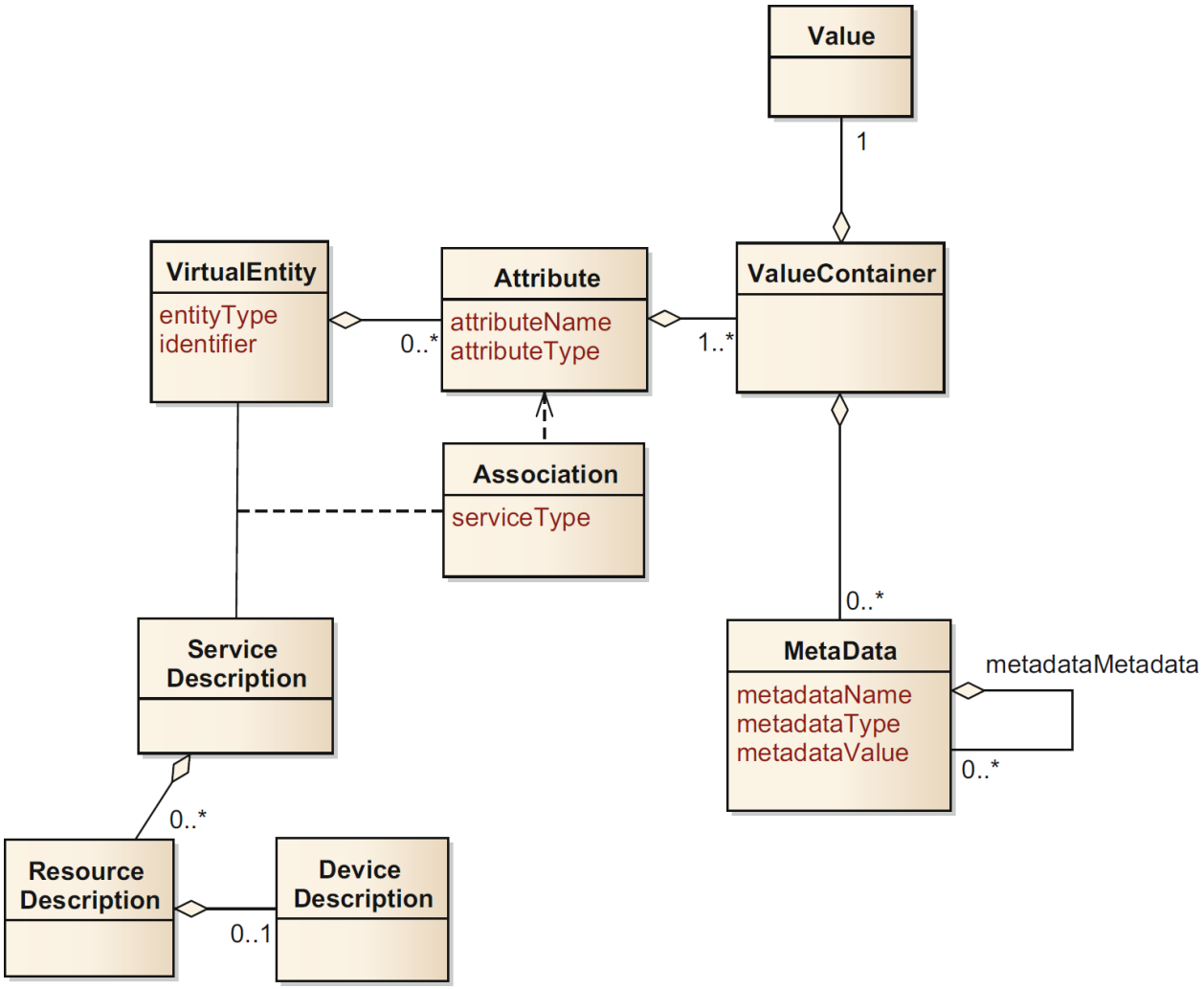


Figure 5 IoT Domain Model

This generic model can be used to model a wide variety of information. In particular, in the implementations of IoT systems there is a need to have a specific definition of what meta-data items, attributes and virtual entities, a given system operates on. In fact, the IoT-A methodology itself suggests that the definition of, for instance, what entity types are available, is left to the implementer. Following the IoT-A suggestion, of using specific schemas and models to describe available types of virtual entities, attributes and meta-data, INTER-IoT uses ontologies and semantic vocabularies to augment the information model.

The INTER-IoT reference meta-data model is a set of ontologies and documentation that is used to define specific implementations of IoT-A information model. The INTER-IoT model, in particular, describes the types of *VirtualEntities*, *Attributes* and *MetaData* for INTER-IoT understanding, and includes descriptions of *Services*, *Resources* and *Devices*, which are included, but not expanded upon in the IoT-A model. The process of creation of the INTER-IoT reference meta-data model is described in the following subsections.

2.4.2 The INTER-IoT reference meta-data model creation process

A reference meta-data model describes concepts, structures and relationships between meta-data items (i.e. data that provides information about other data). In the context of INTER-IoT the reference meta-data model describes meta-data about any entities that appear in the context of interoperable IoT platforms.

Features of meta-data include:

- Enabling data identification
- Enabling data search and retrieval
- Description of links (relationships) between objects
- managing and organizing data

Note that it is not the role of the reference meta-data model to implement any of the described mechanisms. For instance, if the model contains information about authentication mechanisms, it may inform a reader about types of authentication mechanisms and data required for authentication in each of them (e.g. user ID, email, password, checksum, or encrypted key file). It does not implement any mechanism of processing this data, or the actual authentication mechanism.

The process of defining the meta-data reference model is described in Figure 6:

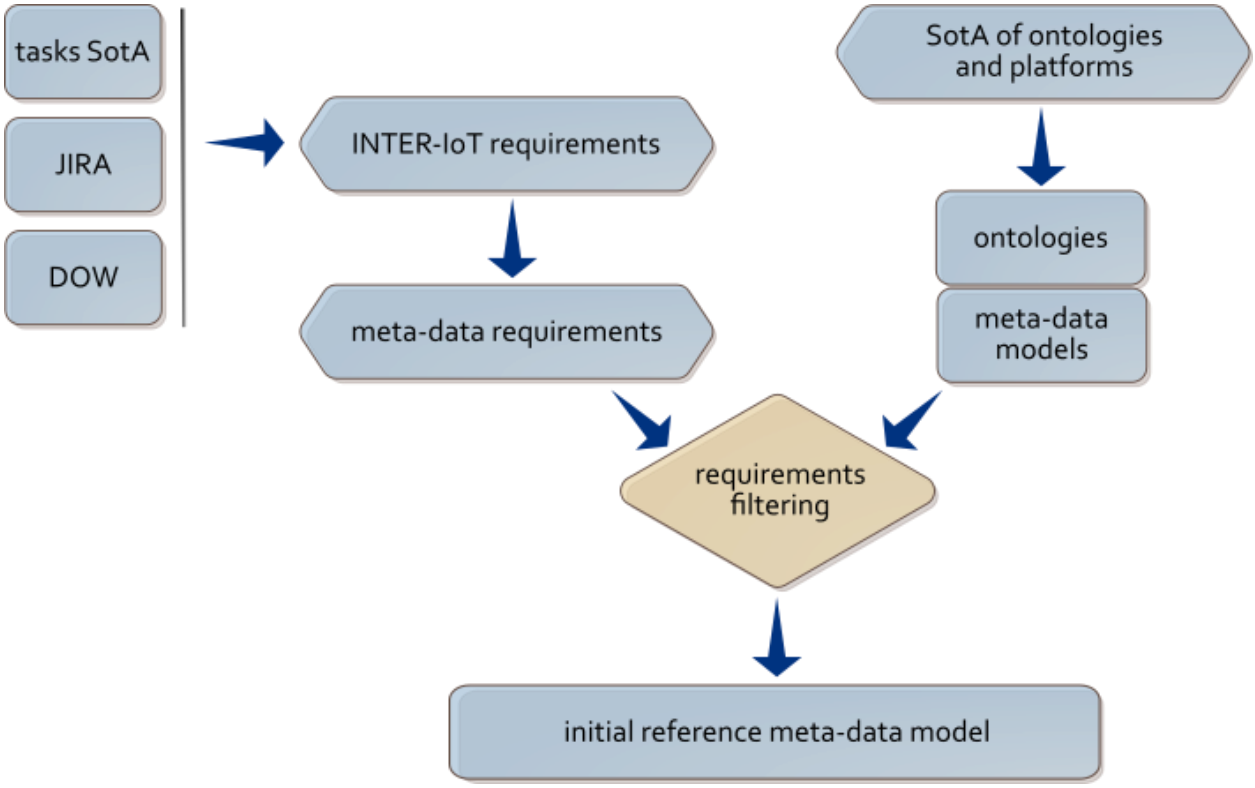


Figure 6 Creation of initial reference meta-data mode

2.4.2.1 Meta-data language

As a preliminary step, we have chosen Web Ontology Language[13] (OWL) [31] as the language for definition of meta-data items. Thus, the final model will consist of a set of OWL ontologies with supporting documentation.

OWL is a formal ontology language rooted in description logic. It is currently the de facto standard ontology language for all kinds of resources, including Linked Data[14]. OWL supports definitions of

[13] https://www.w3.org/TR/owl-features/

[14] https://www.w3.org/standards/semanticweb/data

rich taxonomies and complicated properties and relationships between entities. It is an extension of RDF[15] (Resource Description Framework) and is directly compatible with RDF processing tools and technologies.

Since OWL ontologies are machine-processable, they enable understanding in communication between both people and machines. OWL is an ontology specification language that supports multiple file formats (i.e. its semantics are independent of file format). It does not define any canonical way of implementation of models in software solutions, thus being technology agnostic and implementation-independent. Using OWL one can describe concepts and their properties, as well as concrete entities (instances). OWL ontologies are directly extendable and may be combined to form new ontologies to capture knowledge from many different domains and perspectives. In short, OWL meets the requirements of a language for the reference meta-data model specification.

OWL can be used in a technology-agnostic way. OWL files can also be used with OWL or RDF specific technologies, such as semantic reasoners, triple-stores, ontology editors, ontology alignment tools, ontology viewers, semantic IoT middleware (e.g. OpenIoT[16], UniversAAL[17] and others).

Summary of OWL features:

- Structured data description
- Easy Linked Data integration
- RDF compatibility
- Rich semantics
- Most popular ontology language
- Format and technology independent

OWL ontologies vary in size, scope, and level of detail. Some ontologies contain specialized domain-specific knowledge and, because they describe one specific issue in great detail, they are useful is specialized applications. Other ontologies are very general and inform about very basic concepts that refer to a wide range of domains. This variety is summarized in a general model of ontology modularity, and expanded upon in the specific model of ontology modularity.

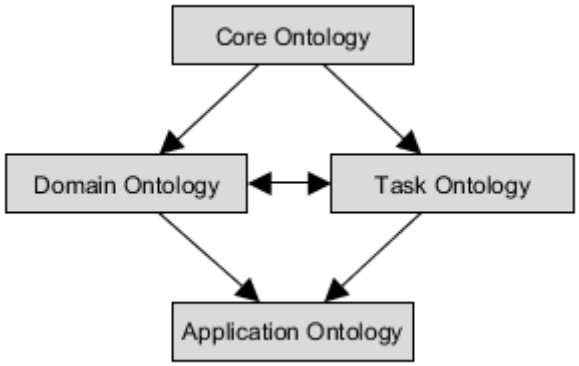


Figure 7 Model of ontology

The general model of ontology modularity (see Figure 7), proposed in [5] defines four types of ontologies and a partial ordering defined by inheritance. The most general type of ontologies, the

[15] https://www.w3.org/RDF/
[16] http://www.openiot.eu/
[17] http://www.universaal.info/

Core ontology (also called Upper) contains very general terms without specific details. It is meant to have a high level of abstraction and, thus, be reusable and widely applicable. Domain and Task ontologies inherit from the Upper ontologies and describe domain-specific knowledge (e.g. in medical domain) and specific actions (e.g. assembly of parts), respectively. These two types of ontologies combine to form description of domain specific tasks (e.g. assembly of a medical equipment). Finally, an Application ontology inherits from all the other types of ontologies to provide knowledge that stems from applying higher-level ontologies to a particular application (e.g. a deployment of a medical system). INTER-IoT reference meta-data model focuses on upper ontologies and domain ontologies specific to internet of things.

The specific model of ontology modularity expands upon the simple ordering of four types of ontologies and proposes that any ontology may be divided into either vertical or horizontal modules. In general, modules are parts of ontologies that are clearly identifiable and separable. Vertical modules form a hierarchy of inheritance. In simple terms, we may say that an upper ontology that is extended by an upper ontology, is its vertical module. A horizontal module is a part of ontology independent from other parts, except possibly by sharing a common base of inheritance. If and ontology is made by a combination of task and domain ontologies, then we may say that those are horizontal modules. Some ontologies explicitly define their modules, while others are monolithic by design. Modular ontologies encourage using only the modules that are needed. For instance, if a supply chain ontology defines multiple modules, we may use only the transportation module and disregard others e.g. one that describes push-pull supply chain characteristics. INTER-IoT reference meta-data model focuses on analysis and choice of only those modules that are relevant to avoid a bloated and unmanageable model.

2.4.2.2 Meta-data requirements

The creation of the reference meta-data model for INTER-IoT starts with defining the scope of the model. This is done in the process of extraction of meta-data requirements from other work done in INTER-IoT. Meta-data requirements are, essentially, items that together form a loosely-defined vocabulary of terms. Each item identifies a small part of the scope of the full model. The items are gathered from the following sources:

- Grant agreement document
- Relevant INTER-IoT requirements (identified in other INTER-IoT work packages)
- Explicit semantic models of IoT platforms
- IoT-EPI Task force
- Partners' expertise

The details of each source of meta-data requirements as well as its relevance to INTER-IoT is described in later sections of this document.

2.4.2.3 Available Ontologies

The second preliminary action done is the identification of state of the art when it comes to available OWL ontologies. Only ontologies relevant to IoT are considered, including those implemented in working IoT systems (e.g. OpenIoT[18]), but also ontologies that are not IoT specific, such as units of measurement ontologies. The list of identified ontologies can be found in Appendix 2.

[18] http://www.openiot.eu/

2.4.2.4 Requirements filtering

The construction of the initial INTER-IoT reference meta-data model is performed in a process called “requirements filtering” (See Figure 8). This process takes as input, previously identified meta-data requirements, and selects existing ontologies (from IoT ontologies SOTA analysis) that fulfil those requirements. The initial choice of ontologies is supported by in-depth analysis of ontologies (see section 3.3) and contains ontologies most relevant to the IoT space. The relevance was decided through analysis of requirements. Because a smart device is the central entity in IoT, ontologies that describe devices were deemed as the most important and relevant.

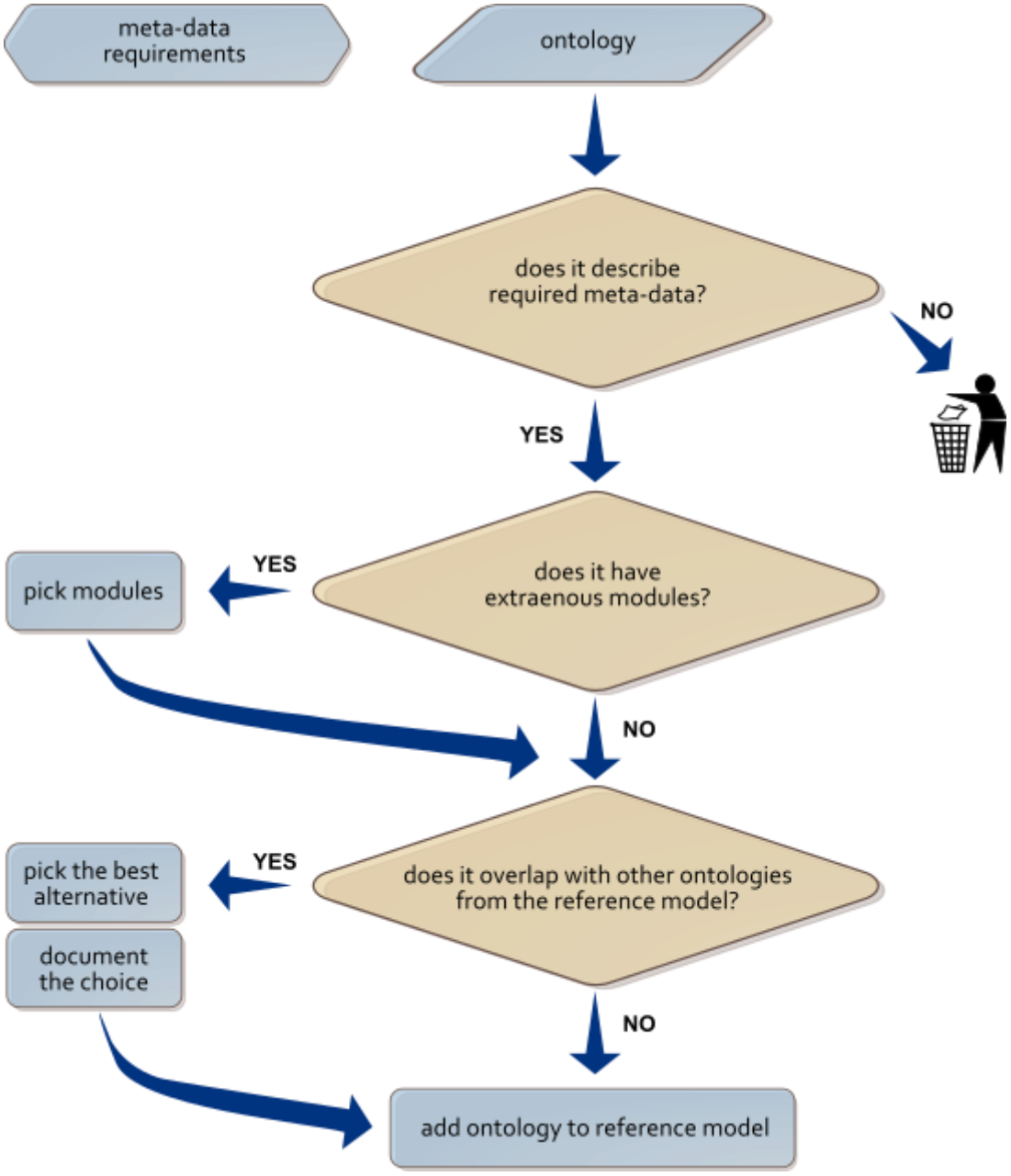


Figure 8 Merging modules (Adding an ontology, or an ontological module, to the reference meta-data model)

The final INTER-IoT reference meta-data model will be constructed in an iterative process (see Figure 9) that builds upon the initial model. Identified ontologies will be analysed and added to the model (possibly replacing ontologies already in the model) if they cover entities identified in meta-data requirements. The requirements may be modified as other INTER-IoT tasks progress and the

changes will be propagated to meta-data requirements, which in turn will prompt an augmentation of the model.

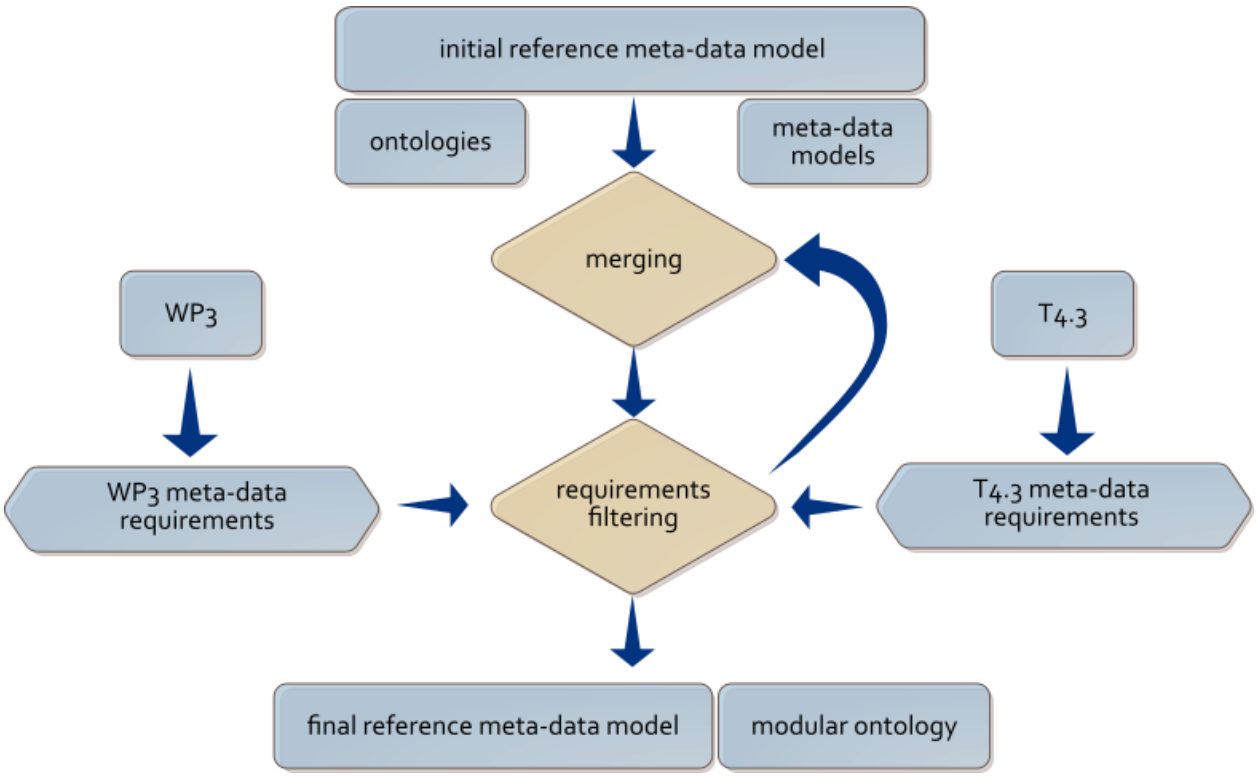


Figure 9 Creation of final reference meta-data model

2.5 Functional Model

The Functional Model, according to MacKenzie et al.[1], is defined as “an abstract framework for understanding the main Functional Groups (FG) and their interactions”. This framework defines the common semantics of the main functionalities, and will be used for the development of Functional Views.

The Functional Model is designed upon the results of the Domain Model and the Information Model. Nevertheless, we are not designing the Reference Model from scratch. It is based on the previous work done in IoT-A. The functional decomposition, made in IoT-A, generated the Functional Model and the Functional View. This means that we have reviewed the Functional Model of IoT-A. Next, we have performed an analysis of a set of IoT Platforms, from the Functional Model point of view. For doing this, we have matched the functional features of each platform against the IoT-A’s Functional Model, using the Functional Model diagram as an enabler, and we have generated a diagram for each platform. Once we collected all the diagrams, we have analysed them, detecting very different compliances that are due to the different nature of te existing platforms.

The set of IoT Platforms to be analysed was gathered from the output of the stakeholder’s analysis and the partner’s expertise. Based on this input, it was decided that a set of 16 IoT platforms was to be analysed.

Within these 16 IoT Platforms, 5 of them have been selected as the 5 IoT Platform that INTER-IoT will give support to, keeping in mind that platform support must be easily extensible. The reasons to have this initial choice are a combination of market presence, open / commercial balance, suitability for the pilot cases (considering especially the feedback of the pilot owners), completeness of the

platforms (coverage of all functional groups expected in IoT stacks) and partners' expertise. As it was previously stated, the initial support does not mean exclusive support, given the fact that the results of INTER-LAYER, INTER-FW and INTER-METH are domain agnostic by definition and intended to be extensible and scalable.

The list of IoT Platforms analysed is as follows:

IoT Platform
FIWARE
Open IoT
UniversAAL
OneM2M
Microsoft Azure
Amazon AWS IoT
All-Joyn
Butler
i-Core
Sofia 2
ThingSpeak
GE Predix
IBM Watson IoT
Contiki
eCare
WSO2

Figure 10 List of platforms analysed

Initially supported platforms are highlighted in the figure above.

Once we know the functional model capabilities of the selected set of IoT Platforms, we have been able to design a brand new Functional Model with INTER-IoT's vision for making IoT Platforms interoperable. This Functional Model has been based on some of the concepts defined by the IoT-A, but has been designed with the aim of dealing with the problem of interoperability among heterogeneous platforms.

2.6 Communication Model

The Communication Model aims at defining the communication paradigms for connecting the elements that compose the IoT system, previously defined by the Domain Model(DM). Also, this model is certainly less critical in some application scenarios than in others, and thus, not strictly mandatory.

Being an IoT system sustained on a network, this model leverages on the ISO OSI 7-layer model, but it highlights those peculiar aspects inherent to the interoperation among different stacks, which we will call, in what follows, interoperability features.

To create this model, it is important to identify the communication system elements and/or the communication Users (Human Users, Services or Digital Artefacts) among those defined in the DM.

The communication among these Users, needs to support different paradigms:

- Unicast: as mandatory solution for one-to-one communication.
- Multicast and anycast: for fulfilling many other IoT-application requirements (data collection, information dissemination, etc.)

Normally, most of communication between Users and Services can be established using standard Internet Protocols but, there are two main exceptions to this approach when two services communicate with each other and:

- One belongs to a constrained network: a gateway and/or proxy must be deployed for ensuring successful communication
- Both belong to a constrained network: then a constrained communication protocol has to be used (e.g., 6LoWPAN, UDP, CoAP, etc.).

Instead of focusing on a specific realization of the communication stack, the Communication Model (CM) provides a transversal approach, from which one or more communication stacks can be derived. As a matter of fact, a single interoperability aspect can be used to describe the interactions of stacks belonging to different communicating systems. Once a system is modelled according to the CM it is easy to derive a set of ISO/OSI interoperable stacks in order to provide the needed interoperability features (see Figure 11).

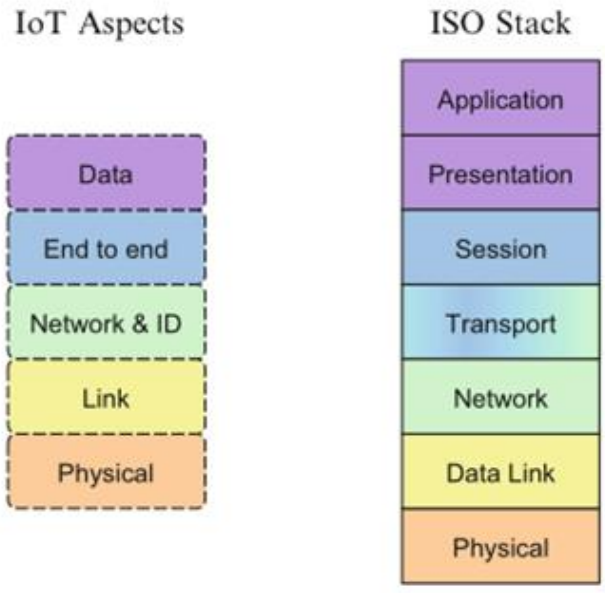


Figure 11 Interoperability aspects of the IoT Communication model compared to the ISO/OSI communication stack

Below, the different interoperability aspects are described:

- Physical aspect: similar to OSI PHY layer, it does not enforce the adoption of any specific technology but it uses the adopted technologies as a base to model the remaining aspects of the system.
- Link aspect: most networks implement similar, but customized communication solutions. This layer must support solution diversity to achieve full interoperability and support heterogeneity. Additionally, it needs to provide upper layers standardized capabilities and interfaces. As this layer needs to abstract a large variety of functionalities, enabling direct communication, IoT systems do not have to restrict the selection among data link layers but must enable coexistence.
- Network and ID aspect: this section combines two aspects; *networking*, same capabilities as the OSI layer, and *identifiers*, resolution functionalities between locators and IDs. The difference between identifiers (unique descriptors of the Digital Artefact; either active or passive), and locators (descriptors of the position of a given IoT element in the network), is the first convergence point in the CM. The interoperability aspect oversees making any two systems addressable independently of the technology adopted.
- End-to-End aspect: this involves reliability, transport, translation, proxies/gateways support and parameter configuration between different networking environments. It provides interoperability aspects on top of Network and ID ones to obtain the final component for achieving a global ThingsToThings (T2T) Communication Model. Connections are also part of this scope. Also, Application Layer aspects are addressed here. Moreover, Application Protocols, in the trend to embed confirmation messages and congestion control techniques, require being more complex than what is achievable in the OSI Transport Layer.
- Data aspect: related with data definition and transfer. Its purpose is to model data exchange between any two actors in the IoT system. This exchange can adopt many different representations, ranging from raw data to complex structures where meta-data information is added to provide context specific links. Additionally, the data aspect needs to model the following characteristics:

    1.- Capability of providing structured attributes for data description;
    2.- Capability of being translated (possibly by compression/decompression) the one to each other (e.g. CoAP to HTTP by decompression or XML to EXI by compression or IPv4 to IPv6 by mapping, etc).
    3.- Constrained device support.

In the Communication model, we define the connection between two or more elements in the model, maybe using a single communication stack. For that reason, there are two options to model a composed communication according to the IoT Communication Model. These options are the configuration with a Gateway that involves the composition of two or more protocol stacks located across different network or a Virtual configuration that implies the composition of two or more protocol stacks, one on top of the other.

Within the composed modelling option, it is known:

- *Gateway configuration* as the composition of two or more protocol stacks that are placed side by side across different media so that they can be seen seamlessly connected.

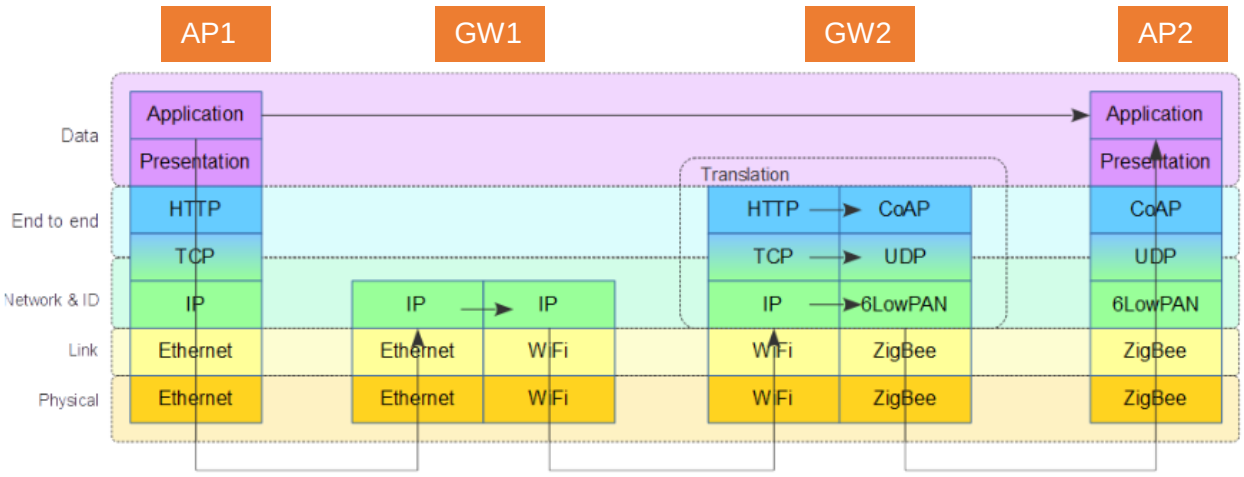


Figure 12 Gateway configuration for multiple protocol stacks

In Figure 12, we can observe the communication between two application layers through two gateways. The first one (left) bridges between Ethernet and WiFi networks, and the second one (right) additionally includes a translation functionality between WiFi and ZigBee, and also the translation between IP to 6LoWPAN, TCP to UDP, HTTP to CoAP and vice versa.

- *Virtual configuration* as the composition of two or more protocol stacks, one on top of the other, where the actual communication path is virtualized by tunnelling the communication using a second protocol stack.
  In Figure 13, we can observe an inner communication path composed of an Ethernet network and a WiFi network using a bridging block and an outer communication path that is independent of the inner path and which allows for the two application layers to communicate. Such a scheme is usually achieved using virtual private network solutions.

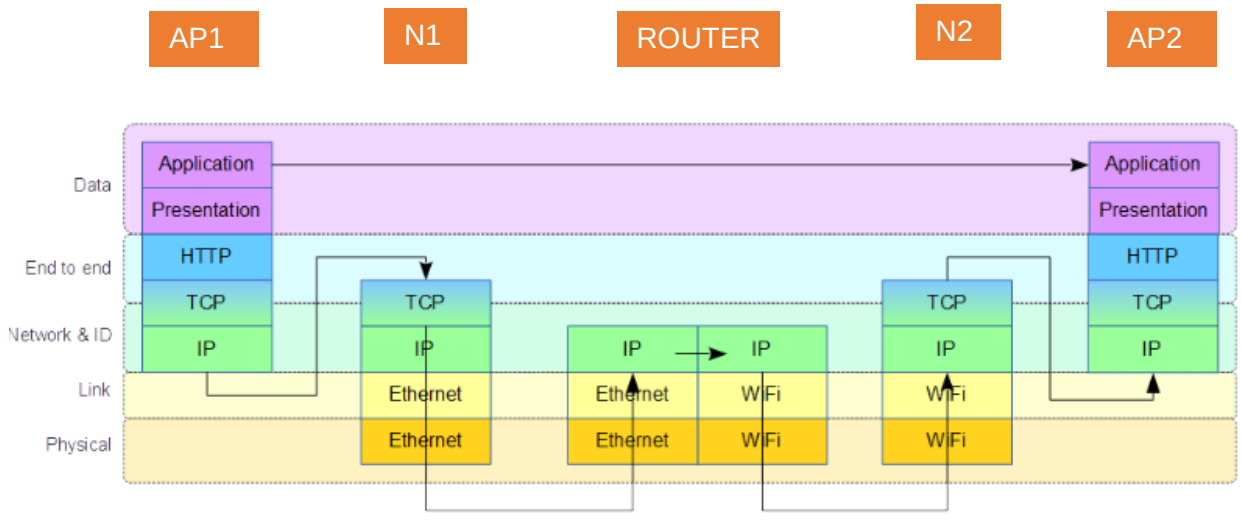


Figure 13 Virtual configuration for multiple protocol stacks

Additionally, we can find a channel model. This describes the content of the channel in the Shannon-Weaver model[19], but in context of the IoT domain. The main objective is not capturing every possible

[19] http://communicationtheory.org/shannon-and-weaver-model-of-communication/

characteristic of IoT technologies, but to provide a common ground to be used to compute overall system performance and benchmarking. To understand this channel modelling we must define:

- Unconstrained networks as a high-speed communication link (as wired Internet). Here, link-level transfer latencies are also small and mainly impacted by congestion events in the network, rather than by the physical transmission technology.
- Constrained networks as communications with relatively low transfer rates (typically smaller than 1Mbit/s) and large latencies. These are due to several factors; the involved low-bitrate physical layer technology and the power-saving policy of the nodes (with periodic radio power-offs).

According to this, heterogeneous networks can be seen as the combination of constrained and unconstrained networks linked together via gateways and/or proxies. In the IoT case, it could be a single constrained network, several constrained networks over different technologies, or even two constrained networks joint by an unconstrained one (as two WSN communicating by Internet).

Additionally, the nature of the constrained networks relies on constrained devices. The communication between these can:

1. Be based on different protocols;
2. Require additional processing in the gateways.

It is important to point out that the characteristics of each network can have a noticeable impact on the overall end-to-end communication [25].

As we can notice following the IoT ARM Reference Manual [33], there are some steps for modelling our systems. These steps involve four usages; the usage of the IoT Domain Model, IoT Information Model, IoT Communication Model and Perspectives.

Within the first Usage, the instantiation of the Domain Model to a particular case is discussed. The main identified concept instances are: Physical Entities and related with Virtual Entities, Resources, Device, Services and User. (As is explained in Section 3.2)

Once this instance has been identified, we can proceed with the first steps of modelling, that includes, for the Domain Model, the first three rules as:

> **Rule 1** *Model as precisely as possible based on the domain model concepts at the time of modelling. Use the more concrete, more fine-granular concepts and instances whenever possible, but only to the granularity that appears reasonable for the given purpose.*
> **Rule 2** *When modelling an autonomous object, an Augmented Entity is used, consisting of a device (Physical Entity) and its software controller (Virtual Entity).*
> **Rule 3** *Only model something as a Physical Entity if it is relevant in the IoT system so that the representing Virtual Entity is also modelled.*

These are applied when we model the IoT Domain Model of our specific use case, and later one the rest of the rules are used for each one of the other usages.

In this case, for the usage of Communication Model that define the architectural process as:

1. Identify homogeneous sub-systems and their capabilities and constraints.

2. Identify suitable protocol stacks and network topologies to be merged in a common system view.
3. Define gateways and other bridging solutions

With this picture in mind, the IoT-A ARM provides different guidelines for using the CM to provide an overall framework for communication within the IoT systems, previously defined the domain and information models. This is carried out following these rules:

> **Rule 4**: *Identify homogeneous sub-systems (as a set of elements with the same communication technology and similar hardware capabilities) from the complete domain model and determine their capabilities and constraints. Analysing these capabilities and constraints to understand the communication specific parameters (data rate, delays, reliability) and technology specific parameters (memory, computational power and supported functionalities).*

Following, we can analyse the communication requirements coming from services in the domain model, and interaction patterns from the information model. So, we obtain a set of interoperable protocol stacks and topologies with the following characteristics:

1. Each stack must grow from a specific communication technology.
2. Interoperability shall be enforced in the lowest possible layer of stack.
3. The combination of identified stacks and topologies must satisfy all the requirements.

> **Rule 5**: *Use existing standard communication mechanisms and related protocols whenever possible. If this is not possible then each of the sub-systems is the starting point for building a protocol stack which is botch technology specific and interoperability prone.*

This Rule is applied for technological optimizations. This Rule enhance the communications and ensures feasibility in all sub-systems by the re-use of the same protocols between as many components as possible.

> **Rule 6:** *Interoperability shall be enforced in the lowest possible layer.*

Enforcing simplicity, and avoiding stack duplication and also reusing protocols horizontally in the system. Usually, the most effective interoperability point is the Network & ID aspect of CM as is the lowest common point not technology specific, so could be the same across different sub-systems.

> **Rule 7**: *In order to allow seamless interaction between sub-systems, gateway and proxies shall be designed for the whole system.*

Finally, the Data Interoperability aspect of the CM considers the remaining aspects of data exchange, compression and representation. Most often, adopting a compressed format which fits constrained network capabilities, provides simpler network interactions, and lower traffic [25].

2.7 Functional View

For the creation of the INTER-IOT functional view, the following steps have been followed:

1) Analysis of the functional aspects of the existing platforms according to the IoT-A Functional Model
2) Review of the Functional Model.
3) Analysis of the project requirements and use cases.
4) Generation of a new functional view with the conclusions of the previous steps.

For the analysis (also referred in this document as platform study), a set of 16 platforms was selected, based on the following criteria: partners' expertise, market relevance and current adoption and support (in industry or in open source communities). Some constraints were added to prioritize promising platforms and to ensure a proper balance between private and open software. The methodology to gather this information was to let all the partners report about a set of question related to these platforms. Finally, a list was made and prioritized, resulting in a list of 16 platforms that is extensively used in this document.

Each platform in the list was carefully analysed with the scope of the IoT-A ARM, with particular attention to the functional view. Data and statistics about the functional components of each platform offer important conclusions about the degree of coverage of the components, and where the interoperability mechanism can be more effective.

Next, the Functional Model was reviewed taking into account the previous analysis of the selected IoT Platforms.

The project requirements and the use cases were analysed to identify the features that the Functional View should accomplish.

Finally, with the knowledge gained in the platform study and the conclusions obtained in the previous steps, a novel Functional View for platform interoperability was developed, accordingly with the functional model (see section 2.4). It is firstly proposed in this document.

3 INTER-IoT Reference Model and Meta Data Model

3.1 Introduction

The IoT-A project used the OASIS definition for describing the reference model. OASIS (Organization for the Advancement of Structured Information Standards) gives the following definition of a reference model:

> [*Reference model is*] an abstract framework for understanding significant relationships among the entities of some environment, and for the development of consistent standards or specifications supporting that environment. A reference model is based on a small number of unifying concepts and may be used as a basis for education and explaining standards to a non-specialist. A reference model is not directly tied to any standards, technologies or other concrete implementation details, but it does seek to provide a common semantics that can be used unambiguously across and between different implementations.

Central features that a reference model needs to exhibit are:

- Clearly defined concepts
- Clearly defined concept relationships
- Clearly defined concept properties
- Does not describe concrete entities (instances)
- Restricted to a specific problem space
- Promotes understanding of the problem space
- Technology agnostic and implementation independent
- Used as reference for implementation
- Enables understanding (i.e. common semantics) in communication

The Reference Model is composed by 5 different models that fully encompass the IoT modelling and are the base for the development of the architectural view and perspectives. The first model is the IoT Domain Model, which describes all the concepts that are relevant in an Internet of Things scenario. All other models and the IoT Reference Architecture are based on the concepts introduced in the IoT Domain Model. The Communication Model is very relevant as describes the different communication that happens in the IoT domain, namely, between constrained and unconstrained networks. The Information model shows how the information flows between entities. The IoT Trust, Security, and Privacy Model shows the importance of dealing with Privacy and Security issues from the very modelling part. Finally, the Functional Model introduces the concepts and the modularity between functional parts that will be key in building a concrete architecture.

Following IoT-A methodology, in INTER-IoT we have developed the following models for the initial Reference Model described in this document:

- Domain Model.
- Communication Model.
- Information Model.
- Functional Model.

3.2 Domain Model

3.2.1 Introduction

In general, a Domain model[20] is a class diagram that is used to describe specific aspects of a set of knowledge or activities. The main use for it is to represent use cases and real-world concepts in a way that can then be used by the technical people to develop a service, application or a product.

The main purpose of a generic domain model is to generate a common understanding of the target domain in question. Such a common understanding is important, not only within a specific project, but also to be able to discuss with stakeholders and external parties. Only with a common understanding of the main concepts it is possible to choose between different architectural solutions and to evaluate them.

3.2.2 IoT-A Domain Model

The IoT-A project defines a domain model as a description of concepts belonging to a specific area of interest. The domain model also defines basic attributes of these concepts, such as name and identifier, and relationships between concepts, for instance "*Services* expose *Resources*". The IoT-A domain model also provides a common lexicon and taxonomy that can be used in the IoT domain. [6]. The IoT-A Domain Model extends two previous models in this specific domain, namely [7].

In a IoT domain, the most generic scenario is that of a generic User who needs to interact with a *Physical Entity* (PE) in the physical world. Here we can already see two of the main entities in IoT:

- a *User* which can be a human person or a *Digital Artefact* (e.g., a *Service*, an application, or a software agent) that needs to interact with
- a *Physical Entity*, which is an object that is under observation and can be modified by automatic means. Physical Entities can be almost any object or environment; from humans or animals to cars; from store or logistics chain items to computers; from electronic appliances to jewellery or clothes.

While in a physical environment, interactions can only happen directly (for instance, by moving a pallet from location X to Y manually), within the IoT world it's possible to interact indirectly or mediated, by calling a *Service* that will either provide information about the *Physical Entity* or act on it. When a *Human User* is accessing a service, he does so through a service client, a *User Interface* for instance. For the scope of the IoT Domain Model, the interaction is usually characterised by a goal that the *User* pursues.

Physical Entities are represented in the digital world by a *Virtual Entity*, which can be seen as a "virtual counterpart ". There are many kinds of digital representations of *Physical Entities*: 3D models, avatars, database entries, objects (or instances of a class in an object-oriented programming language). *Virtual Entities* are associated to a single *Physical Entity* and the *Virtual Entity* represents this very *Physical Entity*. While there is generally only one *Physical Entity* for each *Virtual Entity*, it is possible that the same *Physical* Entity can be associated to several *Virtual Entities*. Each *Virtual*

[20] https://msdn.microsoft.com/en-us/library/bb126581(v=vs.90).aspx

Entity must have one and only one ID that identifies it univocally. *Virtual Entities* are *Digital Artefacts* that can be classified as either active or passive. *Active Digital Artefacts* (ADA) are running software applications, agents or *Services* that may access other *Services* or *Resources*. *Passive Digital Artefacts* (PDA) are passive software elements such as database entries that can be digital representations of the *Physical Entity*. Please note that all *Digital Artefacts* can be classified as either *Active* or *Passive Digital Artefacts*.

As well, *Virtual Entities* are synchronised representations of a given set of aspects (or properties) of the *Physical Entity*. This means that relevant digital parameters representing the characteristics of the *Physical Entity* are updated upon any change of the former. In the same way, changes that affect the *Virtual Entity* could manifest themselves in the *Physical Entity*. For instance, manually locking a door might result in changing the state of the door in home automation software, and correspondingly, setting the door to “locked” in the software might result in triggering an electric lock in the physical world.

The *Augmented Entity* is what enables everyday objects to become part of digital processes. In technical terms, the *Augmented Entity* is a composition of *Physical* and *Virtual Entities*.

The relation between *Virtual Entity* and *Physical Entity* is usually achieved by embedding into, by attaching to, or by simply placing in close vicinity of the *Physical Entity*, one or more ICT *Devices* that provide the technological interface for interacting with, or gaining information about the *Physical Entity*. A *Device* thus mediates the interactions between *Physical Entities* (that have no projections in the digital world) and *Virtual Entities* (which have no projections in the physical world), generating a paired couple that can be seen as an extension of either one; in other words, the Aug*mented* Entity. *Devices* are thus bridging the real world of *Physical Entities* with the digital world of the Internet. This is done by providing monitoring, sensing, actuation, computation, storage and processing capabilities. A *Device* can also be a *Physical E*ntity: an example for such an application is *Device* management, whose main concern is the *Devices* themselves and not the entities or environments that these *Devices* monitor.

Resources are software components that provide data from or are used in the actuation on *Physical Entities*. *Resources* may be *On-Device* and *Network*. As the name suggests, *On-Device Resources* are hosted on *Devices*, while *Network Resources* are *Resources* available somewhere in the network, such as back-end or cloud-based databases. A *Virtual Entity* can also be associated with *Resources* that enable interaction with the *Physical Entity* that the *Virtual Entity* represents.

In contrast to heterogeneous *Resources* implementations of which can be highly dependent on the underlying hardware of the *Device*, a *Service* provides an open and standardised interface, offering all necessary functionalities for interacting with the *Resources* / *Devices* associated with *Physical Entities*. Interaction with the *Service* is done via the network. On the lowest level the one interfacing with the *Resource* and closer to the actual *Device* hardware, *Services* expose the functionality of a *Device* through its hosted *Resources*. Other *Services* may invoke such low-level *Services* for providing higher-level functionalities, for instance executing an activity of a business process.

Since it is the *Service* that makes a *Resource* accessible, the above-mentioned relations between *Resources* and *Virtual Entities* are modelled as associations between *Virtual Entities* and *Services*. For each *Virtual Entity* there can be associations with different *Services* that may provide different functionalities, like retrieving information or enabling the execution of actuation tasks. *Services* can also be redundant, i.e., the same type of *Service* may be provided by different instances (e.g. redundant temperature *Services* provided by different *Devices*). In this case, there could be multiple associations of the same kind for the same *Virtual Entity*.

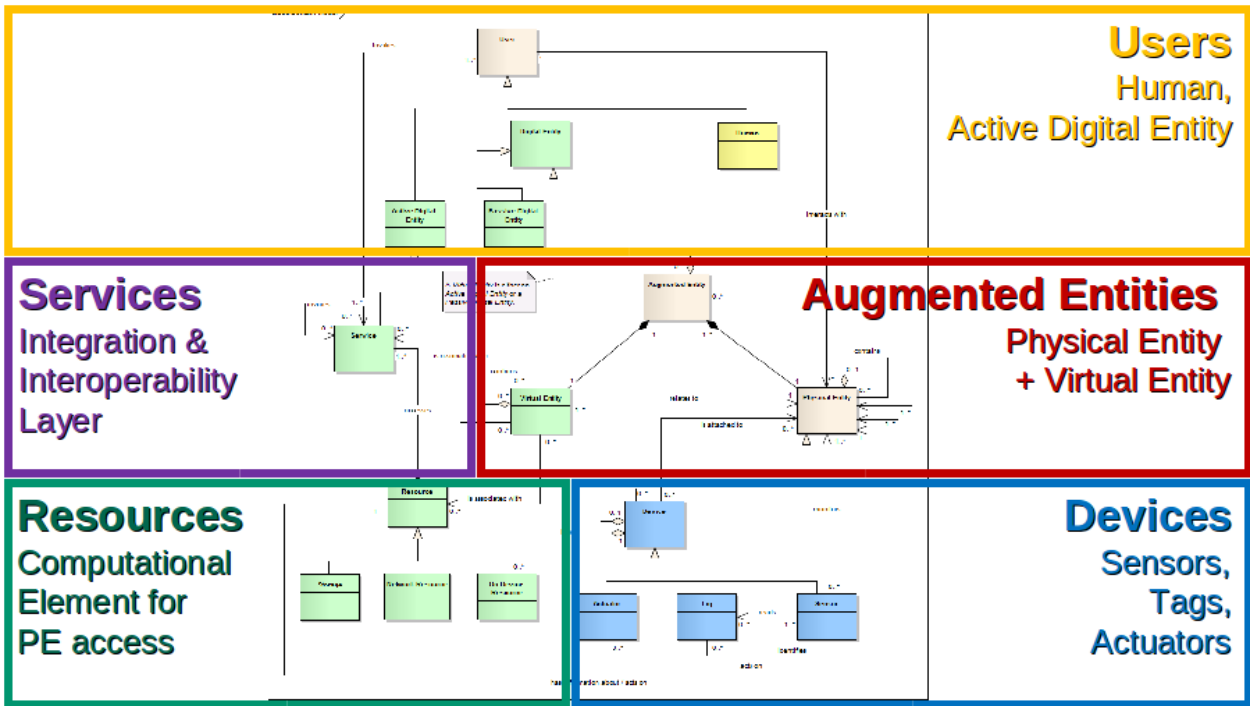


Figure 14: IOT-A's Domain Model with entity classification

3.2.3 INTER-IoT Domain Model

We have extended IoT-A Domain Model to include new entities inherent to IoT platform interoperability.

The clearest entity is an *IoT Platform* itself. In any IoT Domain Model, the platform is intrinsically implicit, as it is really "the whole model". When dealing with platform interoperability, different *IoT Platforms* appear, thus need arises to model them independently.

Any *IoT Platform* relates with the underlying entities, like *Services*, *Things* or *Physical Entities*, and so on. Therefore, an *IoT Platform* can be modelled as a set of composed entities, the entities the platform manages.

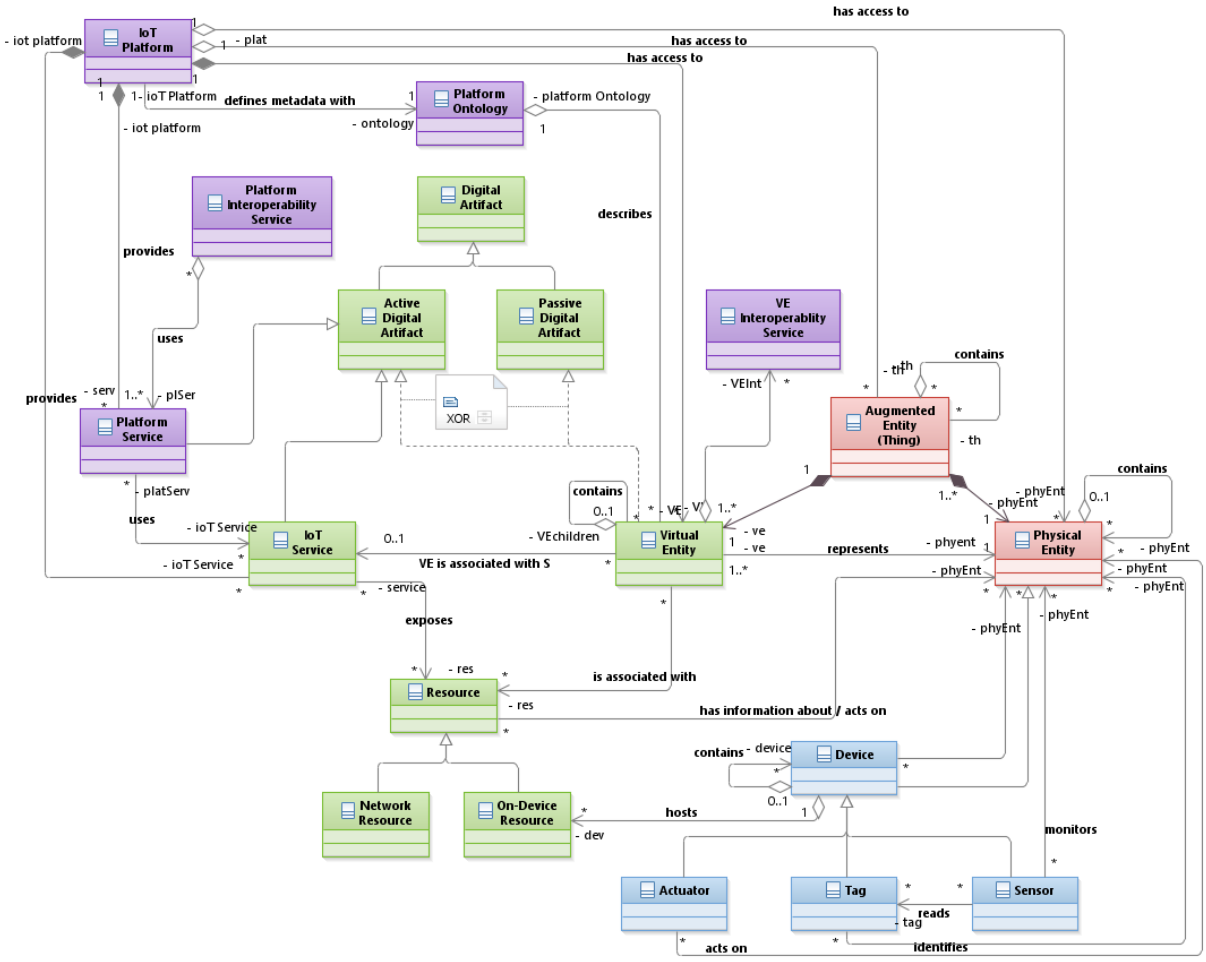


Figure 15: INTER-IoT generic domain model

An *IoT Platform* is comprised of several collections of entities:

- **IoT Service**: An *IoT Service* is what was called a *Service* in the IoT-A. It provides an open and standardised interface, offering all necessary functionalities for interacting with the *Resources* / *Devices* associated with *Physical Entities*. We understand an *IoT Service* as a mechanism to interact with specific *Resources* related to *Virtual Entities*. They are *Active Digital Artefacts* that usually are available in *IoT Platforms* exposing capabilities like *query*, *update*, *subscribe*.
- **Platform Service**. A *Platform Service* is also an *Active Digital Artefact* that exposes functionality about *Resources* related to *Virtual Entities*. However, rather than being attached to specific *Physical Entities* (and its related *Virtual Entities*), they offer more elaborated services that internally make use of *IoT Services*. They are usually placed in an upper layer of IoT architectures, allowing more complex processing, like CEP (Complex Event Processing), Stream Processing, Historical Data Management, Monitoring, etc. They are like the IoT processes of IoT-A, but we think of them not as placed close to enterprise systems, but as derived services that can be used as building blocks for creating more complex interoperability services among different IoT platforms.
- **Virtual Entity**. A *Virtual Entity* is a representation of a *Physical Entity* in the digital world. IoT platforms tend to use this digital representation, especially those based on cloud platforms.

An *IoT Platform* utilizes a set of *Virtual Entities* for managing a thing status regardless of whether the thing is connected to the Internet or not.

- **Physical Entity**. A *Physical Entity* is an object or environment that is of interest to an external user, application, or service. It's something that can be observed and connected to one or more *IoT Platforms* to interact with it.
- **Augmented Entity**. As of IoT-A definition, an *Augmented Entity* is "*the composition of one Virtual Entity and the Physical Entity it is associated to, in order to highlight the fact that these two concepts belong together. The Augmented Entity is what actually enables everyday objects to become part of digital processes, thus, the Augmented Entity can be regarded as constituting the "thing" in the Internet of Things.*". So, we can refer to it as a *Thing*, and later we will see the ontology modelling for things (meta-data model).

A new concept we introduce for the Domain Model is the **Platform Ontology**. The *Platform Ontology* is conceived to store the definition concerning the ontology used by the platform to define its inner structure and components (devices, sensors…). The *Platform Ontology* defines also the ontology used for modelling the observations made by each sensor, in our case available from the *Virtual Entities*. These ontologies will usually be different for each *Physical Entity* type (and its related *Virtual Entity*). This entity is thus, the one that is responsible for handling the corresponding semantics.

Once we have modelled the different entities of the platforms and the platform itself, we have added two entities related, specifically, to defining the interoperability services that can be created at different layers.

The **Platform Interoperability Service** handles the definition of new compound services that appear as a consequence of using, and mixing in any way, *Platform Services* from one or more IoT platforms. An example of this would be the creation of an alert service that may throw an event when weather sensors from platform A exceed predefined thresholds using a rule engine service at platform A, or when weather sensors from platform B send an alert using a CEP (Complex Event Processing) within platform B.

So, the *Platform Interoperability Service* is linked with the different *Platform Services* it uses. Used *Platform Services* are just part of *IoT Platforms*. From the interoperability point of view, they are the building blocks of more complex interoperability services among different *IoT platforms*: *Platform Interoperability Services*.

The **VE Interoperability Service** has a similar role as the *Platform Interoperability Service*, but it defines interoperability services among devices rather than platform services. It is responsible for defining the interoperability at the device layer, what we call D2D (Device to Device) interoperability at INTER-IoT.

The *VE Interoperability Service* can handle the rules for performing the D2D interoperability. For instance, it could have the definition of a rule triggered when a proximity sensor detects presence, switching a light on.

3.3 Information Model

3.3.1 Introduction

The Information Model is one of the 5 Models composing the IoT-A Reference Model. The main aspects are represented by the elements *VirtualEntity*, *ServiceDescription* and *Association*. As a *Virtual Entity* models a *Physical Entity*, a *ServiceDescription* describes a *Service* that provides information about the *Physical Entity* itself or the environment. Through an *Association*, the

connection between an *Attribute* of a *Virtual Entity* and the *ServiceDescription* is modelled; in other words, the *Service* acts as a “get” function for an *Attribute* value.

Every *Virtual Entity* needs to have a unique identifier (*identifier*) or entity type (*entityType*), defining the type of the *Virtual Entity* representation, for instance, a human, a car or a temperature sensor. Furthermore, a *Virtual Enti*ty can have any number of different attributes (*Attribute* class). The *entityType* of the *VirtualEntity* class may refer to concepts in an ontology that defines what attributes a *Virtual Entity* of this type has. Each *Attribute* has a name (*attributeName*), a type (*attributeType*), and one to many values (*ValueContainer*). The *attributeType* specifies the semantic type of an attribute, for example, that the value represents temperature. It can reference an ontology-concepts. This way, it is possible to model an attribute or a list of values, which itself has several values. Each *ValueContainer* groups one Value and zero to many metadata information units belonging to the given *Value*. The metadata can, for instance, be used to save the timestamp of the *Value*, or other quality parameters, such as accuracy or the unit of measurement. The *Virtual Entity* (*Virtual Entity*) is also connected to the *ServiceDescription* via the <*Service Description* / *Virtual Entity*> Association.

A *ServiceDescription* describes the relevant aspects of a *Service*, including its interface. Additionally, it may contain one (or more) *ResourceDescription*(s) describing a *Resource* whose functionality is exposed by the *Service*. The *ResourceDescription* in turn may contain information about the *Device* on which the Resource is hosted.

According to the IoT-A [25] the *IoT Information Model* defines the structure of all the information for *Virtual Entities* on a conceptual level (cf. Section 2.3). This description utilizes meta-data coming from appropriate *ontologies*. The INTER-IoT project uses semantic technologies to deal with meta-level interoperability. Specifically, the semantic interoperability will be established through the use of a modular ontology, ontology alignments, and semantic transformations.

The Inter-IoT reference meta-data model is a set of ontologies, that can also be viewed as one modular ontology, with both horizontal and vertical modules. Following the process described in Section 2.4 this ontology needs to cover fundamental concepts in IoT, such as *thing*, *device*, *observation* and *deployment*.

The meta-data model needs to conform to OASIS guidelines enumerated in section 3.1. OWL ontologies naturally exhibit some of those, such as: clear (and formal) descriptions of concepts and relationships between them; independence of implementation technology; and enabling common semantics. Other than that, the reference meta-data model does not contain references to any specific instances, and is limited to the scope defined by meta-data requirements, described in following sections[17].

3.3.2 Scope of Meta-Data model

The scope of the Inter-IoT reference meta-data model is defined in a process outlined in section 2.4 The process relies on defining meta-data items (entities) that need to be included in the model. Simple examples of meta-data entities are *Service*, *Device* (with sub-types *Actuator*, *Tag* and *Sensor*) that are declared in the IoT-A domain model. The meta-data reference model expands those declarations into definitions by defining properties, class attributes, taxonomy and other elements structuring the meta-data entities. Once the scope (defined by the entities) is prepared, the reference model is constructed by choosing and adjusting (expanding or reducing) modular ontologies. Subsections that follow contain analysis of sources of meta-data items defined in section 2.4.

3.3.2.1 INTER-IoT Grant Agreement

The grant agreement document, along with its amendments, contains broad descriptions of Inter-IoT tasks and work packages. Semantic entities can be identified and extracted from task and work package descriptions. The description of task 4.2 - creation of reference meta-data model - gives the following summary (meta-data entities are underlined):

- IoT Device/Smart Object metadata will basically include identity, type, physical characteristics, location, embedded devices, and provided services.
- Middleware metadata will basically contain communication service type, access protocol, URI, supported data / object domains.
- Application Services metadata will basically include service identifier, type, access protocol, device/smart objects supported, input data and output data.
- Application Data metadata will basically include time, value structure, security features and domain-specific characteristics.
- User Data metadata will basically include identifier, role, personal data, and security/privacy/trust policy information.

3.3.2.2 Inter-IoT project requirements

In the first few months of Inter-IoT, requirements for the whole project (and for each part of it separately) were identified and stored and numbered in a project management software. Some of them contain (explicit or implicit) references to data entities. Each project requirement was analysed with respect to meta-data requirements. The results of this analysis are contained in the table in Appendix 1. Some project requirements were redacted from the appendix, if the information relevant to meta-data they provided was overlapping with information from other requirements already on the list. This was done for the sake of clarity. Note that the reference meta-data model itself should not contain domain-specific entities for the pilot implementations of Inter-IoT. Because of that requirements that relate to domain knowledge (eHealth, transportation & logistics; e.g. INTERIOT-641 to 645) were redacted or interpreted with disregard for the domain-specific metadata.

3.3.2.3 IoT Platforms

Some IoT platforms, like OpenIoT or UniversAAL provide explicit ontologies that model the meta-data used within those platforms. The knowledge contained within those models is an indirect source of meta-data entities. Since INTER-IoT is a set of tools for interoperability between platforms, rather than a platform itself, the models of platforms should not simply be copied. That being said, the analysis of existing platform ontologies provides valuable insight that augments explicit meta-data requirements from other sources, and puts them in context. Section 3.3.3 contains an in-depth analysis of selected ontologies, that, apart from being standards for IoT, are actually used in existing platforms.

3.3.2.4 IoT EPI Task Force on Interoperability

The IoT-EPI task force on semantic interoperability can provide useful considerations when it comes to use of ontologies in all participating projects. If any ontology is going to be used in multiple projects, it should be given special importance when considering our reference meta-data model. Projects involved in IoT-EPI may also have a different perspective on IoT landscape, which partially stems

from the fact that they each have different problems to solve. For instance, in IoT-EPI three software artefact levels are delimited: Cloud platforms, IoT Gateways and IoT Device, which is a slightly different division of IoT space than that proposed by INTER-IoT internally. Different perspectives can potentially bring useful conclusions about interoperability mechanisms. Documents produced by IoT-EPI will be studied in search of meta-data requirements and ontology modules that could be incorporated into Inter-IoT. At this point in time, the IoT EPI participants do not have meta-data models defined yet. Once they are ready and available (shared), INTER-IoT will analyse them and communicate with the rest of the partners to make semantic interoperability between projects as easy as possible. The focus of the reference meta-data model, however, is and will always be on the INTER-IoT.

3.3.2.5 Summary

The table below contains a concise summary of meta-data requirements gathered from all the sources described in section 3.3.2.

Category	Details
Device	Identifier (URI, RFID tags), Systems and subsystems of devices, Type (sensor, actuator, human interface), Capability (actuation, sensing; active or passive), Supported communication protocols, Status and performance information (energy consumption, battery level, usage mode), Device capabilities, stack number, Environmental impact (gas emission levels, heat emission, noise level energy requirements), Location of the thing/device (where the thing is located e.g. thermostat over a door), Feature of interest being measured (where the feature being measured is located, e.g. room being measured by a thermostat)
Middleware	Identifier (name) Communication and access protocols, Protocol type (publish-subscribe, ...), Supported object and data domain,

	Authorization, accounting and authentication credentials and methods (user ID, email, password, checksum, encrypted key file, authentication device, authentication token)	
Service	Identifier, Type, Access protocol, input and output data, Supported objects (e.g. devices, platforms), service provider	
User	Identifier, Role, Privileges, Personal data (names, location), security/privacy/trust policy information	
Application	Security features, Security policy, Extendable domain characteristics (out of scope for the reference model), Event, Access log, system event, event log (out of scope), content type (document – report; image – graph, chart, diagram), data access policy – when, who (user, role, device, platform), Communication protocol features (multicast, single cast, broadcast), read, write, share privileges	
Network	Address (MAC, IP), Protocol and communication method (6LoWPAN, RoLL, ZigBee, Bluetooth, Wi-Fi, ethernet, DSL, PSTN, GSM, 3G, LTE, Satellite), Security method (SSH, SSL, TLS). Location and distance, network coverage	
other	**Message / device priority and importance**	

	Provenance	Ownership, Current assigned entity (current owner, current caretaker, current responsible entity), creation, responsibility, source, provider, physical or virtual location of data storage
	Data	Time (including duration), geolocation, location of the sensing device, feature of interest being measured (can be different) location (address), Data stream, Measures and units, Attribute ranges, energy, temperature, Quality/accuracy

Table 2: Summary of meta-data requirements

3.3.3 Comparing IoT-related ontologies

The space of ontologies is fragmented, regardless of the domain of interest. The richer an ontology is, the larger area it spans. Hence, uniqueness and intersections with other ontologies become more intricate and complex. Internet of Things spans enormous number of domains, and rapidly expands with the growing popularity of “smart devices”. Use of ontologies in the IoT mimics this expansiveness. There are many ontologies that represent models relevant to the IoT, including, but not limited to, devices, units of measurement, data streams, data processing, geolocation, data provenance, computer hardware, methods of communication, etc. We assume that the centrepiece of the IoT is a smart device capable of communication. Therefore, the first iteration of the reference meta-data model is in the form of a device ontology and forms a cornerstone for other ontology modules (that cover other meta-data requirements). The list of identified and analysed ontologies (including the device ontologies) can be found in Appendix 2, along with a short description. The ontologies were selected for analysis based on a simple criterion that they describe some (at least one) of the meta-data requirements summarized in Section 3.3.2.

From this perspective, from the identified ones, we have selected ontologies that capture the idea of a device, and are well established in the IoT space: SSN, SAREF, oneM2M Base Ontology, IoT-Lite,

and OpenIoT. Each of them takes a different approach to modelling the IoT space but, despite the differences in conceptualization, they cover intersecting fragments of the IoT landscape. Below, we discuss divergence, contrariness and similarities between these ontologies.

SSN, or "Semantic Sensor Network" [11,15] is an ontology cantered around sensors and observations. It is a de-facto extension of the SensorML language. SSN focuses on measurements and observations, disregarding hardware information about the device. Specifically, it describes sensors in terms of capabilities, performance, usage conditions, observations, measurement processes, and deployments. It is highly modular and extendable. In fact, it depends on other ontologies in key areas (e.g. time, location, units) and, for all practical purposes, needs to be extended before actual implementation of an SSN-based IoT system. SSN, formulated on top of DUL[21], is an ontological basis for the IoT, as it tries to cover any application of sensors in the IoT.

SAREF [16], or “The Smart Appliances REFerence” ontology covers the area of smart devices in houses, offices, public places, etc. It does not focus on any industrial or scientific implementation. The devices are characterized predominantly by the function(s) they perform, commands they accept, and states they can be in. Those three categories serve as building blocks of the semantic description in SAREF. Elements from each can be combined to produce complex descriptions of multi-functional devices. The description is complemented by device services that offer functions. A noteworthy module of SAREF is the energy and power profile that received considerable attention, shortly after its inception[22]. SAREF uses WGS84 for geolocation and defines its own measurement units.

oneM2M Base Ontology (oneM2M BO; [10,13]) is a recently created ontology, with first non-draft release in August 2016. It is relatively small, prepared for the release 2.0 of oneM2M specifications, and designed with the intention of providing a shared ontological base, to which other ontologies would align. It is similar to the SSN, since any concrete system necessarily needs to extend it before implementation. It describes devices in a very broad scope, enabling (in a very general sense) specification of device functionality, networking properties, operation and services. The philosophy behind this approach was to enable discovery of semantically demarcated resources using a minimal set of concepts. It is a base ontology, as it does not extend any other base models (such as DOLCE+DnS Ultralite DUL or Dublin Core). However, alignments to other ontologies are known [19].

IoT-Lite [14] is an *instantiation* of the SSN, i.e. a direct extension of some of its modules. It is a minimal ontology, to which most of the caveats of the SSN apply. Specifically: focus on sensors and observations, reliance on other ontologies (e.g. time or unit ontologies), high modularity and extendibility. The idea behind the IoT-Lite was to create a small/light semantic model that would be less taxing (than other, more verbose and broader models) on devices that process it. At the same time, it needed to cover enough concepts to be useful. The ontology describes devices, objects, systems and services. The main extension of the SSN, in the IoT-Lite, lies in addition of actuators (to complement sensors, as a device type) and a coverage property. It explicitly uses concepts from a geolocation ontology [8] to demarcate device coverage and deployment location.

OpenIoT [22, 23] ontology was developed within the OpenIoT project. However, here, we use the term “OpenIoT” to refer to the ontology. It is a comparatively big model that (re)uses and combines other ontologies. Those include all modules of the SSN (the main basis for the OpenIoT), SPITFIRE (including sensor networks), Event Model-F, PROV-O, LinkedGeoData, WGS84, CloudDomain, SIOC, Association Ontology and others, including smaller ontologies developed at the DERI (currently, Insight Centre). It also makes use of ontologies that provide basis for those enumerated

[21] http://www.ontologydesignpatterns.org/ont/dul/DUL.owl

[22] https://goo.gl/1OXTJb, https://goo.gl/ZaGjCJ

earlier, e.g. DUL. Other than concepts from the SSN, OpenIoT, uses a large number of SPITFIRE concepts, e.g. network and sensor network descriptions. While some mentioned ontologies are not imported by the OpenIoT explicitly, they appear in all examples, documentation, and project deliverables. Therefore, one can treat OpenIoT as a combination of parts of all of those. Similar to the SSN, OpenIoT does not define its own location concepts and does not explicitly import geolocation ontologies. It relies on other ontologies for that but, in contrast to the SSN, it clearly indicates LinkedGeoData and WGS84 as sources of geolocation descriptions. It defines a limited set of units of measure (e.g. temperature, wind speed), but only when they were relevant to the OpenIoT project pilot implementation.

The rich suite of used ontologies means that OpenIoT provides a very extensive description of devices, their functionalities, capabilities, provenance, measurements, deployments and position, energy, relevant events, users and many others. Interestingly enough, it does not explicitly describe actuators or actuating properties/functions. It can be observed that the broad scope of the ontology makes it rather complicated. This is also because, it is not documented well-enough, i.e. the detail level and ease-of-access of the documentation do not match the range of coverage of concepts in the model. Moreover, it is not clearly and explicitly modularized, despite being an extension of the SSN.

Let us note that, while there are other IoT models of potential interest (such as OGC Sensor Things, UniversAAL ontologies, FAN FPAI, IoT Ontology[23], M3 Vocabulary), we have decided that they are of less importance or relevance to INTER-IoT. This was either because they have generated much less “general interest”, or had scope well outside that of the project.

3.3.3.1 In-depth analysis

Let us now compare the selected ontologies side-by-side. To do this, we have selected key aspects, or categories, *directly pertaining to the IoT*; placed in the first column of Table 1. However, because of intricacies and disparate philosophies behind compared ontologies (see, above), each category needs to be further investigated. In other words, proposed categorization is a tentative way of visualizing and analysing similarities and differences between ontologies of choice. Here, we follow an approach proposed by Raúl García-Castro during June 2016 European Platform Initiative (IOT EPI[24]) meeting.

Before proceeding it should be noted that there are numerous approaches to ontology evaluation, e.g. [24,21]. We have, however, found that applying them would not help in the context of specific, project-related, problem. Specifically, we were more interested in capturing and comparing details of each area that the selected ontologies cover, rather than their overall evaluation by some standard. In other words, we are primarily interested in how well the ontologies can help us solve the problem at hand.

Category (Subdomain)	SSN	SAREF	oneM2M BO	IoT-Lite†	OpenIoT†
Thing	✓	✓	✓	✓	✓

[23] http://ai-group.ds.unipi.gr/kotis/ontologies/IoT-ontology

[24] http://iot-epi.eu/

Device	✓	✓	✓	✓	✓
Device Deployment	✓α	✓	✓α, ∅	✓	✓
Device Properties and Capabilities	✓				✓
Device Energy	✓	✓ε			✓
Function and Service		✓	✓	✓S	
Sensing and Sensor Properties	✓	✓β		✓∅	✓
Observation	✓α	✓	✓		✓
Actuating and Actuator Properties		✓β		✓∅	
Conditionals	✓				

Table 3: Ontology classification

† Extends modules of SSN

α No time or location

β Implicit, implied by device functions

ε Rich energy model

S Service only

∅ Only small or provisional description, or a stub

In what follows, we discuss selected categories from Table 1. While, we have selected only some categories, this discussion provides a valuable insight to key aspects of use of semantic technologies in the IoT.

3.3.3.1.1 Thing

This category describes the general approach and provision of properties to any class of an ontology. All considered ontologies are, understandably, generic in this regard. Each contains only a handful of relevant properties that pertain to the very generic concepts. SSN's Things can have FeatureOfInterest (an abstraction of a real-world phenomenon, such as person, event or, literally, anything) and display Properties (a specification of DUL Quality; needs to be observable and inseparable from the SSN thing). SAREF defines a, similarly general, Property (specifying anything that can be sensed, measured or controlled). IoT-Lite extends the SSN with an Object (any physical entity) and its Attribute (any property exhibited by the Object that can be exposed by a Service). OpenIoT does not provide independent extensions or departures from the approach taken by the SSN. Instead, it provides subclasses for the SSN Property, mostly to describe entities needed in pilots of the project (e.g. WindSpeed, AtmospherePressure).

OneM2M BO is unique in its description of things, because the entire ontology is very general. It defines its own Thing class that captures, quite literally, any entity identifiable in a oneM2M system. OneM2M BO does not extend any upper ontologies, and its Thing is a direct subclass of `owl:Thing`.

Here, a Thing can have ThingProperty (which has a self-explanatory, all-encompassing definition). In this way, oneM2M BO displays characteristics of an upper ontology.

3.3.3.1.2 Device

Devices are at the core of the IoT. This is reflected in all ontologies. OneM2M BO proposes the simplest structure of a Device class that uses a written description, instead of rich ontological relations. Device has a single subclass of InterworkedDevice (one that does not directly implement oneM2M interfaces). A Device can consist of a number of other Devices.

In the SSN, the central taxonomy subtree consists of Device, Sensor, and SensingDevice subsuming both previous classes. An SSN System can represent any part of an infrastructure of devices connected in some way. In particular, it can be any Device in the System. Any System is comprised of subsystems (also of class System). IoT-Lite expands this structure with the addition of an ActuatingDevice and a (passive) TagDevice. Strangely, there is no definition of an Actuator. OpenIoT does not expand the basic structure of the SSN.

SAREF borrows from both, oneM2M and SSN. SAREF Device consistsOf any number of Devices, and has a DeviceCategory that, in turn, has its own subclass structure (which starts with FunctionRelated, EnergyRelated and BuildingRelated categories). It is meant to represent a given perspective (point of view) on a device (e.g. of user, administrator, manufacturer, etc.). On top of that, the ontology defines a couple of subclasses of the Device class, which range from general, such as a Sensor, to quite specific, like a WashingMachine (with classes, such as Switch, in between). Interestingly, Sensor and Actuator are not *neighbours* (the first being a subclass of a Device, and the latter of a DeviceFunction).

3.3.3.1.3 Observation

The second crucial element of any IoT ontology is the way that observations are modelled. They are fundamental data items, and their description very strongly affects possible use of a model and functionality of a concrete systems. In oneM2M BO, observations revolve around three general classes: Variable, Aspect and Metadata. Variable class encompasses input and output variables, as well as a ThingProperty, that pertains to any entity and can have additional Metadata. The latter class is a catch-all way of annotating observations (e.g. with units or precision), which lacks specification, i.e. any property structure is permissible under the BO Metadata. Aspects describe functionality as well as input or output Variables. This simplistic, high-level model of observations allows for great flexibility. On the other hand, there are no examples, and the intended use is very tersely explained. Lack of documentation, combined with elasticity of interpretation, may lead to systems being barely interoperable, despite using the same base ontology.

SSN proceeds differently, by extending the general model proposed by DUL. It introduces the Observation class. Each Observation results in a SensorOutput, a class with relations with other relevant information, such as ObservationValue, or the Sensor that saw the Observation. Observations have FeatureOfInterest that describes their characteristics, e.g. precision, latency, range, response time, etc. In general, the SSN Observation is a record of an occurrence of measurement, along with structured meta-data about the observation value, its properties, as well as the process leading to the Observation. Since the SSN lacks explicit units or time definitions, it needs to be complemented with relevant ontologies.

IoT-Lite does not extend the SSN Observation related modules. Instead, it proposes a vast simplification by introducing a Metadata class, similarly to the oneM2M BO. It is a generic class, intended to model any entity that does not fit the Unit or QuantityKind classes (a separate ontology is needed to describe the actual quantities). Observed values are not stored in the structure of the IoT-Lite. Instead, sensors are described in terms of types/kinds of observations made by them. For instance, one can construct a full description of a temperature sensor with meta-data of precision, unit, etc. However, within IoT-Lite, a series of concrete observations cannot be described.

OpenIoT extends the SSN Observation model by providing a Context, however, because of lack of documentation, the intended usage of this class is not clear. Nevertheless, it preserves the SSN Observation structure.

Finally, SAREF observations are described in terms of device Functions (in particular, SensingFunction and MeteringFunction). While lacking an explicit observation class, Functions have a number of properties that pertain to concrete values of measurements. Every relevant Function has a time value (e.g. hasMeterReadingTime) and an "observation" value (e.g. hasMeterReadingValue). These values are described in terms of Properties, which have concrete values alongside the UnitsOfMeasure. SAREF proposes its own taxonomy of units of measurements (currency, power, temperature…). Other than the values of concrete measurements, Functions have "reading types" (e.g. gas, pressure, energy…), which are implied to be relatively constant, vis-a-vis, for instance, meter readings of time and value. Compared to the SSN, the observation model in SAREF is simpler, and more focused on devices and their functions. It does not treat observations as pieces of data with their own structure and place in the system, which enables advanced data processing, e.g. analysis of historical data (within the structure given by the ontology). Instead, the SAREF model presents observations as tentative "outputs" of a function.

3.3.3.1.4 Device Deployment

A deployment description is a very important information in any system with multiple distributed devices. OneM2M BO interprets this category as a basic information about a network environment (AreaNetwork), but only if the device is proxied (InterworkedDevice). There is no standard way to model deployment information for any oneM2M BO Device.

SSN describes device deployment in terms of Platform(s) a Device is on, and System(s) it is part of. Even though the SSN itself does not define time or location properties, it is strongly implied that Devices, Systems and Platforms should be annotated with such information (no specific ontology to fulfill that function is suggested). SSN also defines a Deployment, a process with subprocesses (DeploymentRelatedProcess) that lead to the device becoming deployed. IoT-Lite extends the deployment aspect of the SSN by explicit use of geolocation from the WGS84 model. OpenIoT, on the other hand, provides a very peculiar extension of the SSN, namely it adds an OperatingProperty of Device, named EaseOfDeployment. No further description or explanation of its usage is provided.

In SAREF, deployment is understood in terms of physical space, in which a device is deployed, i.e. BuildingSpace, annotated with geolocation data from the WGS84. This is an interesting design decision, as it restricts SAREF Devices to be deployed only in buildings. It seems to contradict the design-time assumption that SAREF devices, i.e. smart appliances, can be located also in public spaces.

3.3.4 Summary

Each of considered ontologies proposes a different approach to modelling the IoT space. The biggest differences are in the details.

a) OneM2M BO proposes a small base ontology, similar to upper ontologies that provides only a minimal set of highly abstract entities. This allows for a very broad set of domain ontologies to be easily aligned with it. It also means that the BO itself is not enough to model any concrete problem (or solution) in the IoT. Furthermore, it does not capture some aspects (device, sensor and actuator properties) that are very common in other ontologies.

b) OpenIoT contrasts the oneM2M BO philosophy by providing a detailed model for a specific problem (i.e. pilot implementations from the OpenIoT project) that can be also be applied in a more general case, or in other solutions. Its heavy usage of external ontologies provides high semantic interoperability by design.

c) SSN is a developed model of the IoT in general, but with strong focus on sensor networks. It is based on DUL, and is clearly modularized, which makes it a good candidate for extensions into concrete systems and implementations. This is evidenced by the fact that other ontologies, evaluated here, make good use of it. When it comes to specificity, it places itself in the middle between oneM2M BO and OpenIoT.

d) IoT-Lite is an extension of selected SSN modules, mainly to include actuators. Rather than focusing on providing a detailed description of a delimited problem space within the IoT, it approaches the modelling problem from the perspective of an implementation device. It aims to deliver a small, but complete, model in order to simplify processing of semantic information. This is also its distinctive characteristics.

e) SAREF is a model with a strong focus on its own area—of smart appliances. Even though mappings to other standards exist, SAREF was developed from scratch to represent a specific area of application of the IoT. In this area, it delivers a strong and detailed base, that is also clear and easy to understand. At the same time, it is general enough to be used when extended to other domains, or solutions. Interestingly, all these ontologies almost completely disregard hardware specifications. It seems that the “place” of a device in an IoT system is much more important to ontology engineers than its hardware specification and resulting capabilities.

Results of our investigations show how different the existing conceptualizations of the same domain can be, depending on the context of the approach, and the applied ontology engineering methodology. Separately, we conclude that, while each considered ontology has its uses and caveats, two of them stand out in the context of the INTER-IoT project. These are SSN and SAREF. The first presents a model focused on sensors, but still robust enough, and with strong ontological basis. Those features make it a good choice in terms of interoperability (which is the focus of the project). In addition, the SSN is modular, extendable, and has been actually implemented and extended in other systems and ontologies (e.g. IoT-Lite and OpenIoT). SAREF, on the other hand, is a thoroughly modern ontology with many recommendations and relatively large scope, despite targeting only smart appliances. It already has alignments with other models, thus improving its interoperability.

In light of the facts and analysis presented in this section, the SSN ontology will be used as the basis of INTER-IoT reference meta-data model. We have found that it covers many meta-data requirements identified in INTER-IoT and is designed to be modular and extendible. It is also a core ontology, with many implementations in already deployed and well-tested systems. This makes it, in our view, the best currently available core IoT ontology for INTER-IoT.

3.4 Functional Model

The main purpose of functional decomposition[25] is to break up the complexity of the systems, under investigation, into smaller and more manageable parts, and to understand and illustrate their relationships to each other. Additionally, this produces a superset of functionalities that can be used to build any IoT system. The functional model is not directly tied to technology, application domain, or implementation. It contains both the Functionality Groups and the interaction between parts as a list of the Functionality Groups alone would not be enough to make up the Functional Model.

3.4.1 IoT-A Functional Model

The IoT-A project defines a Functional Model as

> *an abstract framework for understanding the main Functionality Groups (FG) and their interactions*

This framework defines the common semantics of the main functionalities of a system and is used for the development of the Functional Views.

The Functional Model, together with the Unified Requirements, is the base for the Functional View, which describes the system runtime Functional Components, including the responsibilities of components, their default functions, their interfaces, and their primary interactions. Various Functional Views could be derived from the same Functional Model.

Another concept that is very important in the IoT-A Functional Model is the *Functional Decomposition* (FD), that refers to the process by which the different *Functional Components* (FC) composing a specific service or application are identified and related to one another. The main purpose of Functional Decomposition is, on the one hand, to break up the complexity of a system compliant to the IoT ARM in smaller and more manageable parts, and to understand and illustrate their relationship on the other hand.

The IoT Functional Model diagram was derived from the main abstractions identified in the Domain Model, such as Virtual Entities, Devices, Resources and Users. Therefore, the “Application”, “Virtual Entity”, “IoT Service” and “Device” FGs are directly linked to the Domain Model parts. Considering the plethora of communication technologies that IoT needs to support, the need for a “Communication” FG is identified.

Furthermore, requirements expressed by stakeholders regarding the possibility to build services and applications on top of connected objects are covered by the “IoT Process Management” and “Service Organisation” FGs. To address consistently the concern expressed about IoT Trust, Security and Privacy, the need for a “Security” transversal FG is identified. Finally, the “Management” transversal FG is required for the management of and/or interaction between the functionality groups.

All in all, the IoT Functional Model contains seven longitudinal Functionality Groups complemented by two transversal Functionality Groups (as shown in Figure 16). These transversal groups provide functionalities that are required by each of the longitudinal groups. The policies governing the transversal groups will not only be applied to the groups themselves, but do also pertain to the longitudinal groups. As an example: for a security policy to be effective, it must ensure that there is no functionality provided by a component that would circumvent the policy and provide unauthorised access.

[25] http://soapatterns.org/design_patterns/functional_decomposition

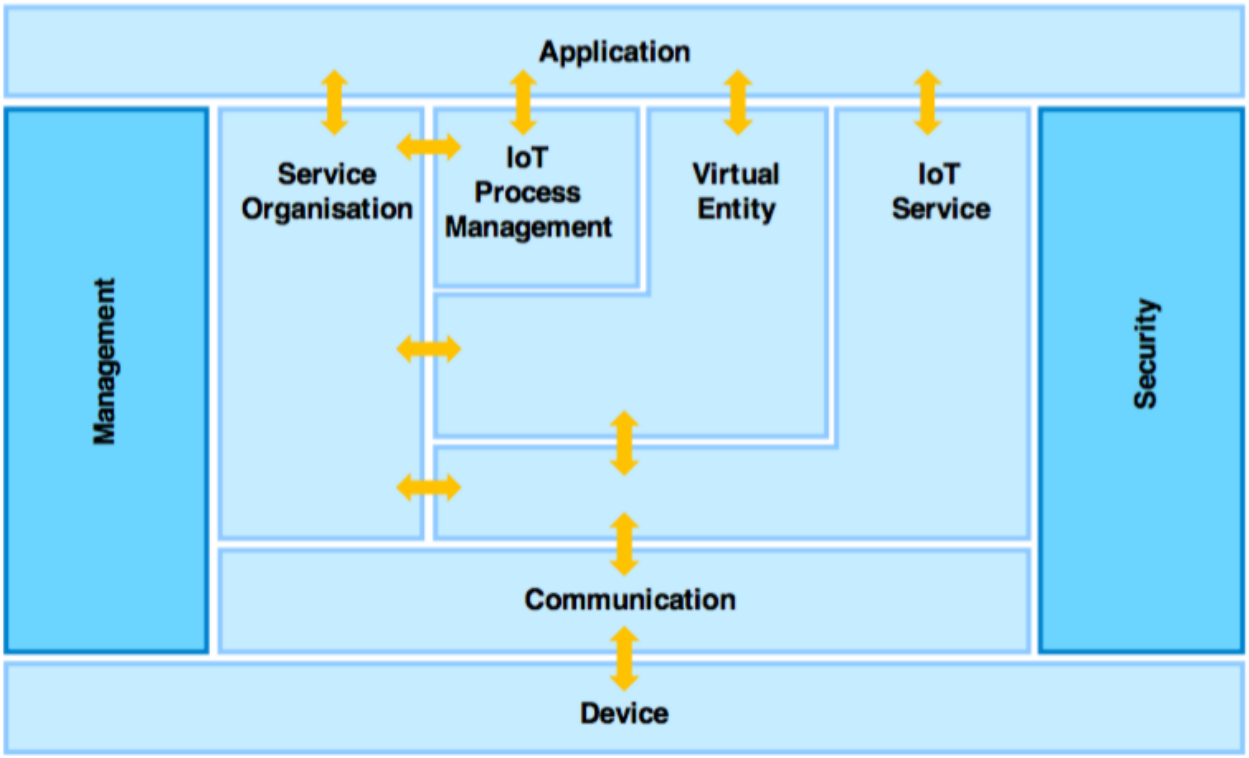


Figure 16: Example of functional model: IOT-A's functional model

3.4.2 IOT-A based functional analysis of IoT Platforms

We have performed an analysis of a set of IoT Platforms from the functional model point of view, matching the functional features of each platform against IoT-A's Functional Model. The aim of this analysis is to better understand the reality of the IoT Platforms, and don't make assumptions that could be erroneous, which could lead to disastrous results when trying to apply INTER-IoT results.

The analysis has also been useful for assessing the way that different IoT Platforms solve similar problems, helping in the design of the Functional Model of INTER-IoT.

Next, a brief description and a Functional Model analysis of each IoT Platform is shown. The list of IoT Platforms is depicted in Figure 10.

3.4.2.1 FIWARE

FIWARE[26] is a middleware platform, driven by the European Union under the Future Internet Public Private Partnership Programme[27], for the development and global deployment of Smart Applications for Future Internet in multiple vertical sectors.

The FIWARE platform provides a rather simple yet powerful set of APIs (Application Programming Interfaces) that ease the development of Smart Applications in multiple vertical sectors. The specifications of these APIs are public and royalty-free. Besides, an open source reference implementation of each of the FIWARE components is publicly available so that multiple FIWARE providers can emerge faster in the market with a low-cost proposition.

[26] https://www.fiware.org/

[27] https://www.fi-ppp.eu/

The key deliverables of FIWARE will be an open architecture and a reference implementation of a novel service infrastructure, building upon generic and reusable building blocks developed in earlier research projects.

FIWARE is based on the following main foundations:

- Service Delivery Framework – the infrastructure to create, publish, manage and consume FI services across their life cycle, addressing all technical and business aspects.
- Cloud Hosting – the fundamental layer which provides the computation, storage and network resources, upon which services are provisioned and managed.
- Support Services – the facilities for effective accessing, processing, and analyzing massive streams of data, and semantically classifying them into valuable knowledge.
- IoT Enablement – the bridge whereby FI services interface and leverage the ubiquity of heterogeneous, resource-constrained devices in the Internet of Things.
- Interface to Networks – open interfaces to networks and devices, providing the connectivity needs of services delivered across the platform.
- Security – the mechanisms which ensure that the delivery and usage of services is trustworthy and meets security and privacy requirements.

FIWARE GEs are grouped and organized in chapters. Each chapter provides a set of GEs that work and communicate together to give support the following areas (see FIWARE catalogue[28]):

- Data/Context
- IoT
- Advanced UI
- Security
- Interface to Networks and Devices (I2ND)
- Apps
- Cloud

FIWARE also provide Domain Specific Enablers (SE) aimed at provide functionality and APIs for these domains:

- Manufacturing
- Transport, logistics and agrifood
- Personal mobility
- Social connected TV, mobile city services and video games
- Smart cities and public security
- eHealth
- Smart energy
- Environment

Usually SEs depend on other SEs and GEs to provide a specific functionality.

[28] http://catalogue.fiware.org/

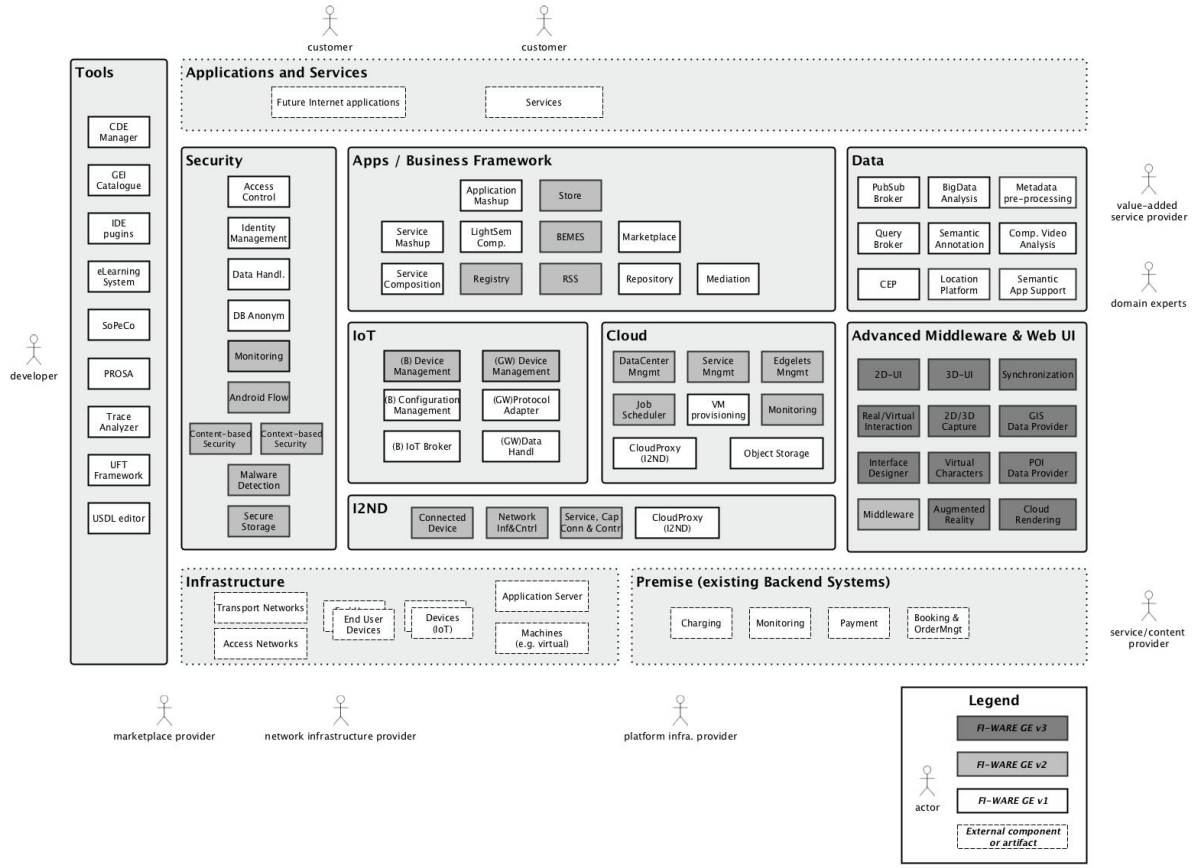


Figure 17: FIWARE architecture with the main Generic Enablers

FIWARE IoT

One of the most successful applications of the FIWARE initiative in real scenarios comprises the usage of its Data/Context Management enablers and infrastructure to build IoT ready scenarios with open source in a reliable way. As shown in the previous picture, FIWARE has a specific area devoted to IoT, separated from the Data Enablers. Since they both are extremely coupled in the domain of INTER-IoT (and IoT in general) they are treated jointly, as suggested in the officially maintained documentation page of the IoT Stack[29]. This IoT Stack is one of the latest association of GEs and definitions within a common domain and purpose[30].

[29] http://fiware-iot-stack.readthedocs.io

[30] Other similar set are also called bundles and those officially supported can be found in https://catalogue.fiware.org/bundles

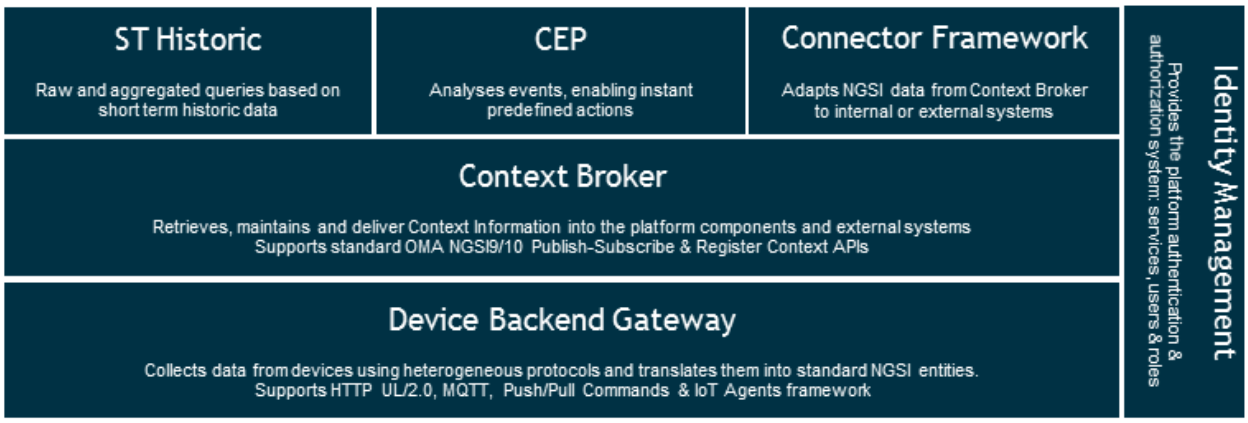


Figure 18 FIWARE IoT stack components

The IoT Stack groups the following FIWARE APIs:

- Authentication API
- Device API
- Data API
- Complex Event Processing API
- Management API

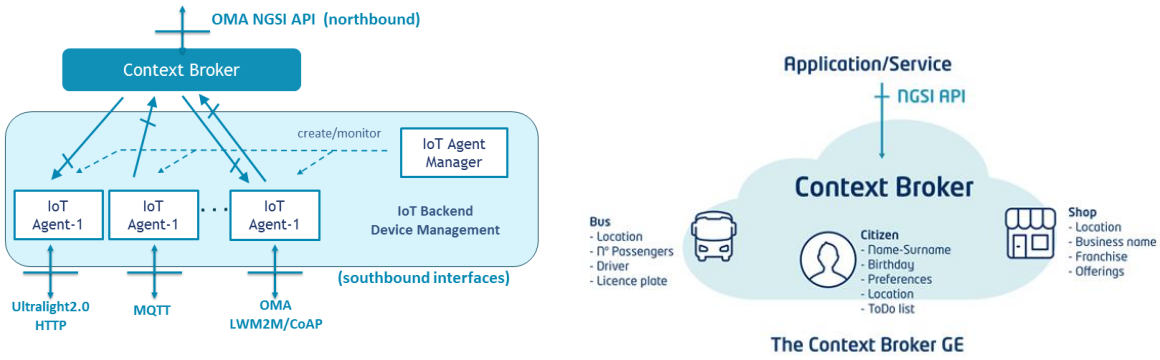


Figure 19: Context broker and IoT agents

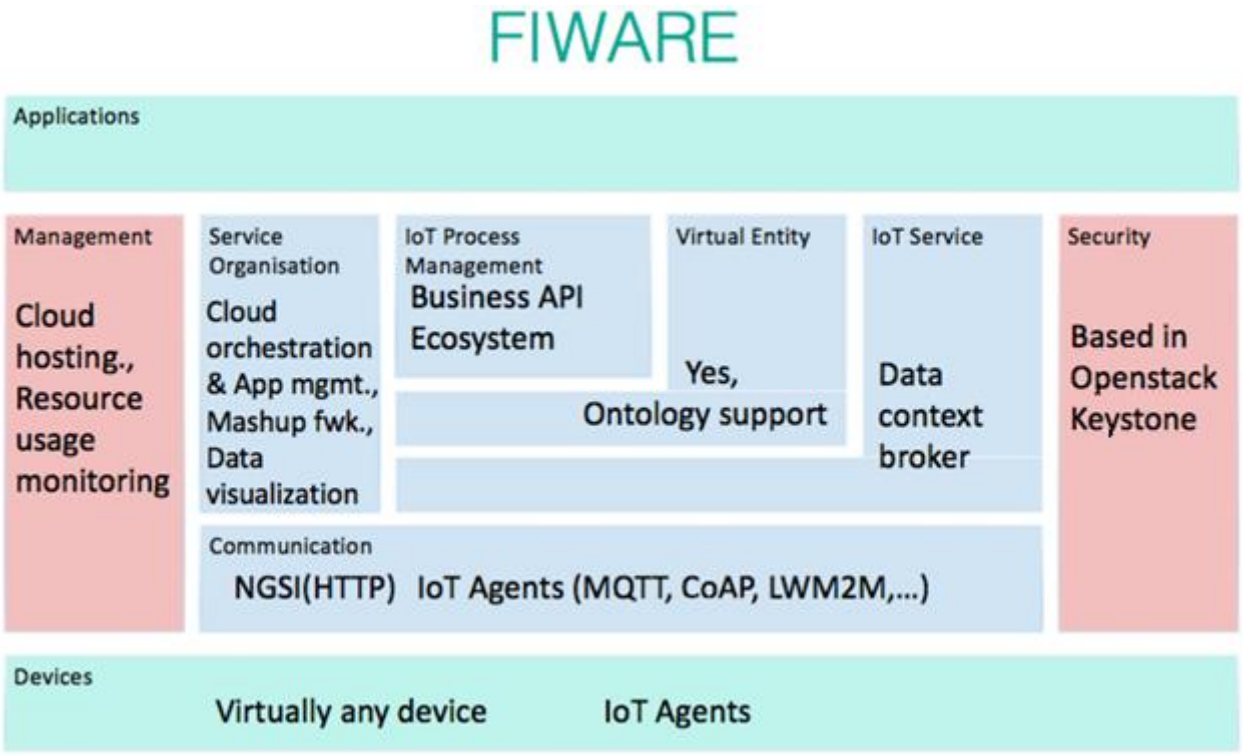


Figure 20: Relation of FIWARE with IOT-A functional model

3.4.2.2 OpenIoT

OpenIoT is open source middleware and development platform infrastructure that aims at:

- Collecting and processing data from virtually sensors, including physical devices, sensor processing algorithms, social media processing algorithms and more. (In OpenIoT the term sensor refers to any components that can provide observations)
- Semantically annotating sensor data, according to the W3C Semantic Sensor Networks (SSN) specifications.
- Streaming the data of the various sensors to a cloud computing infrastructure.
- Dynamically discovering/querying sensors and their data.
- Composing and delivering IoT services that comprise data from multiple sensors.
- Visualizing IoT data based on appropriate mashups (charts, graphs, maps etc.)
- Optimizing resources within the OpenIoT middleware and cloud computing infrastructure.

To achieve this architecture is divided in seven main elements divided in three planes;

Physical Plane

1. The Sensor Middleware (Extended Global Sensor Network, X-GSN), collects, filters, combines, and semantically annotates data streams from virtual sensors or physical devices. Acts as a hub between the OpenIoT platform and the physical world. The Sensor Middleware is deployed on the basis of one or more distributed instances (nodes), which may belong to different administrative entities. The prototype implementation of the OpenIoT platform uses the GSN sensor middleware that has been extended and called X-GSN (Extended GSN).

Virtualized Plane

2. The Cloud Data Storage.(Linked Stream Middleware Light,LSM-Light), enables the storage of data streams stemming from the sensor middleware thereby acting as a cloud database. Also, stores the metadata required for the operation of the OpenIoT platform (functional data). The prototype implementation of the OpenIoT platform uses a re-designed LSM Middleware, with push-pull data functionalities and cloud interfaces for enabling additional cloud-based streaming processing.

3. the Scheduler together with the Discovery Services functionality, processes all the requests for services from the Request Definition and ensures their proper access to the resources (e.g., data streams) that they require. This component undertakes the following tasks: it discovers the sensors and the associated data streams that can contribute to service setup; it manages a service and selects/enables the resources involved in service provision.

4. the Service Delivery and Utility manager, performs a dual role: combines the data streams as indicated by service workflows within the OpenIoT system in order to deliver the requested service (with the help of the SPARQL query provided by the Scheduler) either to the Request presentation or a third-party application (using the service description and resources identified and reserved by the Scheduler component) and acts as a service metering facility, keeping track of utility metrics for each individual service. This metering functionality will be used to drive functionalities as accounting, billing, and utility-driven resource optimization.

Utility/Application Plane

5. the Request Definition, selects mashups from a library in order to make a service presentation in a Web interface. Communicates with the Service Delivery & Utility Manager to visualize these services, obtaining the relevant data.

6. the Request Presentation , component enables specification of service requests to the OpenIoT platform providing a Web interface. It a set of services for specifying and formulating requests, while also submitting them to the Global Scheduler.

7. the Configuration/Monitoring, enables the management and configuration of functionalities over the sensors and the (OpenIoT) services that are deployed within the platform. Also enables the user to monitor the status of the different deployed modules[31].

Everything is running at the top of a JBoss application server, and it provides User Interfaces that include:

- IDE Core
- Request Definition
- Request-Presentation
- Virtual Sensor Schema Editor and
- Management, Monitoring and Editors

The following picture depicts the architecture in the IOT-A functional model to observe how the physical plane is in charge of the parts related with communication and devices (also with application if we take in account that for OpenIoT Twitter could provide data as a device), the green ones, and the virtualized plane and utility/management plane are in charge of the organisation, management and processing parts, the red and blue ones.

[31] OpenIoT draft http://cordis.europa.eu/docs/projects/cnect/5/287305/080/deliverables/001-OpenIoTD431Draft.pdf

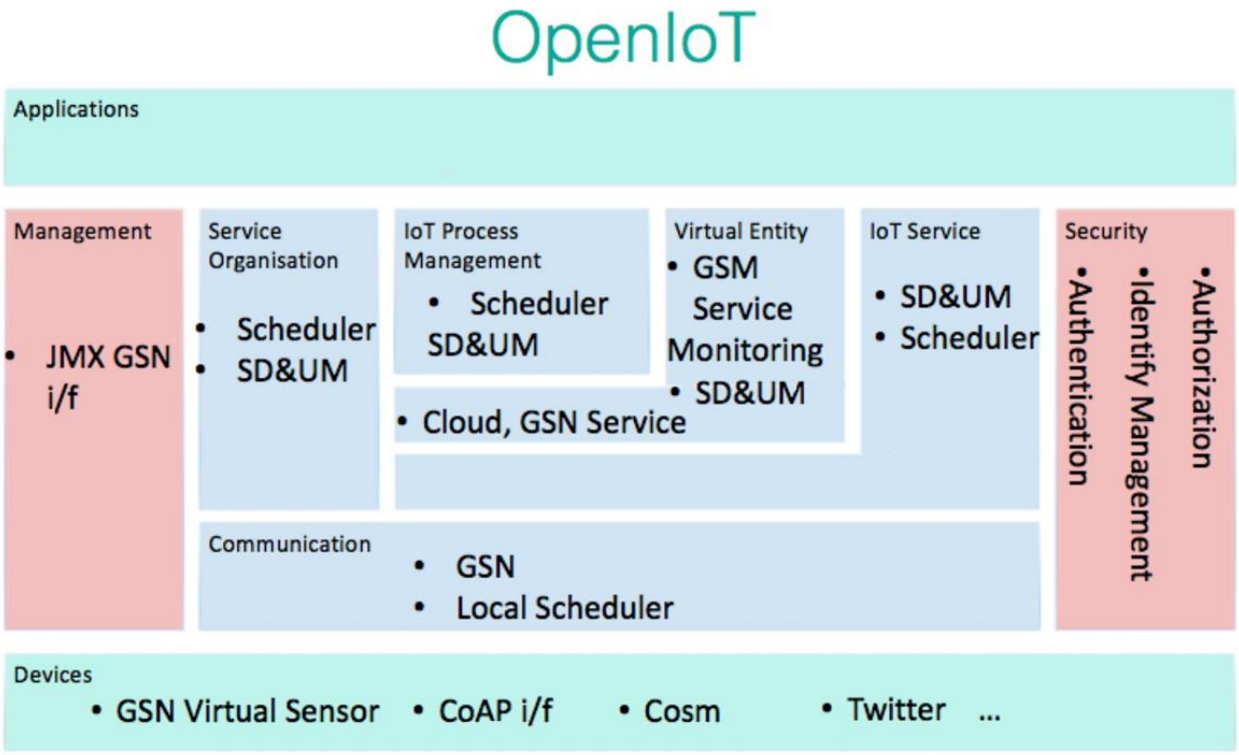


Figure 21: Relationship of OpenIoT with IOT-A functional model

3.4.2.3 UniversAAL

UniversAAL is the result of the homonymous FP7-ICT project UNIVERSAAL: UNIVERsal open platform and reference Specification for Ambient Assisted Living, coordinated by SINTEF[32] and developed by a consortium of 19 partners. Currently is maintained by some of their creators, who are creating the UniversAAL Coalition, expected to be officially announced during the first quarter of 2017.

UniversAAL is defined as an independent (funded publicly so far) IoT platform that provides a service-oriented environment – via an Enterprise Service Bus (ESB) model – enabling developers to utilize the cumulative potential of the sum of capabilities in the environment and compose their software applications over all verticals. This becomes possible through an implementation of semantic interoperability for SOA at the level of communication protocols (existing since 2008); this way, universAAL avoids domain-specific APIs by reducing syntactical dependencies to one single brokerage API, allows dynamic evolution of arbitrary constellations based on loose coupling, and enables integration and interoperability in a domain- and vendor-independent way.

UniversAAL communication protocols hide distribution and heterogeneity, currently with Java based implementations for the OSGi and Android runtime environments. It supports different types of targets, including mobile, embedded, and server- / Cloud-based.

The main features of universAAL are:

- 100% Semantic: it brings a list of extendable ontologies to define every communication that is produced within the system.

[32] https://www.sintef.no/en/

- Context bus: a single channel to publish and consume the context information produced by the attached entities.
- Service bus: a single channel to share the service-related information, defined in a semantic way.
- UI bus (optional): a single channel to transport all the UI interactions produced an requested. It implements multi-modality, so the interactions are adapted to the context and the end-user leveraging the semantic capacity of the system.
- Ontology management: it allows the extension of the existing ontologies and provides tools to support the basic operations with ontologies.
- Remote interoperability among universAAL instance: includes mechanisms to allow the coexistence and collaboration of multiple universAAL instances in a single network.
- Compatibility with some hardware technologies (Continua Alliance, ZigBee): provides adapters to a set of technologies for physical devices attachment.
- Semantic reasoner: includes a semantic reasoner to set triggers according to situations and thresholds in a semantic way.

UniversAAL is publicly available in github[33] under the Apache Software License 2.0.

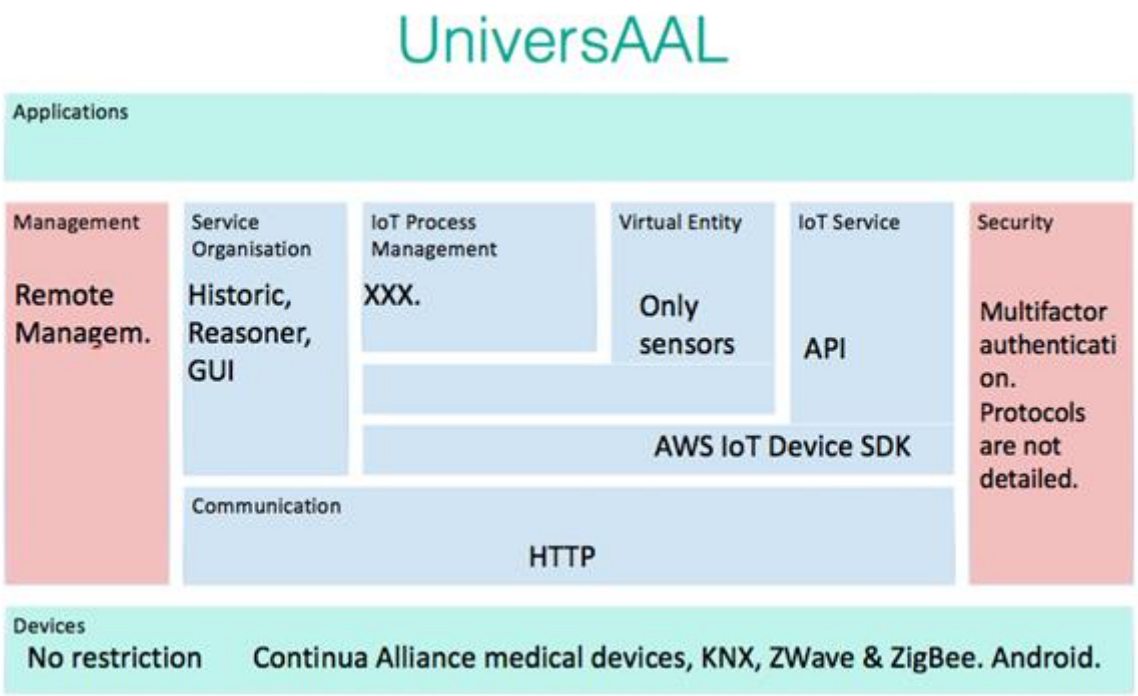


Figure 22: Relationship of UniversAAL with IOT-A Functional Model

3.4.2.4 OM2M

OM2M[34] is an open source project created by Eclipse under EPL license that implements the specification of oneM2M and SmartM2M standards.

[33] https://github.com/universAAL

[34] https://wiki.eclipse.org/OM2M/one

The main characteristic is that it implements a Service Common Entity (CSE), which is similar to a service layer, that can be deployed on a M2M server (CSE-IN) a gateway (CSE-MN) or a device (CSE-AE).

The features that the CSE offers are:

- Application Enablement
- Security
- Triggering
- Notification
- Persistency
- Device Interworking
- Remote Entity Management
- Routing
- and Communication

Is created in Java and runs at the top of an OSGi Framework called Equinox, so is modular and can be extended by plugins (OSGi Bundles). For building it uses Maven and Tyco. Each bundle offers specific functionalities and can be remotely installed, started, stopped, updated or uninstalled without reboot.

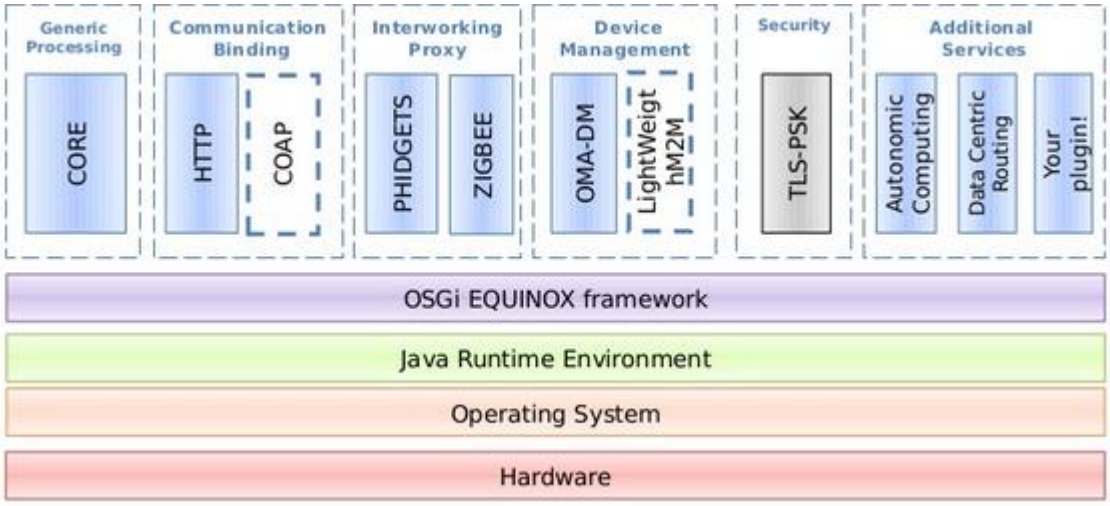


Figure 23: Eclipse OM2M Building Blocks

Additionally, it provides a RESTful API with primitive procedures for machines as:

- Authentication
- Resource Discovery,
- Application registration,
- Containers management
- Synchronous and Asynchronous communication
- Access right authorization
- Groups organization
- And re-targeting.

This API operates on the following primary resource types:

- **CseBase:** describes the hosting CSE, and is the root for all other resources within the hosting CSE.

- **remoteCse**: stores information related to M2M CSEs residing on other M2M machines after successful mutual authentication. It enables Cses interactions using retargeting operations
- **AE**: stores information about the Application Entity after a successful registration on the hosting CSE.
- **Container:** acts as a mediator for data buffering to enable data exchange between applications and CSEs
- **AccessControlPolicies**: manages permissions and permissions holders to limit and protect the access to the resource tree structure.
- **Group**: enhances resources tree operations and simplifying the interactions on the API interfaces by adding the grouping feature. It enables an issuer to send one request to a set of receivers instead of sending requests one by one.
- **Subscription**: stores information related to subscriptions for some resources. It allow subscribers to receive asynchronous notification when an event happens such as the reception of new sensor event or the creation, update, or delete of a resource.

This helps to develop services and applications independently of the underlying network. Also, It supports multiple protocol bindings such as HTTP and CoAP. Various interworking proxies are provided to enable seamless communication with vendor-specific technologies such as Zigbee and Phidgets devices.

We can apply the ARM FM to the OM2M obtaining a comparative very significant and relevant as we can see in the following figure[27].

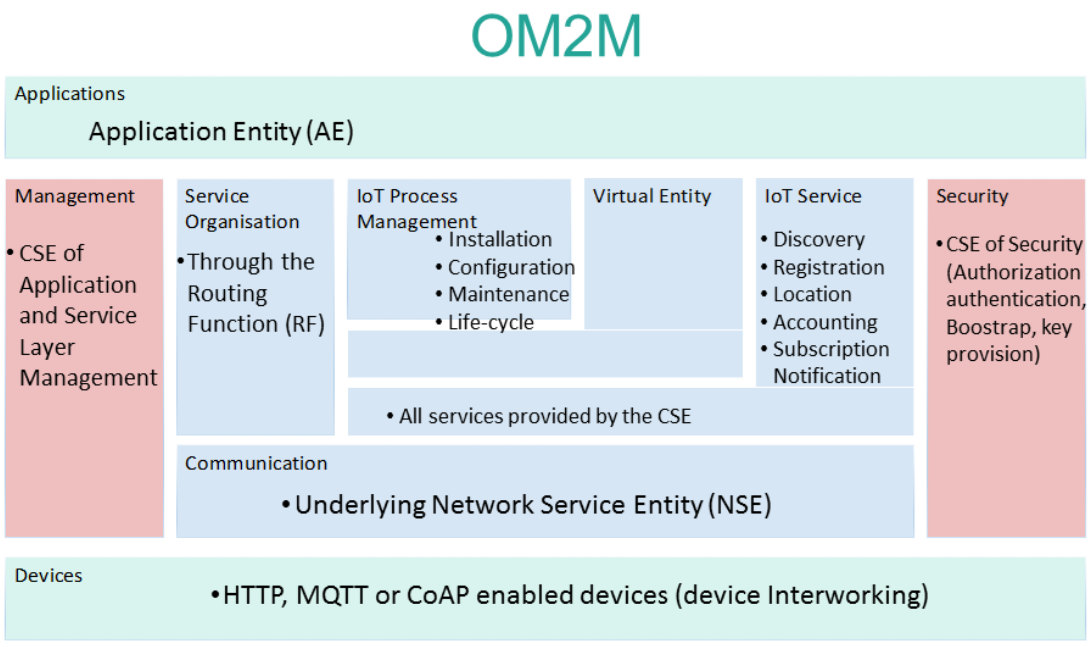


Figure 24: Relationship of OM2M and IOT-A Functional Model

3.4.2.5 Microsoft Azure

Azure IoT Hub is an extension towards the IoT domain that is integrated into Microsoft Azure cloud offering. Its main purpose is to enable reliable and secure bidirectional communications between a large number of IoT devices and a back-end engine, typically cloud-hosted. The Azure IoT Hub provides reliable device-to-cloud and cloud-to-device messaging, secure communications using per-device security credentials and access control. It offers extensive monitoring for device connectivity

and device identity management events and includes device libraries for the most popular languages and platforms. It also provides an IoT gateway SDK for the development processing and application logic at the edge.

The Microsoft Azure IoT platform is composed of core platform services and application-level components to facilitate the processing needs across three major areas of a typical IoT solution. This includes

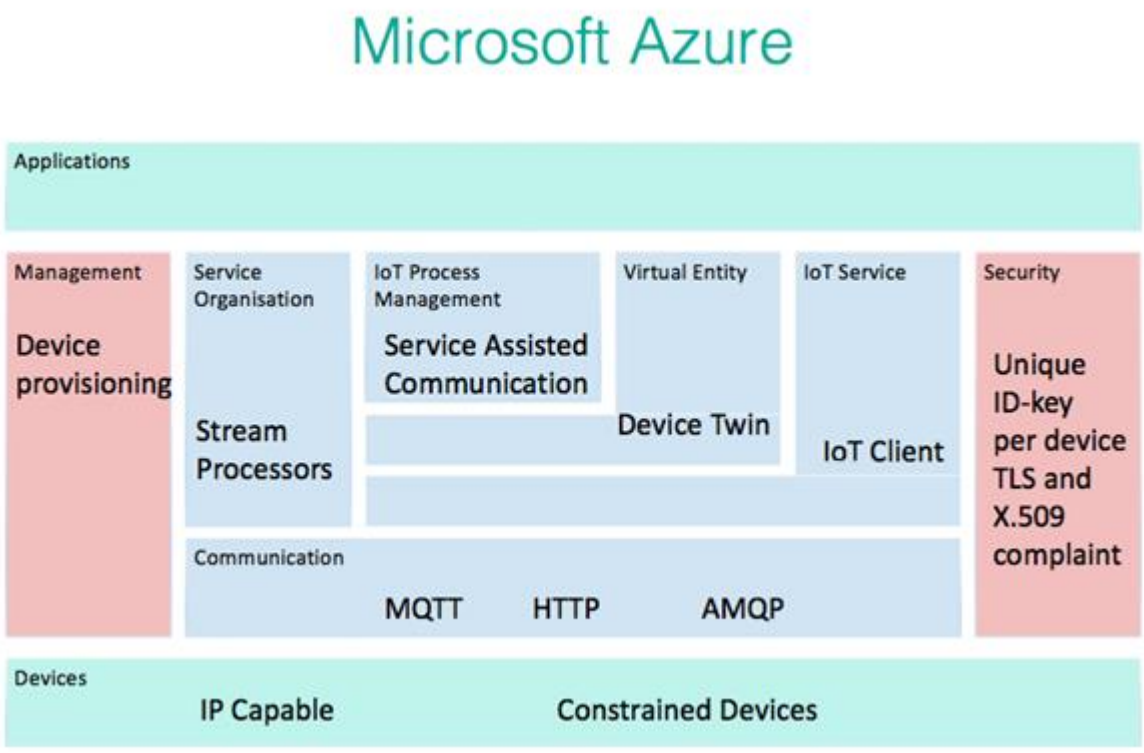
1. device connectivity
2. data processing, analytics, and management and
3. presentation and business connectivity.

Devices can be connected directly or indirectly via a gateway, and both may implement edge intelligence with different levels of processing capabilities. A cloud gateway provides endpoints for device connectivity and facilitates bidirectional communication with the backend system. The back end comprises multiple components to provide device registration and discovery, data collection, transformation, and analytics, as well as business logic and visualizations. The business integration and presentation layer is responsible for the integration of the IoT environment into the business processes of an enterprise.

For what concerns connectivity, Microsoft Azure Hub supports different connectivity options in order to integrate IoT resources, that can be connected directly or indirectly via so called field gateways. The main integration point towards the devices provides is the Azure IoT hub which offers support for three protocols:

- AMQP (with optional WebSocket support)
- MQTT and
- HTTP 1.1 over TLS protocols

The Azure IoT device SDK can be used to simplify the development of IoT clients that can connect to the Azure IoT hub via the options above. More constrained devices require a field gateway implementation to translate from protocols such as CoAP, OMA LWM2M, OPC, Bluetooth or ZigBee.



3.4.2.6 Amazon AWS IoT

AWS IoT is a managed cloud platform that lets connected devices easily and securely interact with cloud applications and other devices. It provides services hosted in the leading cloud services AWS (standing for Amazon Web Services) and leverages most of the services/modules on that to facilitate some of the most common processes such as

- AWS Lambda (serverless cloud computing)
- Kinesis (streaming data operations)
- S3 (cloud storage)
- Machine Learning
- DynamoDB (NoSQL database)
- CloudWatch (monitoring of cloud applications)
- CloudTrail (API calls logging)
- Elasticsearch Service with built-in Kibana integration (data visualization)

The AWS IoT module covers exclusively those aspects that are exclusive (or highly bound to) of the IoT domain (see the available documentation)[35]:

- Device management
- Device SDK connectors
- Message Broker
- Virtual entities (device shadows)
- Rules engine

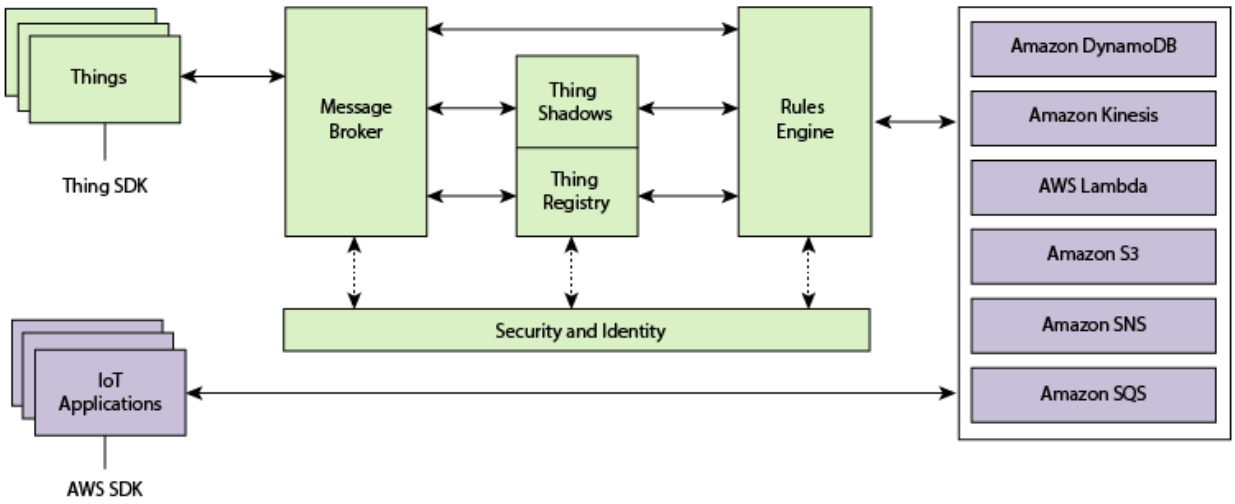


Figure 25: Amazon AWS IoT main architecture

[35] http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html

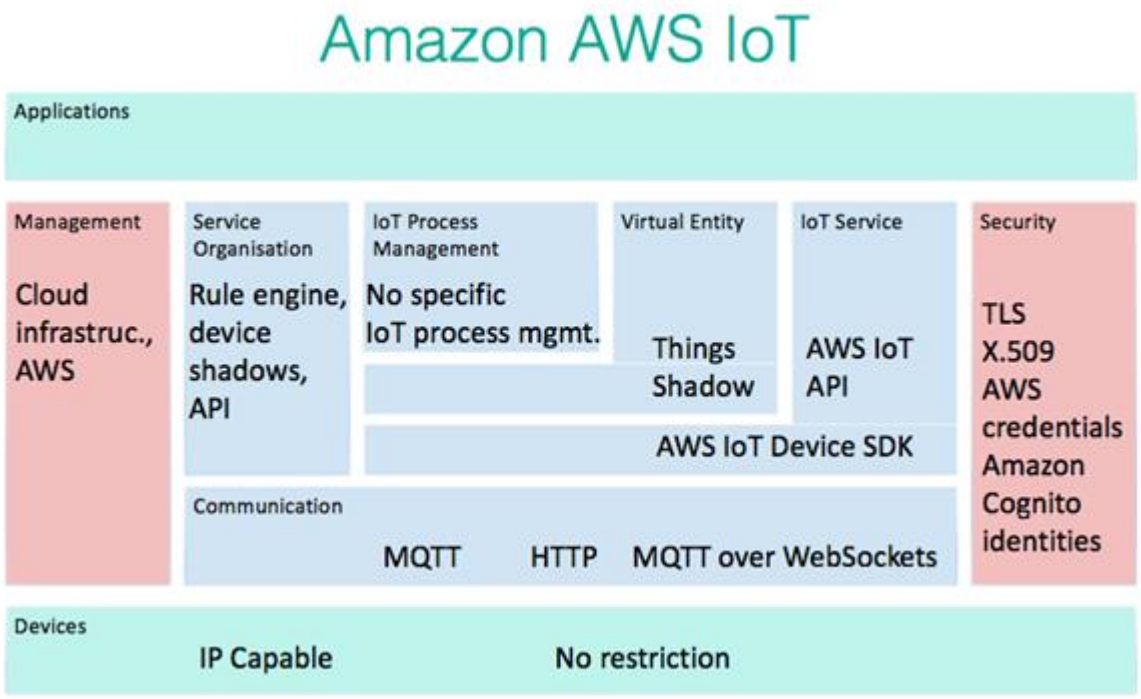


Figure 26: Relationship of Amazon AWS IoT

3.4.2.7 AllJoyn

AllJoyn is a collaborative open source software framework, that has since October 2016 been merged into IoTivity[36]. It is flexible, it promotes proximal network and it has an optional cloud connection. Being an open source project, it is licensed under the Creative Commons License and developed by the AllSeen Alliance[37] (including, among others, Qualcomm, Foxconn, Technicolor, LG-Innotek, LeTV, Microsoft and Xiaomi) and in collaboration with the Linux Foundation.

AllJoyn provides a universal software framework and core set of system services, which enabled interoperability among connected products and software applications. This is achieved by creating dynamic proximal networks using the D-Bus message bus. Compatible devices and applications could find each other and communicate in a client-server model across the boundaries of product categories, platforms, brands and connection types – including fields such as Connected Home, Smart TV, Smart Audio and Broadband Gateways. This is made possible by usage of introspection XML files, which are owned by each device on the network and they advertise device's abilities. AllJoyn is a core component in Windows 10.

The framework of AllJoyn has both routers and apps; communication between the latter always goes through the former. The AllJoyn application advertises its services, and when a neighbouring application discovers the application that does the advertising, it can create a session by connection to a specific port. Sessions between applications can be either point-to-point or multi-point.

Apps and routers can run on the same physical device. In the case that the app uses its own router, the router is called *bundled router*. If all apps on the device share a common router, then this router is called a *standalone router*. This is common on Linux systems where router runs as a daemon

[36] https://www.iotivity.org/

[37] https://allseenalliance.org/

process. However, if an app uses a router on a different device, which is common on embedded devices, we use for it the term *thin app*.

To facilitate interoperability between apps, AllJoyn implements service frameworks, which implement a set of common services, like onboarding, notification and control panel. These are divided into AllSeen working groups, such as:

- **Onboarding** - to provide a consistent way to bring a new device onto the wi-fi network,
- **Configuration** - to allow one to configure certain attributes of an application or device, such as its name,
- **Notifications** - applications can send and receive text-based, as well as audio and image messages directly or via URLs,
- **Control panel** - to allow remote access to a virtual control panel for the device.

AllJoyn apps can communicate with one another through wi-fi, Ethernet, serial or Power Line (PLC).

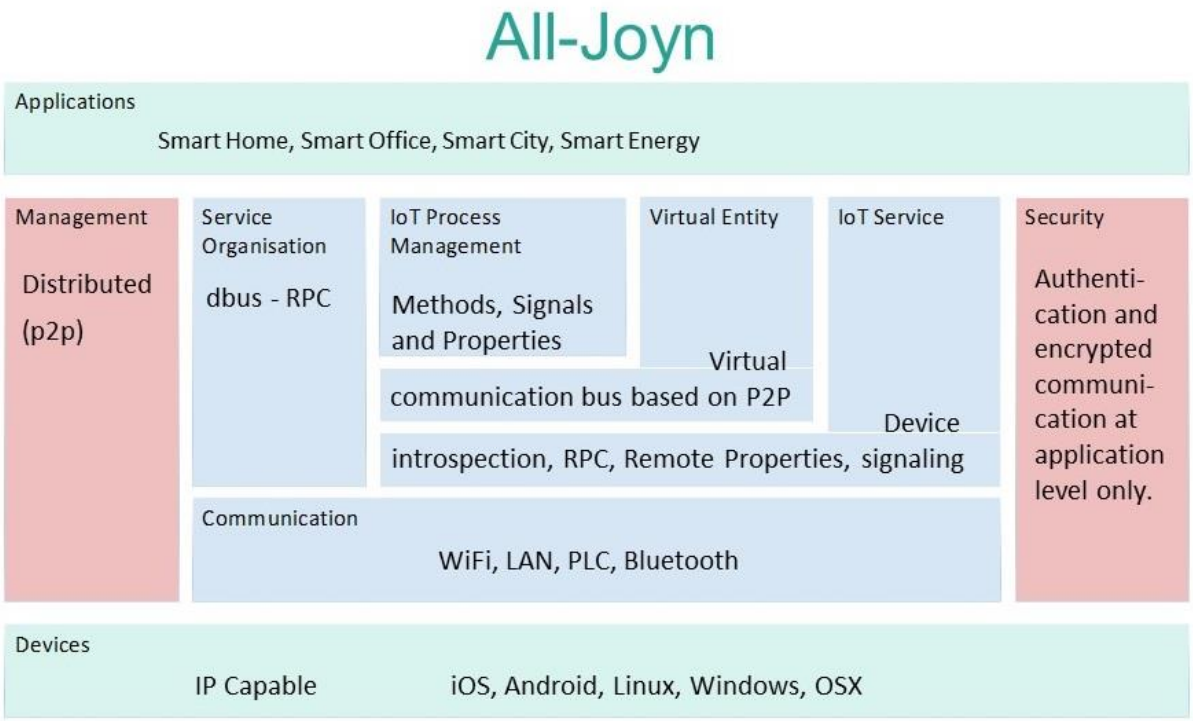


Figure 27: Relationship of AllJoyn with IOT-A functional model

3.4.2.8 Butler

BUTLER (uBiquitous, secUre inTernet-of-things with Location and contEx-awaReness) is a FP7-ICT project ended in 2014 and involved 17 partners (6 academic institutions and 11 companies), coordinated by INNO. Leveraging on a context and location aware, pervasive information system, *BUTLER* aimed at the development of inherently secure (from physical to application layer), energy-efficient and optimized applications spreading across different scenarios (Home, Office, Transportation, Health, etc.). Indeed, *BUTLER* provides a horizontal platform where IoT devices can be reused by various applications from different domains via intermediate value added services such as localization, context capturing, behaviour capturing, security management, etc.

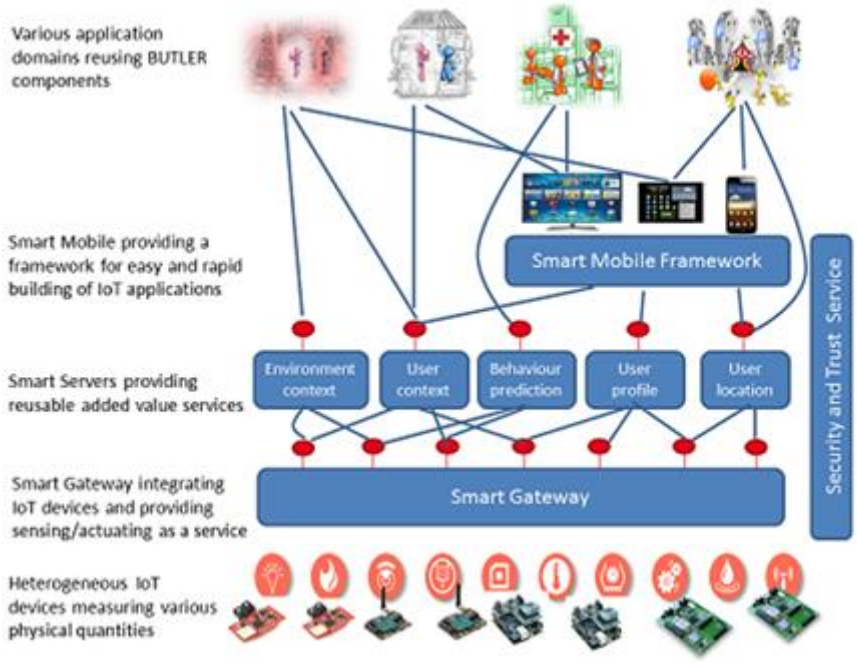


Figure 28: Butler generic architecture

To such purpose, *BUTLER* presents a smartDevice-centric network architecture where smartObject (sensors, actuators), smartMobile (user's personal device) and smartServers (providers of contents and services) are interconnected directly over IPv6 or by means of a SmartGateway (for devices adopting CoAP, ZigBee, BT, NFC, etc.). Just the Butler SmartGateway plays a crucial role for the integration of heterogeneous SmartObjects by representing different devices in a homogeneous way through a Service-Oriented approach and through several IoT Protocol Adapters. In particular, *BUTLER* layered architecture (Communications Layer, Data/Context Management Layer, System/Device Management Layer, Service Layer) is modular, extensible and domain independent since it implements a set of principles and guidelines that can be used to build any kind of IoT systems. Indeed, integrates existing and develops new technologies to form a "bundle" of applications, platform features and services that will bring IoT to life.

BUTLER has taken advantage of existing efforts, either within the framework of the research-related EU initiatives (e.g., especially IoT-A for the Butler Information Model definition and FIWARE for the context abstraction), or supported by other industry standards body (e.g., OMA, OAuth, SAML 2.0 or OSGi). All the components developed in the project, integrated demo (e.g. with *iCore* platform) and trials are available on the IOT OPEN PLATFORMS portal.

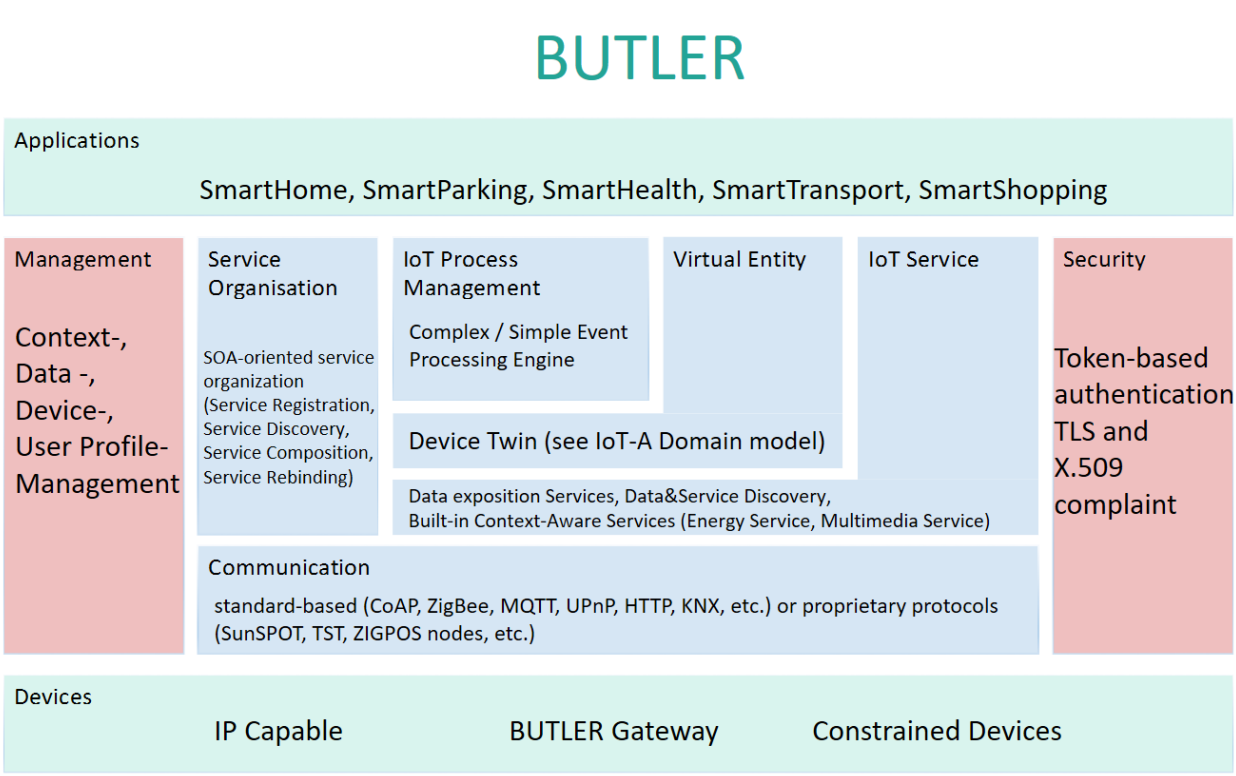


Figure 29: Relationship of BUTLER with IOT-A functional model

3.4.2.9 i-Core

iCore is a FP7-ICT project ended in 2014 and involved 19 partners from industry, research and academia, coordinated by CREATE-NET. The *iCore* proposed solution for addressing the heterogeneity of objects and the need for resilience in very large IoT scenarios is a cognitive application domain neutral management framework. Although most of the *iCore* concepts have been inherited by the IoT-A (Internet of Thing- Architecture), *iCore* building blocks refer to four specific concepts (virtualization, composition, cognition and proximity) spread among a three-layered (VO Level, CVO Level, Service Level) architecture.

At the first level the focus is on the virtualization activity, that allows linking every real-world object (RWO) with a digital always-on alter ego, called virtual object (VO). VOs reflect RWOs status and capabilities, and can be dynamically created, destroyed or changed. At the second level, the focus in on the composition activity, since VOs are aggregated in more sophisticated entities, called composite virtual objects (CVOs). CVOs are cognitive mashup of semantically interoperable VOs aiming at rendering services in accordance with both the application and user requirements. At the third and last level, the Service one, mechanisms related to User Characterization, Situation Awareness and Intent Recognition support the Service Request Analysis, whose output provides the input parameters for the composition processes of CVO Level. Learning mechanisms, Semantic Query Matchers, and RDF Rules Inference Engines are the enabling supports for the Service Execution Request process, in accordance to the stored policies.

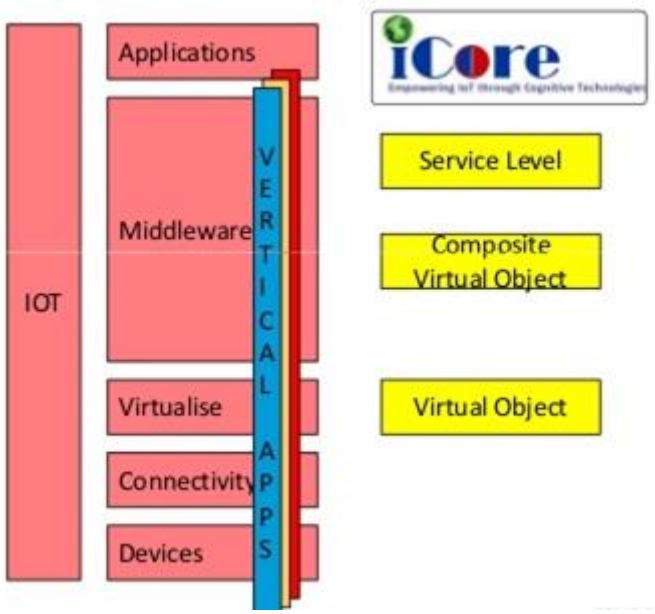


Figure 30: i-Core generic architecture

Cognition spreads in all the three aforementioned architectural levels, under different forms (optimization techniques, learning mechanism, ontology, etc.). In detail, at VO Level cognition needs for VOs self-management and self-configuration in order to handle data flows, to optimize resources, to monitor relevant RWOs. At CVO Level, cognition needs for meeting the application requirements and the VOs/CVOs capabilities, choosing between VOs/CVOs candidates, recognizing already faced scenarios (pattern recognition and machine learning techniques) and reuse or adapt already built solutions. Finally, at Service Level cognition is used as semantic reasoning in order to capture the application requirements, translate them into appropriate request service format and so guide the selection process at the lower levels.

The proximity concept instead expresses the level of relatedness/usefulness between any IoT user/application and any object in order to achieve more and more automation and scalability in the cognitive selection of VOs/CVOs.

Several *iCore* trials and integrated demo (e.g. with *Butler* platform) related to different application domains (home automation, logistics, security, etc.) have been realized as proof-of-concept. *iCore* resources are subject to different licenses.

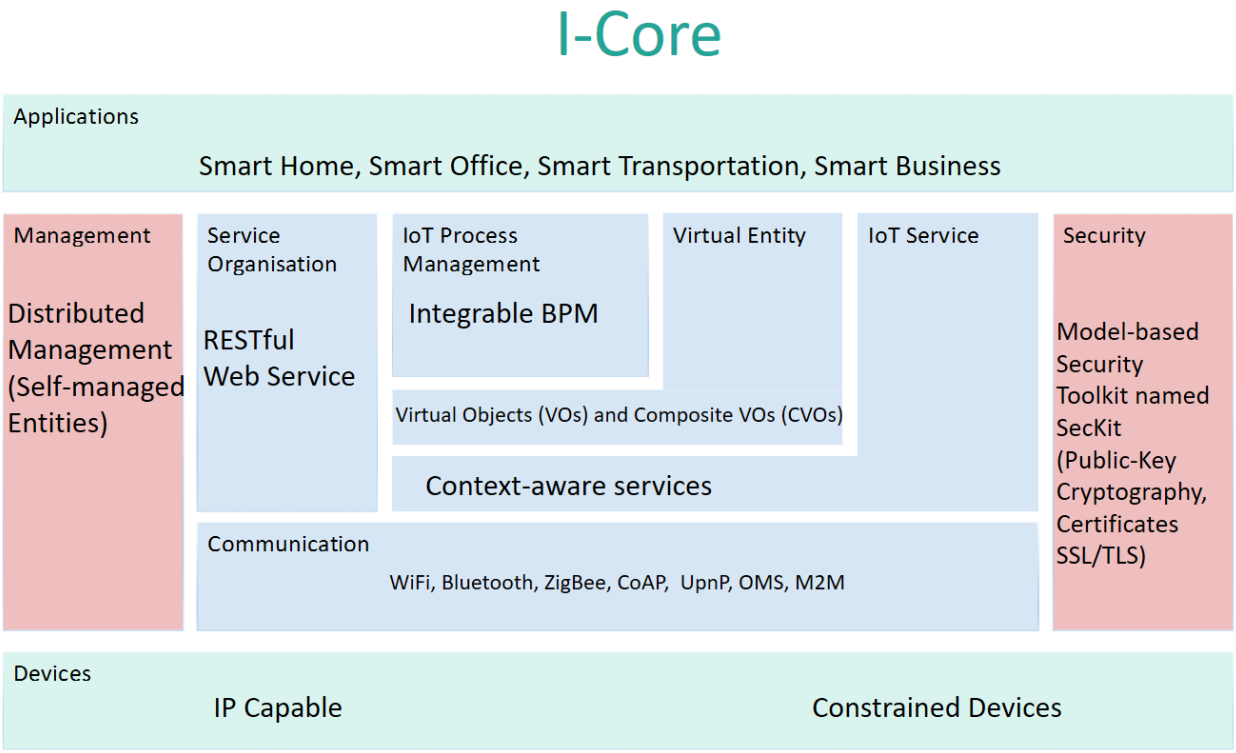


Figure 31: Relationship of i-Core with IOT-A functional model

3.4.2.10 Sofia 2

SOFIA2[38] IoT is a Platform created by the union of an Open Source project called SOFIA (Smart Objects For Intelligent Applications), which is a middleware, and the Indra company effort.

SOFIA was a middleware architecture that allowed the interoperability of several systems and devices. It allowed making real information available for intelligent. This interoperability was achieved using different applications that share semantic concepts.

Some of its main advantages are:

- Open-source
- Multi-platform: Available for MS Windows, Android, Linux, iOS…
- Multi-language: It has libraries in Java, JavaScript, C++, Arduino…
- Communication agnostic: With implementations for TCP, MQTT, HTTP (REST and WebServices), Ajax Push, …

Late on, Indra company kept evolving the original SOFIA project, creating a platform that focuses on enterprise use. The current version of the Platform is called SOFIA2.

This platform allows the interoperability between multiples IT systems and IoT devices, joining the aforementioned middleware with a repository capable of processing thousands of events per second, with huge storage and Big Data analytics and additionally offers:

- Real time interoperability between systems, networks, devices and sensors in a feasible and secure way.
- Design of actuation rules from data received and learning through Big Data Advanced Analytics

[38] http://sofia2.com/

- Incorporation of georeferenced visualization tools, integrating information from several sources and synoptics about the operation.

Even though it's not Open Source, it provides an Open API and client to access its services. Furthermore, SOFIA2 is focused on these areas:

- Adapting it to the enterprise environment: High availability operation with distributed data centres.
- Working with the Platform was simplified, particularly in the following areas:
  - Ontology development (ontologies became lightweight)
  - Query language
  - Smart Space Access Protocol: With a JSON implementation besides the XML one.
- Big Data Interfaces (Hadoop) to host huge amounts of data and data warehouse.
- Integration capacities with back-ends using standard protocols, e.g. Web Services.
- Plug-in concept to expand the Semantic Information Broker
- Integrated storage and GIS queries
- Addition of pluggable security mechanisms.
- REST interfaces to connect easily from smart phones, devices, RIA applications, …

To better understand the architecture of the platform, SOFIA2 can be conceptualized through these concepts:

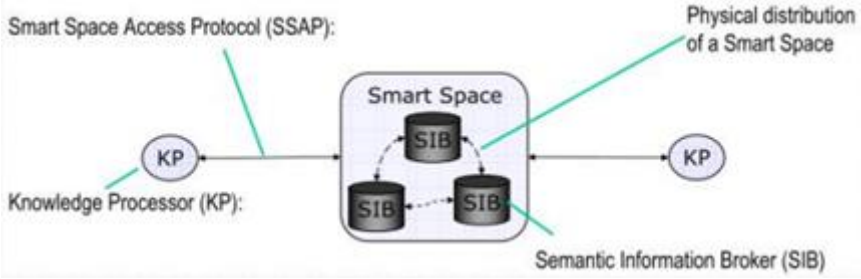


Figure 32: SOFIA2's conceptual blocks

- **Ontology**: The entities handled inside the Smart Space and exchanged between the Things and the SIB.
- **Smart Space:** is the virtual environment where different devices and applications interoperate with each other to provide a complex functionality.
- **Semantic Information Broker (SIB):** core of the Platform. It receives, processes and stores all the information of applications connected to the SOFIA Platform, thus acting as the Interoperability Bus. All the existing concepts in the domain (reflected in the ontologies) and their current states (specific instants of the ontologies) are reflected on it.
- **Knowledge Processor (KP):** Represents each element which communicates with a Smart Space by producing and/or consuming information.
- **Smart Space Access Protocol (SSAP):** This is the standard messaging language to communicate between the SIBs and the KPs. There are two implementations: XML or JSON.

It can be appreciated that the main field of work is on the Ontologies, and the interoperability in this area. Thus, applications sharing classes (commonly called concepts) from the same ontology can easily exchange information using specific instances of those common classes. Sofia2 represents ontologies in JSON format to be used by the KP, representing determined data[28].

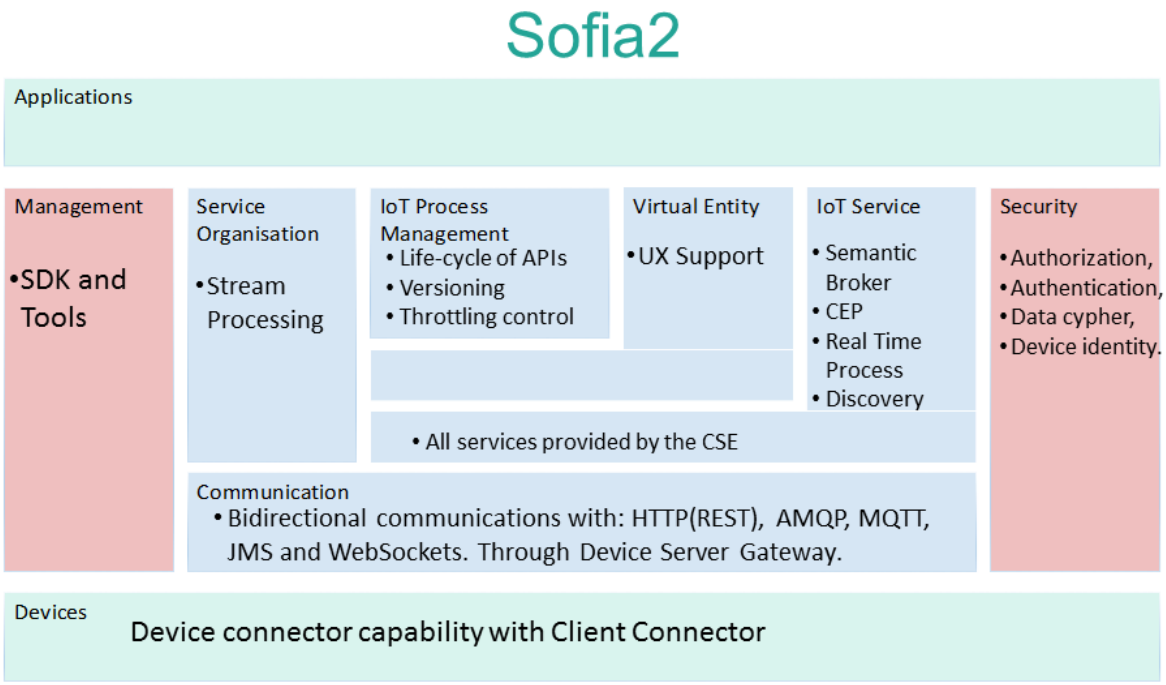


Figure 33: Relationship of Sofia2 with IOT-A functional model

3.4.2.11 ThingSpeak

Attending to the description of its developers; "ThingSpeak[39] is an open source IoT application and API to store and retrieve data from things using **HTTP** over the **Internet** or via a **Local Area Network**."[29]

So is not properly said a platform but an application to support and build IoT information ecosystem on the application and service level. The main characteristics it that you can create sensor logging applications, location tracking applications, and a social network of things with status updates to have a handler application for data.

Between its functionalities, ThingSpeak allow to storing and retrieving numeric and alphanumeric data, the API allows numeric data processing such as time-scaling, averaging, median, summing, and rounding.

Also, there are the called Channels, which are the main build block of the system and store all the data that a ThingSpeak application collects, and supports data entries of up to 8 fields that can hold any type of data, plus three fields for location data and one for status data (latitude, longitude, elevation, and status). The channel feeds support the following formats: JSON, XML, and CSV.

One can get data into a channel from a device, website, or another ThingSpeak channel. Once one collects data in a channel, you can use ThingSpeak Apps to analyze and visualize it. Also it supports

[39] https://thingspeak.com/

time zone management, read/write API key management and JavaScript-based charts from Highslide Software / Torstein Hønsi.

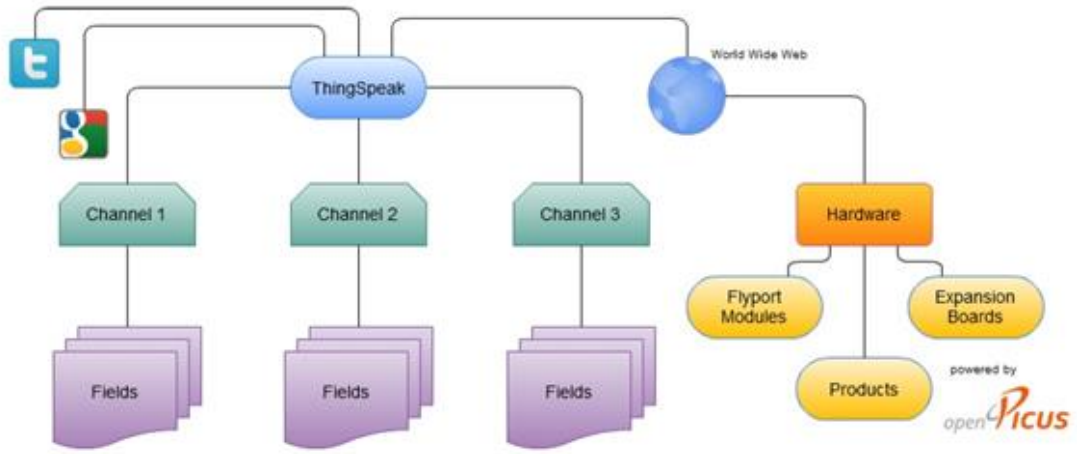


Figure 34: Architecture of ThingSpeak

Detailed study of channels allows one to obtain the settings or parameters needed to create the communication and to start to retrieve data from the devices to the ThingSpeak applications.

Channel Settings

- **Channel Name:** Enter a unique name for the ThingSpeak channel.
- **Description:** Enter a description of the ThingSpeak channel.
- **Field#:** Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.
- **Metadata:** Enter information about channel data, including JSON, XML, or CSV data.
- **Tags:** Enter keywords that identify the channel. Separate tags with commas.
- **Latitude:** Specify the position of the sensor or thing that collects data in decimal degrees. For example, the latitude of the city of London is 51.5072.
- **Longitude:** Specify the position of the sensor or thing that collects data in decimal degrees. For example, the longitude of the city of London is -0.1275.
- **Elevation:** Specify the position of the sensor or thing that collects data in meters. For example, the elevation of the city of London is 35.052.
- **Make Public:** If you want to make the channel publicly available, check this box.
- **URL:** If you have a website that contains information about your ThingSpeak channel, specify the URL.
- **Video ID:** If you have a YouTube or Vimeo video that displays your channel information, specify the full path of the video URL.

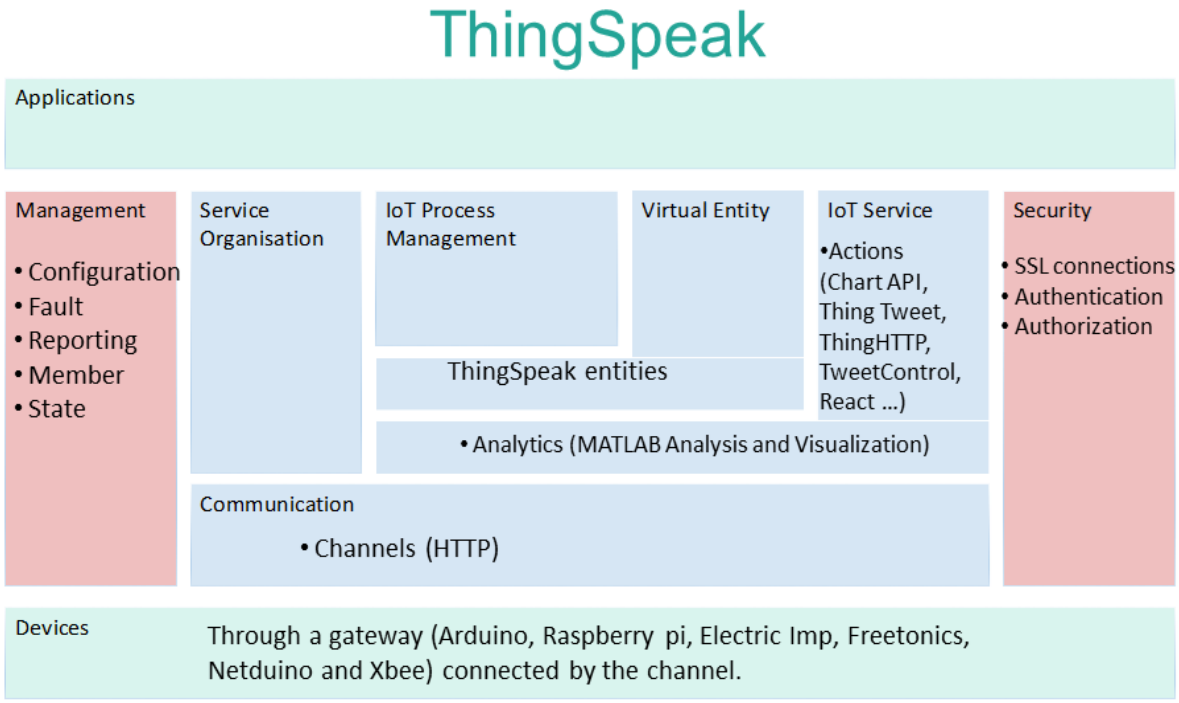


Figure 35: Relationship of ThingSpeak platform with IOT-A functional model

3.4.2.12 GE Predix

Predix is the key product from GE in the IoT Platform field, and it is targeted for the Industrial Internet. According to GE, this platform should help Business in creating different innovative solutions based on Predix capability of handling real-time operational data and transform this into valuable knowledge. The platform should be a one-stop shop allowing users to have secure, fast and effective deployments for industrial apps.

Clearly, GE has a very large industrial know-how and this can help companies transform themselves. Predix is used first and foremost within GE business, and this knowledge and experience based on GE manufacturing operations, securing and monitoring the approximately $1 trillion GE industrial assets deployed worldwide, is for sure a unique asset to this tool.

GE decided on a platform because it offers a standardized way to enable an entire business to quickly take advantage of operational and business innovations. By using a platform that is designed around a reusable building block approach, developers can:

- build apps quickly,
- leverage work elsewhere,
- reduce sources of error,
- develop and share best practices,
- lower risk of cost and time overruns,
- future-proof their initial investments .

Independent third parties can also build apps and services on the platform, allowing businesses to extend capabilities easily by tapping the industrial ecosystem. The cloud model allows businesses to take advantage of key capabilities including:

- economics of a centrally managed and shared infrastructure in a pay-as-you-go subscription model,
- scale to meet different business and application workloads by easily adjusting capacity on-demand,
- assets can be connected across the entire business so data can be captured,
- analytics can be developed and run to deliver insights at all levels of the organization.

A common cloud architecture also enables improved system governance, standardized security vulnerability assessments, and release management control and consistency

By combining cutting-edge IT with leading-edge OT, Predix brings world-class software innovation to your assets and operations, while integrating within your organization's existing IT systems. Predix is the only platform designed to:

- address the key challenges that prevent growth and market competitiveness,
- capture and analyse the velocity, volume, variety, and complexity of industrial data,
- meet the demanding needs for industrial grade, end-to-end cyber, informational, and operational security.
- innovate faster by eliminating the barriers to entry to develop industrial apps for new business outcomes,
- take advantage of an industry-wide ecosystem of partners to extend capabilities through integrated software, hardware, and services.

Predix provides fast access to data and timely analytics while minimizing storage and compute costs It offers a secure, multi-tenancy model that includes network-level data isolation and encrypted key-management capabilities It also supports the ability to plug in analytic engines and languages to interact and process the data. There are four key components:

1. **Connection to the source**: Connections are established with GE and non-GE machine sensors, controllers, gateways, enterprise databases, historians, at les, and cloud-based applications.
2. **Data ingestion**: Data is ingested from the source in real time, and by bulk upload Workflow tools allow the user to identify specific sources and to create default data flows for all—or specific—data sets and data types, including unstructured, semi-structured, and structured These tools speed the design, testing, and generation of code, making it easier to manage and monitor simple, one- time projects to complex, ongoing data synchronization project.
3. **Pipeline processing**: The ingestion pipeline can efficiently ingest massive amounts of data from millions of assets However, data can be messy, arrive in different formats, and come from multiple sources, all of which make running predictive analytics di cult Pipeline processing allows the data to be converted to the correct format so that predictive analysis and data modelling can be done in real time The pipeline policy framework provides governance and catalogue services, allowing users to perform data cleansing, increase data quality, data enrichment (for example, merging with location or weather data), data tagging, and real-time data processing.
4. **Data management**: Data needs to be stored in the appropriate data store, whether it be time series for machine sensor data, Binary Large.

Object (BLOB) (for example, MRI images), or an RDBMS This allows use of the data for both operational and analytical purposes It also provides data blending capabilities, where users can

deploy tools to extract value from these data sources to patterns and process complex events (i.e., look for a combination of certain types of events to create a higher-level business event)

To take advantage of the Industrial Internet, integration with existing—and future—equipment, data, and analytics is critical, especially in brownfield sites Predix achieves this at a number of levels:

- **Machines**: Connect machines of any vendor or vintage Predix machine supports a number of protocols, including OPC-UA, DDS, and MODBUS, as well as TCP-based sockets communication.
- **Data**: Standard connectors are included for time series, location, ERP, and CRM systems Custom connectors can also be built to incorporate proprietary data schemas.
- **Programming languages / tools**: Support is provided for Java, Node js, Python, Artefactory, GitHub, JaCoCo, and Ruby on Rails
- **Analytics**: Support is provided for Java, Matlab, and Python
- **Mobile devices**: By supporting HTML5, existing desktop browsers, smartphones, and tablets can be used across the business.

3.4.2.13 Contiki

Contiki is a highly portable operating system for constrained systems with a focus on low-power wireless IoT devices. While there are many similar OS such as TinyOS, what makes Contiki different is the completeness and flexibility it offers to the programmers.

Contiki can fit into 10kB of RAM and 100kB of ROM. It runs on small microcontroller architectures such as Atmel AVR, 8051 SoC, ARM-powered and MSP430 devices and includes a very light implementation of IP called uIP. uIP otherwise known as “micro IP”, was designed to incorporate minimal set of components, that are necessary for a full TCP/IP stack. It was meant for tiny 8 and 16 Bit microcontrollers, and this stack includes TCP, UDP, and ICMP protocols along with an implementation of IPv6, called uIPv6. uIPv6 is claimed to be the world’s smallest certified IPv6 stack for low-cost networked device such as sensors and actuators.

Operating system features include:

- Multitasking kernel
- A Graphical User Interface
- Process and memory management (The 'protothreads' allow memory-efficient concurrent programming on constrained devices)
- Communication management (Contiki supports both IPv4 and IPv6 stack implementations, which include TCP, UDP and HTTP protocols with the smallest footprints)

LoWPAN (IEEE 802.15.4) and 6LoWPAN are supported by Contiki OS. Low-power wireless personal area networks have the characteristics of small packet sizes, low data rates, low-power devices and large number of devices.

Contiki is a completely open source software, distributed using the 3-clause BSD-style licence. The complete code is available on github for use or further development. The software was created by Adam Dunkels in 2002 for Cisco and has been further developed by a worldwide team of developers from Texas Instruments, ST Microelectronics, and many others. Contiki comes with much documentation apart from well documented code. There are also community forums where active discussions happen.

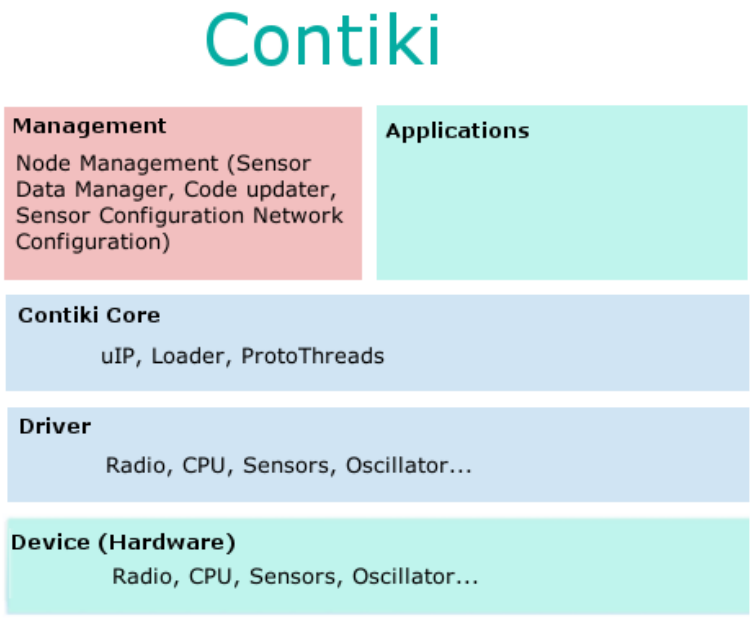


Figure 36 Contiki architecture

3.4.2.14 eCare

The e-Care Telecom Italia Lab (Tilab) Platform is an innovative cloud based platform and an evolution of the commercial service “Nuvola It Home Doctor”, the distance monitoring system, developed by TI for the prevention and cure environment. It is composed of two modules: quantitative measurement management (collection and analysis of physiological parameters), and qualitative measure management (health status analysis through questionnaires).

The e-Care Tilab is focused on non-mobile remote monitoring based on nonwearable measurement devices. It is based on Cloud infrastructures to enable data storing, off-line analysis, and data visualization through a remote services.

During the pilot the e-Care platform consists of a solution used to monitoring Lifestyles at subject’s home, recording periodically weight, blood pressure, eating habits and physical activity practice through use of electromedical devices interconnected to the same platform. In particular, during the experimental nutritional counselling (m-Health) will be carried out: - Recordings weight at home by electromedical devices (weekly) - Recordings blood pressure at home, only for subjects with borderline blood pressure values (daily). - Real-time recordings of eating habits and physical activity practice through computerized questionnaires on e-Care platform (Twice a month). The Electromedical devices used are equipped with wireless bluetooth interface, which allows the transmission of the detected physiological parameters automatically and wireless from medical Devices to Smartphone / Tablet. The electro-medical devices used have the CE mark according to Directive 93/42 / CEE certifying that the device respects the Operators and Patients Minimum Essential Safety Requirements. It is required the Smartphone / Tablet, appropriately equipped with a special application software, so they can to connect to Electromedical Devices During the measurement of the subject's parameters, using Bluetooth wireless technology. The gateway receives the measurements from devices and sends them to the platform via 2G/3G/4G/Wi-Fi/ADSL connectivity. The measurements detected by Electromedical Devices and received on Smartphone

/ Tablet are transferred in real time or deferred, via GPRS or UMTS, to the Collection Center (back end portal of Central Platform) for next web consultation on web by the health operator and by the subject.

The measurements make by the subject are also stored on their smartphone. Accessing to the tele-monitoring application the subject can consult all the values: a different graphic connotation of measurement also allows to distinguish between measurements already sent to the Platform and the measurements still to send. Doctors have at their disposal the instruments to evaluate the results (by web access to the medical platform) and, on the basis of patients' condition, are able to interact with them through the available means (SMS, telephone, videocalling) and modify their treatments.

The Smartphones compatible with the tele-monitoring application must to have Android operating system.

The e-Care Tilab Platform is made of basic components used as middleware: (i) THP (telemedicine horizontal platform) that works like a hub to exchange data between other systems; it combines multiple frameworks (such as Liferay, Hibernate, etc.) to perform its tasks, and (ii) SH (service module) to receive measurements from gateways/devices and send configuration parameters to gateways.

The e-Care Tilab Platform provides different web services and an API through which offers the possibility for other applications to use basic services like Calendar, Forum, Rating, etc. Data is stored in Oracle databases: one standard that records directory and personal user information, and one custom for recording measurements and vertical health information. Data is stored in a cloud architecture and could stay on virtual machines different from the application server.

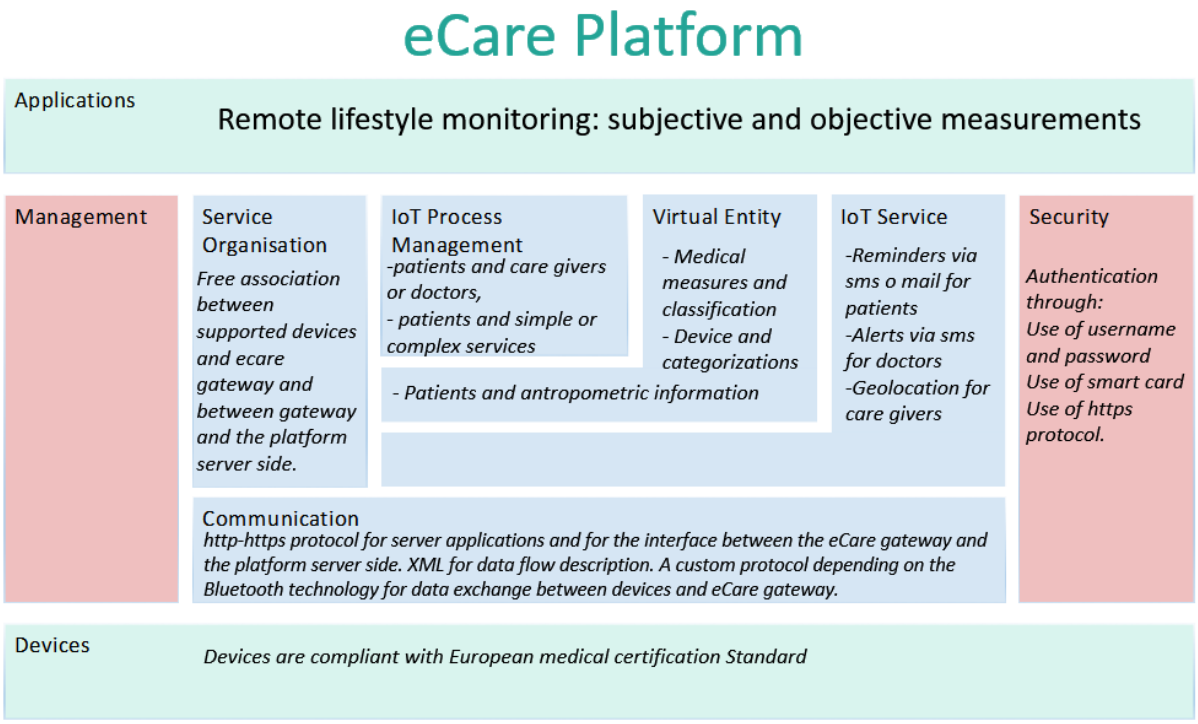


Figure 37 Relationship of IBM Watson with IOT-A reference model

3.4.2.15 IBM Watson

The internet of things on Bluemix utilizes the IBM Watson IoT Platform. Generally, Bluemix functions as a cloud platform as a service (PaaS) powered by open source projects and developed by IBM. It supports multiple programming languages such as Java, Node.js, Go, PHP, Python, Ruby Sinatra, Ruby on Rails and can be extended to support other languages such as Scala using buildpacks. It also supports multiple services as well as integrated DevOps to build, run, deploy and manage applications on the cloud. Bluemix is based on Cloud Foundry open technology and runs on SoftLayer infrastructure. There are initial free plans that include up to 20 devices, 10 applications bindings and 100MB of data exchange. Additional usage is billed at a per MB rate.

Besides common IoT services, Bluemix provides extensions for Business Rules, Hadoop processing, Cloudant and MongoDB NoSQL database layer, different DevOps tools, Messaging, GeoSpatial analysis, and access to the Watson services, particularly for Natural Language Processing.

Connecting to the platform is possible for devices and gateways. The data is secured in the cloud by connecting using MQTT messaging protocol or HTTP. Watson is the hub allowing set up and management of connected devices and applications allowing access to live and historical data. Rest and real-time APIs are available to facilitate connections between devices and applications.

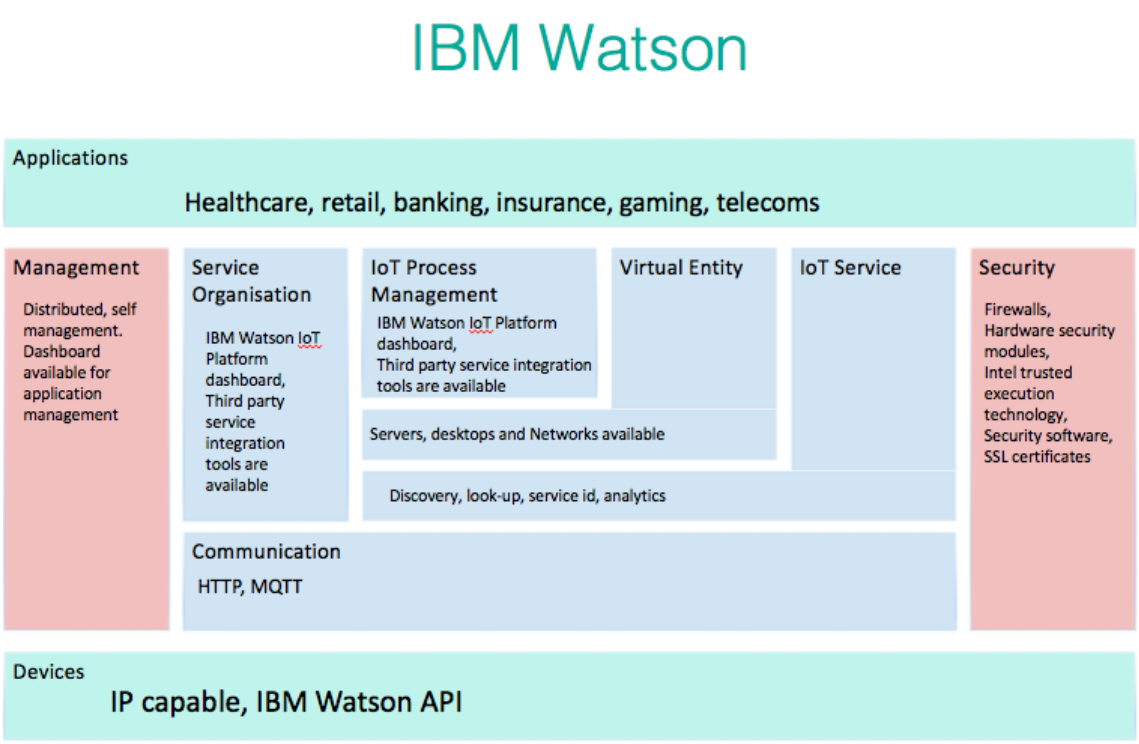


Figure 38: Relationship of IBM Watson with IOT-A reference model

3.4.2.16 WSO2

WSO2 is an open source service-oriented architecture (SOA) middleware. It is designed with independent components, so it can be adapted for a lean targeted solution to enterprise applications. The entire WSO2 middleware stack works seamlessly across private, public, WSO2 managed and hybrid clouds, as well as on-premise.

To completely protect from lock-in, all WSO2 products are 100% Open Source and based on Open Standards. Furthermore, WSO2 products released under the Apache License Version 2.0. WSO2 it's open to anyone who is interested in their products to get involved in the WSO2 community. Developers can extend the platform, customize code and use any programming model they like, report bugs or security vulnerabilities, prepare training materials, participate in forums and events, subscribe to public mailing lists, etc.

WSO2 products make heavy use of Java technology and are built on top of WSO2 Carbon, the company's SOA middleware platform. Carbon makes use of Apache Axis2 and encapsulates SOA functionality such as data services, business process management, ESB routing/transformation, rules, security, throttling, caching, logging and monitoring.

Not all components are used as stand-alone implementations. Many of them are used to supplement the capabilities or add functionality to an implementation of the Enterprise Service Bus. The main components that can be used in the WSO2 middleware are:

API Management

API Manager: API management platform for creating, deploying and managing APIs to expose data and functionality of backend systems.

API Cloud: Hosted API management service.

Integration

Enterprise Service Bus: Allows developers to connect and manage systems and software in accordance with SOA Governance principles.

Data Services Server: Provides a Web service interface for data stores.

Message Broker: Translates, validates and routes messages between systems.

Business Process Server: A graphical console to manage business processes and human tasks.

Analytics

Data Analytics Server: Real-time, batch, interactive and predictive analytics using enterprise data.

Complex Event Processor: Real-time event processing and detection. Identify patterns from multiple data sources, analyse their impacts. Uses WSO2 Siddhi and Apache Storm.

Machine Learner: Explorative data analysis using models to generate predictions. Uses Apache Spark.

Identity Management and Security

Identity Server: Connects and manages multiple identities across applications, APIs, the cloud, mobile, and Internet of Things devices.

Services and App Dev

Application server: Allows share business logic, data, and process across the entire IT ecosystem. It provides hosting shared, multi-tenant, elastically scaling SaaS applications.

App Cloud: Provides a comprehensive cloud ecosystem that evolves to enable easy and efficient integration, identity and API management for your digital enterprise.

Microservices Framework for Java: Allow to create microservices in Java with container-based deployment.

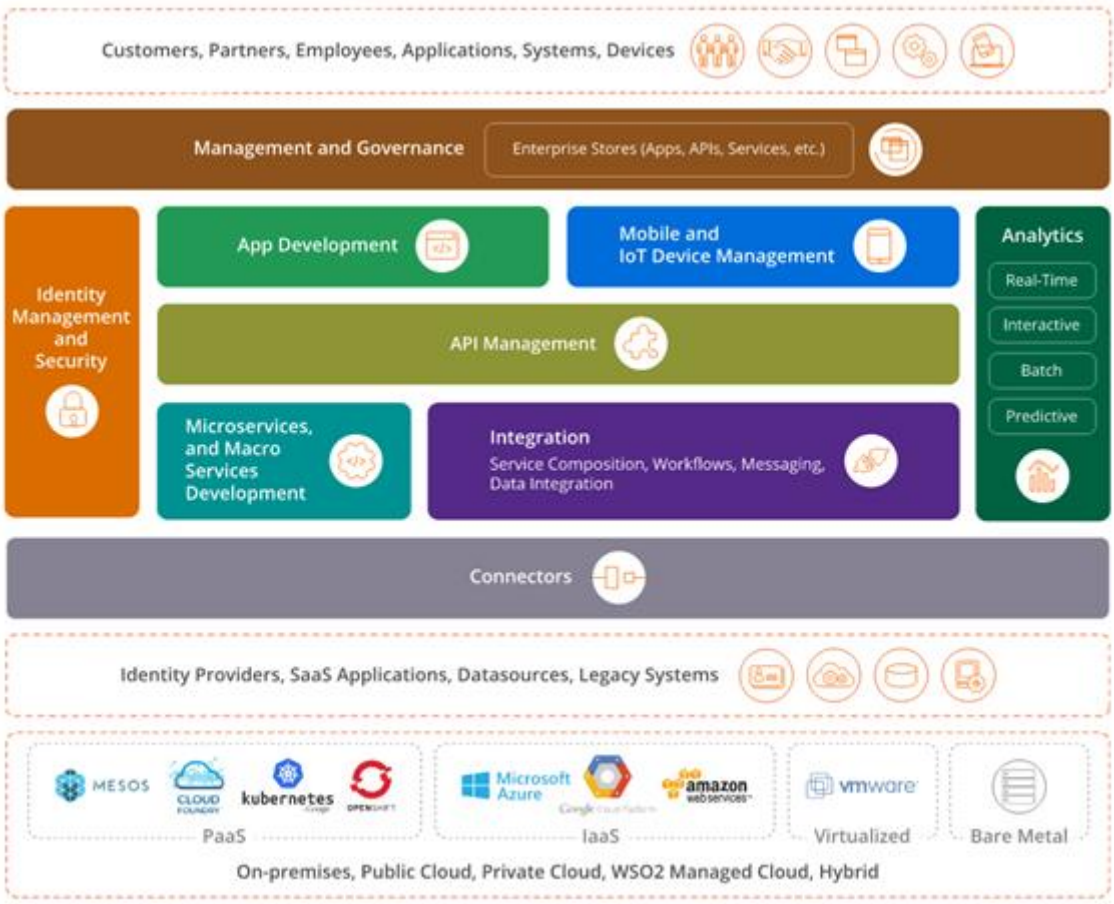


Figure 39: WSO2 components and generic architecture

Management and Governance

App Manager: Facilitates the process of creating, deploying and managing applications.

Governance Registry: Storage, cataloguing, indexing, managing and governing metadata related to enterprise assets.

Mobile and IoT

IoT Server: Internet of things platform for device management.

Enterprise Mobility Manager: Device management and business policy enforcement for mobile devices.

The following picture depicts how these components are related to the IOT-A model:

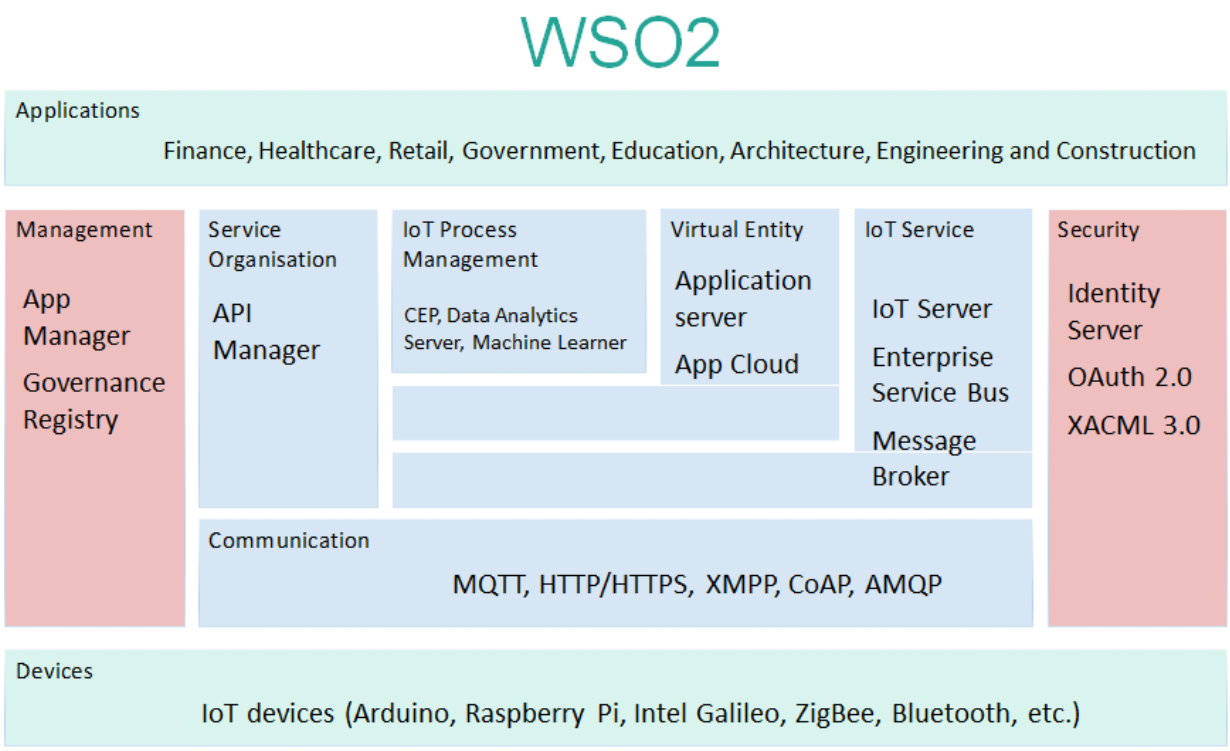


Figure 40: Relationship of WSO2 with IOT-A reference model

3.4.3 INTER-IoT Functional Model

Once we learned the functional model capabilities of the selected set of IoT Platforms (see, sections 3.4.2), we have been able to design a brand new Functional Model with INTER-IoT's vision for making IoT Platforms interoperable. This Functional Model is based on some concepts defined by the IoT-A, but is designed with the aim of dealing with the problem of interoperability among platforms.

The Functional Model to be used in the INTER-IoT has been generated taking in mind that the interoperability among IoT Platforms can be done at different layers, as we stated in the Description of Work of the INTER-IoT proposal. Therefore, the Functional Model of INTER-IoT is not an IoT system model, but a model to enable interoperability among platforms, each of which may follow the IoT-A Functional model. The Functional Model is comprised of a set of Functional Groups (FG) of INTER-IoT, which have been derived as follows:

- From some of the main abstractions identified in the Domain Model (IoT Platform, Platform Interoperability Services, Platform Ontologies, VE Interoperability Services), the "IoT Platform", "Service Interoperability", "Semantics" and "Device Interoperability" FGs are derived (see 3.2).
- From some other of the main abstractions identified in the Domain Model related to offering access and interactions with devices, which already existed in the IoT-A Domain Model (Virtual Entities, IoT Services, Resources), the "Device Access" FGs is derived.
- From the abstraction identified in the Domain Model related to the physical devices that already existed in the IoT-A Domain Model (Device, Sensor, Actuator, Tag), the "Device" FGs is derived.

- As defined in the requirements, there is a need to access different IoT Platforms to make them interoperable at several layers. To address this, the “Platform Interoperability” FG has been identified.
- As defined in the requirements, the users of INTER-IoT will be also applications or systems willing to access the different platforms, so an Application FG has been identified for this purpose.
- To address consistently the concern expressed about IoT Trust, Security and Privacy in the interoperability realm, the need for a transversal “Security” FG is identified.
- Finally, the transversal “Management” FG is required for the management of and/or interaction between the functionality groups.

We have generated a novel Functional Model for the INTER-IoT Reference Model that is depicted in the figure below. This new Functional Model is fully oriented to the interoperability among IoT Platforms. It contains eight longitudinal Functionality Groups (light blue) complemented by two transversal Functionality Groups (Management and Security, dark blue). These transversal groups provide functionalities that are required by any of the longitudinal groups.

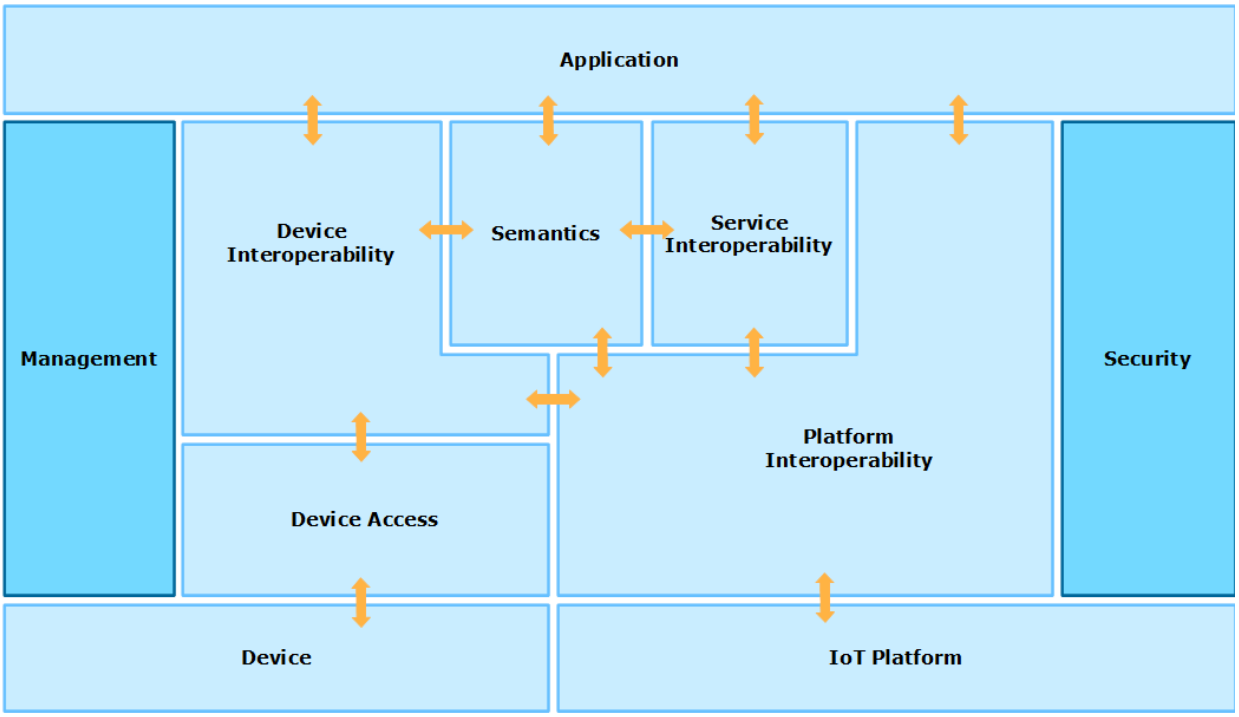


Figure 41: Functional Model of INTER-IoT Reference Model

The interactions among the different FGs has also been included. Depicted with orange arrows in Figure X, is the interaction between two FGs. As it can be seen, the layout of the interaction among FGs is quite vertical between the Application FG and the Device/IoT Platform Interoperability FGs. The Management FG and the Security FG interact with almost all the FGs in the model, so we have decided not to include them in the diagram, being considered these FGs as transversal ones.

Talking about interoperability, three FGs are out of the scope of this nature of solutions. The Application FG is responsible for using the underlying FGs to make use of the interoperability features through INTER-IoT. The Device FG can be considered a legacy FG to be integrated, thus being too generic and diverse. The IoT Platform Interoperability FG represents external existing IoT Platforms that are going to be interconnected or accessed through the INTER-IoT. The Application FG, the Device FG, and the IoT Platform Interoperability FG have been excluded from the description of FGs as their description won’t add any value from the interoperability point of view.

Hereafter, the different five interoperability-oriented FGs (Service Interoperability, Semantics, Platform Interoperability and Device Access) are described in detail.

3.4.3.1 Service Interoperability

The Service Interoperability FG relates to the need to interoperate different IoT Platforms at the Service layers. Interoperability between IoT Platforms can be handled at different layers (e.g. device, middleware, service, etc.). The Service Interoperability FG works at the service layer of each platform, regardless of their underlying infrastructure.

The overall goal of the Service Interoperability FG is to provide “compound” services to the Application FG, which are comprised of existing services that different IoT Platforms expose. An example of this would be a service aimed at offering access to historical data about traffic intensity in a region, when that service needs to access a historical data service from different government organizations (e.g. National Roads, Regional Road and Local Road Agencies).

Therefore, the Service Interoperability FG is responsible for accessing and using services that already exist in heterogeneous IoT Platforms. It also needs to provide a means to design the new “compound” services where some components are the services that exist in the concerning IoT Platforms to be interoperated.

Those new services were done using existing IoT Platform’s services need to be stored. A client of this FG will usually be an actor at the Application Layer, who will define new services and will manage them. Later, these new services will be executed when requested by the Application FG. The Service Interoperability FG will be responsible for executing them, accessing IoT Platform Services and providing the results back to the Application FG.

The Service Interoperability FG can also interact with the Semantics FG for requesting semantics features needed for the execution of the services, like, for instance, aligning different ontologies used by different IoT Platforms.

3.4.3.2 Semantics

The role of the Semantics FG is to deal with the management of ontologies that are needed for making IoT Platforms interoperable. Traditional interoperability designs leave semantics tasks to the Application side, but this approach lacks the necessary interoperability features. For instance, no common data processing can be made at any component, as data ontologies are unknown. Compound services are then very limited without semantic support, as the data from different platforms is not compatible due to the lack of ontology.

We consider semantics essential for interoperating IoT Platforms without transferring responsibilities to the end user. The Semantics FG is the responsible for providing support to all the management of ontologies needed at INTER-IoT. It defines the core ontology used for interoperating a specific set of IoT Platforms, each with its own ontology. It is also able to identify the ontologies used at the different platforms interoperated for the different devices or services providing information.

One of the main functions of the Semantics FG is to perform, so called, ontology alignment, which means to perform the translation from an origin ontology (maybe from an IoT Platform) to a target ontology (maybe needed by a destination IoT Platform). This ontology alignment process is just a step needed to perform the semantic translation of content among IoT Platforms, which is the final goal of the Semantics FG. The semantic translation among platforms, provided by the Semantics FG offers the following functions:

- Identify or define the origin or destination ontologies of the data involved in a data communication between IoT Platforms.
- Perform the ontology alignment from these origin ontologies to a common ontology.
- Perform the ontology alignment from the common ontology to the destination ontology.

The Semantics FG can provide its capabilities to several FGs with different purposes:

- Service Interoperability FG. It allows the Service Interoperability FG to perform alignment of data ontologies from different IoT Platform services so that common service processing can be done.
- Platform Interoperability FG. The Platform Interoperability FG can use this FG when particular services need to translate ontologies from data flowing from heterogeneous IoT Platforms with its own ontology into a common one to be provided to a user at the Application FG or, for instance, to interconnect sensor data from one to another platform each one of them having different ontologies.
- Device Interoperability FG. It allows this layer to perform ontology translation of data between devices, when making Device to Device interconnections, if data format or data ontology is different.
- Application FG. Although users, at the Application FG, will usually need to use the Service Interoperability FG, Platform Interoperability FG or Device Interoperability FG to make IoT Platforms interoperable in different ways, there is a possibility that the services provided by the Semantics FG can be of high value to an external user. This is a secondary functionality of interoperable IoT Platforms, but it's considered interesting when, for instance, a user wants to orchestrate its own services using raw data from different IoT Platforms and this data needs to be semantically homogenized.

3.4.3.3 Platform Interoperability

The Platform Interoperability FG is a central group that takes place in the most cases of interoperability between IoT Platforms. Its main goal is to interact with the different IoT Platforms to be interconnected. This FG abstracts the remaining groups from knowing about the details of the IoT Platforms, so much in terms of communications as in terms of capabilities, communication, security and so on.

It's important to highlight that the Platform Interoperability FG is the responsible for talking with the IoT Platforms, not for implementing any of the features that the IoT Platforms provide (what, in IoT-A's Functional Model, is described in the IoT Process, IoT Service or Virtual Entities FGs).

The Platform Interoperability FG has three main functions:

- To enable the access to different IoT Platforms. This includes the use of the appropriate protocols and APIs at middleware level that each platform exposes.
- To keep track of the interconnected IoT Platforms and their devices, so that they can easily be found, when needed. This allows the remaining groups to not to know about the location of the platforms, or how the devices are connected to them.
- To perform device and platform interactions, like querying data from different devices and platforms in a common way, mapping sensor data flows from a source to a destination, offering subscriptions to sensor data, etc.

This FG makes use of the Semantics FG, for instance, to translate ontologies from data flowing from heterogeneous IoT Platforms with its own ontology, into a common one to be provided to a user at

the Application FG. The Platform Interoperability FG talks also with the Service Interoperability FG to enable the access to services existing in an IoT Platform, or to provide a subscription to a flow of data coming from any IoT Platform.

The Platform Interoperability FG is the only FG that interacts with the IoT Platform FG. It's responsible for the aspects of dealing with these IoT Platforms (protocol communication, APIs, security features, data access, etc.).

On the device side, the Platform Interoperability FG does not interact directly with devices connected to the IoT Platforms. Note that, regarding the interoperability at the device layer, the Platform Interoperability FG can communicate with the Device Interoperability FG for two reasons: to enable the flow of data coming from devices not connected to an IoT Platform towards an existing IoT Platform, and for using this very data as another data source to be accessed by the Application or Service interoperability FGs for interconnection purposes.

3.4.3.4 Device Access

As it has been described in the Document of Work of the project (DoW) and has widely addressed in D3.1, there is a great need for making legacy sensor systems or disparate devices interoperable, as those connected to the real IoT Platforms. In order to allow the upper functional groups to be able to access these devices, this functional group is needed.

The main role of Device Access FG is to provide transparent access to very different devices when they are not connected to real IoT Platforms.

The main functions of this FG are:

- To enable the communication with the devices, independently of the access protocols, acting as the edge of the interoperability at low level.
- To abstract the physical entities and their related devices, which work in physical plane, from the concepts of IoT Service and Virtual Entity. IoT Service and Virtual Entities are closely related, and provide access to the devices and their resources in the virtual plane. The relationship between a Virtual Entity FG and an IoT Service reflect the features of the Domain Model related to the VE, Physical Entity, IoT Service and Resource.

The Virtual Entity and the IoT Service are like the groups described in IoT-A functional model with the same name. In the figure below, the basic concepts of Virtual Entities, IoT Services, Resources and Devices are described:

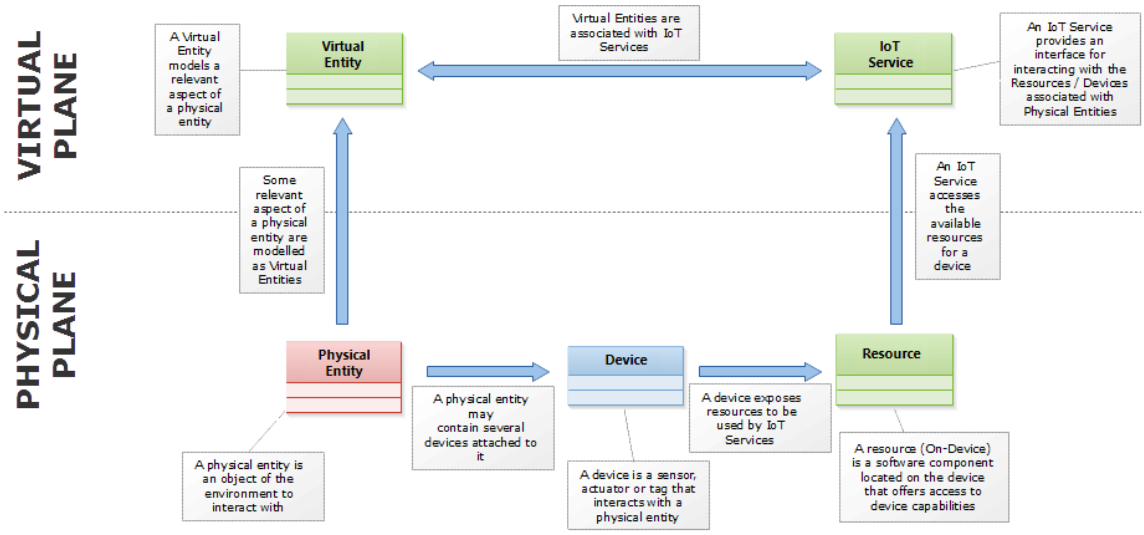


Figure 42: Relationship among main entities about devices in the physical and virtual plane

To understand the different concepts, an example has been depicted as follows:

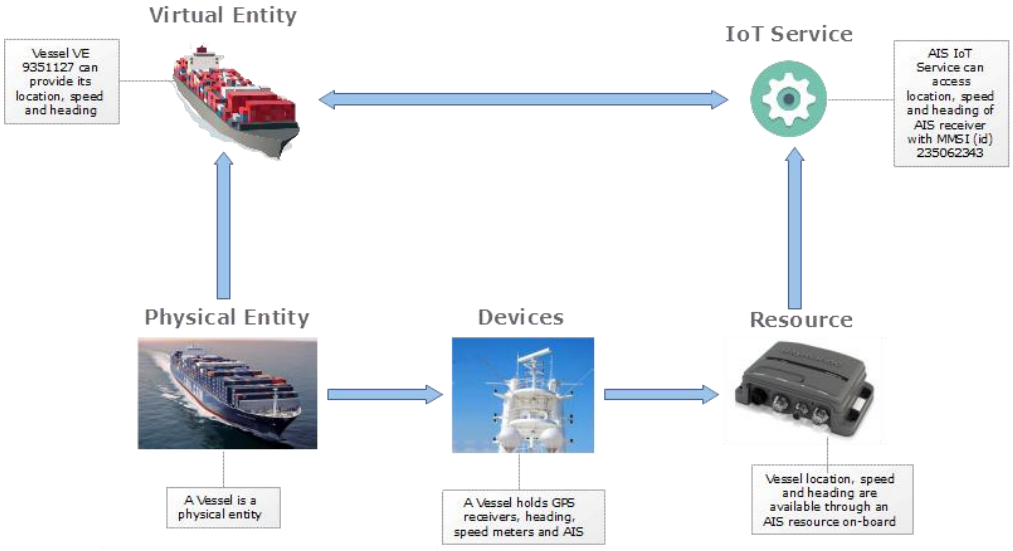


Figure 43: Example of relationship among main entities about devices

As of IoT-A: "*The Virtual Entity contains functions for interacting with the IoT System on the basis of VEs, as well as functionalities for discovering and looking up Services that can provide information about VEs, or which allow the interaction with VEs*". "*The IoT Service contains IoT Services as well as functionalities for discovery, look-up, and name resolution of IoT Services*".

The Device Access FG interacts with the Device FG to interact physically with the devices. It also interacts with the Device Interoperability FG, to enable the interoperability at device and network layer and also to integrate with the Platform Interoperability FG as described in section 3.4.3.5.

3.4.3.5 IoT-A Background

The role of the Device Interoperability FG appears once the Virtual Entities and IoT Services are available. The Device Interoperability FG addresses the functionalities around making the devices interoperable.

The Device Interoperability FG is responsible for defining the rules to interconnect devices among them, achieving the D2D (Device to Device) interoperability. It may enable that, for instance, when a person leaves a house, its heating system goes in low mode.

This FG is also in charge to perform the Network to Network Interoperability for the networks the devices are connected to. This implies routing, roaming and off-loading capabilities to enable the devices to move among different networks.

The Device Interoperability FG can interact with the Platform Interoperability FG in two ways: it can act as a client of IoT Platforms, for interconnecting legacy or disparate devices to existing IoT Platforms through the appropriate device register and data retrieval/actuating functions typical these platforms. It can also act as a kind of IoT Platform for the Platform Interoperability FG, when there is no IoT Platform where to attach the devices, but an application can access these devices for interoperating with other IoT Platform information. In this last case, the Platform Interoperability FG would interact with devices through the Device Interoperability FG.

Semantics can also be a resource to be used for translating sensor data. the Device Interoperability FG can communicate with the Semantics FG to achieve this.

3.4.3.6 Management

The Management FC considers all the functionalities needed to rule the interoperability among different IoT Platforms. The Management FC is thus, responsible for initializing, monitoring and modifying the operation of the interoperability among IoT Platforms.

According to Pras A. [34], the main reasons for needing management fall within the following groups:

Cost Reduction

Users, obviously want to operate a system at the lowest possible cost. This implies that the design of the solution should satisfy a great number of potential users and situations so that the cost can be recovered among many users. To achieve this, the design should be as multipurpose as possible. It means that the system should parametrized to a wide range of scenarios and user needs. The Management FC will be responsible for setting up these parameters for any final deployment of INTER-IoT.

Lack of design experience

We cannot assume that users of INTER-IoT will always be high-skilled IT engineers that can easily understand all the concepts and apply them right, finding good solutions for each and every problem. Some of the problems that will face our end users will arise during the operation phase of the system, not during the design phase. For instance, an IoT Platform can decelerate its performance or even shutdown completely, some devices can have malfunctions overloading with irrelevant data, some external component can inject too much traffic in form of requests, like a DDoS attack or a service become unavailable at a certain moment.

To address this reality, the Management FC will need to include capabilities to mitigate the impact of these issues without a necessary good design of the interoperability made by an INTER-IoT user. Some examples of this would be to monitor IoT Platforms state or to handle incoming requests.

Fault Handling

Failures are inherent to any operating system. They can have many causes, not being possible to prevent all the failures. As the consequences of these failures can be very severe, it's necessary that the Management FC includes strategies and actions to control the operation of the interoperability solution.

Such control implies the monitoring of the whole system, the prediction of potential failures, the detection of existing failures, the mitigation of their effects and, if possible, to repair them. The Management FC is responsible for addressing these features, through monitoring capabilities and the possibility to change operational parameters during run time, such as platform and device registries, communication channel re-mapping, service catalogue status, etc.

Flexibility

Traditional interoperability design is based on specific user requirements, which drive the design of a specific solution by, for instance, defining specific communications or translations between two IoT Platforms. The danger of this approach is that, on one hand, requirements can change in time, affecting the already deployed solution, and on the other hand, each interoperability scenario may have its own specific requirements.

Instead of designing a new interoperability solution each time, it is better to include some flexibility in INTER-IoT, so that the Management FC can adapt to different situations and react to changes during the operational phase.

Some flexibility features have been included in the requirements. The Management FC is responsible for supporting these features during the operational phase. Some examples of this is the ability to use different ontologies for the same iot Platform and change them during runtime, to be able to define new services and have them available, or to define new rules for making devices interoperable among them.

3.4.3.7 Security

The Security FG is responsible for ensuring all the security aspects involved in the interoperability of IoT Platforms. The security in our realm has two faces:

- Management of the security aspects related to the connection with underlying IoT Platforms. This implies to accomplish with the different security features that the platforms require. INTER-IoT will need to tackle the user authentication for connecting to a platform, the authorisation management (e.g. use of authentication tokens) and the encryption of some communications. Moreover, the access to the different IoT Platforms maybe user-based or anonymized depending on the decision of platform owners, so it must be handled by INTER-IoT with flexibility for each scenario.
- Management of the internal security of INTER-IoT. The connection to INTER-IoT must be secured, with appropriate authentication capabilities, and authorisation management. The identity of each user must be preserved, so much for keeping the identification until the IoT Platform, as to keep track of the anonymization when talking with the IoT Platforms. This internal security also implies the permission assignment to specific IoT Platforms and its resources (devices, services, etc.) under certain conditions. for instance, a platform owner

may will to give access to a subset of devices to a set of user roles, but only within a time range, or when mobile devices are at a certain location.

The Security FG interacts with all the different groups and will allow that certain accesses are made, or that certain interconnections between two platforms are authorised or not.

3.4.4 Conclusions

In this section, we analysed the most relevant IoT platforms. Clearly, as the number of them is over 300, this is not a whole comprehensive study, not a taxonomy or a benchmarking, but rather an understanding of what are the main characteristics of the different efforts in this area, trying to see through a reverse-mapping on the IoT-A Functional model where it would be possible to find analogies and similarities to the different IoT Platforms.

Given the work above, it's then possible to derive the Functional View of the different schemas, as explained in the introductory section. The Functional View will allow us to assess the different functional features of the platforms, and to propose an architecture for our INTER-FW product.

3.5 Communication Model

3.5.1 Introduction

The approach of INTER-IoT to the CM, is the creation of a layered solution that is equivalent to each of the IoT aspects as is shown in the figure.

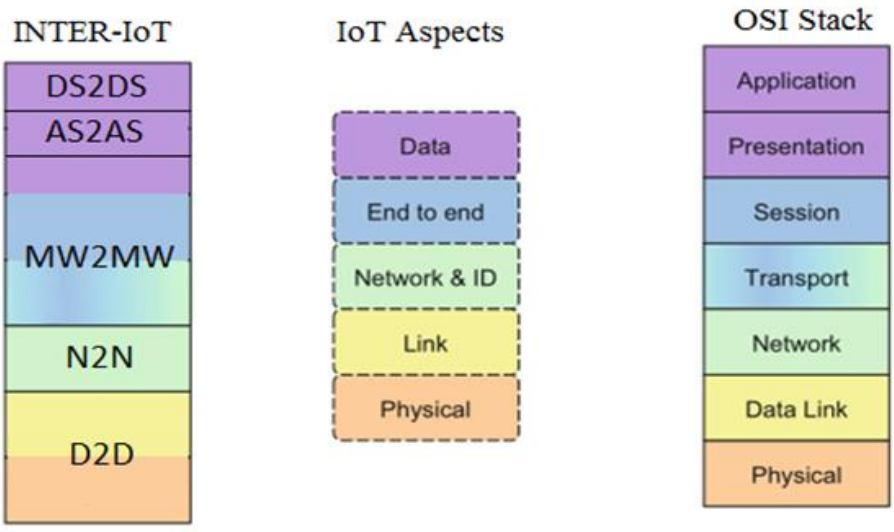


Figure 44: Comparison between traditional OSI model, IoT stack and INTER-IoT stack

- D2D: is equivalent to the Physical and Link aspects, due to is in charge of manage the communication technologies of the low levels of the stack, that includes technologies as WiFi, Ethernet, Bluetooth, etc. that belongs to these abstraction layers.
- N2N: is equivalent to the Network & ID aspect, due to is in charge of addressing and routing the information through the nodes of the IoT system as well as the packet filtering and traffic control.
- MW2MW: is equivalent to End-to-end communication, that involves application protocols to exchange the information (HTTP, MQTT, CoAP…) and part of the presentation and accessibility to the data, stored in the IoT platforms.

- AS2AS: Is in this case equivalent to the Data aspect, due to the interoperability carried out in this layer is mostly through the translation from one data format to another, (e.g. CoAP is translatable to HTTP by decompression or XML is translatable to EXI by compression, JSON is translatable to XML by mapping…).
- DS2DS: involves, for sure, the Data aspect because is in charge to translate semantics of messages exchanged by IoT Artefacts (platforms, gateways, application, etc) within the INTER-IoT system.

3.5.2 Communication Protocols on IoT Platforms

	Application Layer						Network/Link Layer			
	HTTP	MQTT	CoAP	LWM2M	AMQP	UPnP	BLE (802.15)	ZigBee (802.15)	WiFi (802.11)	Others
AllJoyn							X		X	Ethernet,Serial, PLC and DNS
AWS	X	X								WebSockets
Azure	X	X			X					Other protocol as CoaP has to be adapted
FIWARE	X	X	X	X						IoT-Agents
IBM Watson	X	X								
OM2M	X		X					X		KNX and HUE
OpenIoT		X	X							GSN Wappers
SOFIA2	X	X								Through KPs (Knowledge Processor a.k.a. Gateway)
UniversAAL	X							X		ZWave
BUTLER	X	X	X			X		X		

I-CORE			X			X	X	X	X	

Table 4: Summary of most used communication protocols in IoT Platforms.

What is notable in the above table is that, even with the heterogeneity in the IoT environment, some protocols are winning places between the one most used for this environment. Thus, it can be noticed that low-level layers (that involve PHY, MAC and network), are implemented mostly in all gateways or frameworks. Examples of protocols in these are: Bluetooth, WiFi, ZigBee or those covered by the IEEE 802.15 standard.

It is true that some other protocols are widely used, by sensor or devices, as LoRa, ANT or cellular network protocols, but still the most common protocols implemented in the systems analysed are the ones listed in Table 4

From the other side, at application level something similar occurs, some protocols are already widely spread meanwhile others have been appearing with the increasing of smart devices connected to the network. The most common web service protocol used over the transport layer was HTTP, over TCP, but with this growth other protocols as CoAP, over UDP, or even AMQP, over TCP, have been introduced in the frame of communication technologies. So, this, HTTP, MQTT, CoAP and AMQP are the protocols implemented in most IoT platform with services, normally, web services. Even if the most suitable protocol for IoT could be CoAP, or even others as LWM2M, for its size, speediness, low power consumption… yet HTTP, together with REST architecture of web services, is commonly used for all platforms.

This way, implementing these two protocols, the platform can communicate for one side with constrained devices to obtain the information using CoAP, and communicates with other less constrained devices or applications using HTTP or MQTT, creating a trade-off between performance and complexity.

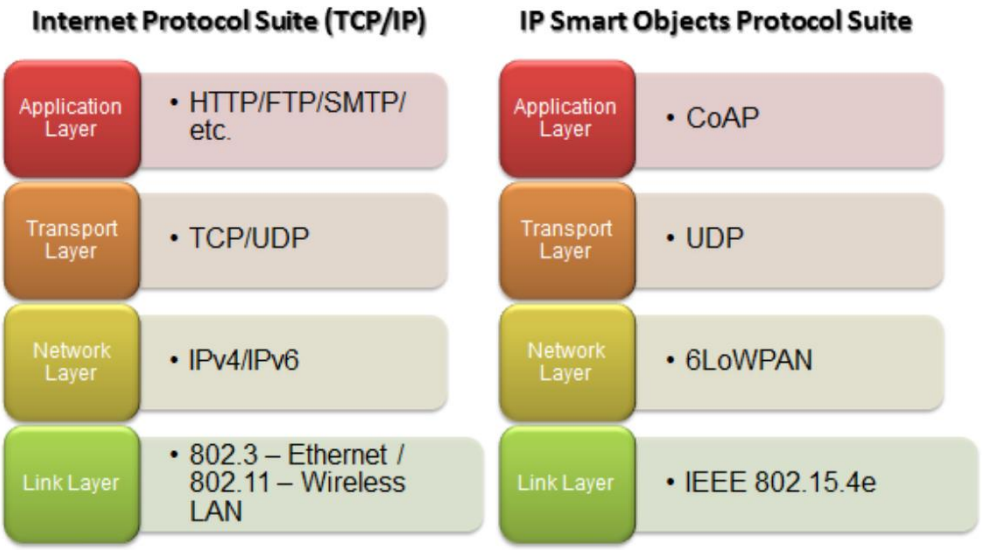


Figure 45: Comparison between traditional Internet stack and the IoT network stack. [32]

For that reason, INTER-IoT implements the most common protocols within each layer to allow the communication and interoperability between different services and resources that belong to different devices, platforms, or even in different sub-networks.

3.5.3 INTER-IoT Domain Model element communications

The Communication Model is used to identify the communication system elements and/or Users among those defined in the Domain Model, but, taking in account that the CM define Users in different categories as: Human Users, Services or Active digital artefacts.

So that, the INTER-IoT Communication Model explains the interaction between elements on the INTER-IoT Domain Model, previously identified in Section 3.2.3, and how to communicate two Entities through its relevant layers.

3.5.3.1 Device to Device Interactions

In Device to Device interactions the *Device* is composed by a *Sensor* and *Actuator*, or both, and with a unique direction for it to be addressable. Optionally, a *Device* could be just a simple *Tag* referencing this *Device*. This is the main component for the interaction and with this is representing the *Physical Device* in our Domain Model. Most of cases, but not always, the *Physical Device* is represented or described in the *Virtual* plane by the *Virtual Device* component and together with the *Physical Device* counterpoint they conform the *Augmented Entity*; with the characteristics and features of the *Physical Device* and the extra-information Metadata of the *Virtual Device*.

Additionally, other resources of the D2D gateway, the N2N network or the Platform will appear in the communication. The implementation of the Domain model in the Device architecture described in the deliverable D3.1 is as follows: *Device (Sensor, Actuator or Tag)* represents the objects we desire to interconnect, *Physical* and *Virtual Entities* are concepts implemented by the gateway as well as the *Augmented Entity* and the *Interoperability resources or Services* are provided by the components within the D2D gateway solution(see next figure).

Examples of interaction:

- **A device communicates through the Physical gateway**

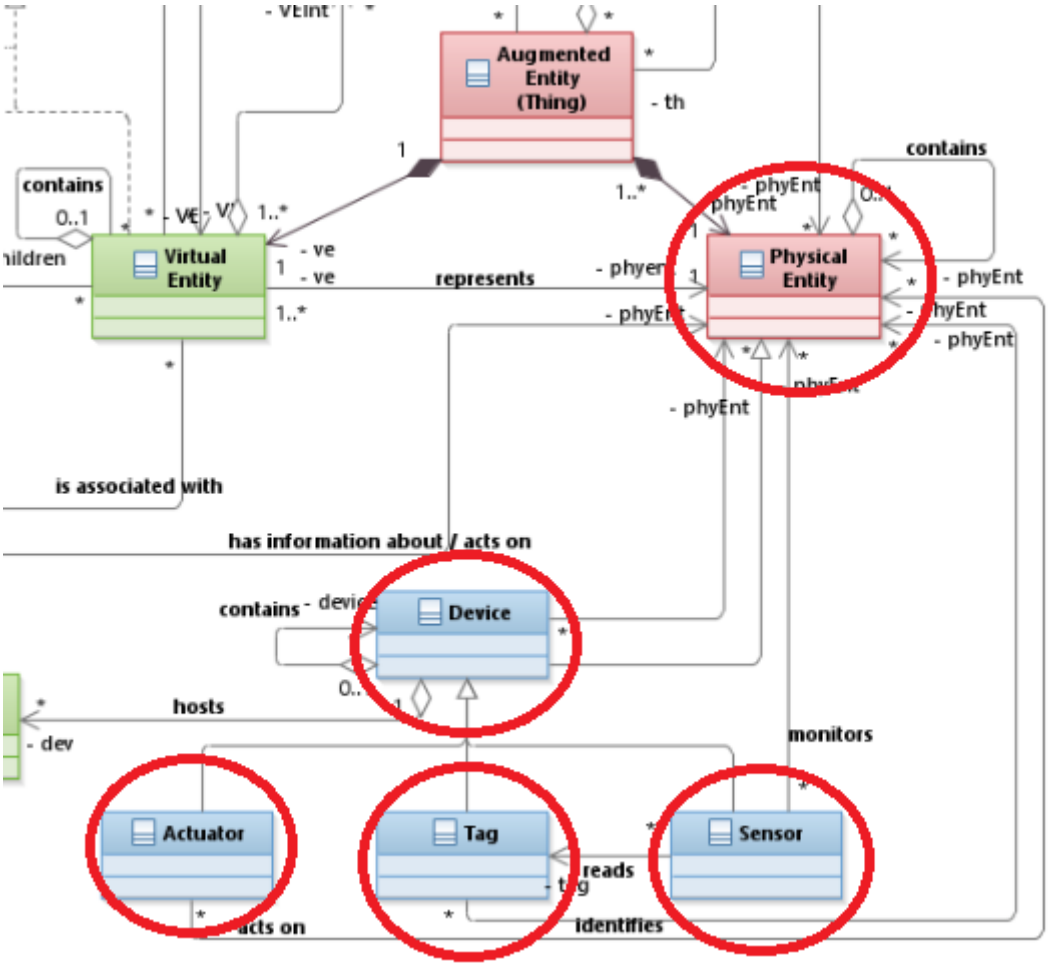


Figure 46: Domain Model entities involved in Device-to-Device communication when the device communicates through the physical gateway Device communicates through the virtual gateway

The same way, the values obtained by one Device (sensor) could be measured until the virtual part of the gateway to be forwarded to another device (actuator) connected to the virtual part.

In this interaction, the Virtual Entity located in the gateway takes part on the communication (see Figure 47).

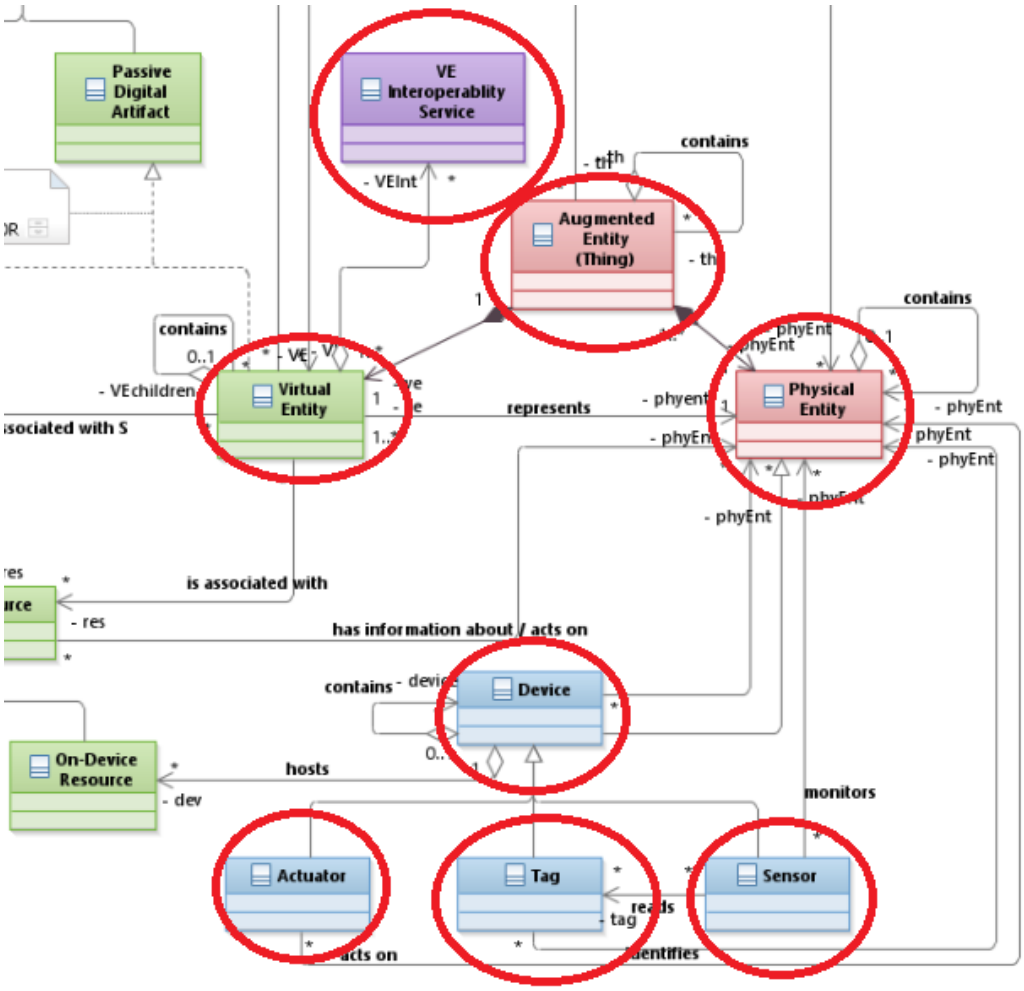


Figure 47: Domain Model entities involved in Device-to-Device communication when the device communicates through the virtual gateway

In both cases, one of the devices could be another Entity, as a Platform, or another gateway but this interaction will be contemplated in future sub-sections.

3.5.3.2 Network to Network Interactions

Examples of interaction:

- **Device communicates with resource in the network.**

In this case, the interaction takes place within the network, when a device or entity interacts with a resource or service hosted and running in the in the network, or even when one or both element that want to interact belong to this network (see next figure).

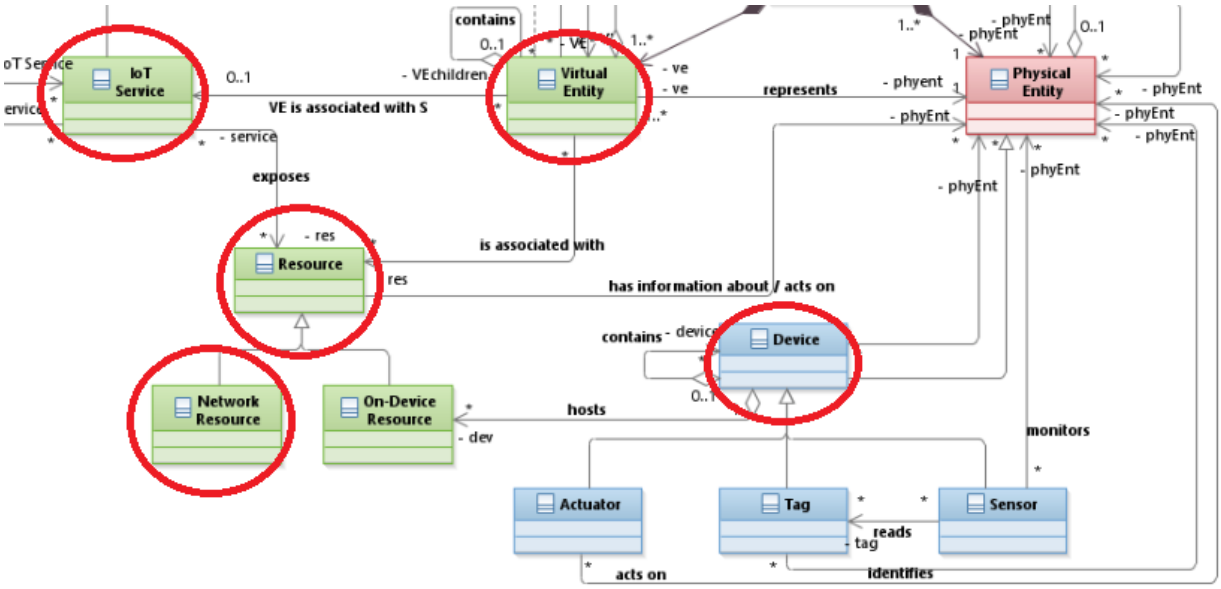


Figure 48: Domain Model entities involved in Network-to-Network communication when the device communicates with resource in the network

- **Platform service or a resource communicates with another resource in the network**

The domain model element communications affecting the network layer are the IoT Platforms and Platform services (red) requesting information about the IoT Services (green) available in the resources specifically network resources (yellow). See Figure 49.

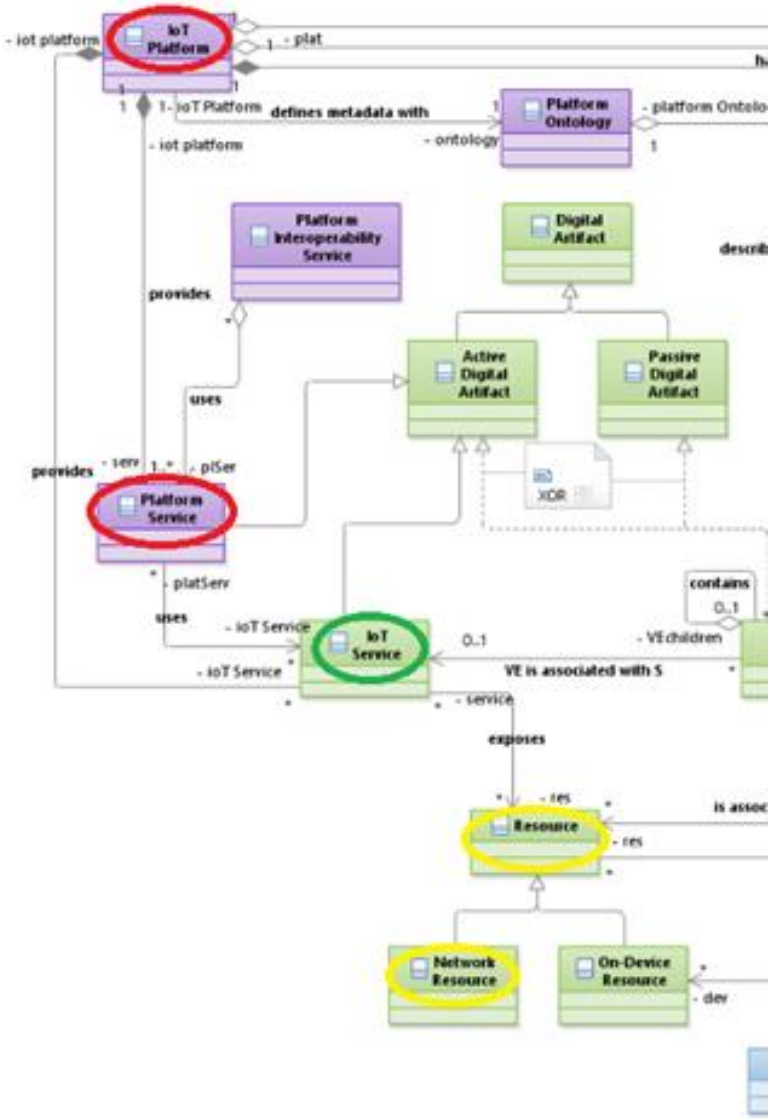


Figure 49: Domain Model entities involved in Device-to-Device communication when platform communicates with another resource in network

3.5.3.3 Middleware to Middleware Interactions

In middleware to middleware interactions, *Platform Ontology* and *IoT Platform* represent key components that interact with underlying middleware IoT platforms. *Platform Ontology* holds the "knowledge" about how to talk to and understand an IoT middleware implementation, while the *IoT Platform* entity "knows" about a specific IoT middleware deployment. Those two components thus appear in all Middleware to Middleware communication scenarios. In specific cases, they are aided by the other two additions to the INTER-IoT domain model to the IoT-A model: *Platform Interoperability Service* and *Platform Service*.

The implementation of the domain model in the middleware architecture described in the deliverable D3.1 is as follows: *Platform Ontology* and *IoT Platform* are implemented through the Communication and control segment (*Platform Ontology* and *IoT Platform)* and Bridges segment (*IoT Platform*), while *Interoperability Service* and *Platform Service* are partially implemented through the MW2MW services segment.

Examples of interaction between Middleware and Middleware are:

- **User – Middleware of IoT Platform**

The interaction between user and middleware entails the usage of the platform's ontology, knowledge of which is stored within the Platform Ontology component, as well as the usage of the IoT platform itself, which is represented with the IoT Platform component. With the aid of these two components the interaction with specific segments of underlying IoT platforms (marked in yellow) is made possible (IoT Service, Virtual Entity, Augmented Entity, Physical Entity).

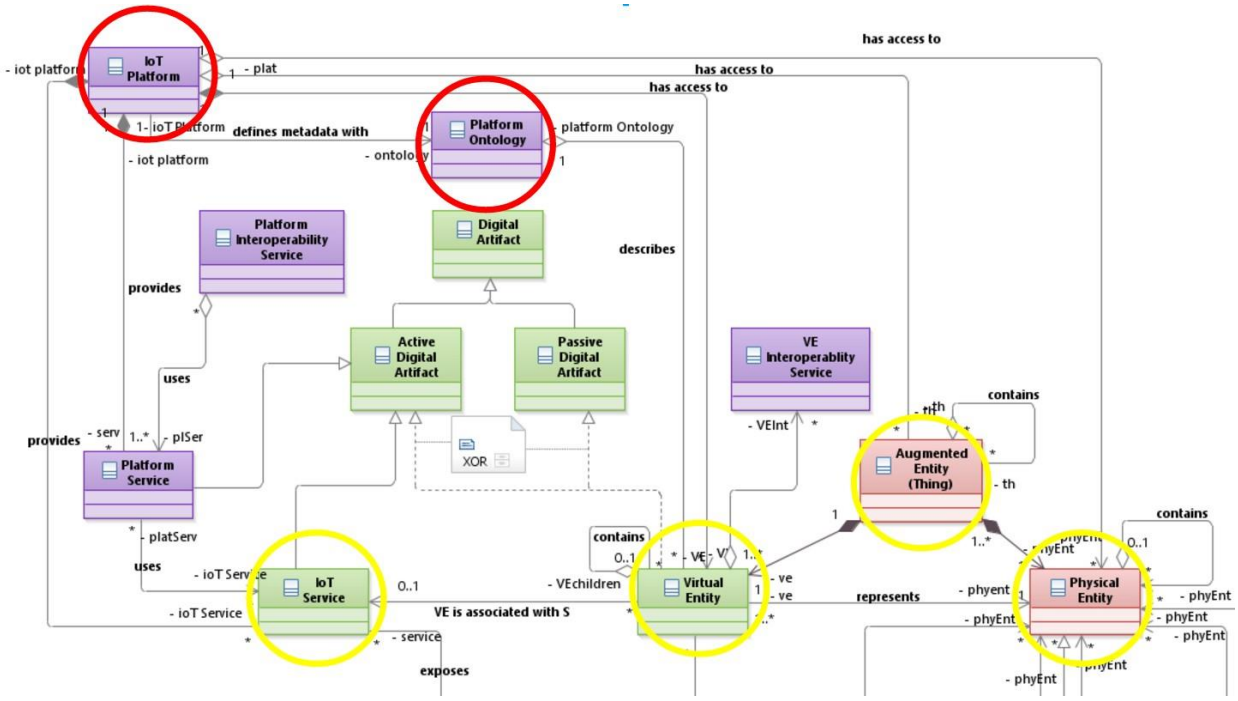


Figure 50: Domain Model entities involved in Middleware-to-Middleware communication when the user communicates with an IoT Platform

- **User configuring a Middleware – Middleware between IoT Platforms**

In the case of a user attempting to configure the INTER-IoT middleware, we highlight the component Platform Ontology, due to the need to understand platform's specific ontology, as well as the components IoT Platform (implementation of the platform itself), Platform Service (providing the support for configuration of a specific platform) and Platform Interoperability Service (providing support for configuration of interoperability between platforms).

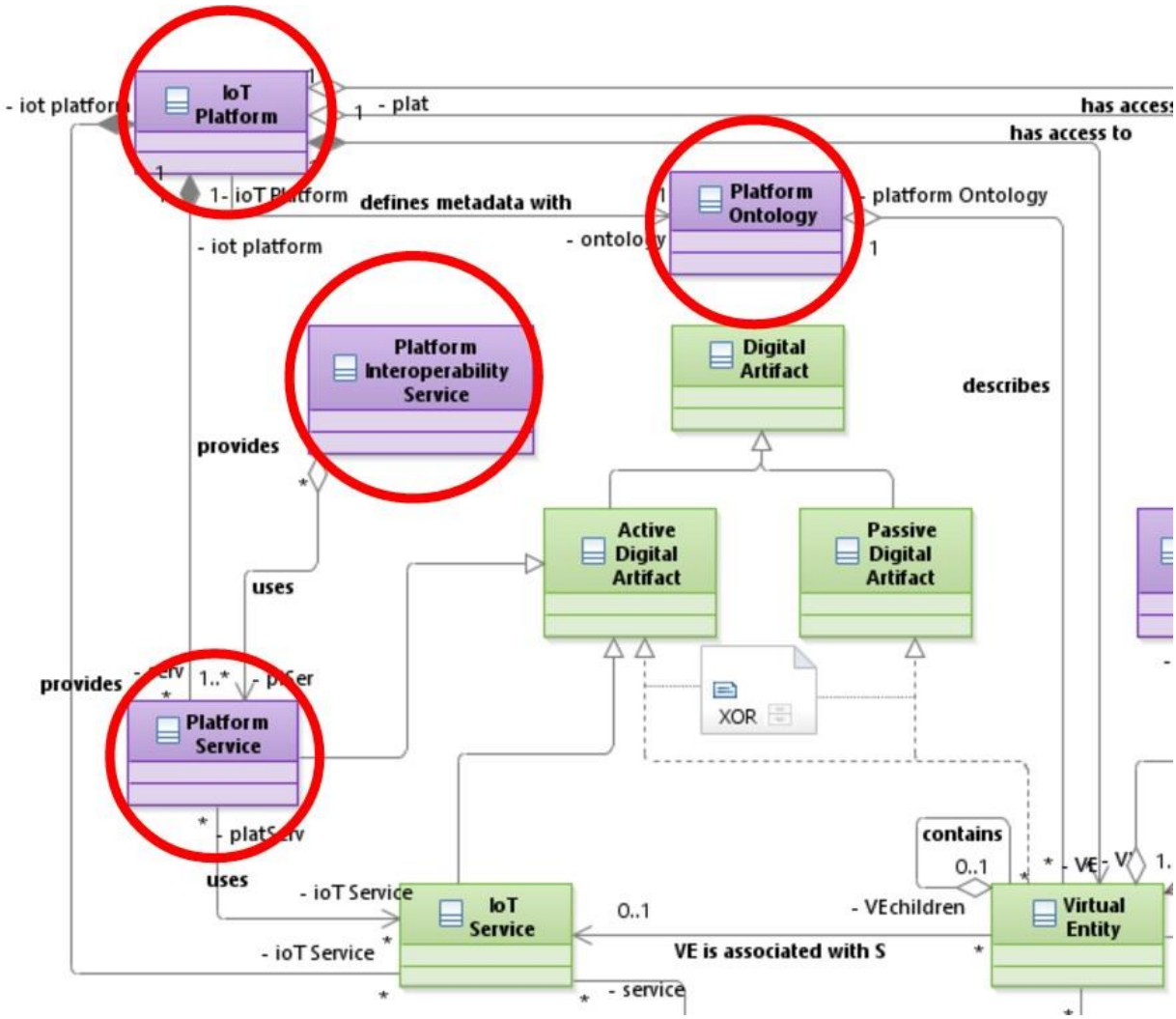


Figure 51: Domain Model entities involved in Middleware-to-Middleware communication when the user configures a Middleware to Middleware communication between two IoT Platforms

- **Direct communication Middleware – Middleware between IoT Platforms**

Direct communication between two middlewares includes the usage of the Platform Ontology component (providing the basis for ontology translations between different platforms, operating under the two middlewares). The IoT Service component of one middleware platform initiates communication through the IoT Platform component, using the knowledge about the ontology of the other middleware and accessing the Virtual Entity component (which could be a device, to which it tries to write data) in the other middleware.

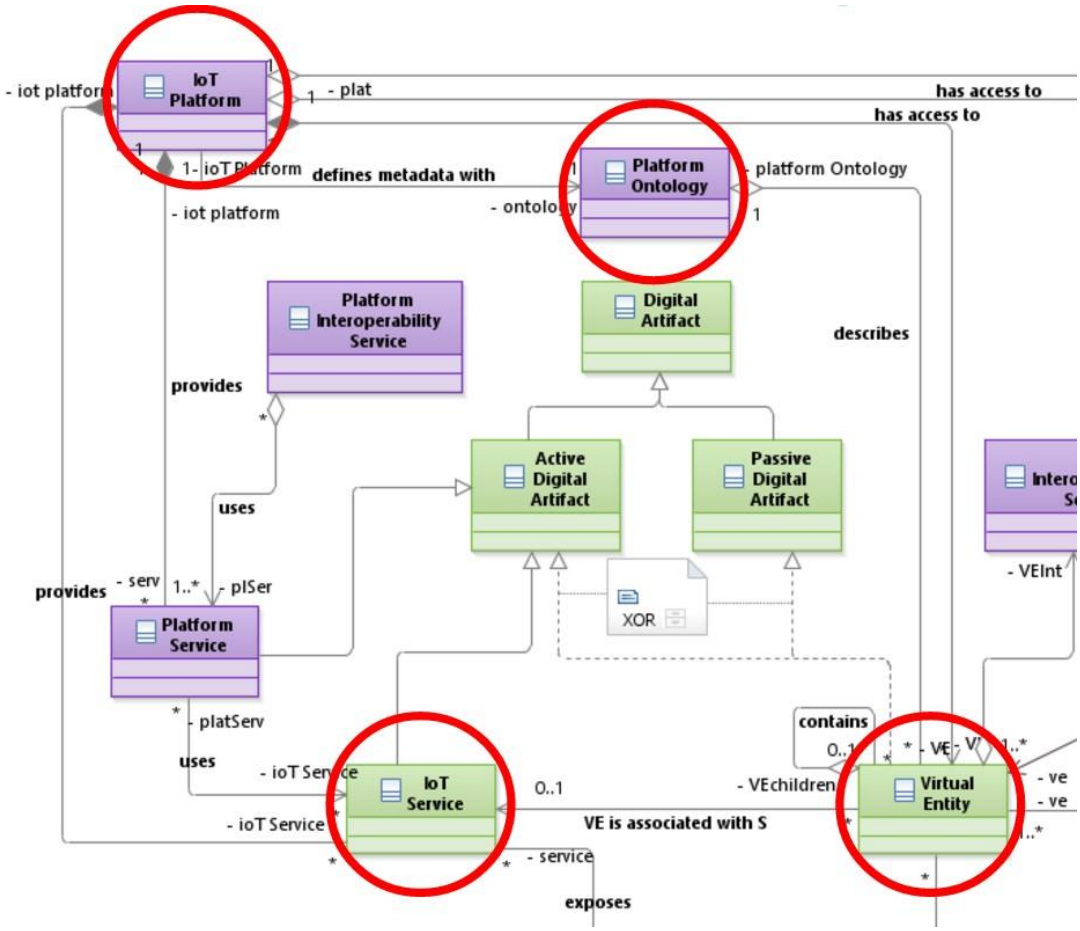


Figure 52: Domain Model entities involved in a direct Middleware-to-Middleware communication between IoT Platforms

3.5.3.4 Application & Services to Application & Services Interactions

In AS2AS interactions *Platform Service*, *IoT platforms* and *Platform Interoperability Service* represent the key components that interact with the IoT Platforms.

Platforms Service exposes functionality about Resources related to Virtual Entities, they offer more elaborated services that internally make use of IoT Services. For that reason, they are placed in the Application and Service Layer and can be used as building blocks for creating more complex interoperability services among different *IoT platforms*.

The *Platform Interoperability Service* handles the definition of new compound services that appear as a consequence of using, and mixing in any way the *Platform Services* from one or more IoT platforms. So, the Platform Interoperability Service is linked with the different Platform Services it uses. The used *Platform Services* are just part of *IoT Platforms*. In some cases, it would be necessary the interaction with *Platform Ontology*, because it holds the information about how to understand the services from the IoT Platforms.Examples of interaction:

- **User creating a Composed Service.**

A user wants to create a composite service, he needs to interact with the *Platform Service* of the *IoT Platforms.* He also can use a service that have been already composed communicating with the *Platform Interoperability Service* to be possible to compose it with other *Platform Services.*

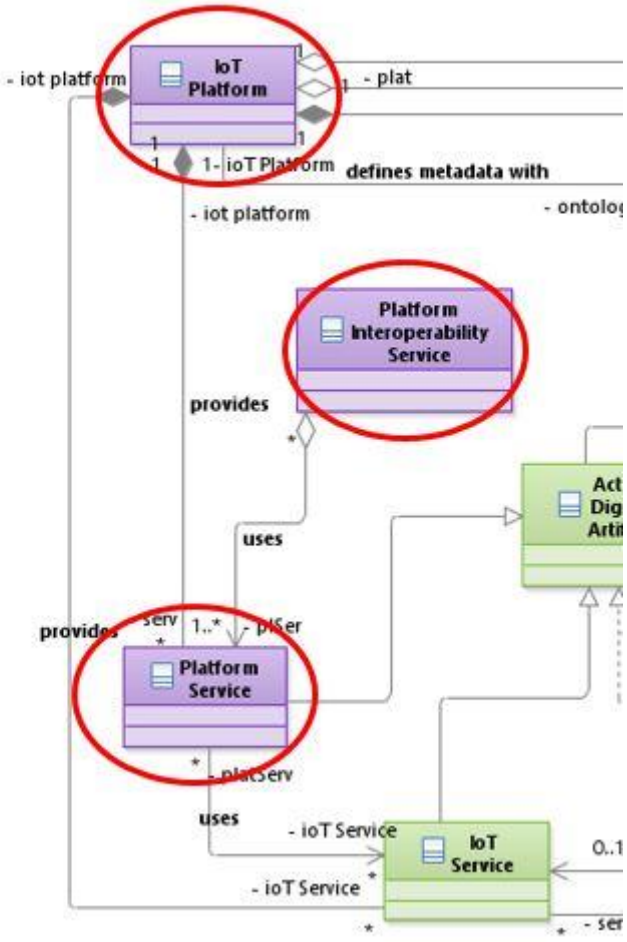


Figure 53: Domain Model entities involved in Application&Services-to-Application&Services communication when the user creates a compound service

- **Service of an Iot Platform communicates with a Service from another IoT Platform**

The following interaction takes place when a Service from a platform desires to communicate with another service from another platform. When two services communicate it is reflected in the *Platform Interoperability Service* that indicates which *Platforms Services* from the *IoT Platforms* participate in this composition.

Finally, the *Platform Ontology* will perform the need of ontology translation between these services, to understand each other.

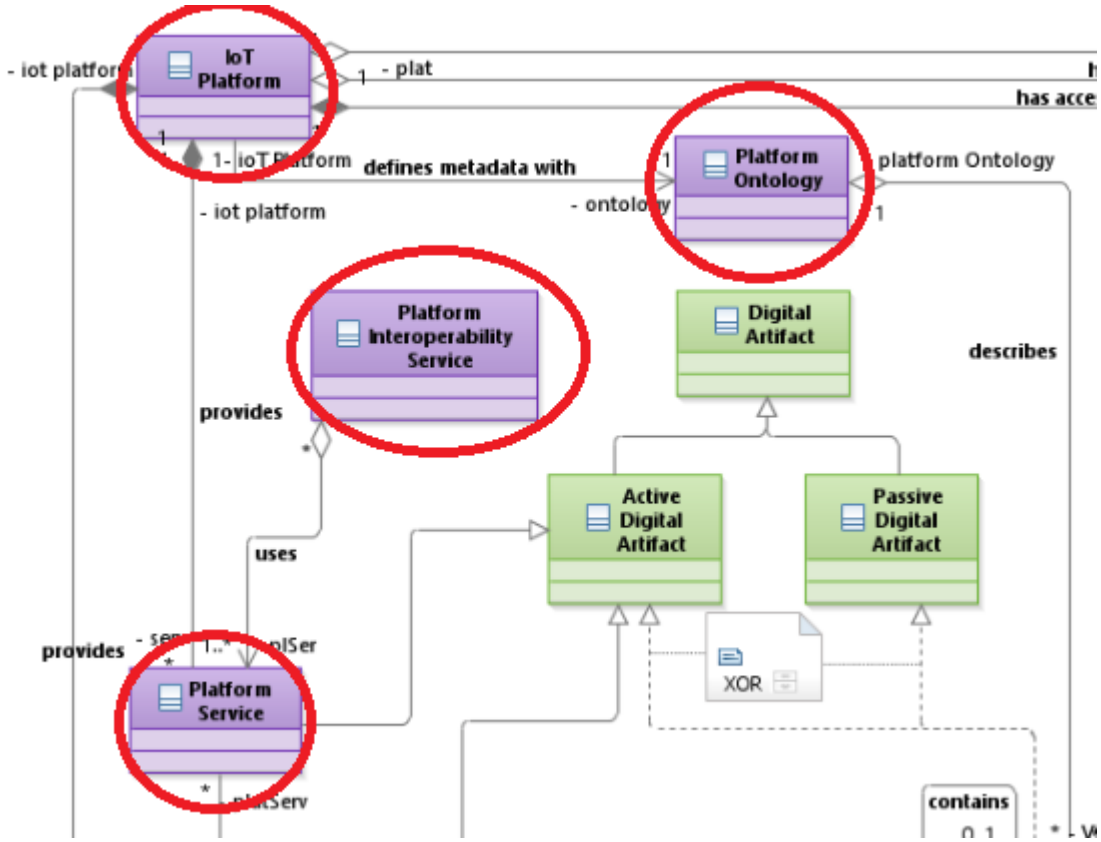


Figure 54: Domain Model entities involved in Application&Services-to-Application&Services communication when the compound service communicates with a Service from another IoT Platform

3.5.3.5 Data & Semantics Interactions

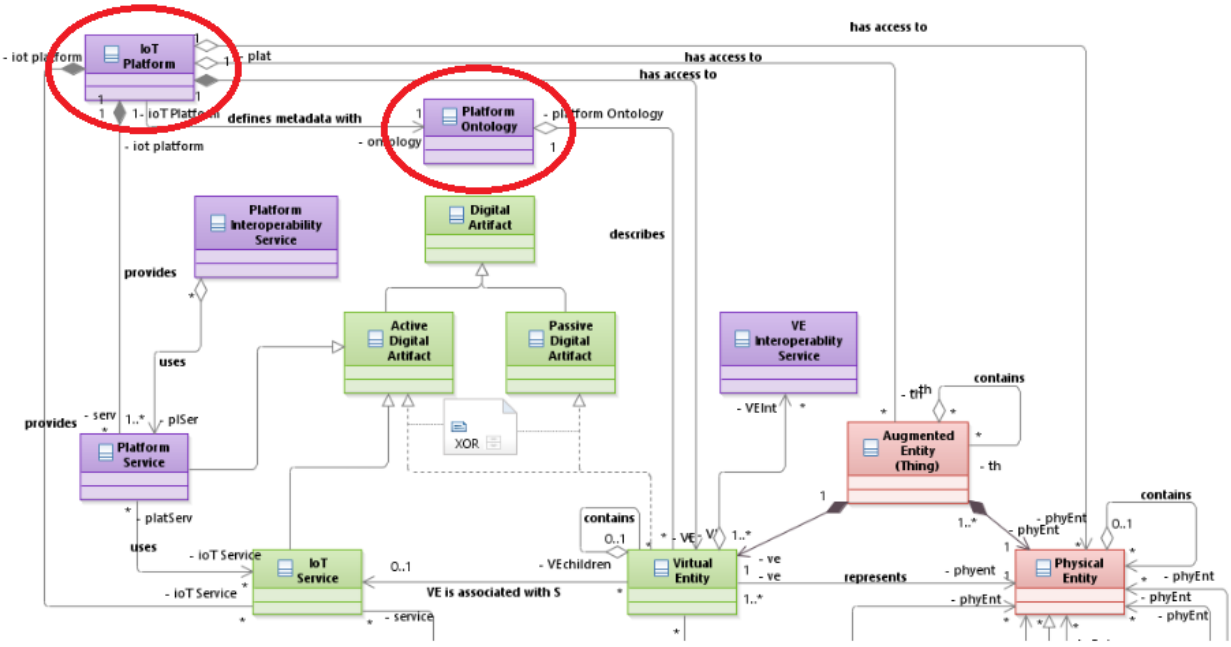


Figure 55: Domain Model entities involved in Data&Semantics Interactions

Interactions between Domain Model elements on DS2DS layer are exclusively between an IoT artefact (platform) and its ontology. They are two-fold and divided into preparation of ontology and its usage.

In the INTER-IoT approach, every platform (system, application, etc.), which would voluntarily like to interoperate with one or more other platforms needs to be prepared and willing, first. In order to enable semantic interoperability, and explicit ontology is needed. Some platforms, or middlewares (e.g. UniversAAL, OpenIoT) already need to have OWL ontologies ready before deployment. These ontologies can be used in INTER-IoT.

In other cases, any semantics present in a platform needs to be extracted and formalized into an OWL ontology. This process is called “lifting to OWL”, and is described in [1]. In short, lifting to OWL is a process in which semantics of platforms, sometimes contained in data schemas, are made explicit and stored in an ontology. The most popular languages that can be used in lifting are: XML, RDF, JSON LD, but other formalisms are also acceptable. Such formal description must cover all aspects of data communication that will be needed for interoperability. It must represent entities (and their properties), which exist “inside” of the artefact. This formal description is to be used in creation of platform ontology, as well as in instantiation of communication channel(s) needed to send/receive messages to/from other artefacts (platforms, devices, middleware, services, or applications).

Once a platform has an ontology it is then used to create alignments to and from the GOIoTP, that later serve as configuration for IPSM. The semantic translation process that takes place inside IPSM is non-discriminative when it comes to contents or intentions of communication. It simply translates the meaning of messages, according to configuration of the communication channel that received the messages. Dynamic creation of communication channels allows DS2DS interactions to serve multiple purposes and assist in operation of other INTER-IoT components, as well as other artefacts, if they wish to use IPSM as a “stand-alone” service within INTER-IoT.

It should be stressed that data processing within a single artefact can be represented through more than one ontology (or, possibly, modules within a single modular ontology). Such situation can materialize when different ontologies are used in different “conversations” (concerning different aspects of data, usually with different artefacts). In this way, proposed approach gains flexibility and addresses the issue of scalability (semantic processing is applied to smaller (sub-)ontologies).

3.5.4 INTER-IoT Channel Model for Interoperability

3.5.4.1 Introduction

The channel model depicted in this section comes from the concepts presented in 7.6.4 of [25]. The following points aim at explain how are the different layer protocols involved in the interoperability. This first draft helps on selection the appropriate mechanisms to build an effective software architecture capable to fulfil the objectives of the project.

Interoperability can only be achieved if every layer can communicate to its counterpart (at the same layer level) on another device. For this reason, an architecture has been set up in a way that allows virtualization at low-levels already.

Each paragraph of this section will describe a certain interoperability action. These actions will be illustrated by both the gateway configuration, as well as the virtual configuration for multiple protocol stacks.

3.5.4.2 Device to Device Interactions

Device to device interaction or D2D interoperability will be made possible by the Rules Engine that is acting on the dispatcher. The image below is a screen-capture given from the upper part of the device layer. This is where the Rules Engine is located.

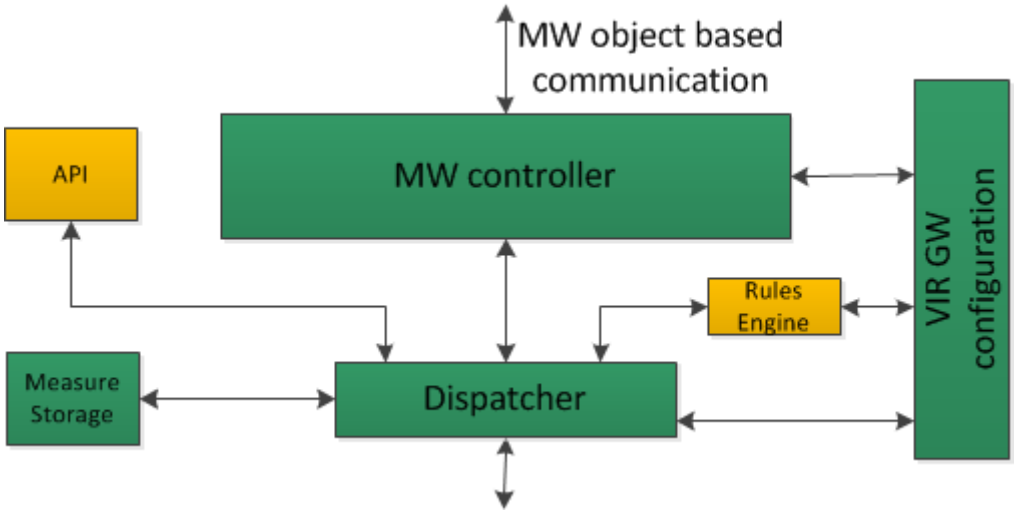


Figure 56 Device-to-Device interactions with location of Rules Engine

The Rules Engine will instruct the dispatcher to route a signal not to the MW-layers above, but via the MW modules and bridges to another device dispatcher to initiate D2D interoperability. Which bridge is used will be managed by the GW configuration, the Rules Engine will only interfere when interoperability is needed. When data needs to go up into the higher layers the dispatcher will simply get the specific bridge information from the GW configuration manager.

The Interoperability actions are programmed into the rules engine through the API. The user must set up the mapping of the communication in an easy way through a user interface which will be running on the API to allow the creation of inter-operability communication mapping.

The entire upper part of the gateway as shown in the figure can be implemented in the virtual world and does not necessarily have to be implemented on the device. When preferred however it can also be implemented on the device.

Device to device interoperability is possible in two ways. The first way is the simplest way, that is when the sensor and actuator are both connected to the same gateway. In this case, there is a direct link inside the gateway. The dispatcher will route the communication via the rules engine to the correct protocol module which will route it further to the connected access network.

The schematic representation is depicted below.

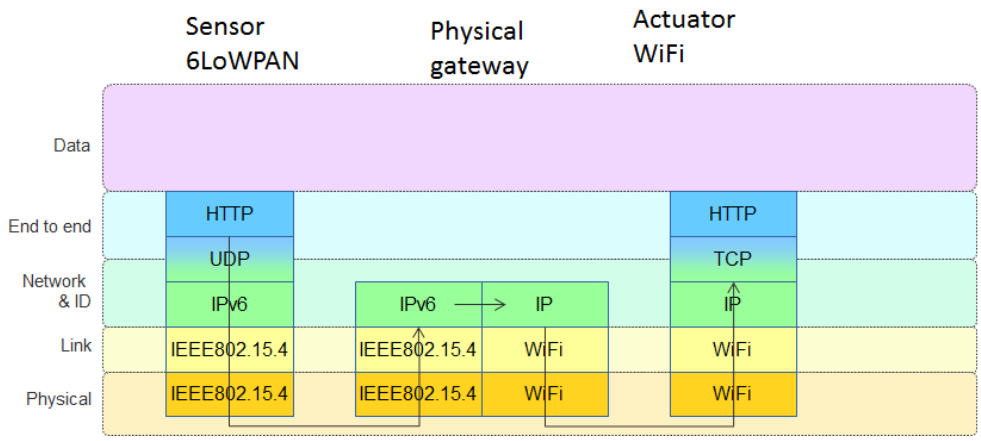


Figure 57 Communication diagram in device-to-device interoperability

In this case the gateway is implemented in the physical device. When the gateway is virtual or when the sensor and actuator are connected to different gateways, a more complex situation occurs.

In case of a virtual gateway and sensor and actuator connected to the same device there is only 1 gateway, in the representation below is the situation given for sensor and actuator on different gateways.

For this case the communication is going through the physical gateway, which translates it to Ethernet and sends it to the virtual part. Here the dispatcher will route the communication via the rules engine, only this time the communication channel is routed to the middleware layer. In this layer, can, depending on the settings, interconnectivity be created on several levels.

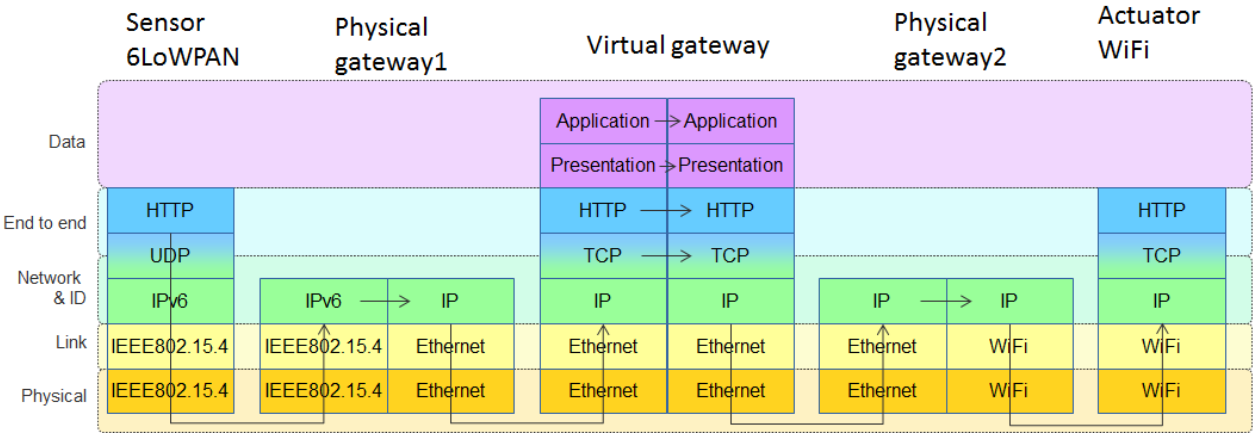


Figure 58 Communication diagram in device-to-device interoperability with virtual gateway

3.5.4.3 Network to Network Interactions

The interaction of a resource from an IoT Device that communicates with other resource from another IoT Platform will be used as an example of interaction in the N2N layer. This interaction is described in the image bellow through an example with the protocol stacks that take part in the interaction.

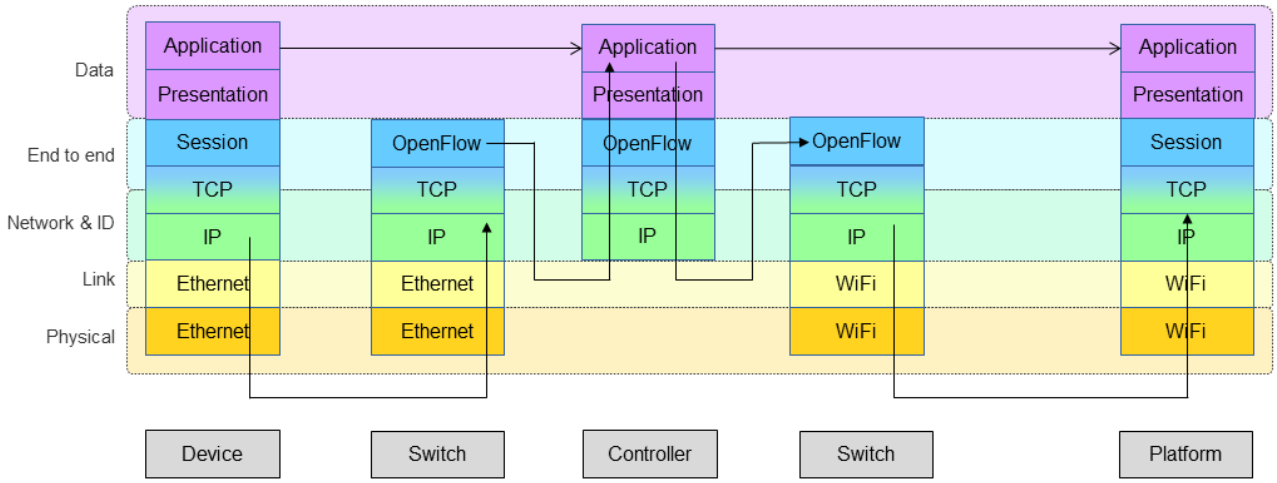


Figure 59 Communication diagram in network-to-network interoperability (SDN)

A specific resource from an IoT Device located within it wants to communicate among the SDN network. For that purpose, the Devices connect with the AP or Switch that is connected to the network as a network gateway. This switch, if needed connects with the SDN controller to updates the routing information table to know the next hop. Finally, the information arrives until the last switch to which is connected our IoT Platform destination.

An example of software defined radio communication is indicated in the image below.

In this example, an application on the left side of the image wants to send a message to another application. The platform supplying the message must first connect to the SDR via Ethernet. The SDR, functioning as a level 2 bridge, will send the information from the external SDR unit to the SDR unit connected to the Inter-IoT SDR gateway module. The information is then passed on to the receiving application. The SDR unit in this example provides an additional entry/exit point to the Inter-IoT that can operate to meet end user requirements if standard methods are not available.

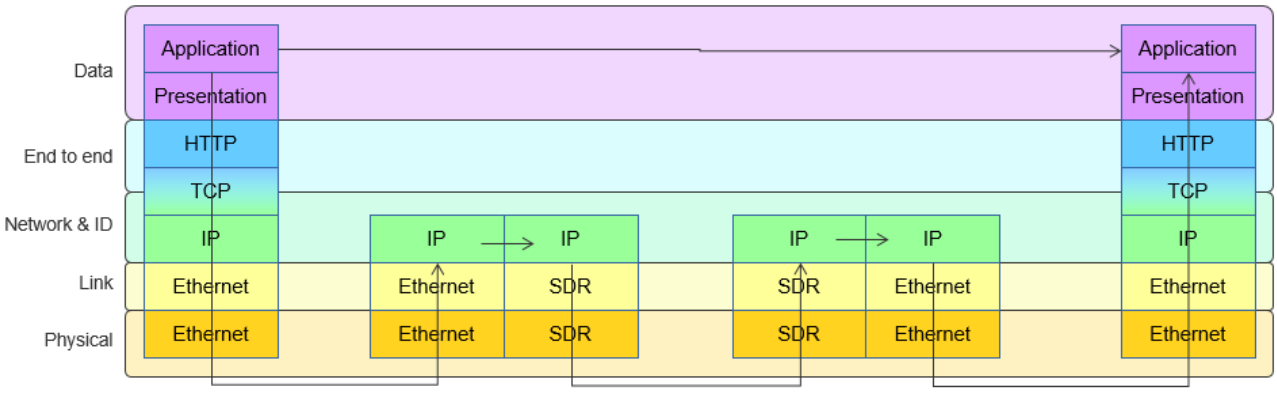


Figure 60 Communication diagram in SDR

3.5.4.4 Middleware to Middleware Interactions

An example of middleware to middleware communication is indicated in the image bellow. Application on IoT middleware on the left side of the image wants to send a message to another application on another IoT middleware on the right side. The message must first cross a bridge from middleware of the sending platform into INTER-IoT middleware, then be semantically translated in the presentation layer of the INTER-IoT middleware into the format, understood by the target middleware, only to cross another bridge that is associated with the middleware of the receiving platform. The second bridge also routes the message to the target middleware through WiFi instead of Ethernet, demonstrating the technical agnosticism of the INTER-IoT middleware.

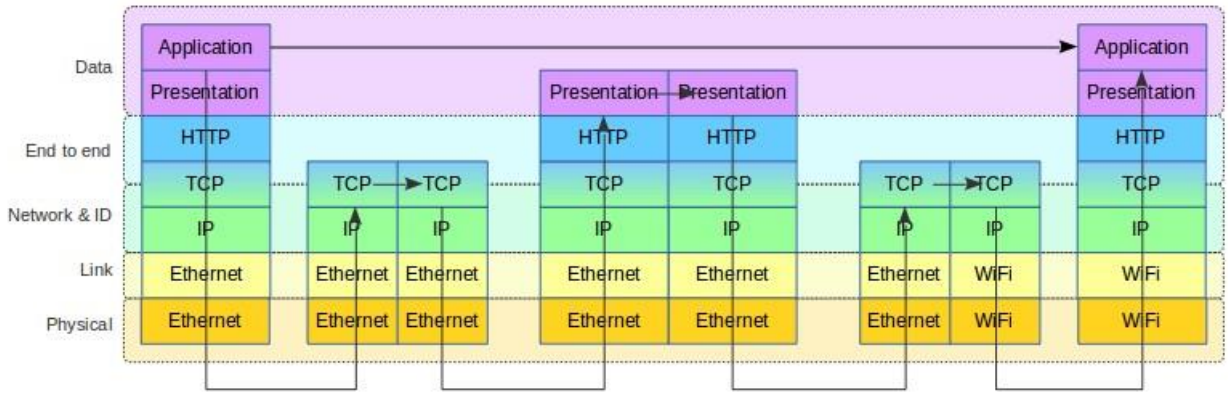


Figure 61: Communication diagram in middleware-to-middleware interoperability

3.5.4.5 Application & Services to Application & Services Interactions

The interaction of a Service from an IoT Platform that communicates with a Service from another IoT Platform will be used as an example of interaction in the AS2AS layer. This communication is described in the image bellow through an example of stack protocol that takes part in the interaction.

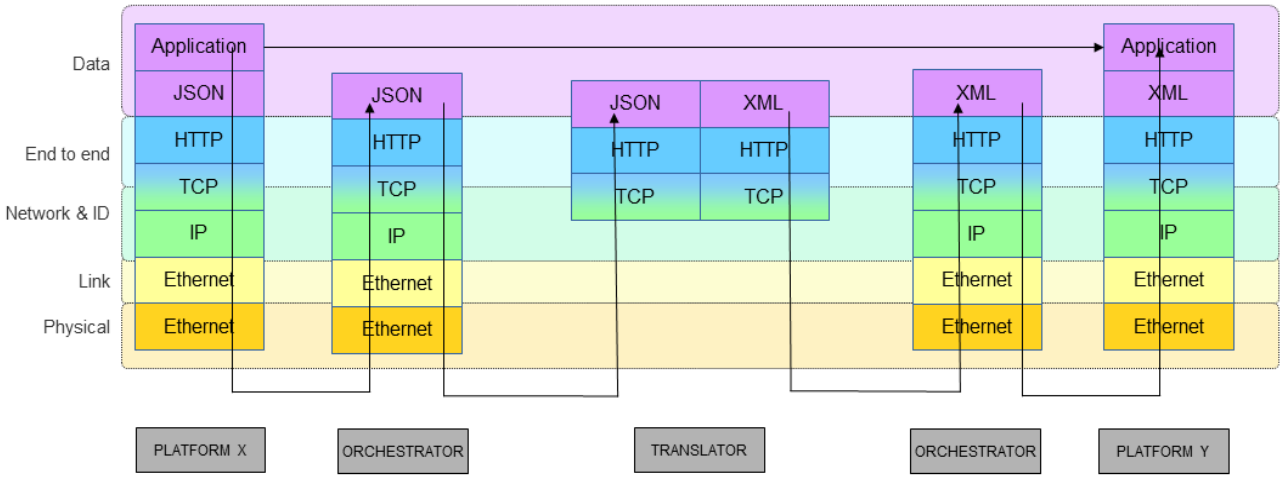


Figure 62: Communication diagram in AS-to-AS interoperability

Application within an IoT Platform on the left side of the image wants to send a message to another application in another IoT Platform on the right side. It works similarly as in the MW2MW layer.

The message comes from the Platform X Service to the Orchestrator inside INTER-IoT AS2AS solution. The orchestration module would be responsible of making calls to IoT Platform Services and carry out the internal processes necessary to make the composition successful.

If needed, a specific translation of presentation format takes place in the translator component of the AS2AS. It performs translation of input messages, expressed in source format, to output messages, expressed in target format.

Later, the parsed message, with the format of the destination platforms, goes from the Orchestrator to the receiving Platform Y Service.

3.5.4.6 Data & Semantics Interactions

DS2DS layer interactions are limited exclusively to application layer software. In DS2DS a central component acts as an intermediary in communication, and has its own communication infrastructure.

To achieve interoperability on data and semantics layer, IoT platforms (and other artefacts) use this central mediation component that acts as a “semantic bridge” between IoT artefacts ((platforms, gateways, applications, etc.). This mediation component, which can be named IPSM (Inter Platform Semantic Mediator), performs the semantic translation (configured with ontology alignments) of incoming messages, representing semantics of artefact P1 to semantics of artefacts e.g. P2, P3. Figure 1 shows a general view on the sample interaction, whereas Figure 63 shows details of the interaction between source artefact, IPSM and target artefacts.

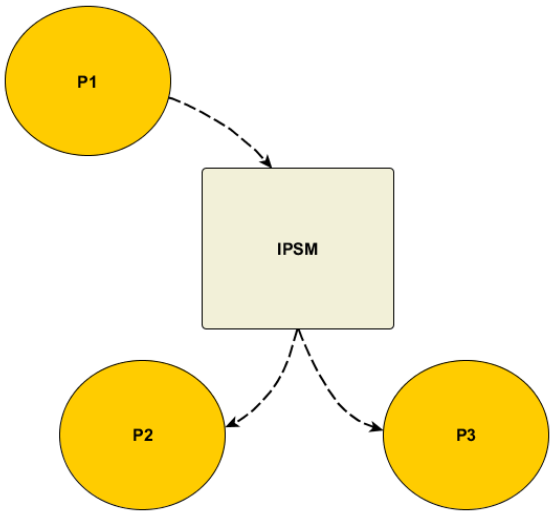


Figure 63 Interaction between source artefact, IPSM and target artefacts

IPSM must expose a mechanism for configuration. An additional communication infrastructure is required to enable communication between IPSM and all other artefacts that are to use its semantic translation services. Communication infrastructure can be based on communication channels consisting of a source, sink, and a series of flows between sources and sinks. The IPSM will work concurrently servicing multiple communication channels each representing a single “communication”. While each channel will have one input and output, multiple artefacts will be allowed to use its output, and write to the input. Channels may be combined in the communication infrastructure to facilitate one-to-many, many-to-one and many-to-many communication.

4 INTER-IoT Reference Architecture

4.1 Functional View

4.1.1 IoT-A's Functional View

IoT-A's Functional View has the same nine functional views than the Functional Model:

- Application
- Service Organisation
- IoT Process Management
- Virtual Entity
- IoT Service
- Device
- Management
- Security

The Application FG and the Device FG are considered out of the scope of the IoT-A Reference Architecture, so they are not described in the Functional View. In IoT-A's diagram of Functional View, they are coloured in yellow to indicate they are not described.

The Functional Components of the seven functional groups included in the IoT-A Reference Architecture are included in the following diagram:

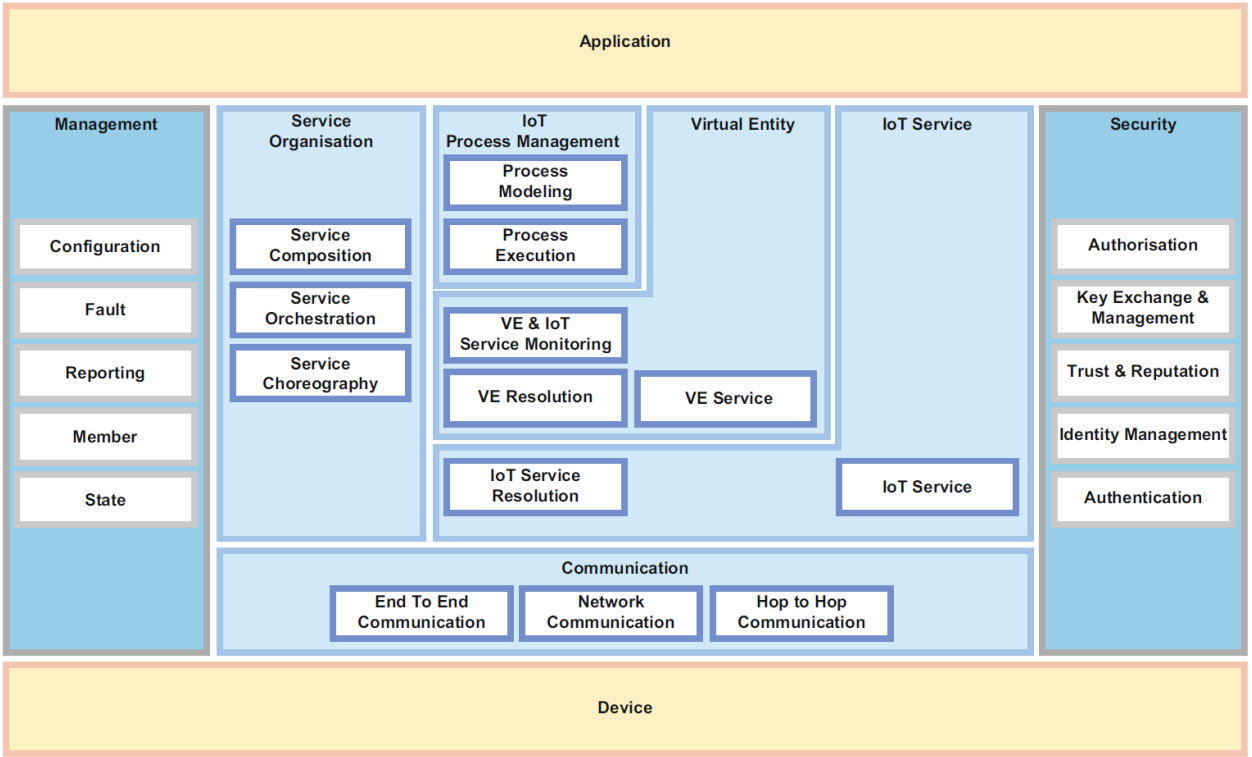


Figure 64: IOT-A functional components

The frames inside each functional group are the Functional Components identified for each Functional Group. The description of each Functional Group has been done in the Functional Model. Describing each of the Functional Components of the Functional Groups does not add any value to this document. If the reader is interested in more details, a read to [25] is recommended.

4.1.2 IoT Functional View Platform Analysis

Following the functional view proposed in IOT-A, an analysis of the platforms under study has been done. In the next subsections, commonalities and discrepancies between platforms will be analysed for each functional group, paying attention to the relevant functional components in IoT-A.

To perform the study, each platform has been studied and determined if they provide at least one feature to cover partially or completely each of the functional components of the IOT-A. Then, the number of the FC-compliant has been aggregated, giving an idea of the availability of the features, also revealing what is considered important in the industry and academia, which has a relevance in order to prioritize functionalities when a great adoption is intended.

This analysis aims at two main objectives, on one hand, an analysis of the so-identified market and research relevant platforms, which (as aforementioned) helps to find overall connection between different solutions and determine which ways of interoperability will be more effective and beneficial in the long term. On the other hand, the project has a limited scope and it will offer support (at integration-ready level) to a limited number of platforms as already stated in several points of this documents. The re-elaboration of the analysis with the initially supported platforms will give also an idea of which interoperability layers' mechanisms prioritize in order to provide early results to support pilots and third parties to join.

Complete dataset with annotations are available in this document in the Annex 3.

4.1.2.1 Application

The application comprises all those features that are domain/application specific and thus, they are out of the scope of the definition of the platform interoperability, as is defined in INTER-IoT.

While in the study performed some aspects of the application FG were described (such as the domains where the platform operates or offer specific solutions (see 3.4 and subsections), the conclusions of them are not relevant for the functional view or the development of a reference architecture to build interoperability mechanism between IoT platforms.

For completeness, the prevalence of the domain aimed in the platforms under study is shown in the following histogram:

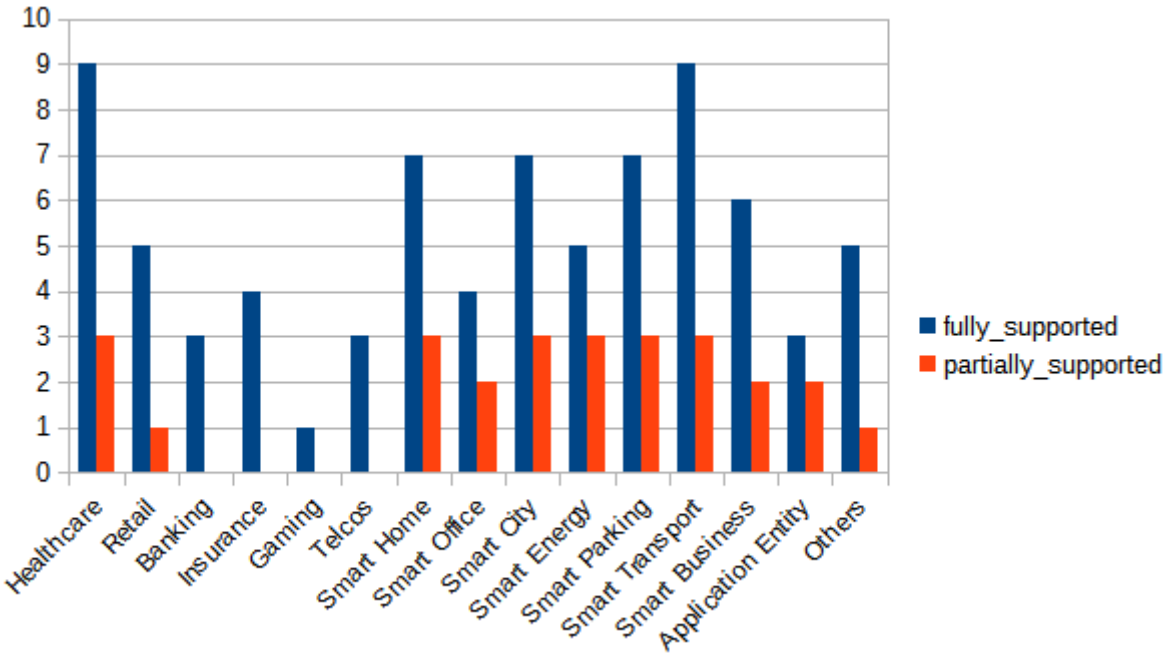


Figure 65 Domain prevalence in studied platforms

The analysis reveals that the domains with more support are healthcare, transport, home, city and parking. This shows the areas where the IoT is expanding faster (such as transport) or it has a smoother implantation (as in cities).

4.1.2.2 Management

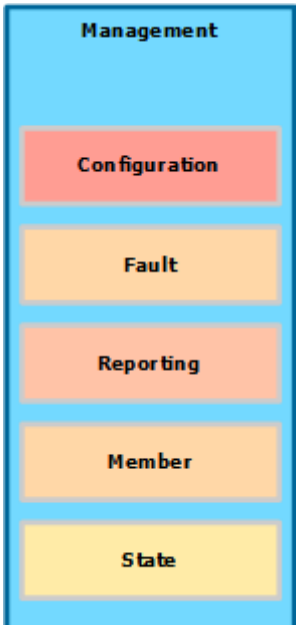


Figure 66 Management FCs prevalence in 15 platforms study

The study of the Management functional components for the set of 16 platforms shows a clear predominance of the Configuration FC (implemented in 12 out of 16 platforms analysed) and the Reporting FC (10/16).

This shows that system initialization including the attached devices and the assessment of the overall performed are considered key pieces for an IoT platform. However, in general, the implementation of the IOT-A FCs of this group is high in the analysed set, finding only a significant lack of coverage on the State FC, which can be related to the legacy systems support that many of the platforms present. The legacy systems coming from home automation or telemetry often do not support queries or network info reporting, making virtually impossible to feature the state of the device networks. As a matter of the facts, those platforms that were born with a strong support of these communication protocol, do not care much about those features that were not available for their focus technologies.

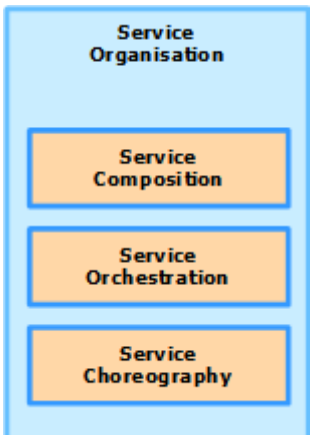
In the case of the selected platforms to be natively supported in INTER-IoT (see Figure 10), all the FC similarly covered, being particularly remarkable, as in the case of the Configuration, which is implemented by all of them. In

the rest of FCs, the 3/5 platform cover them, a prevalence similar to the results of the complete study.

For the purpose of the INTER-IoT Project, this means that, at management level, a common configuration interface could be implemented for the INTER-IoT user, as part of the framework planned. Other features would have compatibility with some but not all the initially supported platforms, so that the feasibility is limited and the decision of implementation depends strongly on the requirements of the project and the INTER-FW (see deliverables D2.3 and D2.4). In general, FIWARE, MS Azure and Open IoT are the most complete platforms in terms of system management among the selected platforms, while extending the scope to the complete study, GE Predix and Sofia2 also have full feature set.

Figure 67 Management FCs prevalence in INTER-IoT initial platforms

4.1.2.3 Management



The service organisation group, which evaluates the ability to manage services in the platforms is highly supported by all platforms, presenting figures of coverage about the 60% ~ 70% of each functional component. In this case, the degree of implementation is similar in the three components, being slightly more popular the service orchestration. Service organisation is a concept highly bound to the service presence itself, so it can be observed in the detail that is very usual that platform with more upper layers implement at least two components of this group while more middleware-centric platforms (such as AllJoyn or OneM2M) do not implement any at all, since they are device focused in spite of the service focus of the former.

Figure 68: Service organisation FCs prevalence in 15 platforms study

For the selection of platforms initially supported, the situation is different. OneM2M is a middleware centric platform, so it does not give support to any kind of service composition, not including services in its domain model. The rest of platforms have a wider scope and support in some way operations and combinations with services. Consequently, it can be observed a coverage of 80% in orchestration and choreography.

This FG is especially relevant for the AS2AS, as it gives a first idea of which platforms will be able to connect services and also provides an idea of the service interoperability mechanisms supported in the focus group.

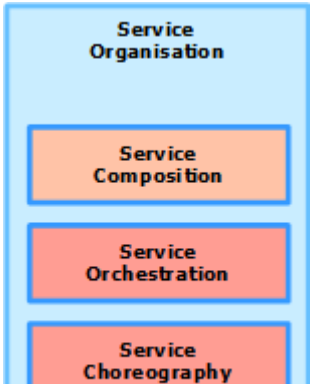


Figure 69 Service organisation FCs prevalence in INTER-IoT intial platforms

4.1.2.4 IoT Process Management

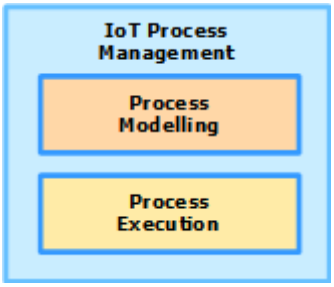


Figure 70 IoT process management FCs prevalence in 16 platforms study

This group has a significant lower coverage in all platforms, being present in around the 50% of platforms, regardless if it is analysed the full set of platforms or the focus group.

The concept of IoT processes in IOT-A is related to the Business Management and how the IoT specific constraints are mapped there. As the definition of IoT Process is very specific and new, part of the platforms does not offer a particular solution for this characteristics, transferring the responsibility to the end user (using external services or custom logic). This is particularly true in the device or middleware centric platforms which provide more features in the lower layers.

In the case of the focus group the situation is similar. Process modelling is supported in Azure, Open IoT and UniversAAL (with limitations in the last two, though) while process execution is supported only in Azure and OpenIoT.

With the perspective of interoperability, this FG is not very relevant, since the business processes concerning two or more different platforms can be modelled (and executed) externally leveraging the already proposed AS2AS layer.

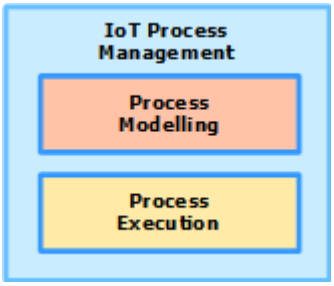


Figure 71 IoT process management FCs prevalence in INTER-IoT initial platforms

4.1.2.5 Virtual Entity

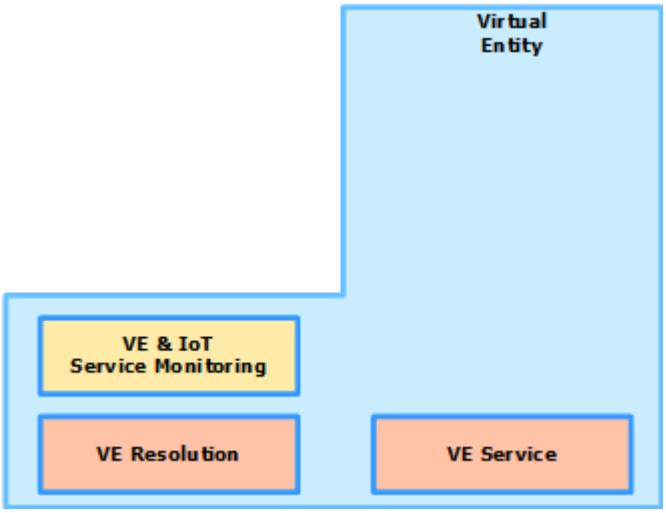


Figure 72 Virtual Entity FCs prevalence in 15 platforms study

The Virtual Entity Functional Group has the mission of handle the relations between virtual entities and associated services, providing the needed mechanisms to discovering, updating and accessing to entity level services and features. In the global study, it has been found that the accomplishment of these features is high except for the VE & IoT Service.

Monitoring component, which is probably the most complex of the three components identified in IOT-A for this FG. Therefore, it is declared to be implemented (or partially covered in less than the 50% of the analysed platforms (6 out of 16). However, the VE Resolution and the VE service capabilities are supported in more than the 70% of platforms, which shows the relevancy of these components for the platforms.

For the focus group, the implementation of these components reaches higher levels, reaching the 80% in the case of VE Services component. In this case, the OneM2M platform makes the difference since it does not support virtual entity related components, despite the rest of the platforms of the subset.

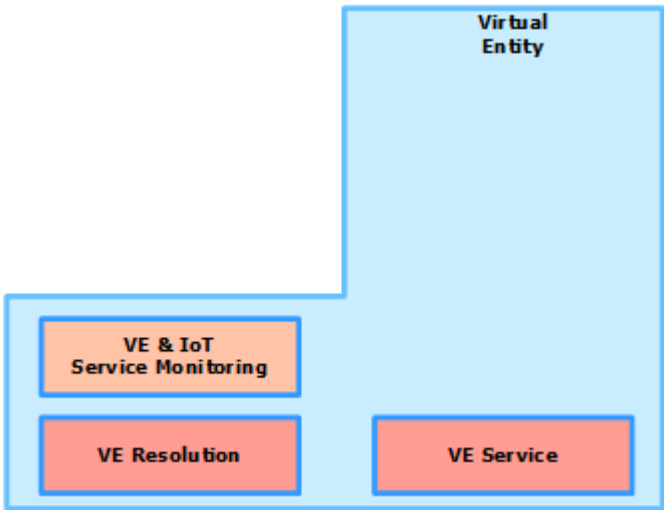


Figure 73 Virtual Entity FCs prevalence in the INTER-IoT initial platforms

Virtual entities are a key concept in the INTER-IoT concept, architecture and framework. It is thanks to the entity virtualization that heterogeneous data can be harmonized, stored, transmitted and even translated into different ontologies. As explained in previous sections and in INTER-IOT Deliverable D3.1, released at the same time of this document, the concept is largely used at interoperability level, being a keystone for D2D and MW2MW. The high accomplishment of this functional group in the focus group and in the global study guarantees the viability of the solutions proposed in the Deliverable D3.1.

4.1.2.6 IoT Service

The IoT Service FG and its components are well covered in the IoT platforms, according to the study done. In this case, the study was one step further and analysed the prevalence of specific services, from a set of platform services typically present in sensor-related scenarios:

- Query information
- Update information
- Use resource operation/service
- Subscribe to information
- Subscription with filters
- Registration
- Historic data access
- CEP
- Big data storage
- Others
- IoT Client

While for the IoT Service Resolution, a list of features was also provided, based on the definition of IOT-A.

- Discovery
- Lookup
- Service Id. Resolution

- Service Descr. Mgmt.
- Others

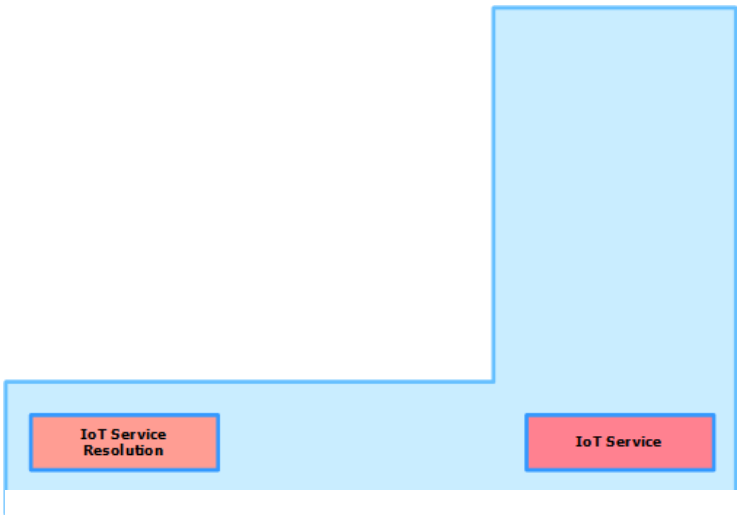


Figure 74 IoT Service FCs prevalence in 15 platforms study

In general terms, the service implementation levels are high, reaching the 100% or near in cases as the registration or the complex event processing (CEP). This also occurs in the IoT Service Resolution, which is supported in the ~70% of platforms on average for the four specific features analysed.

For the initially supported platforms, the conclusions are similar, showing again the lack of services implemented in OneM2M, much more centred in communications than in services. The IoT Service Resolution is also well covered by all the platforms with the known exception of OneM2M.

The following two histograms show the number of platforms that implement a version of the listed features for each of

the FC related to IoT Services:

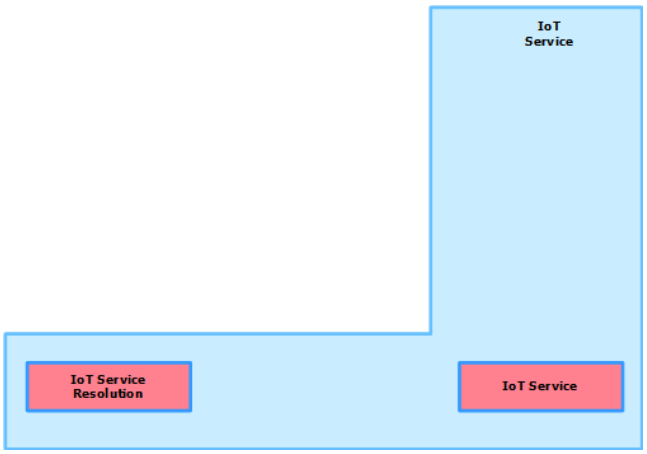


Figure 75 IoT Service FCs prevalence in INTER-IoT initial platforms

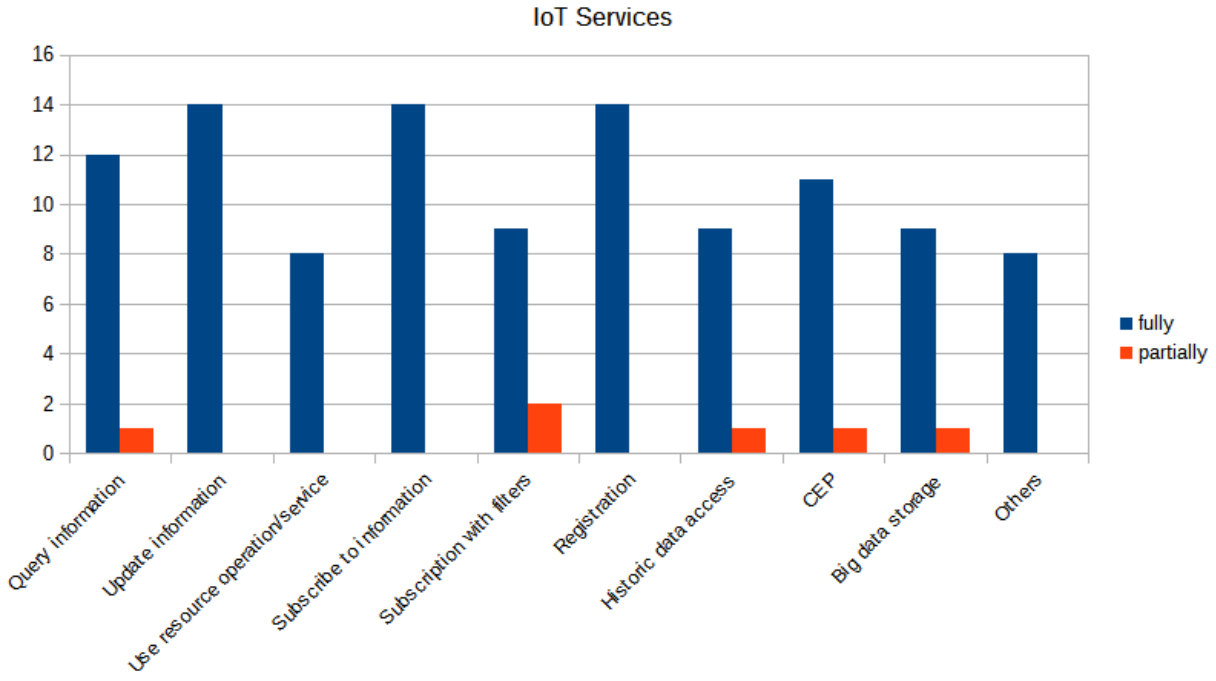


Figure 76: IoT services implemented in the studied IoT platforms

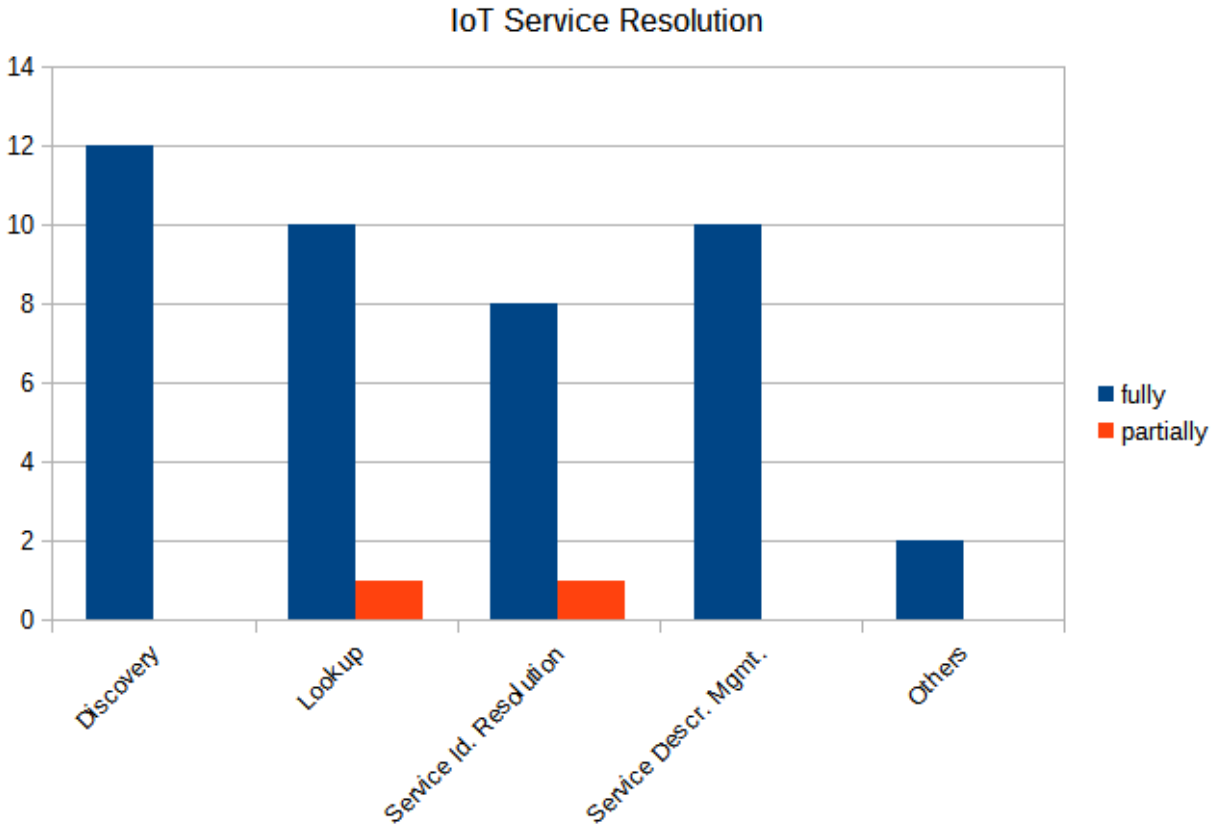


Figure 77: IoT service resolution policies in the studied IoT platforms

4.1.2.7 Security

Figure 78 Security FCs prevalence in 15 platforms study

The Security FG is a transversal group that applies to all the rest of FGs per the IOT-A guidelines. With this in mind, a thorough analysis of the security capabilities was performed, assessing not only if each component is implemented somehow in every platform but also describing, when possible, the technologies used and more specific details related to the implementation of the components (see Annex 3 for further details).

The results show that the security is a common concern in the platforms analysed. All the analysed platforms implement more than one component of the identified in the Functional View. However, the strategies to accomplish the Security related operations differ between platform. While authorisation and authentication are the preferred mechanisms (present in ~ 90% of the platforms), trust and reputation is significantly less prevalent with only the ~ 30% of platforms implementing policies or components in this way.

For the focus group, results are similar, with a better support (practically full support) of the authorisation and authentication components and a better coverage of the rest of components.

The security FG is a concern of each interoperability layer and the INTER-IOT framework, which is in charge to coordinate and orchestrate all the security policies in order to maintain or improve the existing security standards in the platforms.

According to the results obtained, the interoperability efforts here should go on the direction of ensuring and, when possible, centralising the authentication and authorisation in platforms.

Figure 79 Security FCs prevalence in INTER-IoT intial study

4.1.2.8 Communication

The diversity of the device-to-device or device-to-gateway communications is one of the reasons of having so heterogeneous platforms. It is usually a starting point for creating a so-called information silo, since the lack of device interoperability with other devices or, more important, with other platforms usually ends in a domain/application specific deployment that forgets completely about interoperability due to the difficulties to achieve it.

This is the closest FG to the physical level, and thus it should better be implemented in the device and middleware centric platforms. However, probably due to the reasons described previously, the implementation of the components in this group is still poor in the platforms analysed. The most spread mechanism of communication supported is the end to end communication, while the network communication and the hop to hop communications are marginally covered.

Figure 80: Communication FCs prevalence in 15 platforms study

The situation in the general study and in the case of the focus group is similar. With this results, it keeps clear that further efforts in communication standardization are needed. From the project point of view, the end to end communication is the best option to implement interoperability mechanisms, as devised in D2D layer (see Deliverable D3.1).

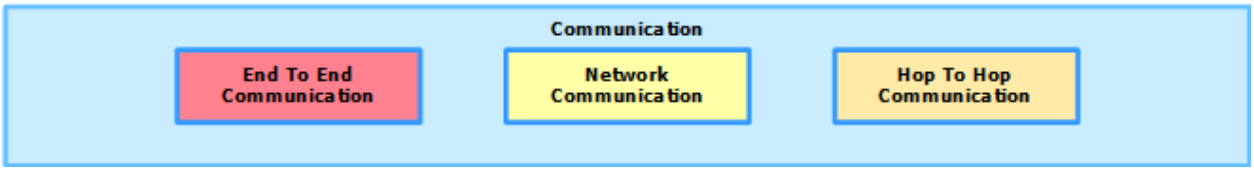


Figure 81: Communication FCs prevalence in INTER-IoT initial platforms

As mentioned, the range of different communications/standards/protocols to (mostly) physically transmit information from one device to a gateway, a dongle or another device is vast. The study considered this and analysed the full set of possible communication in all the platforms reviewed, whose result is depicted in the following histogram:

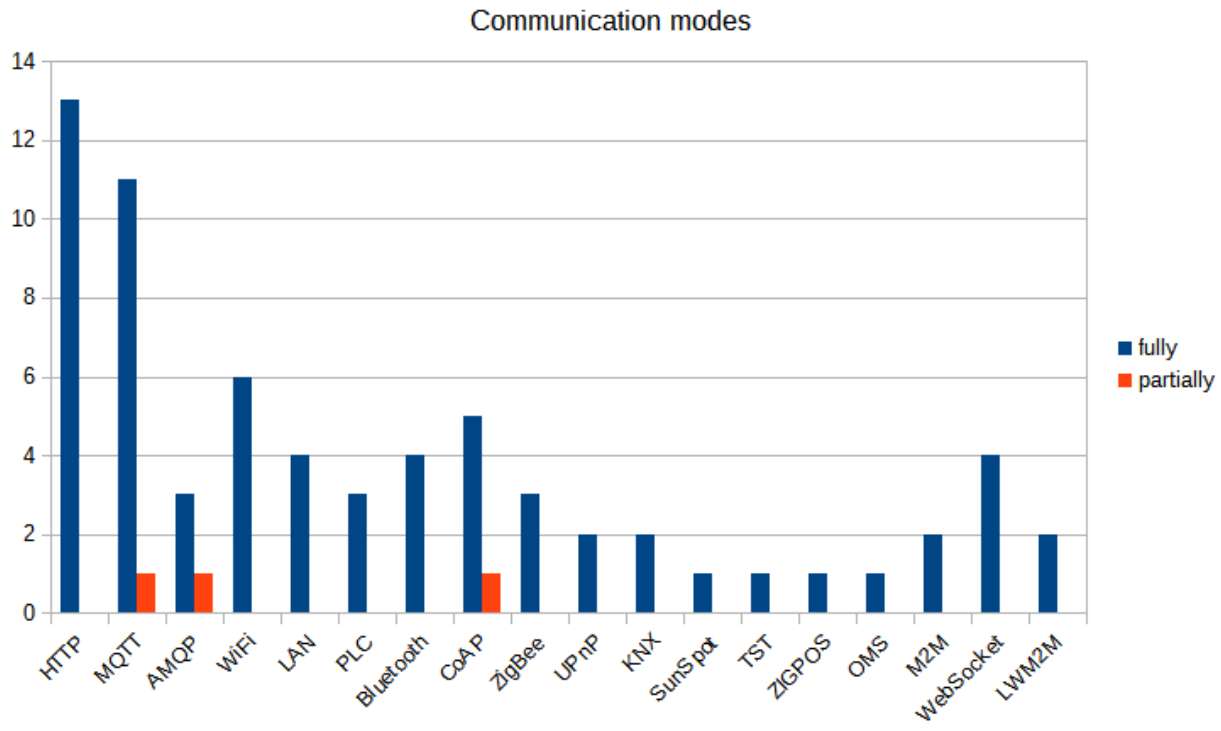


Figure 82: Communication protocols at different layers supported by the platforms studied

It is obvious that this chart mixes very different communication modes, some of them compatible between them (e.g. it is possible to have Bluetooth and MQTT communications at the same time). The analysis has considered each supported communication regardless the OSI layer which is

aimed at, to show on one hand the variety of possibilities and also to find the most popular device to platform communication methods that are supported in the cohort under study.

4.1.2.9 Device

This group is considered out of the scope of this document and is not included in the analysis.

However, since information about supported devices is offered, there has been created a histogram with the main groups of device groups and its support in the studied platforms.

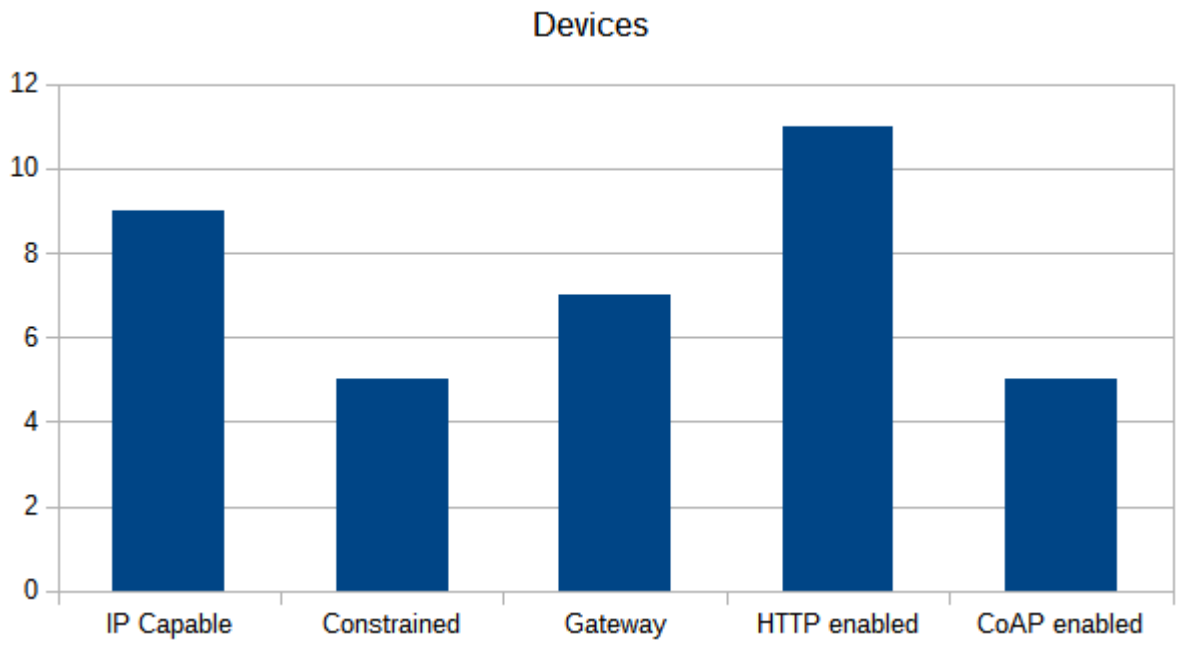


Figure 83: Type of devices aimed by the platforms studied

4.1.3 INTER-IoT Functional View

The Functional View of INTER-IoT has been derived from two inputs:

- INTER-IoT Functional Model, as described in section 3.4.3. The Functional Groups and their relationship were kept for the Functional View.
- The requirements and use cases identified in WP2.

The requirements and use cases have been analysed to group them in clusters of common functionality. Then these clusters have been reviewed with the technical leaders for the different interoperability layers and the Software Architect to refine them.

We have elaborated a novel Functional View for the INTER-IoT Reference Architecture that is depicted in the following figure:

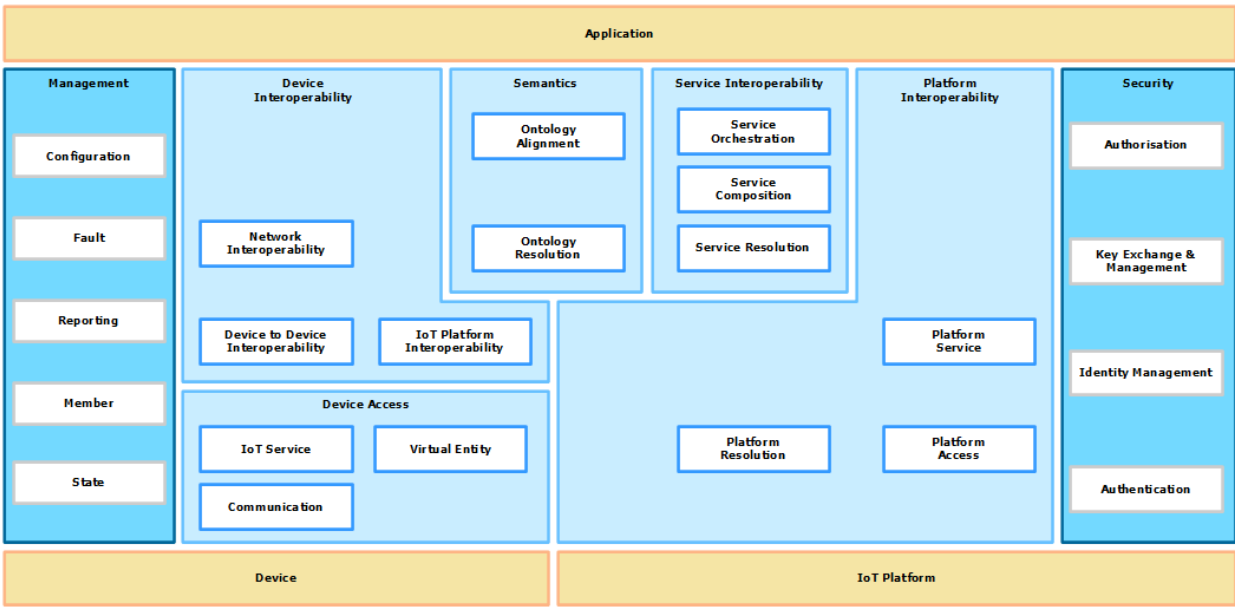


Figure 84: Functional-decomposition viewpoint of the INTER-IoT Reference Architecture

The Functional View of INTER-IoT is focussed on the challenge of interoperability between IoT platforms. This has led to manage eleven Functional Groups. Among these, it's important to remark that:

- The Application FG, the Device FG and the IoT Platform FG are out-of-scope of the INTER-IoT Reference Architecture, consequently they have not been described and have been represented in yellow colour. They are considered out of the scope because they represent external groups that are going to be interoperated through the INTER-IoT. They already exist, they are not going to be created in the project, they are just going to be used.
- The remaining five longitudinal Functionality Groups are represented in light blue colour.
- The Management FG and the Security FG are transversal Functionality Groups and are shown in dark blue colour. These transversal groups provide functionalities that are required by any of the longitudinal groups.

In the following sections, we describe each of the Functional Components grouped by Functional Group.

4.1.3.1 Service Interoperability

The role of the Service Interoperability FG is to support the Application & Service to Application & Service (AS2AS) interoperability through the definition and execution of new compound services that make use of already existing services in the underlying IoT Platforms. Its goal is to use services from different IoT and create new services based on them.

The Service Interoperability FG consists of three Functional Components (see figure below):

- Service Resolution;
- Service Composition;
- Service Orchestration.

Figure 85: Service Interoperability

The **Service Resolution FC** is responsible for the storage of what we call *flows*. A flow is a logical definition of a sequence of steps, each of which can be a service existing in an IoT Platform. The functions of the Service Resolution FC are three: (1) to resolve the access to IoT Platform services that can be used in a flow, (2) to store the definition of services and atomic components so that they can be used by the Service Composition FC and instantiated by the Service Orchestration FC and (3) to provide storage and access to the logical definition of flows.

The flows that are defined for service interoperability have to be stored by the Service Resolution FC, also enabling the semantic cataloguing of services and their discovery.

The main role of the **Service Composition FC** is to design new compound services based on services that IoT Platforms exposes. These services have been previously defined and catalogued by the Service Resolution FC. The new services are designed like flows which will be later executed.

The flows that are designed by the Service composition FC are stored by the Service Resolution FC.

Finally, the **Service Orchestration FC** is responsible for the execution of the flows that are stored in the catalogue managed by the Service Resolution FC. The execution of these flows are initiated by triggers (user request, IoT Platform event or alert, data received, etc.) which have been defined for each flow.

4.1.3.2 Semantics

The **Semantics FG** is the central Functional Group that addresses the challenges related to semantic interoperability of IoT Platforms. It provides support for the other FGs dealing with interoperability about IoT: The Service Interoperability FG, the Platform Interoperability FG and the Device Interoperability FG.

The Semantics FG consists of two Functional Components (see Figure 86):

- Ontology Resolution;
- Ontology Alignment.

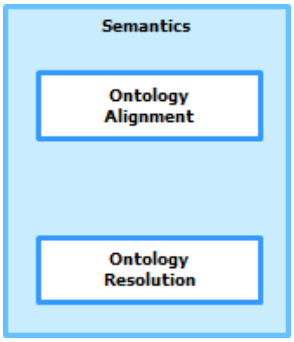


Figure 86: Semantics

The **Ontology Resolution FC** is responsible for managing the different ontologies used at the various IoT Platforms that are connected through INTER-IoT. These ontologies have a double approach:

- Syntactic knowledge;
- Semantic knowledge.

The syntactic knowledge is about being aware of the syntax that the IoT Platforms uses for interchanging data, what usually is related to the communication protocol being used or the type of the API: JSON, XML, etc.

The semantic knowledge is about being aware of the structure and meaning of the data, usually through OWL or similar definitions (JSON-schema, XSD, etc.).

The Ontology Resolution FC is the component that stores these data descriptions and offers access to them for the Ontology Alignment FC.

The **Ontology Alignment FC** is responsible for performing the alignment from a source data with an ontology to a target data with its own ontology. It makes the data translation between two ontologies, using the ontology definitions resolved by the Ontology Resolution FC.

4.1.3.3 Platform Interoperability

The overall goal of the Platform Interoperability FG is to interact with the different IoT Platforms to be interconnected. It is the responsible for accessing the IoT Platforms, not for implementing any of the features that the IoT Platforms provide.

The Platform Interoperability FG has three Functional Components (see Figure 87):

- Platform Resolution;
- Platform Access;
- Platform Service.

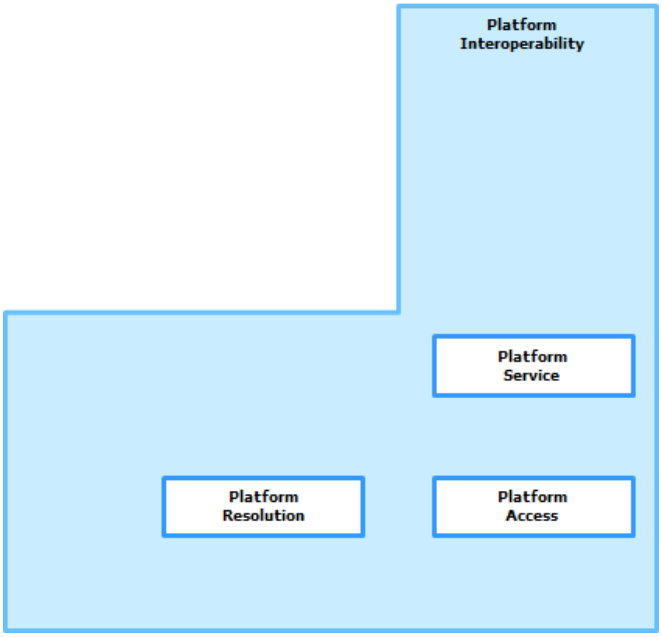


Figure 87: Platform Interoperability

The **Platform Resolution FC** is responsible for discovering and cataloguing the IoT Platforms that are available at a specific deployment of INTER-IoT as well as their devices, capabilities and IoT Platform Services, so that they can easily be found, when needed. This allows the remaining groups to not to know about the location of the platforms, or how the devices are connected to them. When any Functional Component needs to access a specific resource of an IoT Platform, it will use the Platform Resolution FC to get its identification, location and way of accessing.

The Platform Resolution FC is also responsible for roaming capabilities of mobile devices between IoT Platforms.

The main role of the **Platform Access FC** is to implement the functions needed for connecting to an IoT Platform and accessing their resources (specific discovery, lookup, data query, data subscription, device registry, etc.). This includes the use of the appropriate protocols and APIs that each platform exposes.

The Platform Access FC depends on the specific details of implementation for each of the IoT Platforms supported.

The **Platform Service FC** is responsible for performing device and platform interactions, like querying data from different devices and platforms in a common way, mapping sensor data flows from a source to a destination, offering subscriptions to sensor data, etc.

4.1.3.4 Device Interoperability

The Device Interoperability FG addresses the challenges of making legacy devices and non-real IoT Platform interoperable with other IoT Platforms and systems.

It consists of three Functional Components (see Figure 88):

- Device to Device Interoperability;
- Network Interoperability;
- IoT Platform Interoperability.

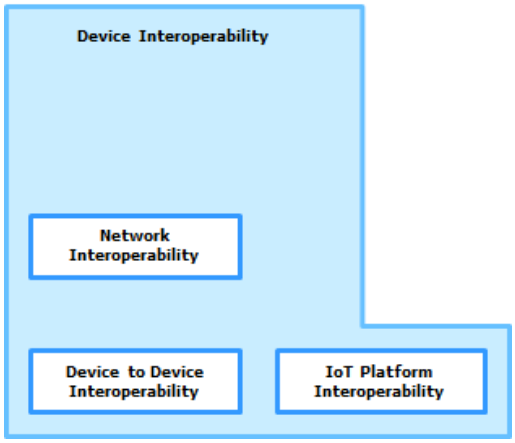


Figure 88: Device Interoperability

The three identified Functional Components deal only with interoperability related to devices.

The **Device to Device Interoperability FC** implements the needed functionalities to achieve the interoperability among devices which are available through the Device Access FG.

To enable this interoperability among devices rules are defined. These rules define the actions to be followed when some update is received from a device through the Device Access FG, like for instance to send a request to another device(s), also through the Device Access FG. An example of this would be to switch on a set of outdoor lights when a proximity sensor detects that someone is close.

The **Network Interoperability FC** is responsible for managing the interoperability between networks or parts of the network that belong to an IoT deployment, and which are accessible through the Device Access FG. We understand the network level of an IoT deployment as the protocols, systems, and devices that work on the layer 2 and 3 of the OSI stack of protocol. The particularity of the network on the IoT is the treatment of many different types of data flows as well as protocols to support this communication.

The Network Interoperability FC addresses the mobility of objects through different access networks or secure seamless mobility and the backing of real time data among the network. The operation in highly constrained environment is also an important issue. The interoperability solution is based on software defined paradigms but mainly on two approaches: SDR for interoperability on access network and SDN/NFV for the core network.

The role of the **IoT Platform Interoperability FC** is to enable the interaction between the devices available from the Device Access FG and the Platform Interoperability FG. Please, note that the devices available from the Device Access FG are not devices tied to existing IoT Platforms. The devices connected to an IoT Platform are accessed through the interaction between the Platform Interoperability FG and the IoT Platform FG, while the devices not tied to an IoT Platform (those connected to legacy sensor systems that cannot be considered as IoT Platform), are accessed through the Device Access FG.

The interaction between the IoT Platform Interoperability FC and the Platform Interoperability FG works in two ways:

- The IoT Platform Interoperability FC can act as a client of IoT Platforms, thus being responsible for interconnecting legacy or disparate devices into existing IoT Platforms. This is achieved through the appropriate device register and data retrieval/actuating functions, typical in these platforms.

- The IoT Platform Interoperability FC can also act as a kind of legacy IoT Platform from the point of view of Platform Interoperability FG. This may happen when there is no IoT Platform where to attach the devices, but there is a need from an external application to access these devices and, maybe, interoperating their data with information from other IoT Platforms. In this last case, the Platform Interoperability FG would interact with devices through the Device Interoperability FG.

4.1.3.5 Device Access

The Device Access FC embraces the functionality described in Communication, IoT Service and Virtual Entity FG's of the IoT-A Functional View. This Functional Component is responsible for offering a common interface to services and virtual entities that represent and expose functionality of physical devices.

It abstracts all the necessary functions for managing the devices and interacting with them.

The Device Access FC consists of three Functional Components (see Figure 89 below):

- Communication;
- IoT Service;
- Virtual Entity;

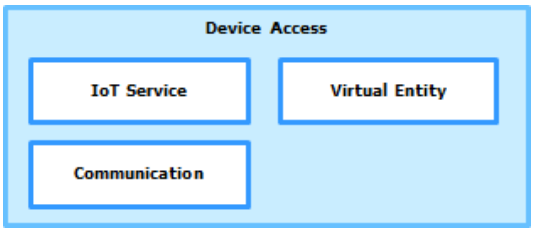


Figure 89: Device Access

The aim of the **Communication FC** is to perform the function of dealing with the devices through very different techniques, abstracting the IoT Service from the technical details of the communication with the devices. It provides a common interface to the IoT Service so that it can access or interact with very different devices in a common way.

The whole communication protocol stack under the transport layer must be handled by the Communication FG. This protocol stack management implies to address all the features related to the communication tasks (flow control, network access, protocol conversion, etc.). Therefore, it's responsibility of the Communication FG to manage the communication with the devices with two different aspects:

- Access network. Handling the access to the different communication networks that may appear to establish the contact with the devices (WiFi, LTE, Bluetooth Low Energy, Serial, etc.).
- Transport Protocol Management. Managing the necessary actions to provide end-to-end communication between devices and gateways, specifically supporting transport protocols like MQTT, CoAP, LWM2M, Raw, etc.

The Communication FG interacts with the Device FG on its southbound interface and with the IoT Service FC to provide it the interaction with devices in a common way and to keep updated the necessary functions of Service Resolution.

The **IoT Service FC** is responsible for managing IoT Services as well as functionalities for discovery, look-up, and name resolution of IoT Services. These services expose resources of devices to the rest of the components. It may allow to gather information about a sensor in a continuous asynchronous way, after a subscription, for instance. Or it may allow to submit requests to an actuator. A specific IoT Service could be to provide access to recent history of sensor observations.

The typical functions of the IoT Service FC are two:

- To access resources, interacting in three different ways: (1) to query information about a resource of a device, e.g. get current temperature of thermometer X, (2) to subscribe to observations about a resource of a device and receive notifications asynchronous for each new observation, e.g. receive all temperature measurement under 0ºC for thermometer X, (3) to submit a request to a resource of an actuator, e.g. switch light actuator Y on.
- To provide the necessary functions for finding the appropriate IoT Services, which may include: discovery, lookup, service locators, service management, etc.

The IoT Service runs in the virtual plane, decoupling the interaction with the resources of devices from their usage.

The **Virtual Entity FC** allows the interaction with an IoT Platform on the basis of Virtual Entities rather than IoT Services. It contains the functions to associate the Virtual Entities with the IoT Services and with the physical things they represent.

The typical functions of the Virtual Entity FC are:

- Discovery and lookup functions to find VE's and their resources and register of new ones.

Handling VE's, which includes getting the values of the entities' attributes, updating this data, and accessing its recent history.

4.1.4 Interactions of the Functional View

To better understand the Reference Architecture from a functional point of view, it's recommended to describe the communication among the Functional Components for some relevant interactions. We have gathered the most relevant use cases from the Deliverables D2.4 INTER-IoT Requirements and Business Analysis, and D2.3 (Use cases manual).

The different interactions are described below:

4.1.4.1 Subscription merged data flows from two IoT Platforms to an external user

This interaction can appear when an external user uses INTER-IoT to receive a continuous single data flow with information from different sensors placed at different IoT Platforms: IoT Platform 1 and IoT Platform 2.

This interaction assumes that the external user has previously established the subscription to the two IoT Platforms 1 and 2.

The interaction among the different components is depicted in the diagram below:

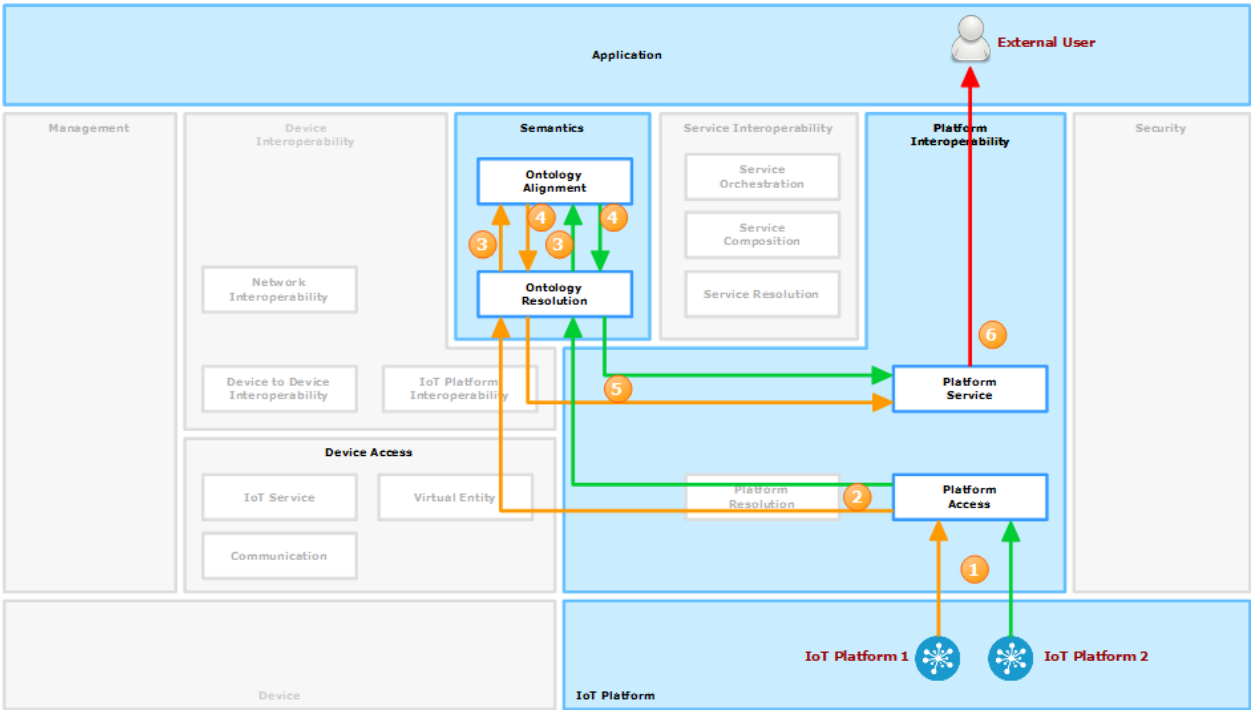


Figure 90: Functional View interaction for subscription to 2 IoT Platforms

In the diagram, it can be observed that each of the IoT Platforms 1 & 2 notify the Platform Access FC that a new sensor observation is available as the Platform Access FC is subscribed to both IoT Platforms.

1. Just after each notification, the Platform Access FC receives the data and sends it to the Ontology Resolution FC with an indication of the specific platform so that the Ontology Resolution FC can find the syntactic and semantic description of the input data for each one of the observations from each IoT Platform.
2. In the subscription that the user previously made, a desired output ontology may have been requested. This ontology is now being resolved by the Ontology Resolution FC.
3. The pair <input ontology, output ontology> along with the observation data from each IoT Platform is submitted to the Ontology Alignment FC.
4. The Ontology Alignment FC then performs the semantic translation into the expected output ontology.
5. The translated data from each IoT Platform now translated into the expected output ontology is sent to the Platform Service FC.
6. The Platform Service FC has a register of the subscriptions that external users have made, and can identify that the incoming data has to merged into a single data flow that is pushed in form of notifications to the external user.

4.1.4.2 Device to Device Interoperability

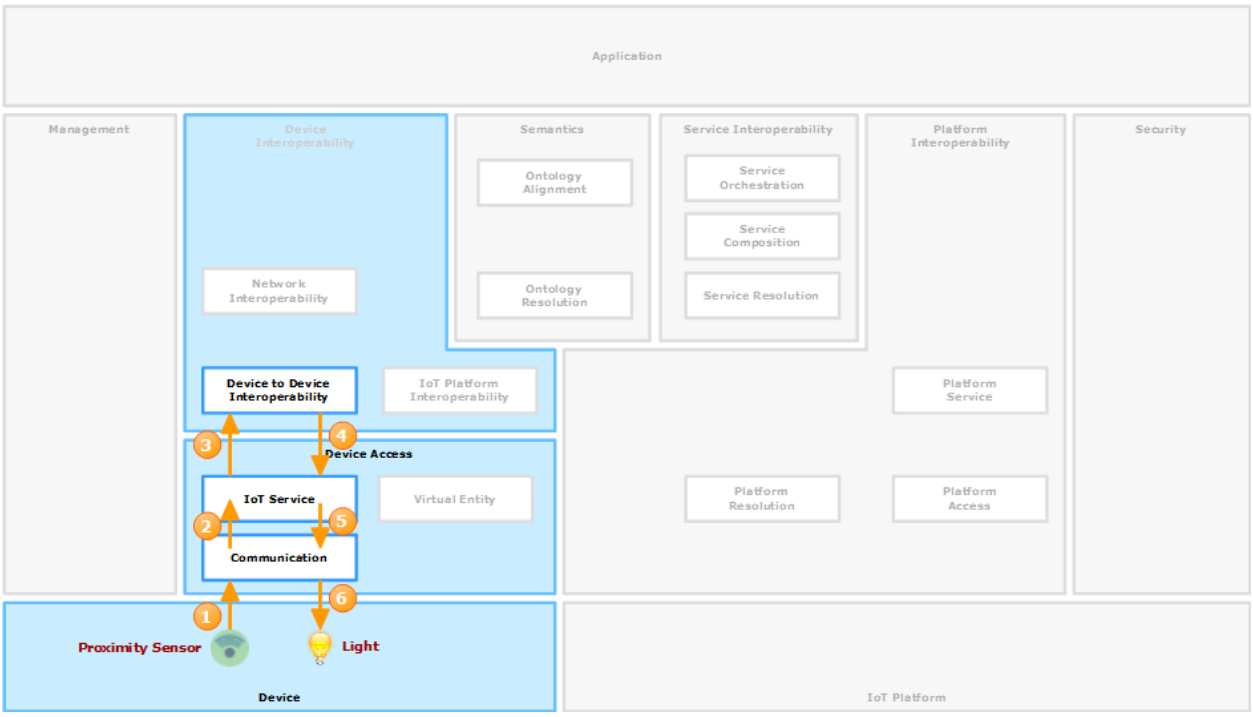


Figure 91: Functional View interaction of device to device interoperability

In the previous diagram the following FCs take part:

1. A device pushes data using the communication features.
2. An IoT Service is attached to the Communication FC so it is fed with the data produced by the device.
3. An IoT Service is provided, so that data pushed can be consumed by upper layers in a IoT standardized way.
4. The service publishes data to the Device to Device Interoperability FC, which manages the routing (source and destination) of the data, and also is able to unify/adapt formats between technologies.
5. Data is pushed into destination device(light) IoT service.
6. The communication FC converts data to the protocol of the destination device and forwards the message.

4.1.4.3 Service Composition

For describing the service composition interaction, we are going to use the use case #13: IoT interoperability for Vessel Arrivals of D2.4. The reason of selecting this case is that involves two different platforms with different ontologies, so that all the FCs have a role and need to interact. The main idea around this use case is as follows:

> A CEP Service within ValenciaPort IoT Platform can send an event/raise an alert when a vessel has berthed to a dock. When a berthing has been detected, we need to search for the nearest cranes in the terminal and instantiate a Web Service in the terminal ICT infrastructure.

In this example, we will use a Platform Service from ValenciaPort (CEP), a search service of Noatum IoT Platform and an external Web Service from Noatum for orchestrating business processes.

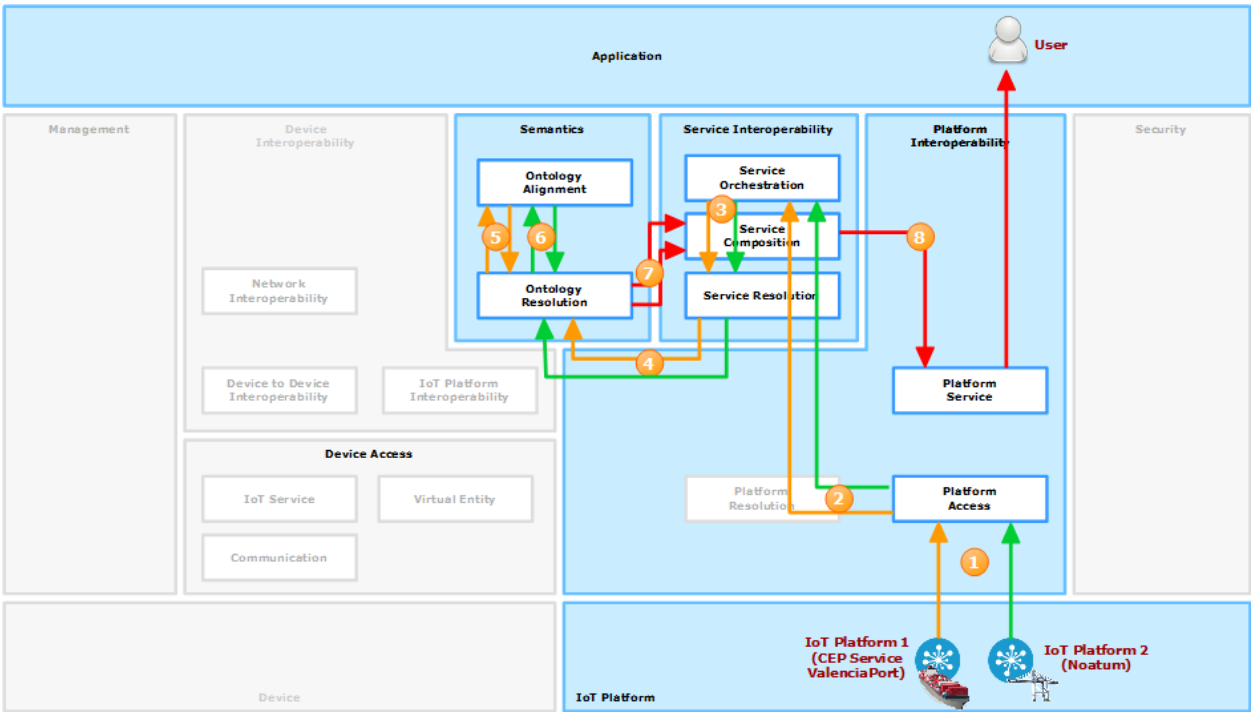


Figure 92: Functional View interaction of service composition (service to service interoperability)

Former diagram contains the following interactions between FCs:

1. CEP service in ValenciaPort and available crane search service are connected to the platform access, so that information can be pushed and accessed in that platforms.
2. Service orchestration FC is able to access native services of the platforms. Native in this context means that these services consume platform-specific APIs and communicate data exclusively in the platform domain with its platform ontology.
3. Service resolution FC consumes discover services and provide the means to effectively consume them.
4. As the consumed services provide data in native ontologies, ontology resolution is required.
5. Ontology alignment works to put the data in the expected output format.
6. Same as in 5.
7. Ontology resolution is used to provide aligned data to the service composition FC.
8. Data is composed (appropriate operations are performed) and send to user.
9. User destination is resolved in platform service FC.

4.2 Other views

This section summarizes other views that are more related to the instantiation of the architecture and thus will be reported in future documents. This, an initial introduction is given and, when possible, are applied the works (such as requirement specification or scenario definition) performed in the project.

4.2.1 Information View

The main reason about connecting objects is to allow an information exchange between each object and external systems, and within each other. Therefore, the way to define, structure, store, process, manage and exchange information is fundamental in this domain. IoT-A created a specific view (the information view) in order to specify a static information structure and a dynamic information flow.

Based on the IoT Information Model, the Information View gives more details about how the relevant information is represented in an IoT system. As the Information View belongs to the reference architecture space, and not a specific system architecture, concrete representation alternatives are not part of this view.

The information view also describes the components that handle the information, the flow of information through the system and the life cycle of information in the system.

As described earlier, the Virtual Entity is a key concept of any IoT system as it models the Physical Entity that is the real element of interest. As specified in the IoT IM, Virtual Entities have an identifier (ID), an EntityType and a number of attributes that provide information about the entity or can be used for changing the state of the Virtual Entity, triggering an actuation on the modelled Physical Entity. The modelling of the EntityType is of special importance, as it can be used to determine what attributes a Virtual Entity instance can have, defining its semantics. The EntityType can be modelled in two different ways: either based on a flat type system or as a type hierarchy, enabling sub-type matching.

EntityTypes are similar to classes in object-oriented programming, so UML class diagrams are suitable for modelling EntityTypes. Similarly, the generalization relation can be used for modelling sub-classes of EntiyTypes, creating a hierarchy of several EntityTypes inheriting attributes from its super-classes.

Services provide access to functions for retrieving information or executing actuation tasks on IoT Devices. Service Descriptions contain information about Service interfaces, both on a syntactic as well as a semantic level. Furthermore, the Service Description may include information regarding the functionality of the resources, or information regarding the device on which the resource is running.

The association between Virtual Entities and Services captures the information on what kind of actuation or data is possible to obtain by which Virtual Entity. The association includes the attribute of the Virtual Entity for which the Service provides the information or enables the actuation as a result of a change in its value.

Information in the system is handled by IoT Services. IoT Services may provide access to On-Device Resources, that provide real-time information about the physical world accessible to the system. Other IoT Services may further process and aggregate the information provided by IoT Services/Resources, deriving additional higher-level information. Furthermore, information that has been gathered by the mentioned IoT Services or has been added directly by a user of the IoT system can be stored by a special class of IoT Service, the history storage. A history storage may exist on the level of data values directly gathered from sensor resources as a resource history storage or as a history storage providing information about a Virtual Entity as a Virtual Entity history storage.

4.2.1.1 Information Handling by Functional Components

There are four message exchanges patterns considered for information exchange between IoT Functional Components: Push, Request/Response, Subscribe/Notify, Publish/Subscribe.

The Push-pattern is a one-way communication between two parties in which a server sends data to a predefined client that receives the data. The server hereby knows the address of the client beforehand and the client is constantly awaiting messages from the server. The communication channel in this pattern is pre-defined and meant to be applied in scenarios in which the communication partners do not changed often. For example, the server can be a constrained device that sends data to a gateway dedicated to this device. The gateway is listening constantly to the device and is consuming the data received from this device.

The Request/Response-pattern is a synchronous way of communication between two parties. A client sends a request to a server. The server will receive the request and will send a response back to the client. The client is waiting for the response until the server has sent it.

The Subscribe/Notify-pattern allows an asynchronous way of communication between two parties without the client waiting for the server response. The client just indicates the interest in a service on the server by sending a subscribe-call to the server. The server stores the subscription together with the address of the client wants to get notified on and sends notifications to this address whenever they are ready to be sent.

The Publish/Subscribe-pattern allows a loose coupling between communication partners. There are services offering information and advertise those offers on a broker component. When clients declare their interest in certain information on the broker the component will make sure the information flow between service and client will be established.

4.2.1.2 Deployment and Operation View

The Deployment and Operation View aims at developing a set of guidelines to drive users through the different design choices that they must face while designing the actual implementation of their services. To this extent this view will discuss how to move from the service description and the identification of the different functional elements to the selection among the many available technologies in the IoT to build up the overall networking behaviour for the deployment.

Since a complete analysis of all the technological possibilities and their combination may be extremely complex, IoT-A focus is on those categories that have the strongest impact on IoT systems realization. Starting from the IoT Domain Model, there are three main element groups: Devices, Resources, and Services. Each of them poses a different deployment problem, which, in turn, reflects on the operational capabilities of the system.

In particular, the viewpoints used in the Deployment and Operation view are the following:

- The IoT Domain Model diagram is used as a guideline to describe the specific application domain;
- The Functional Model is used as a reference to the system definition, as it defines Functional Groups;
- Network connectivity diagrams can be used to plan the connectivity topology to enable the desired networking capability of the target application; at the deployment level, the connectivity diagram will be used to define the hierarchies and the type of the sub-networks composing the complete system network;
- Device Descriptions (such as datasheets and user manuals) can be used to map actual hardware on the service and resource requirements of the target system.

Devices in IoT systems include the whole spectrum of technologies ranging from the simplest of the radiofrequency tags to the smartest objects able to understand the environment and take real-time decisions. The unifying characteristics are the connection and the capability of performing computation. These two characteristics are the subject of the first choices a system designer has to make.

Selecting the computational complexity for a given device is intrinsic to the target application and to the planned roadmap: for instance, a system architect may choose to have a large amount of memory that may seem unnecessary at first, but may be used for future releases and upgrades. On the other hand, choosing among the different connectivity types is not as straightforward as different choices may provide comparable advantages, but in different areas. For the same reason, it is possible to realize different systems implementing the same or similar application from the functional view, which are extremely different from the Deployment and Operation view.

Because of the coexistence of different communication technologies in the same system, the second choice the system designer must account for is related to communication protocols. Connectivity functionalities for IoT system are defined within the ARM in the Communication FG of the FM; in addition, to better understand the application, it is important to describe it within the Functional View. The following possibilities have been identified:

1. **IoT protocol suite**: This is supposed to be the best solution for interoperability;
2. **Ad-hoc proprietary solutions**: Whenever the performance requirements of the target application are more important than the system versatility, ad hoc solutions may be the only way to go;
3. **Other standards**: Depending on the target application domain, regulations may exist forcing the system designer to adopt standards, different from those suggested by the IoT protocol suite, that solved a given past issue and have been maintained for continuity.

After having selected the devices and their communication methods, the system designer has to account for services and resources, as defined in the IoT Service FG section. These are pieces of software that range from simple binary application and increasing their complexity up to full-blown control software. Both in the case of resources and for services the key point here is to choose where to deploy the software related to a given device. The options are as follows:

1. **On smart objects**: This choice applies to simple resource definitions and lightweight services, such as web-services that may be realized in few tens or hundreds of bytes;
2. **On gateways**: Whenever the target devices are not powerful enough to run the needed software themselves, gateways or other more capable devices have to be deployed to assist the less capable ones;
3. **In the cloud**: Software can be also deployed on web-farms. This solution improves the availability of the services, but may decrease the performance in terms of latency and throughput.

Note that this choice must be made per type of resource and service and depending on the related device. As an example, a temperature sensor can be deployed on a wireless constrained device, which can host the temperature resource with a simple service for providing it, but, if a more complex service (for instance, when the Service Organisation FG is called in) is needed, the software should be deployed on a more powerful device as per option 2) or 3).

On the same line, it is important to select where to store the information collected by the system, let their data be gathered by sensor networks or through additional information provided by users. In such a choice, a designer must take into consideration the sensitiveness (e.g.: is the device capable

of running the security framework), the needed data availability and the degree of redundancy needed for data resiliency. This choice is also very important for what concerns interoperability, as the location of the data may ease the interaction between different systems – or, at the contrary, may prove very complex to overcome. The foreseen options are the following:

1. **Local only**: Data is stored on the device that produced it, only. In such a case, the locality of data is enforced and the system does not require complex distributed databases, but, depending on the location of a given request, the response might take longer time to be delivered and, in the worst-case scenario, it may get lost;
2. **Web only**: No local copy is maintained by devices. As soon as data is sent to the aggregator, they are dispatched in databases;
3. **Local with web cache**: A hierarchical structure for storing data is maintained from devices up to database servers.

Finally, one of the core features of IoT systems is the resolution of services and entities, which is provided by the Entity and Service Resolution FCs, respectively and oversees semantically retrieving resources and services, discovering new elements and binding users with data, resources, and services. This is performed adopting the definitions of the Virtual Entity FG. This choice, while one of the most important for the designer, has only two options:

1. **Internal deployment**: The core engine is installed on servers belonging to the system and is dedicated to the target application or shared between different applications of the same provider;
2. **External usage**: The core engine is provided by a third party and the system designer has to drive the service development on the third-party APIs.

Differently from the other choices, this is driven by the cost associated to the maintenance of the core engine software. In fact, since it is a critical component of the system, security, availability and robustness must be enforced. Hence, for small enterprises the most feasible solution is the external one.

4.3 IoT Architecture Perspective: Non-Functional Properties

Architectural decisions often address concerns that are common to more than one view, or even all of them. These concerns are often related to non-functional or quality properties. In this respect, IoT-A follows the approach of [3]: these aspects need to be addressed by special perspectives, to build a concrete architecture. One important aspects are that these perspectives help to introduce Stakeholders requirements in the architectural building process. IoT-A uses the [3] definition for perspectives:

> *An architectural perspective is a collection of activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the system's architectural view*s.

where a quality property is defined as:

> *A quality property is an externally visible, non-functional property of a system such as performance, security, or scalability*.

The requirements we collected in D2.3 clearly show a need of addressing non-functional requirements. Based on them, we identified the perspectives, which are the most important for IoT-systems:

- Evolution and Interoperability;
- Availability and Resilience;
- Trust, Security and Privacy and
- Performance and Scalability.

In INTER-IoT we collected 4 non-functional requirements related to INTER-FW concerning the *Evolution and Interoperability* perspective, 12 concerning *Availability and Resilience*, 7 related to *Trust, Security and Privacy*, and 5 related to *Performance and Scalability*. As can be seen from the definition above there is a close relationship between Perspectives, Views and Guidance. Using the table template that IoT-A proposes, we can derive the following tables:

Evolution and Interoperability:

Desired Quality	The ability of the system to be flexible in the face of the inevitable change that all systems experience after deployment, balanced against the costs of providing such flexibility
INTER-IoT Requirements	1,33,34,35
Applicability	Important for all systems to some extent; more important for longer-lived and more widely used systems. IoT systems are expected, as an emerging technology, to be highly affected by evolution and interoperability issues
Activities	• Characterize the evolution needs • Assess the current ease of evolution • Consider the evolution trade-offs • Rework the architecture
Tactics	• Contain change • Create extensible interfaces • Apply design techniques that facilitate change • Apply meta-model-based architectural styles • Build variation points into the software • Use standard extension points • Achieve reliable change • Preserve development environments

Performance and Scalability:

Desired Quality	The ability of the system to predictably execute within its mandated performance profile and to handle increased processing volumes in the future if required
INTER-IoT Requirements	3,132,114,115,142
Applicability	Any system with complex, unclear, or ambitious performance requirements; systems whose architecture includes elements whose performance is unknown; and systems where future expansion is likely to be significant. IoT systems are very likely to have unclear performance characteristics, due to their heterogeneity and high connectivity of devices.
Activities	• Capture the performance requirements • Create the performance models • Analyze the performance model • Conduct practical testing • Assess against the requirements • Rework the architecture
Tactics	• Optimize repeated processing • Reduce contention via replication • Prioritize processing • Consolidate related workload • Distribute processing over time • Minimize the use of shared resources • Reuse resources and results • Partition and parallelize • Scale up or scale out • Degrade gracefully • Use asynchronous processing • Relax transactional consistency • Make design compromises

Table 5: Interoperability requirements

Availability and Resilience:

Desired Quality	The ability of the system to be fully or partly operational as and when required and to effectively handle failures that could affect system availability.
INTER-IoT Requirements	10, 44, 58, 83, 134, 92, 109, 110, 113, 119, 120, 126

Applicability	Any system that has complex or extended availability requirements, complex recovery processes, or a high profile (e.g., is visible to the public)
Activities	• Capture the availability requirements • Produce the availability schedule • Estimate platform availability • Estimate functional availability • Assess against the requirements • Rework the architecture
Tactics	• Select fault-tolerant hardware • Use high-availability clustering and load balancing • Log transactions • Apply software availability solutions • Select or create fault-tolerant software • Design for failure • Allow for component replication • Relax transactional consistency • Identify backup and disaster recovery solution

Figure 93 Availability and resilience requirements

Trust, Security and Privacy:

Desired Quality	Ability of the system to enforce the intended confidentiality, integrity and service access policies and to detect and recover from failure in these security mechanisms, and to be able to build dependability.
INTER-IoT Requirements	30, 36, 37, 69,77, 104, 117
Applicability	Relevant to all IoT systems.
Activities	• Capture the privacy requirements • Conduct risk analysis • Evaluate compliance with existing privacy frameworks. • Capture the security requirements • Check interoperability requirements for impacts on security processes between heterogeneous peers • Conduct risk analysis • Use infrastructural Authentication components that support more Identity Frameworks for scalability and interoperability • Use infrastructural or federated Key Exchange Management to secure communication initiation and tunnelling between gateways for interoperability

	• Use an Authorization component to enable interoperability with other systems • Define security impact on interaction model • Address all aspects of Service and Communication Security • Integrate the trust model and support privacy features • Identify security hardware requirements • Consider performance/security trade-offs • Validate against requirements
Tactics	• Use an extended Internet Threat Model for which takes into account specific IoT communication vulnerabilities • Harden infrastructural functional components Authenticate subjects • Define and enforce access policies • Secure communication infrastructure (gateways, infrastructure services) • Secure communication between subjects • Secure peripheral networks (data link layer security, network entry, secure routing, mobility and handover) • Avoid wherever possible wireless communication • Physically protect peripheral devices or consider peripheral devices as available to malicious users in the attacker model • Use an Identity Management component that supports Pseudonymization • Avoid transmitting identifiers in clear especially over wireless connections • Minimize unauthorized access to implicit information (e.g. deriving location information from service access requests) • Validate against requirements • Consider the impact of security/performance trade-offs on privacy • Enable the user to control the privacy (and thus security and trust) settings • Balance privacy vs. non-repudiation (accountability)

Figure 94 Trust, security and privacy requirements

5 Relationship with INTER-IoT Architecture

5.1 Introduction

In the previous chapter, we have described a generic ARM (Architecture Reference Model) for the interoperability of IoT Platforms. This was one of the objectives of the INTER-IoT project (see Objective 2).

We consider this, a novel approach and a high valuable output of the project. Nevertheless, an ARM is a generic model, and we need a concrete model for the design of the architecture of the different components of INTER-IoT. This design has been performed in D3.1 using the ongoing work done in this deliverable.

Thus, an architecture has been designed for the different interoperability layers of the INTER-LAYER, using the INTER-IoT ARM as an input. These architectures for the different layers has gone through an iterative process along with the INTER-IoT ARM as described in the Figure 95 below:

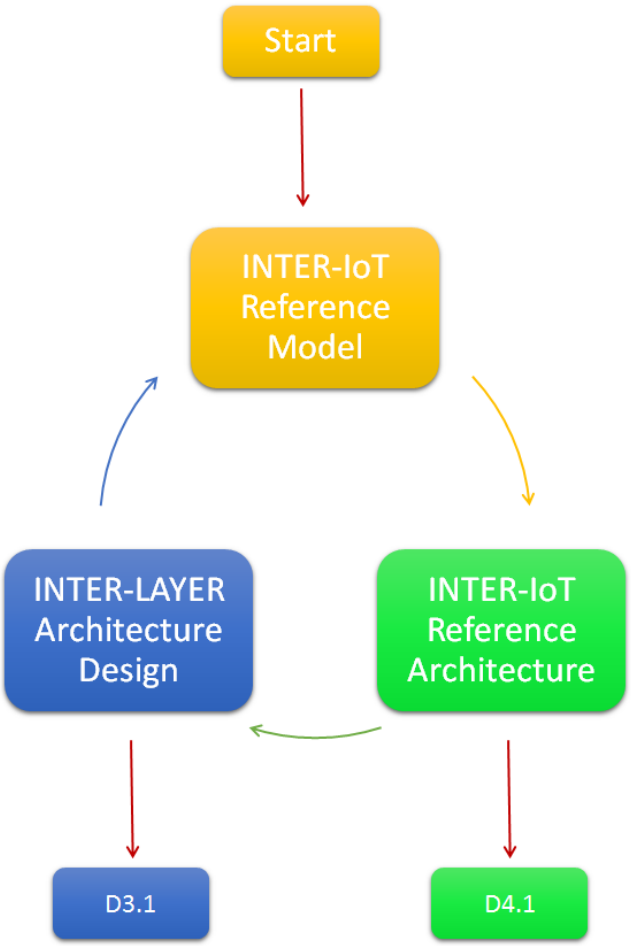


Figure 95: Process for generating D3.1 and D4.1

For both deliverables, the review of requirements, scenarios and use cases has been the main inputs, as it's described in both. The starting point for the process of generating an architecture for

the INTER-LAYER has been the definition of a Reference Model (RM) for INTER-IoT. Based upon it, the Reference Architecture (RA) has been created, and this has been used as an input for the INTER-LAYER Architecture Design.

This Architecture Design has identified some needs and discrepancies with the RM and RA models, initiating an iterative process, that will run until the end of the project as needed. As a matter of fact, the second versions of both deliverables (D3.2 and D4.2) are expected to reflect this, along ship with the INTER-FW design (D4.3), which will fill-in some missing components of the RA, like those dealing with aspects of security and management.

In order to show the matching of the INTER-IoT RA with the architecture design of INTER-LAYER, a mapping of the INTER-IoT Functional View with the different layers of INTER_LAYER has been performed.

This mapping is described in two phases. First, an analysis of the compliance of the RA against the DoW has been done to prove its validity, and assess the achievements of the project. Next, an exact mapping of the different Functional Components of the INTER-IoT RA with the main design components of the INTER-LAYER has been sketched to show the instantiation of the INTER-IoT RA to the INTER-LAYER.

It's important to notice that the analysis of the relationship of some models of the INTER-IoT Reference Model with the INTER-LAYER interoperability layers has already been done (see sections 3.5.3 INTER-IoT Domain Model Element Communications and 3.5.4 INTER-IoT Channel Model for Interoperability).

This instantiation and alignment of the results will be improved in D4.2 with more views and new added changes.

5.2 Mapping of the Functional View with the multi-layered interoperability approach

A way to **assess the validity** of the Functional View is to try to match it against the Description of Work. To evaluate this assessment, we are first going to extract some relevant excerpts of the INTER-IoT proposal and next we will map the Functional View against it:

1. In the overall goal description of INTER-IoT it's stated that:

> *INTER-IoT uses a layer-oriented approach.*

2. The research & innovation objective 1 ("*Design and Implementation of an Open Cross-Layer Framework for Interoperability of IoT Platforms*") states this:

> *By using the INTER-FW, any IoT platform can be made interoperable with respect to its fundamental layers: device, networking, middleware, application service, and data/semantics.*

3. The research & innovation objective 2 ("*Definition of Techniques and Tools for interoperability at the different IoT Platform Layers*") states this:

> *Layer (and cross-layer) interoperability is fundamental to provide global interoperability between IoT platforms.*

4. The INTER-IoT approach (see Figure 96 below) is described as:

> *The INTER-IoT approach will be fundamentally based on three main building blocks:*
>
> *1. Methods and tools for providing interoperability among and across each layers of IoT platforms;*
>
> *2. Global framework (INTER-FW) for programming and managing interoperable IoT platforms,including INTER-API and several interoperability tools for every layer;*
>
> *3. Engineering Methodology based on CASE tool for IoT platforms integration/interconnection.*

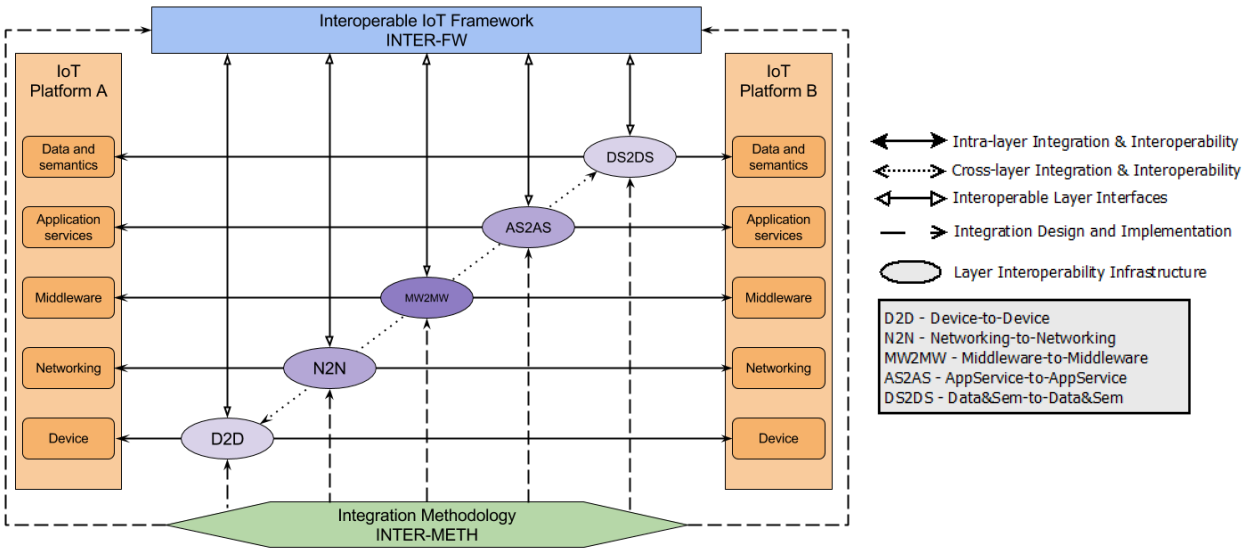


Figure 96: INTER-IoT approach abstract schema

We have matched the different interoperability layers identified in the proposal with the different Functional Groups and Functional Components of the INTER-IoT functional View. The mapping of the Functional View of INTER-IoT against the interoperability layers, is depicted in the figures below.

We have represented filled in light blue colour the Functional Groups involved in the related interoperability layer, and in blue border the specific Functional Components that implement the necessary functions to provide the expected interoperability layer.

Device-to-Device (D2D)

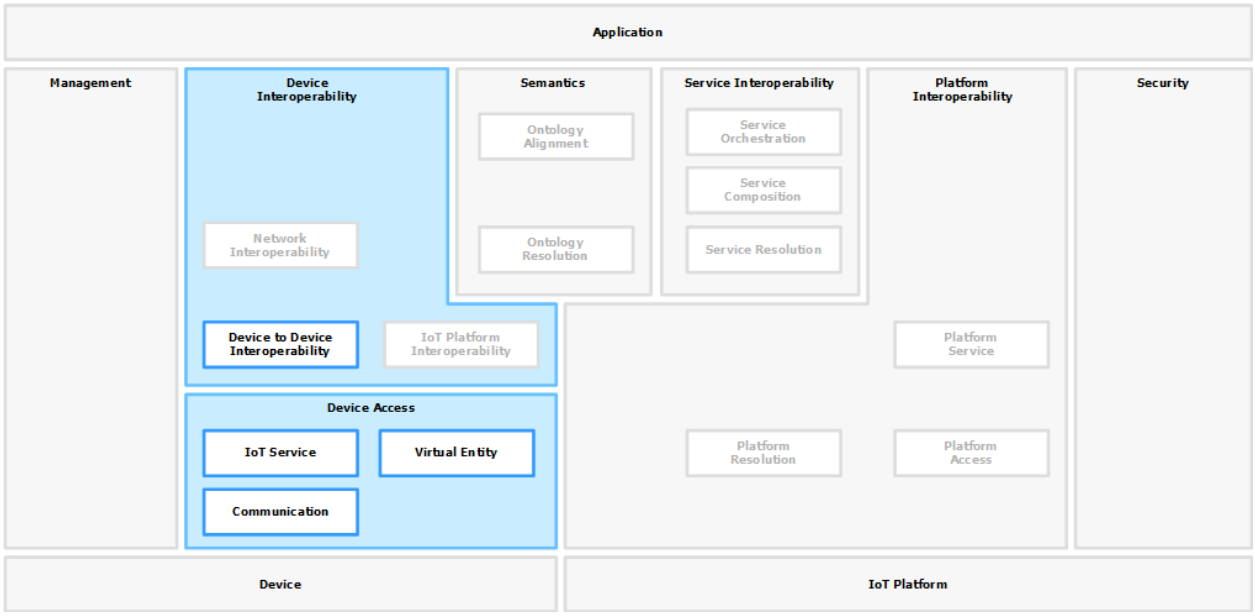


Figure 97: Mapping the functional View with the Device-to-Device Interoperability

Network-to-Network (N2N)

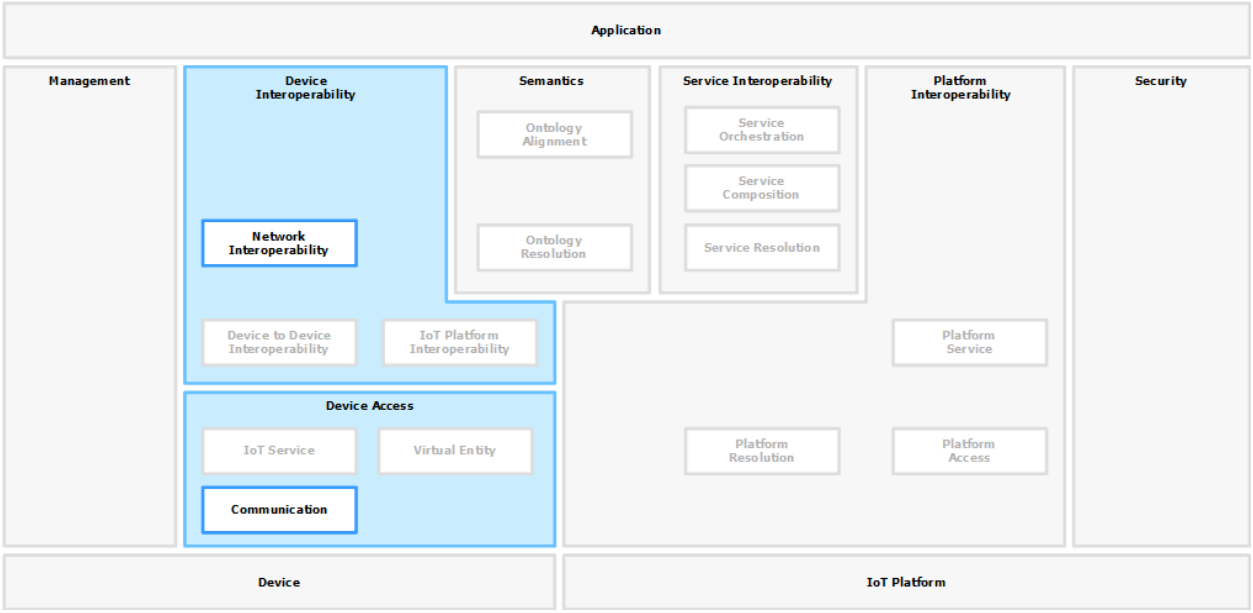


Figure 98: Mapping the Functional View with the Network-to-Network Interoperability

We have included the Communication FC from the Device Access FG because the Network Interoperability is tightly related with communications aspects.

Middleware-to-Middleware (MW2MW)

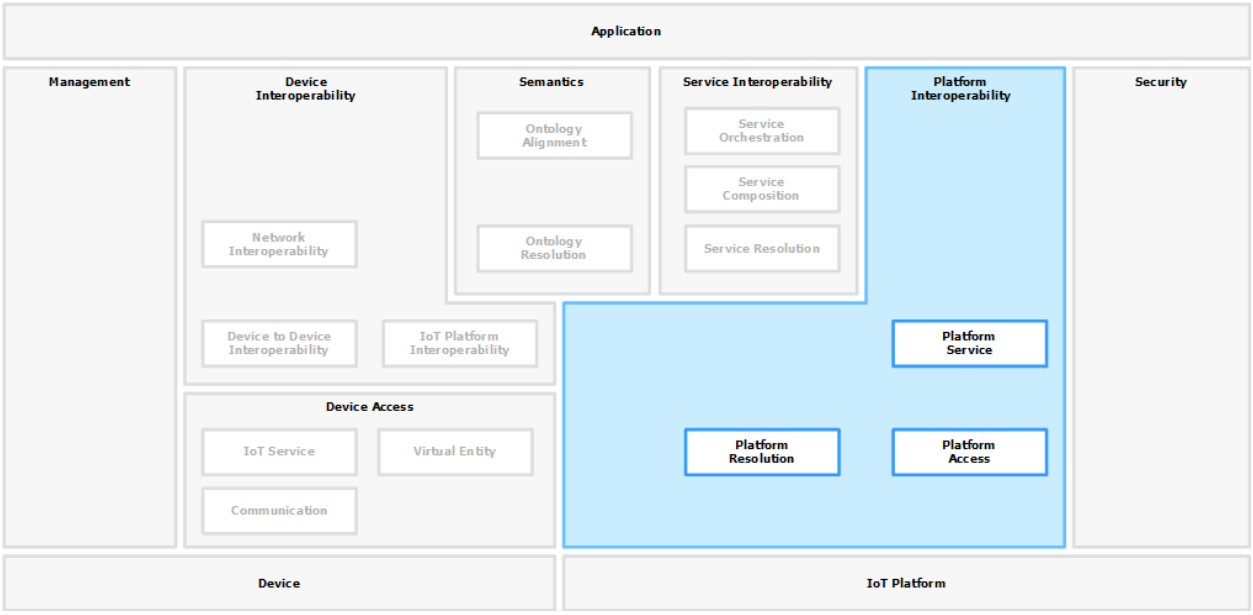


Figure 99: Mapping the Functional View with the Middleware-to-Middleware Interoperability

Application Service-to Application Service (AS2AS)

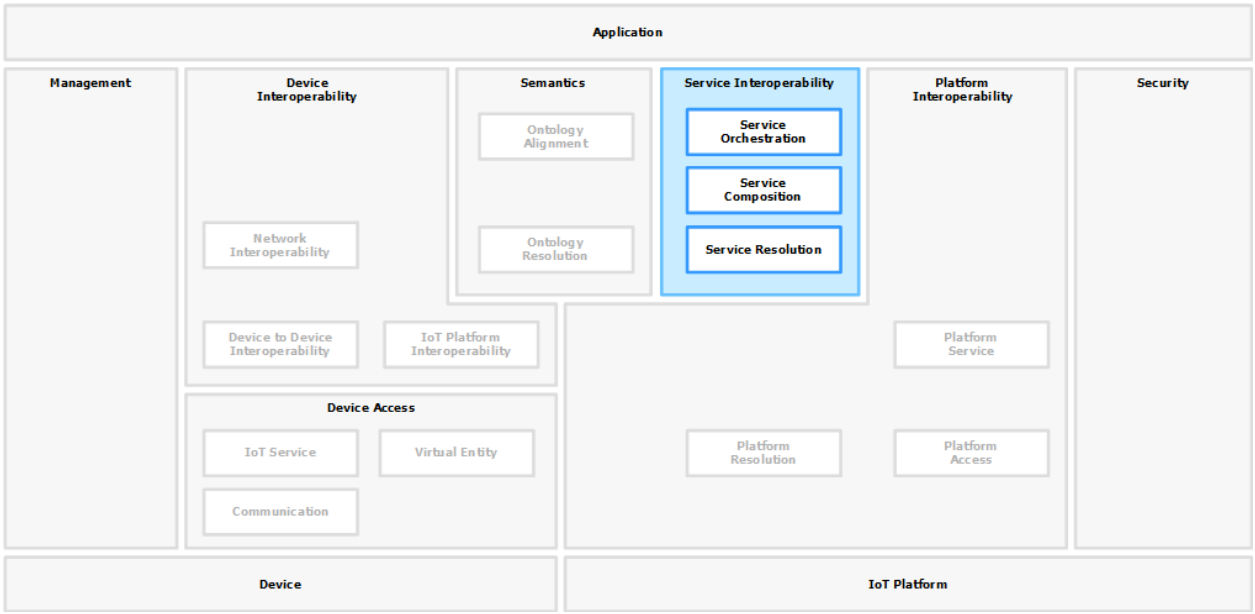


Figure 100: Mapping the Functional View with the Application Service-to Application Service Interoperability

Data&Semantics-to-Data&Semantics

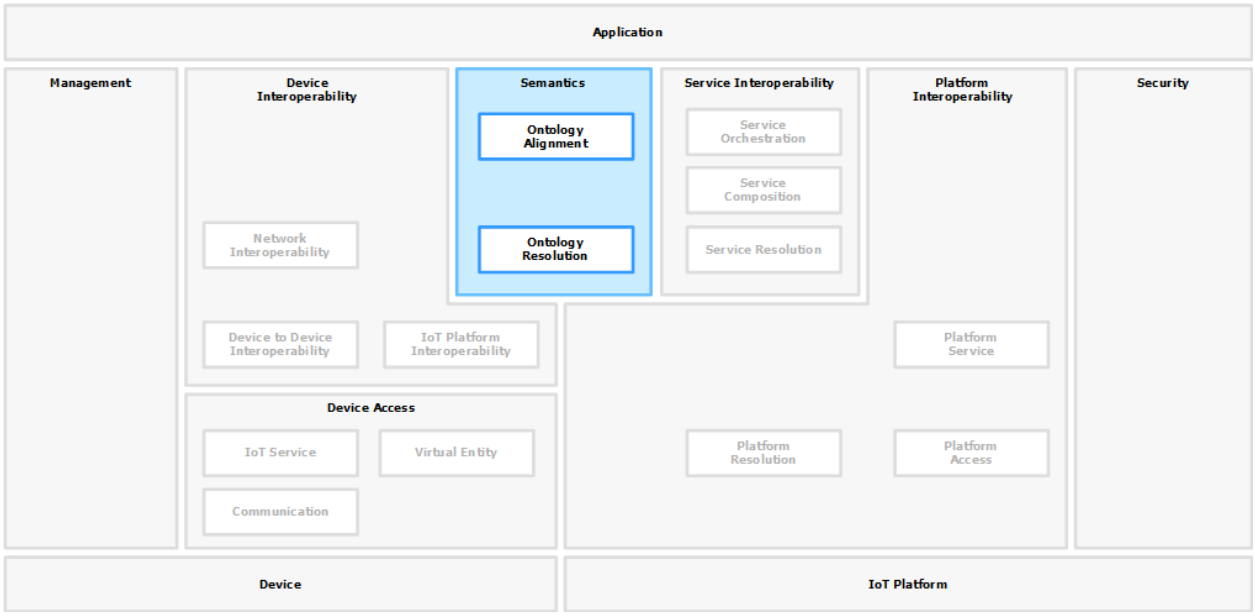


Figure 101: Mapping the Functional View with the Data&Semantics-to-Data&Semantics Interoperability

As we can see, all the proposed interoperability layers are covered in the INTER-IoT Functional View.

5.3 Instantiation of the INTER-IoT RA to INTER-LAYER

As described in section 5.1, an instantiation of the INTER-IoT Reference Architecture to the different layers of the INTER-LAYER has been done. The result is widely described in D3.1. Repeating the description is not in the scope of this deliverable, however, an exact mapping of the generic Functional Components of the INTER-IoT RA with the main design components of INTER-LAYER has been sketched.

This relationship will also help the reader to understand the real difference between the generic INTER-IoT Reference Architecture, which can be applicable to any solution willing to interoperate different IoT Platforms, and the concrete instantiation made for the INTER-LAYER.

5.3.1 Device to Device Interoperability

The mapping of the D2D Interoperability with the INTER-IoT Functional View is described in two steps to help the understanding of a wide set of components. First a mapping of the Functional Groups of the INTER-IoT Reference Model with the main modules of the D2D Interoperability Architecture has been sketched, using coloured lines to show the relationships. Next, a detailed relationship including Functional Components of the INTER-IoT RA is described.

The D2D Interoperability Architecture corresponds to the also called Gateway Architecture, described in section 3.1 of the D3.1. We have rearranged a bit the Gateway Architecture to make it more simple to see the mappings through the lines with minimal crossings. We have also changed some colours, and we have removed the interaction arrows, as they don't add value to the functional

mapping. As a result, the Gateway Architecture has been flipped horizontally, keeping all the components.

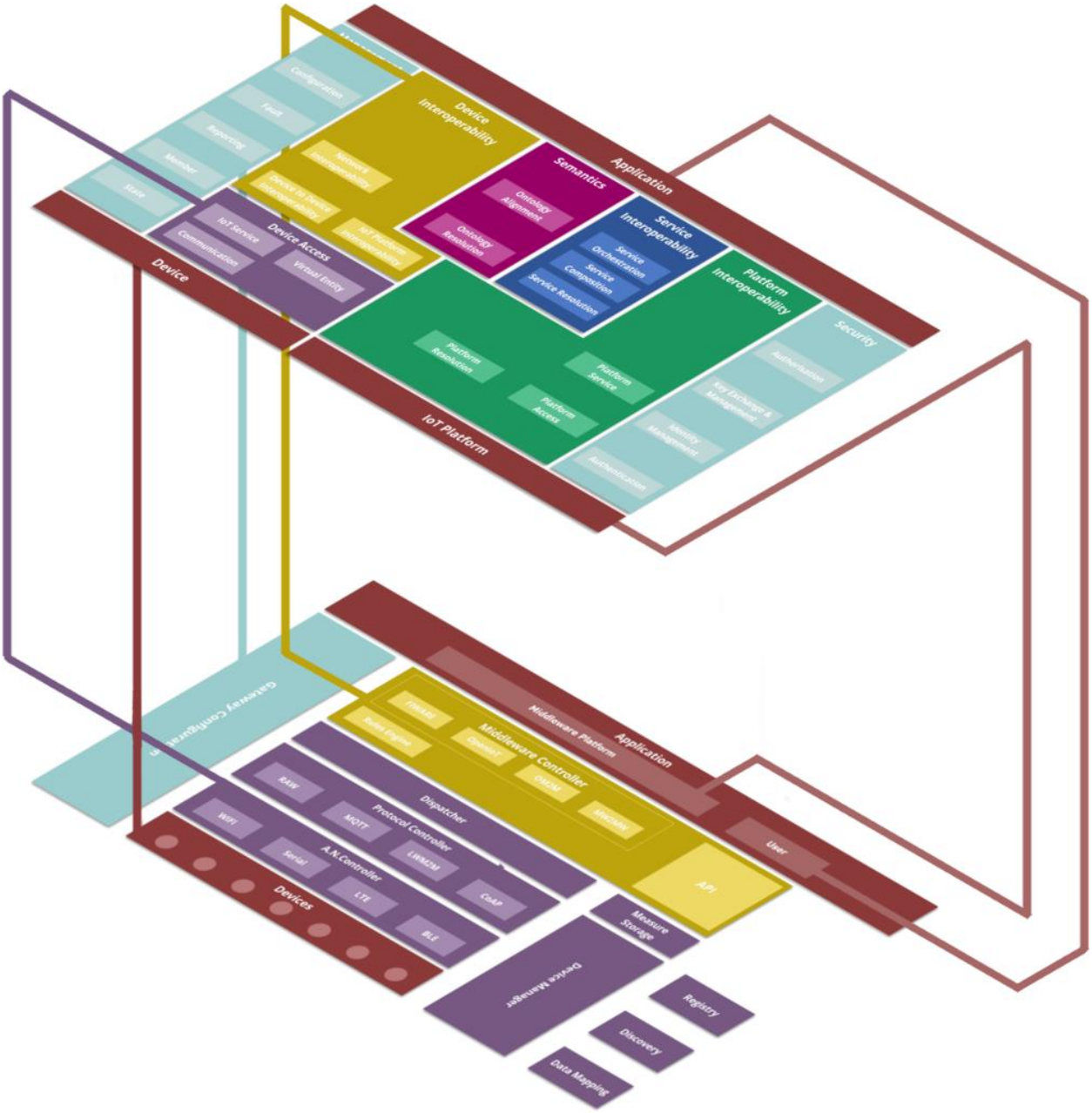


Figure 102: Alignment of the INTER-IoT Functional Groups with the D2D Interoperability layer of the INTER-LAYER.

Applications	▶▶▶	Application
Device Interoperability	▶▶▶	Middleware Controller
Device Access	▶▶▶	A.N. Controller
Devices	▶▶▶	Devices

Table 6 Alignment of INTER-IoT FGs and D2D Layer Ineroperabilty Infrastructure

As described in the Figure 97: Mapping the functional View with the Device-to-Device Interoperability, the D2D Interoperability uses two Functional Groups:

- The Device Access FG.
- The Device Interoperability FG.

Apart from this non-transversal Functional Groups of the Reference Model, some other transversal or generic Functional Groups have been instantiated for the Device-to-Device Interoperability:

- The Device FG.
- The Application FG.
- The Management FG.

The **Device Access** FG has been exploited into a set of components that are represented in purple colour. This is the FG with more functionality to be implemented for the D2D Interoperability, so a wide set of components were expected. The detailed description of this instantiation is described below after Figure 103: Mapping of the Functional Components of the INTER-IoT RA with the components of the D2D interoperability layer of the INTER-LAYER.Figure 103, in a detailed figure with the relationship of the specific Functional Components of each Functional Group.

The **Device Interoperability** FG has produced the yellow components of the D2D Interoperability Architecture, which are also described in the detailed figure.

The **Device** FG corresponds directly to the Device component of the D2D Interoperability Architecture, where the devices are located. It's really an external module, not part of the D2D Interoperability Architecture, but that interacts with the Access Network Controller, as is described in D3.1.

The **Application** FG is mapped to a generic user component which can access directly the gateway, and which is placed in the Application component of the D2D Interoperability Architecture. This Application module also hosts another component, the Middleware Platform, which is the instantiation of the **IoT Platform Functional** Group of the INTER-IoT RA.

The **Management** FG corresponds to the Gateway Configuration components of the D2D Architecture.

Now we are going to show the mapping of the exact Functional Components of the INTER-IoT RA with the lower-level components of the D2D interoperability layer of the INTER-LAYER. The aim is to see how each Functional Component has been instantiated and to check that there are no missing Functional Components in the INTER-IoT RA. Please note that each interoperability layer uses only a subset of the existing Functional Groups in INTER-IoT RM & RA.

The mapping of the exact Functional Components of the INTER-IoT RA with the components of the D2D of the INTER-LAYER is depicted as follows:

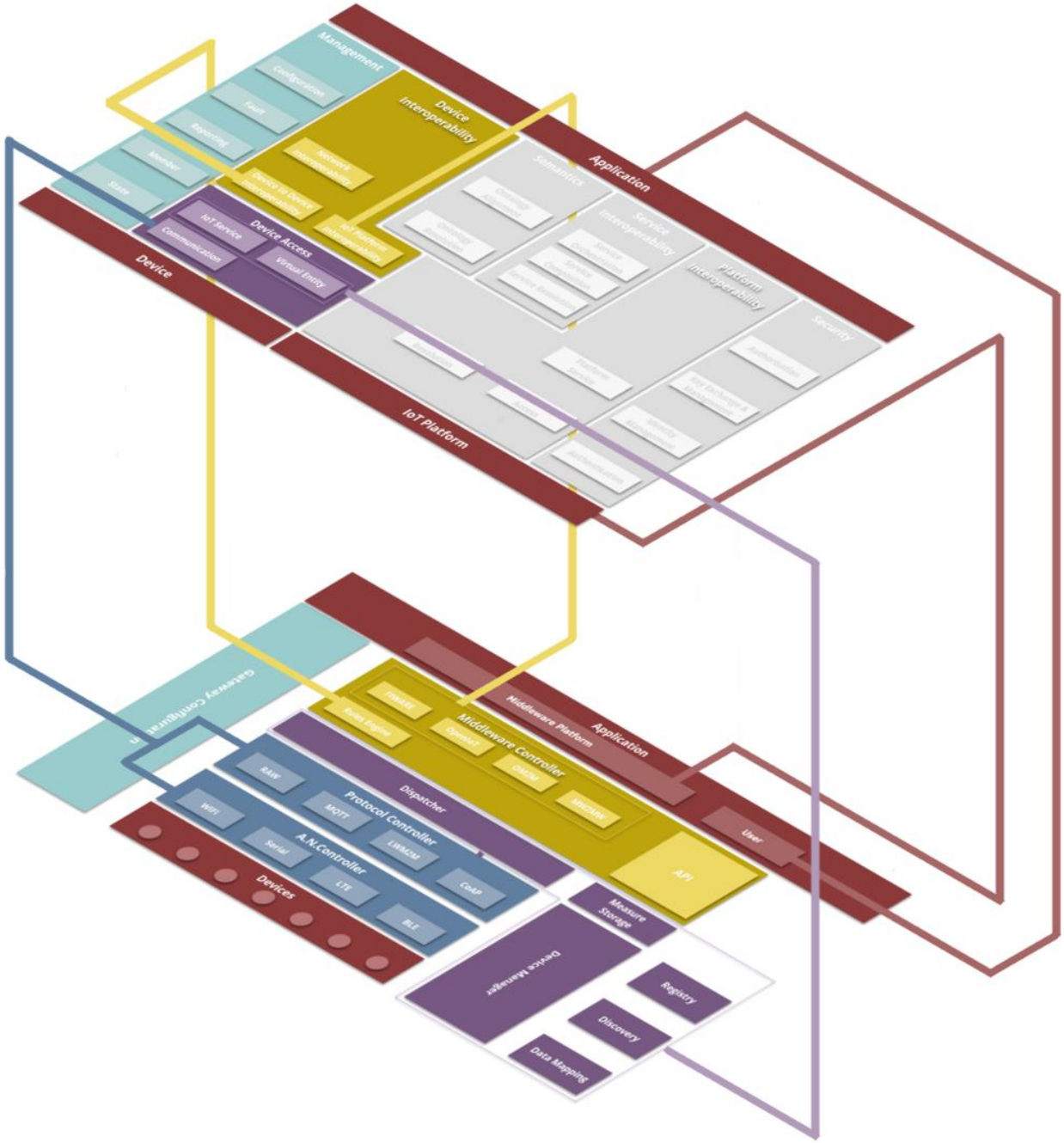


Figure 103: Mapping of the Functional Components of the INTER-IoT RA with the components of the D2D interoperability layer of the INTER-LAYER.

Applications	▶▶▶	Middleware platform
IoT Platform Interoperability	▶▶▶	Middleware Controller
Device to device interoperability	▶▶▶	Rules engine
IoT Service + Virtual Entity	▶▶▶	Discovery
Communications		Protocol Controller A.N. Controller

Table 7 Mapping of INTER-IoT FGs and D2D Layer Interoperabilty Infrastructure

In this figure, we show the instantiation of the different Functional Components of the INTER-IoT RA in the Gateway Architecture Design made in D3.1. The following Functional Components of the INTER-IoT RA have been instantiated:

- The **IoT Platform Interoperability FC** has been directly mapped to a component called Middleware Controller, that acts as a mediator between the MW module and the rest of the gateway.
- The **Device-to-Device Interoperability FC** is implemented in the Rules Engine, which performs that D2D interaction through configurable rules.
- The **Communication FC** has been instantiated as a set of components enclosed into two main groups:
    - A.N. (Access Network) Controller. It allows access to the devices, offering interfaces between the devices and the protocol modules. It includes the different A.N. Modules for providing access to different communication channels.
    - Protocol Controller. It contains all the communication protocols supported by the Gateway, also implementing the common interfaces between those protocols and the other components.
- The **IoT Service FC** and the **Virtual Entity FC**, are instantiated together through a set of modules in the Gateway Architecture. These modules are represented in purple colour enclosed by a dashed purple line. They include:
    - Dispatcher: It handles all traffic between the Protocols layer (Physical device) and the Middleware controller (Virtual device). It provides the entry point to the main functions of the IoT Service and Virtual entity services.
    - Device Manager: It provides information of any sensor/actuator.
    - Registry, Discovery, Data Mapping: They provide the necessary functions for finding the appropriate IoT Services.
    - Measure Storage: It stores measurements from the sensors to offer a history service through the Dispatcher.

5.3.2 Network to Network Interoperability

We don’t describe the mapping between the Functional Components of the Functional View of the INTER-IoT RA and the Network to Network Interoperability Architecture Design. The reason is that only one Functional Component was identified in the INTER-IoT RA related to network interoperability, as it is shown in Figure 104 below.

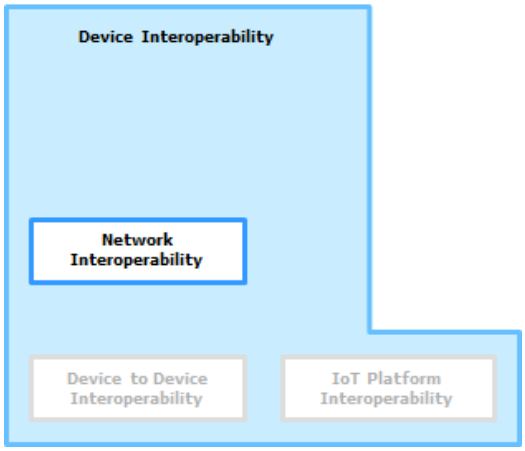


Figure 104: Functional Components for Network Interoperability.

Therefore, the instantiation of this **Network Interoperability FC** has really been an explosion to all the components designed in section 3.2 of D3.1.

During the iterations of the INTER-IoT Reference Architecture and the INTER-LAYER Architecture Design described in section 5.1, some discussion arose about the convenience of splitting the Network Interoperability FC into two or more FCs, or even whether to create or not a new Functional Group. The specific features of network interoperability, which are part of the INTER-IoT, but seem not widely addressed as a key issue in interoperability among IoT Platforms, advised us to keep identified as a Functional Component, but only as a single component part of the Device Interoperability Functional Group.

For the description of the different components of the Network to Network Interoperability Architecture Design, we suggest to read the section 3.2 of D3.1.

5.3.3 Middleware to Middleware Interoperability

The instantiation of the Functional View to the INTER-Middleware Interoperability layer has followed the paths depicted in the Figure 105, as shown below.

The Middleware to Middleware Interoperability Architecture corresponds to the also called INTER-MW Architecture, described in section 3.3 of the D3.1. We have rearranged a bit the INTER-MW Architecture to make it more simple to see the mappings through the lines with minimal crossings. We have also changed some colours, and we have removed the interaction arrows, as they don't add value to the functional mapping.

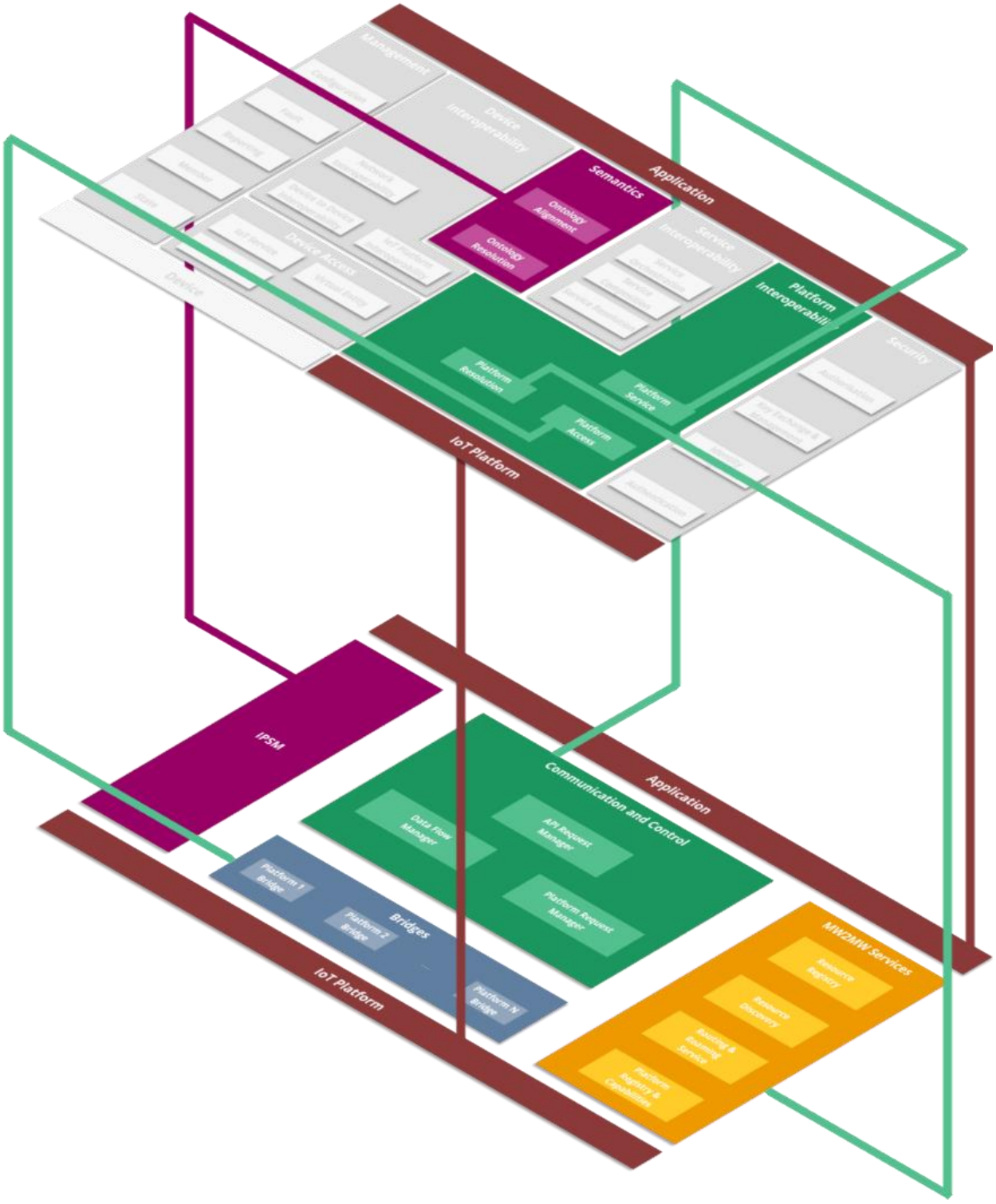


Figure 105: Functional Components of the INTER-IoT RA with the components of the INTER-MW interoperability layer of the INTER-LAYER.

Platform service	▶▶▶	Communication and control
Platform resolution	▶▶▶	MW2MW services
Platform access	▶▶▶	Bridges
Ontology alignment	▶▶▶	IPSM

Table 8 Mapping of INTER-IoT FGs and MW2MW Layer Interoperability Infrastructure

In this figure, we show the instantiation of the different Functional Components of the INTER-IoT RA in the INTER-MW Architecture Design made in D3.1. The following Functional Components of the INTER-IoT RA have been instantiated:

- The **Platform Service FC**, which was responsible for performing device and platform interactions, has been instantiated as a set of components grouped as Communication and Control. This group encloses three components:
  - API Request Manager. It handles requests, received from the API proxy.
  - Data Flow Manager. It orchestrates data flows from the platforms (bridges) to the original caller.
  - Platform Request Manager. It arranges and manages flow of requests to underlying platforms.
- The **Platform Resolution FC** which was responsible for discovering and cataloguing the IoT Platforms that are available at a specific deployment of INTER-IoT as well as their devices, capabilities and IoT Platform Services, has been instantiated as a set of components grouped as MW2MW Services. This encloses the following components:
  - Resource Registry. It contains a list of devices and their properties that can be quickly consulted when needed.
  - Resource Discovery. It finds resources based on queries.
  - Routing & Roaming Service. It allows the communication with a particular device independently of the platform it is currently connected to.
  - Platform Registry & Capabilities. It contains the information of all connected Platforms including their type and service capabilities.
- The **Platform Access FC**, that has the role of implementing the functions needed for connecting to an IoT Platform and accessing their resources, has been instantiated in the bridges component. It is just a collection of bridges for interacting with the different IoT Platforms. A bridge manages the communication with the underlying platforms by translating requests and answers in and out.
- The Semantics components, which are the **Ontology Alignment FC** and the **Ontology Resolution FC** have been instantiating in a common component called IPSM (IoT Platform Semantic Mediator). It is responsible, among other features, for translating incoming information, representing semantics of artefact X to semantics of artefact Y. The IPSM will use ontological alignments to perform ontology-to-ontology translations.

5.3.4 Application&Services to Application&Services Interoperability

The instantiation of the Functional View to the Application&Service Interoperability layer has followed the paths depicted in the Figure 106, as shown below.

The AS2AS Interoperability Architecture has been described in section 3.4 of the D3.1. We have rearranged a bit the AS2AS Architecture to make it more simple to see the mappings through the

lines with minimal crossings. We have also changed some colours, and we have removed the interaction arrows, as they don't add value to the functional mapping. As a result, the AS2AS Architecture has been flipped horizontally, keeping all the components.

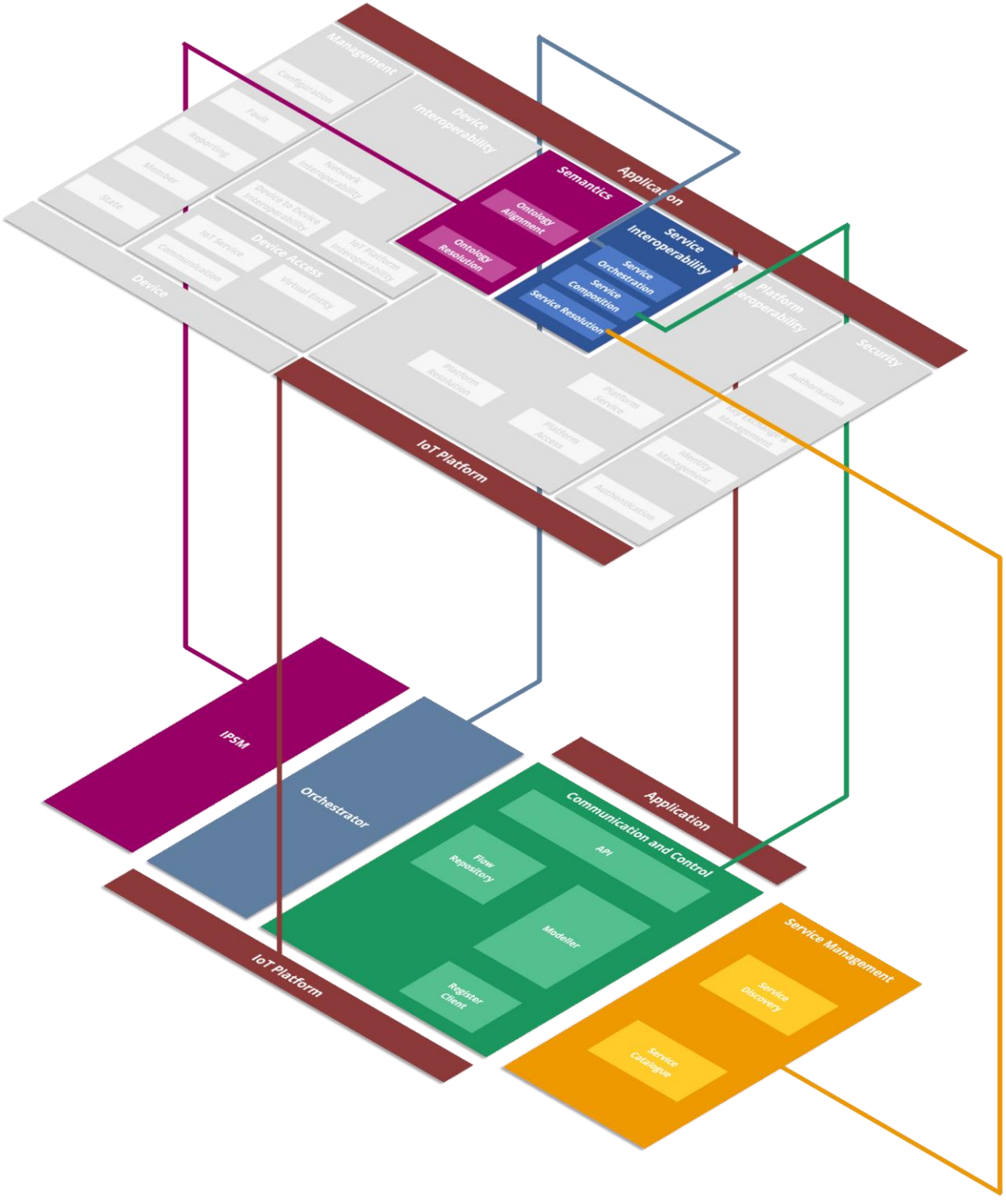


Figure 106: Functional Components of the INTER-IoT RA with the components of the Application&Service interoperability layer of the INTER-LAYER.

Service orchestration	▶▶▶	Orchestrator
Service composition	▶▶▶	Communication and control
Service resolution	▶▶▶	Service management
Semantics	▶▶▶	IPSM

Table 9 Mapping of INTER-IoT FGs and AS2AS Layer Interoperability Infrastructure

In this figure, we show the instantiation of the different Functional Components of the INTER-IoT RA in the AS2AS Architecture Design made in D3.1. The following Functional Components of the INTER-IoT RA have been instantiated:

- The **Service Resolution FC** was responsible for the storage of what we call *flows*, understood as a logical definition of a sequence of steps, each of which can be a service existing in an IoT Platform. It has been instantiated as a group that we have called Service Management here for mapping purposes. This group contains two components:
  - Service Discovery. It manages the detection of services provided by each IoT platform attending certain features.
  - Service Catalogue. It provides storage and access to a uniform catalogue of existing and new services.
- The role of the **Service Composition FC** was to design new compound services based on services that IoT Platforms exposes. The new services are designed like flows which will be later executed. In the AS2AS Architecture, it has been instantiated as a set of components depicted in green colour:
  - Modeller. It allows to make a composition of services with a graphical tool
  - Register Client. It allows the registration of new services through the graphical environment.
  - Flow Repository. It stores the composite services designed with the modeller.
- The **Service Orchestration FC** was responsible for the execution of the flows. It has been directly mapped to an Orchestrator component of the AS2AS Architecture with all the functions of the Service Orchestration FC.
- As in the MW2MW case, the Semantics components, which are the **Ontology Alignment FC** and the **Ontology Resolution FC** have been instantiating in a common component called IPSM (IoT Platform Semantic Mediator). It is responsible, among other features, for translating incoming information, representing semantics of artefact X to semantics of artefact Y. The IPSM will use ontological alignments to perform ontology-to-ontology translations.

5.3.5 Data&Semantics to Data&Semantics Interoperability

The Semantics FG of the INTER-IoT RA is used in different layers of the interoperability. It has been described in the MW2MW Interoperability and AS2AS Interoperability Architectures, the mapping of the Functional Components of the Semantics FG of the INTER-IoT RA to the specific components used in D3.1.

Basically, the instantiation has been done designing a common component called IPSM (IoT Platform Semantic Mediator), which is widely described in section 3.5 of D3.1. We have rearranged a bit the DS2DS Architecture to make it more simple to see the mappings through the lines with minimal crossings. We have also changed some colours, and we have removed the interaction arrows, as they do not add value to the functional mapping. We have added an enclosing dashed-box to some components to facilitate the interpretation of the mapping.

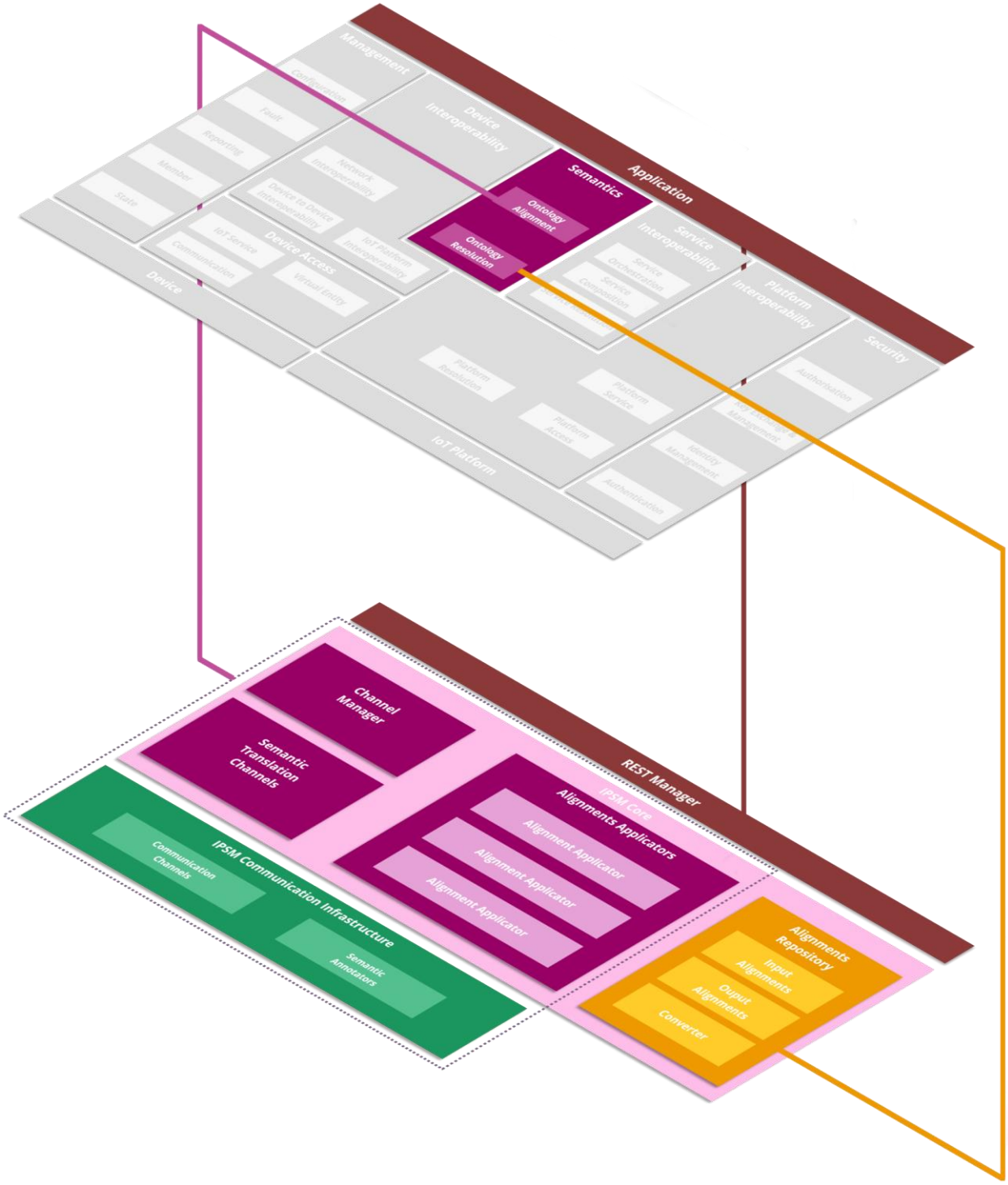


Figure 107: Functional Components of the INTER-IoT RA with the components of the Data&Semantics interoperability layer of the INTER-LAYER.

Ontology alignment	▶▶▶	Alignments repository
Ontology resolution	▶▶▶	IPSM Core
Application	▶▶▶	REST Manager

Table 10 Mapping of INTER-IoT FGs and DS2DS Layer Interoperability Infrastructure

In this figure, we show the instantiation of the different Functional Components of the INTER-IoT RA in the DS2DS Architecture Design made in D3.1. The following Functional Components of the INTER-IoT RA have been instantiated:

- The **Ontology Resolution FC** was responsible for managing the different ontologies used at the various IoT Platforms that are connected through INTER-IoT. It has been instantiated as the Alignments Repository in the DS2DS Architecture Design.It stores and manages alignments (read/write alignments) used in the translation process. It contains:
  - Input Alignments.
  - Output Alignments.
  - Converter. It performs one-time conversion for each new alignment written into the repository.
- The **Ontology Alignment FC** was responsible for performing the alignment from a source data with an ontology to a target data with its own ontology. It makes the data translation between two ontologies, using the ontology definitions resolved by the Ontology Resolution FC. It has been instantiated as a set of components enclosed by a purple dashed-box. These components are depicted in purple/pink or green colour:
  - Channel Manager. It manages (creates, destroys, lists) Communication Channels i.e. flows in message broker and Semantic Translation Channels.
  - Semantic Translation Channels. A lightweight component that stores information about: where to receive data from, which alignment to use, and where to send data to.
  - Alignment Applicator. A component, instances of which are performing semantic translation.
  - IPSM Communication Infrastructure. It facilitates the communication between IoT artefacts and the IPSM.

6 Appendices

6.1 Appendix 1 - INTER-IoT requirements relevant to meta-data

Inter-IoT Requirement	Description	Meta-data entities + *comments*
INTERIOT-203 (Id 1) Roaming across platforms	Users want to get information about a device independently of the platform it is. Objects that are moving can switch platform to which they are connected. Change between a platform and the other should be automatic and transparent to the device. From INTER-FW point of view, a moving device could be set as 'roamable' to specify INTER-LAYER that if it's not available in the expected platform, it should try to discover it in the rest of connected platforms and update the device registry	Device ID, location, position
INTERIOT-278 (Id 42) Support for heterogeneous information representation	The method of integration of multiple information and knowledge representing the same real-world sensing object into a consistent, accurate, and useful representation. It will help to fully take the usage of the IoT information resources for different application and service within an IoT system or between different information systems.	Sensor, *Meta-model design*
INTERIOT-325 (Id 10) Extensibility (feature evolution)	Functionality must be updated over time, and the system should be capable to integrate these updates. The system should expose functionality to the infrastructure maintenance to update the functionality when needed with new INTER-FW versions, without affecting existing clients.	*Meta-model design - extensibility*
INTERIOT-466 (Id 58) Auditability and Accountability	Configured operations performed in the system must be tracked uniquely to the entity that generated it.	Provenance information (ownership, creation, responsibility), users

	The platform should allow: - To retrieve users and/or devices that carried out or are in charge of the activities in the system and their logged operations. - Producing an Audit log with trace of the most important data and their values before and after changes; - Maintain records for a period not less than six months; - Provide synchronization technologies in order to keep aligned the date and time recorded in the logs associated with the access. The criteria for registration of the aforesaid Log (products so as to not be editable) must at least enable the identification: - The event that triggered the log (login, logout, login failure); - The user, the date and the start / end connection. - The sensitive data updates (before and after)	
INTERIOT-473 (Id 63) Provision of authentication credentials	Conformity to the legal rules and criteria (Privacy code) it must be defined procedures and roles for authentication credentials management process to enable proper management of authentication credentials of persons in charge of the data processing. As regards the management of the User credentials, the platform will have to: - allow access only through individual authentication credentials (consisting of a User ID and an authentication device, e.g. Password); - prevent the reassignment of User ID to another user; - allow the definition of access profiles sets that guarantee the principles of "need to know" and “segregation of duties”;	Authentication credentials and methods (user ID, email, password, checksum, encrypted key file,authentication device,authentication token)

	- allow the extraction of the information required to verify the correct allocation of authentication credentials and their authorization profiles; - carry out automatic checks at least monthly of the users inactive for more than six months in order to suspend, unless the users for which it has been required and authorized a derogation on the basis of an operational need.	
INTERIOT-479 (Id 69) Confidentiality	Conformity to the legal rules and criteria (Privacy code): In order to ensure the confidentiality of data, it will have to ensure compliance with the principle of "need to know" through the implementation of appropriate measures. Avoid data falsification or disclosure. - If the need of data processing ended, such data must be deleted permanently and irreversibly in order to prevent unauthorized treatment. - It must be guaranteed the logical isolation of data belonging to different customers on a single platform. In particular, it must be guaranteed the segregation of single customer views, in order to allow processing of data only to persons in charge of the processing (preventing access / views by unauthorized persons). - Special procedures for extraction and transmission of the data processed by the platform must be available. - In order to ensure the confidentiality of data stored in the platform encryption must be provide of identification codes or other solutions that make health data unintelligible to those who are authorized to access (i.e. identification data decoupled from health / sensitive ones).	Authentication credentials and methods
INTERIOT-483 (Id 4) Alignment with AIOTI architecture and view	A key requirement for the system architecture is the alignment with the architectural reference models of other IoT projects, and especially AIOTI.	Meta-model design

	AIOTI architectural model is suitable for guiding the development of INTER-IoT architecture. The use of AIOTI view of the architecture of Internet of Things will be useful, in order to utilize its results and from other projects to avoid re-inventing a new architectural model from scratch, and to be aligned and compatible with those projects.	
INTERIOT-540 (Id 98) Data provenance	Data provenance metadata should allow to identify what is the origin of data e.g. which artefact collected the data.	Provenance (source of information – platform, device or user)
INTERIOT-547 (Id 77) Users manage how their public data is seen	Devices/IoT platforms as data sources are owned by different third parties. The owner of the object should be able to manage who and when other users have access to their information. IoT platforms should support data ownership management, data-flow monitoring, and access management. Data visibility is managed according to owning entities policies. This is managed globally (platform independent) At the configuration of an IoT platform registrated into INTER-IoT, the software integrator may be able to specify a list of devices and/or operations which will be accessible from external agents through INTER-IoT, how long, with whom, etc.	Data access policy – when, who (user, role, device, platform)
INTERIOT-616 (Id 186) Design of required ontologies	To achieve semantic interoperability generic ontology(ies) should be used. Use of required ontologies - a generic ontology of the Internet of Things. Creation of GOIoTP, a global IoT ontology, providing common understanding of the IoT (generic) meta-structure, and enabling semantic interoperability. It is required to be designed or chosen from available ones in order to produce semantic alignment. GOIoTP will be based on current main IoT ontologies, such as W3C SSN, SAREF, etc.	Meta-model design

INTERIOT-662 (Id 223) Semantic support for virtual smart objects, not only sensors	More broad definition of smart object in the ontology, not only referred to physical sensors but to other types of smart object. INTER-IoT ontology, GOIoTP, will include support for smart objects that are not sensors, but act as smart devices, such as virtual devices, human interfaces or algorithms. Many ontologies do not include objects that are not sensors, although they are potential and relevant IoT smart objects.	Device (sensor, actuator, human interface)
INTERIOT-663 (Id 224) Location semantic support for mobile smart objects	The location of smart objects may be a critical information in order to analyse data from them, specially in the case of mobile sensors, and it is not considered in many ontologies.	Device location / position
INTERIOT-693 (Id 132) Portability	Unique names, usage, disambiguation. Service providers must be able to switch between customers / users.	Entities (users, services) IDs
INTERIOT-699 (Id 254) Each data unit is identified univocally	Allow traceability, storage and decoupling between transmissions. Each minimal unit of meaningful data transmission (e.g. a heart rate measurement or a truck location event) must contain an identifier allowing retrieve the source of data and the network/platform for traceability.	Provenance (source of information – platform, network, device)
INTERIOT-702 (Id 256) Each device has a unique INTER-IoT identifier	Each device connected to the network must be recognized in order to be able to process data to and from the device. There should not be a limitation to the number of devices that can connect. An identifier system must be developed to be able to identify each device. Granularity in identification must reach the device level.	Device ID
INTERIOT-703 (Id 257) The INTER-IoT unique ID is used to find the platform-specific ID of the device	The platform specific ID needs to be retrieved from the INTER-IoT ID. The platform specific ID of each element needs to be retrieved from que unique ID assigned in INTER-IoT. This ensures traceability.	Device ID

INTERIOT-706 (Id 260) Manages user permission	Users have permissions to access different platform/devices that need to be managed.	User permissions for platforms/devices.
INTERIOT-708 (Id 262) Manages group-based permissions	Permissions can be managed at group level in order to simplify business processes.	User groups
INTERIOT-709 (Id 263) Access to personal data needs to be previously authorized	Personal data access must meet the EU policies. Access to personal information must be previously authorized by the owner.	Authorization
INTERIOT-712 (Id 266) API allows resources/capabilities discovery	Applications and/or physical devices needs to know the resources and capabilities of the connected platforms. API allow applications to discover resources and capabilities of the platforms.	Platform
INTERIOT-723 (Id 278) Future-proof	Future-proof: Future versions of the protocol must work with prior versions and provide all the same capabilities as prior versions.	*Meta-model design*
INTERIOT-729 (Id 91) The implementation must be done by phases and progressively	When a complex system migrates to a new IoT protocol it is impossible to do it all at the same time. The process of implementation a new IoT protocol has to be compatible with both at the same time, i.e. at least it should have a gateway between the new and the old systems.	*Meta-model design*
INTERIOT-829 (Id 280) Requests filtering	Access needs can be very different for each situation, so tools must be provided to the user to select what he needs. When sending a requests to INTER-FW, it will be possible to specify filtering: The system shares a common filter format when possible. This filtering will allow: - Selection of platform(s). - Selection of device(s). - Selection of property type(s). - Selection of property filtering(s).	Platform, Device, Geolocation

	- Selection of geo-queries (if allowed by the IoT platform).	

6.2 Appendix 2 - IoT ontologies

Ontology	*Dublin Core*
URI	http://purl.org/dc/terms
Available at:	http://dublincore.org/
Documentation at:	http://dublincore.org/documents/dcmi-terms/
Description: A set of vocabulary terms that can be used to describe web such as web pages and physical resources such as books or CDs, and objects like artworks.	

Ontology	*FoaF (Friend of a Friend)*
URI	http://xmlns.com/foaf/spec/
Available at:	http://xmlns.com/foaf/spec/
Documentation at:	http://www.foaf-project.org/
Description: FoaF (Friend of a Friend) describes persons, their activities and their relations to other people and objects. FoaF allows to describe social networks without the need for a centralised database.	

Ontology	*DUL (Dolce Ultra Lite)*
URI	http://www.ontologydesignpatterns.org/ont/dul/DUL.owl
Available at:	http://www.ontologydesignpatterns.org/ont/dul/DUL.owl
Documentation at:	http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite
Description: It is a simplification and an improvement of some parts of DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) Lite-Plus library and Descriptions and Situations ontology (DnS). Its purpose is to provide a set of upper	

level concepts that can be the basis for easier interoperability among many middle and lower level ontologies.

Ontology	*Prov-O*
URI	http://www.w3.org/ns/prov#
Available at:	http://www.w3.org/ns/prov-o
Documentation at:	https://www.w3.org/TR/prov-o/
Description: The PROV-O ontology is a realization of the PROV model in OWL. The model itself is used to represent provenance, i.e. information about actors, entities and activities involved in producing a piece of data (e.g. a document) or thing (e.g. a physical book), regarding quality, reliability, trustworthiness.	

Ontology	*Schema.org*
URI	http://topbraid.org/schema/
Available at:	http://topbraid.org/schema/schema.rdf
Documentation at:	http://topbraid.org/schema/
Description: Schema.org is a collection of terms that webmasters can use to markup their pages to improve the display of search results. There is an up-to-date OWL version of the ontology produced by TopQuadrant.	

Ontology	*SAO (Stream Annotation Ontology)*
URI	http://purl.oclc.org/NET/UNIS/sao/sao
Available at:	http://iot.ee.surrey.ac.uk/citypulse/ontologies/sao/saov06.rdf
Documentation at:	http://iot.ee.surrey.ac.uk/citypulse/ontologies/sao/sao
Description: A lightweight ontology used to represent the features of a stream data. It is built on top of Semantic Sensor Networks (SSN), PROV-O and TimeLine Ontologies, and involves connections with the Complex Event Processing Ontology.and Quality Ontology.	

Ontology	*M3-lite*
URI	http://purl.org/iot/vocab/m3-lite
Available at:	http://purl.org/iot/vocab/m3-lite
Documentation at:	https://mimove-apps.paris.inria.fr/ontology/m3lite.html
Description: Machine-to-Machine measurement ontology is refactored, cleaned and simplified version of M3 ontology.	

Ontology	*Open-IoT Ontology*
URI	http://openiot.eu/ontology/ns/
Available at:	http://openiot.eu/ontology/ns/openiot.owl
Documentation at:	https://github.com/OpenIotOrg/openiot/wiki/X-GSN-Use
Description: It was developed within the OpenIoT project. The ontology is a comparatively big model that (re)uses and combines other ontologies. Those include all modules of the SSN (the main basis for the OpenIoT), SPITFIRE (including sensor networks), Event Model-F, PROV-O, LinkedGeoData, WGS84, CloudDomain, SIOC, Association Ontology and others, including smaller ontologies developed at the DERI (currently, Insight Centre). It also makes use of ontologies that provide base for those enumerated before, e.g. DUL. Other than those from the SSN, OpenIoT, uses a large number of SPITFIRE concepts, e.g. network and sensor network descriptions. Although some of the mentioned ontologies are not imported by the OpenIoT explicitly, they appear in all examples, documentation and project deliverables. Therefore, we treat OpenIoT as a combination of parts of all of those. Similarly to the SSN, OpenIoT does not define its own location concepts and does not explicitly import geolocation ontologies. It relies on other ontologies for that but, in contrast to the SSN, it clearly indicates LinkedGeoData and WGS84 as sources of geolocation descriptions. It defines a limited set of units of measure (e.g. temperature, wind speed), but only when they were relevant to the OpenIoT project pilot implementation.	

Ontology	*OneM2M Base Ontology*
URI	http://www.onem2m.org/ontology/Base_Ontology/base_ontology
Available at:	http://www.onem2m.org/ontology/Base_Ontology/oneM2M_Base_Ontology-V_2_0_0.owl

Documentation at:	http://www.onem2m.org/technical/onem2m-ontologies

Description:

It is a recently created ontology, with first non-draft release in August 2016. It is relatively small, prepared for the release 2.0 of oneM2M specifications, and designed with the intention of providing a shared ontological base to which other ontologies align to. It is similar to the SSN, since any concrete system necessarily needs to extend it before implementation. It describes devices in a very broad scope, enabling (in a very general sense) specification of device functionality, networking properties, operation and services. The philosophy behind this approach was to enable discovery of semantically demarcated resources using a minimal set of concepts. It is a base ontology, as it does not extend any other base models (such as DUL or Dublin Core).

Ontologies	***UniversAAL ontologies***
URI	http://ontology.universAAL.org/[ontology name].owl
Available at:	http://ontology.universaal.org/ https://github.com/universAAL/ontology
Documentation at:	https://github.com/universAAL/ontology/wiki

Description:

A set of ontologies developed within UniversAAL (Universal open platform and reference Specification for Ambient Assisted Living) project. The following ontologies were used as data models for information shared thourgh the middleware buses. The following ontologies were selected as relevant in the INTER-IoT context:

- Devices - unified device ontology.
- Measurement - ontology for representing different measurement capabilities e.g. measurement, signal, error.
- Data Representation - basic data representation model with concepts representing e.g. root class for all locations, root class for comparable individuals, enumeration for QoS rating.
- Unit - ontology for unit representation such as ampere, bit, gram.
- Physical Things - ontology for physical things. It is part of the Physical World upper ontology concept, which defines the most general concepts from the physical world as opposed to the virtual realm.
- Security - ontology defining the most general concepts dealing security.
- Location - ontology for locations. It is part of the Physical World upper ontology concept, which defines the most general concepts from the physical world as opposed to the virtual realm.
- Service Bus - ontology of the universAAL Service Bus
- Health, HealthMeasurement, - health ontologies as an example of domain ontologies defining the health service, based on the treatment concept and measurements of health parameters

- PersonalHealthDevice - ontology for person-related health devices (Continua certified devices) e.g. blood pressure monitor, weighing scale,...

Ontology	*SSN Ontology*
URI	http://purl.oclc.org/NET/ssnx/ssn
Available at:	https://www.w3.org/ns/ssn/
Documentation at:	https://www.w3.org/TR/vocab-ssn/
Description: This ontology describes sensors and observations, and related concepts. It does not describe domain concepts, time, locations, etc. as these are intended to be included from other ontologies via OWL imports.	

Ontology	*SAREF*
URI	https://w3id.org/saref
Available at:	http://ontology.tno.nl/saref.owl
Documentation at:	http://ontology.tno.nl/saref/
Description: It covers the area of smart devices in houses, offices, public places, etc. It does not focus on any industrial or scientific implementation. The devices are characterized predominantly by the function(s) they perform, commands they accept, and states they can be in. Those three categories serve as basic building blocks of the semantic description in SAREF. Elements from each can be combined to produce complex descriptions of multi-functional devices. The description is complemented by device services that offer functions. A noteworthy module of SAREF is the energy and power profile that has received considerable attention shortly after its inception. SAREF uses WGS84 for geolocation and defines its own set of measurement units.	

Ontology	*Fiesta-IoT Ontology*
URI	https://mimove-apps.paris.inria.fr/ontology/fiesta-iot.owl
Available at:	https://mimove-apps.paris.inria.fr/ontology/fiesta-iot.owl
Documentation at:	http://ontology.fiesta-iot.eu/ontologyDocs/fiesta-iot/doc
Description:	

FIESTA-IoT Ontology is designed with a goal to achieve semantic interoperability among heterogeneous testbeds. To build the ontology, a number of core concepts from various mainstream ontologies and taxonomies were merged, such as W3C SSN, M3-lite, WGS84, IoT-lite, Time, and DUL ontology.

Ontology	***iot-lite***
URI	http://purl.oclc.org/NET/UNIS/fiware/iot-lite
Available at:	http://iot.ee.surrey.ac.uk/fiware/ontologies/iot-lite/iot-lite.rdf
Documentation at:	http://iot.ee.surrey.ac.uk/fiware/ontologies/iot-lite
Description: It is an instantiation of the SSN, i.e. a direct extension of some of its modules. It is a minimal ontology, to which most of the caveats of the SSN apply. Those include: focus on sensors and observations, reliance on other ontologies (e.g. time or units ontologies), high modularity and extendability. The idea behind IoT-Lite was to create a small/light semantic model that would be less taxing (than other, more verbose and broader models) on devices that process it. At the same time, it needed to cover enough concepts to be useful. The ontology describes devices, objects, systems and services. The main extension of the SSN in the IoT-Lite lies in addition of actuators (to complement sensors, as a device type) and a coverage property. It explicitly uses concepts from a geolocation ontology (WGS84) to demarcate device coverage and deployment location.	

Ontology	***Ontology Modeling for Intelligent Domotic Environments (dogont)***
URI	http://elite.polito.it/ontologies/dogont
Available at:	http://elite.polito.it/ontologies/dogont.owl
Documentation at:	http://www.cad.polito.it/pap/exact/iswc08.html
Description: The DogOnt ontology supports device/network independent description of houses, including both controllable and architectural elements.	

Ontology	*IoT-O*
URI	http://www.irit.fr/recherches/MELODI/ontologies/IoT-O#

Available at:	http://lov.okfn.org/dataset/lov/vocabs/ioto/versions/2015-02-20.n3
Documentation at:	https://www.irit.fr/recherches/MELODI/ontologies/IoT-O.html

Description:

IoT-O is a core domain Internet of Things ontology. It is intended to model horizontal knowledge about IoT systems and applications, and to be extended with vertical, application specific knowledge. It is constituted of different modules : - A sensing module, based on W3C's SSN (http://purl.oclc.org/NET/ssnx/ssn) - An acting module, based on SAN (http://www.irit.fr/recherches/MELODI/ontologies/SAN) - A service module, based on MSM (http://iserve.kmi.open.ac.uk/ns/msm/msm-2014-09-03.rdf) and hRest (http://www.wsmo.org/ns/hrests) - A lifecycle module, based on a lifecycle vocabulary (http://vocab.org/lifecycle/schema-20080603.rdf) and an iot-specific extension (http://www.irit.fr/recherches/MELODI/ontologies/IoT-Lifecycle) - An energy module, based on powerOnt (ttp://elite.polito.it/ontologies/poweront.owl) IoT-O developping team also contributes to the oneM2M IoT interoperability standard

Ontology	***Spitfire Ontology***
URI	http://spitfire-project.eu/ontology/ns
Available at:	http://spitfire-project.eu/ontology.owl
Documentation at:	http://spitfire-project.eu/incontextsensing/ontology.php

Description:

The SPITFIRE Ontology (spt) is based on the alignment among Dolce+DnS Ultralite(dul), the W3C Semantic Sensor Network ontology (ssn) and the Event Model-F ontology (event).

Ontology	***SAN (Semantic Actuator Network)***
URI	
Available at:	
Documentation at:	http://www.irit.fr/recherches/MELODI/ontologies/SAN

Description:

This ontology is intended to describe Semantic Actuator Networks, as a counterpoint to SSN definition of Semantic Sensor Networks. An actuator is a physical device having an effect on the world (see Actuator for more information). It is worth noticing that some concepts are imported from SSN, but not SSN as a whole. This is a design choice intended to separate as much as possible the

	definition on actuator from the definition of sensor, which are completely different concept that can be used independently from each other. This ontology is used as a ontological module in IoT-O ontology.

Model	***SensorThings***
URI	N/A
Available at:	https://github.com/opengeospatial/sensorthings
Documentation at:	http://docs.opengeospatial.org/is/15-078r6/15-078r6.html
Description: API: http://cite.opengeospatial.org/te2/about/sta10/1.0/site/apidocs/index.html Documentation at: http://docs.opengeospatial.org/is/15-078r6/15-078r6.html OGC SensorML for sensor description URI: http://www.opengeospatial.org/standards/sensorml OGC SensorML for sensor description specification: http://www.opengeospatial.org/standards/om OGC Observations and Measurements: http://www.opengeospatial.org/standards/om	

Ontology	*W3C OWL-Time ontology*
URI	http://www.w3.org/2006/time#
Available at:	https://www.w3.org/2006/time
Documentation at:	https://www.w3.org/TR/owl-time/
Description: The ontology provides a vocabulary for expressing facts about topological relations among instants and intervals, together with information about durations, and about temporal position including date-time information.	

Ontology	***Timeline Ontology***
URI	http://purl.org/NET/c4dm/timeline.owl
Available at:	http://motools.sf.net/timeline/timeline.n3
Documentation at:	http://motools.sourceforge.net/timeline/timeline.html
Description: The ontology defines the TimeLine concept, that is meant to identify a temporal backbone. Each temporal object (signal, video, performance, work, etc.) can be associated to such a timeline. Then, a number of Interval and Instant can be defined on this timeline.	

Ontology	*Process Execution Ontology*
URI	https://w3id.org/pep/
Available at:	https://w3id.org/pep/
Documentation at:	http://ci.emse.fr/pep/
Description: The process execution ontology is a proposal for a simple extension of both the [W3C Semantic Sensor Network](https://www.w3.org/TR/vocab-ssn/) and the [Semantic Actuator Network](https://www.irit.fr/recherches/MELODI/ontologies/SAN.owl) ontology cores.	

Ontology	*Event Ontology*
URI	http://purl.org/NET/c4dm/event.owl
Available at:	http://motools.sf.net/event/event.n3
Documentation at:	http://motools.sourceforge.net/event/event.html
Description: This ontology deals with the notion of reified events. It defines one main Event concept that may have a location, a time, active agents, factors and products.	

Ontology	*Geoposition Ontology (wgs84_pos)*
URI	http://www.w3.org/2003/01/geo/wgs84_pos
Available at:	https://www.w3.org/2003/01/geo/wgs84_pos.rdf
Documentation at:	https://www.w3.org/2003/01/geo/
Description: A vocabulary for representing latitude, longitude and altitude information in the WGS84 geodetic reference datum. Basic classes are: SpatialThing (anything with special extent) and Point (a point described using a coordinate system such as WGS84). Basic properties are: latitude, longitude, location, altitude, lat/long.	

Ontology	***GeoSPARQL***
URI	http://www.opengis.net/ont/geosparql

Available at:	http://schemas.opengis.net/geosparql/1.0/geosparql_vocab_all.rdf
Documentation at:	http://www.opengeospatial.org/standards/geosparql
Description: The OGC GeoSPARQL standard supports representing and querying geospatial data on the Semantic Web. GeoSPARQL defines a vocabulary for representing geospatial data in RDF, and it defines an extension to the SPARQL query language for processing geospatial data. GeoSPARQL ontology defines a list of spatial concepts described in OGC/ISO Simple Features e.g. point, line, polygon that can be placed in a geometry concept hierarchies.	

Model / Ontology	**GeoRSS**
URI	http://www.georss.org/georss/
Available at:	https://www.w3.org/2005/Incubator/geo/XGR-geo-20071023/W3C_XGR_Geo_files/geo_2007.owl
Documentation at:	http://www.georss.org/
Description: Geographically Encoded Objects for RSS feeds is an emerging standard for encoding location as part of a Web feed. Two encodings of GeoRSS are available: GeoRSS-Simple - a lightweight format that supports basic geometries and covers the typical use cases when encoding locations. GeoRSS GML - a formal Open Geospatial Consortium (OGC) GML Application Profile, that supports a greater range of features than GeoRSS Simple e.g. coordinate reference systems other than WGS84 latitude/longitude. W3C Geo OWL provides an ontology which closely matches the GeoRSS feature model and which utilizes the existing GeoRSS vocabulary for geographic properties and classes.	

Format	***GeoJSON (IETF RFC 7946)***
URI	N/A
Available at:	http://geojson.org/
Documentation at:	https://tools.ietf.org/html/rfc7946
Description: GeoJSON is a geospatial data interchange format based on JSON proposed by Internet Engineering Task Force (IETF). It defines several types of JSON objects and the way in which they can be combined to represent data about geographic	

	features, their properties, and their spatial extents. GeoJSON uses a geographic coordinate reference system, WGS84 and units of decimal degrees.

Ontology	***Library for Quantity Kinds and Units***
URI	http://purl.oclc.org/NET/ssnx/qu/qu
Available at:	http://purl.oclc.org/NET/ssnx/qu/qu
Documentation at:	https://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu
Description: This ontology is partially based on the SysML QUDV (Quantities, Units, Dimensions and Values) proposed by a working group of the SysML 1.2 Revision Task Force (RTF), working in close coordination with the OMG MARTE specification group.	

Ontology	***Ontology for Quantity Kinds and Units***
URI	http://purl.oclc.org/NET/ssnx/qu/qu-rec20
Available at:	http://purl.oclc.org/NET/ssnx/qu/qu-rec20
Documentation at:	https://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu-rec20.html
Description: Ontology units and quantities definitions that imports the qu ontology. It defines numerous dimensions and can be used as a common model for describing the type of data measured by sensors.	

Ontology	***Units of Measurement (UO)***
URI	http://purl.obolibrary.org/obo/uo.owl
Available at:	http://www.berkeleybop.org/ontologies/uo.owl
Documentation at:	https://bioportal.bioontology.org/ontologies/UO
Description: The Ontology of Units of Measurement is developed as part of the OBO Foundry initiative.	

Model	***Unified Code for Units of Measure (UCUM)***

URI	http://unitsofmeasure.org/trac
Available at:	http://unitsofmeasure.org/trac
Documentation at:	http://unitsofmeasure.org/ucum.html
Description: A code system intended to include all units of measures being contemporarily used in international science, engineering, and business. A typical application of UCUM are electronic data interchange (EDI) protocols, but it can also be used in other types of machine communication.	

Ontology	***IoT-A Ontologies***
URI	
Available at:	http://www.surrey.ac.uk/ccsr/ontologies/DeviceModel.owl
Documentation at:	http://iot.ee.surrey.ac.uk/s2w/share/ontologies/iot-a/original/
Description: The following ontologies were developed within IoT-A projects: DeviceModel.owl - extends W3C SSN and DUL ontologies with TagDevice, NodeDevice, ActuatingDevice, Actuator concepts LocationModel.owl - describes location with concepts such as SpatialThing, Room, Building, Floor, Premises, Compass_Area ResourceModel.owl - describes a resource i.e. a computational element that gives access to information about or actuation capabilities on a Physical Entity ServiceModel.owl - exposes resource functionalities in terms of the input, output, precondition, and effect ServiceInstance.owl - sample instances built on the service model VirtualEntityModel.owl - describes a physical object that is relevant from a user or application perspective VirtualEntityInstance.owl - sample instances built on the virtual entity model AssociationModel.owl - imports VirtualEntityModel.owl and ServiceModel.owl and adds concepts to relate virtual entites to services. e.g. ServiceCabalilities, VEServiceDescription AssociationInstance.owl - sample instances built on the association model	

Ontology	***Quantities, Units, Dimensions, and Types Ontology***

URI	http://qudt.org/1.1/schema/qudt
Available at:	http://qudt.org/1.1/schema/qudt
Documentation at:	https://bioportal.bioontology.org/ontologies/QUDT
Description: The ontology specifies the base classes properties, and restrictions used for modeling measurable quantities, units for measuring different kinds of quantities and and their dimensions in various measurement systems.	

Model	***Web Service Modelling Ontology (WSMO)***
URI	N/A
Available at:	https://www.w3.org/Submission/WSMO/#appendixA
Documentation at:	https://www.w3.org/Submission/WSMO/
Description: WSMO is a top-down conceptual framework for describing semantic web services in order to facilitate the automation of discovering, combining and invoking. It provides ontology-based framework with components, ontologies, web service descriptions (describe the functional and behavioral aspects), goals (user desires) and mediators (interoperability between different WSMO elements).	

Ontology	***WSMO-Lite***
URI	http://www.wsmo.org/ns/wsmo-lite/index.rdfxml
Available at:	http://www.wsmo.org/ns/wsmo-lite#
Documentation at:	https://www.w3.org/Submission/WSMO-Lite/
Description: This is a lightweight ontology for semantic annotations of services, intended for use with SAWSDL and the Minimal Service Model.	

Ontology	***OWL-S: Semantic Markup for Web Services***
URI	N/A

Available at:	http://www.daml.org/services/owl-s/1.0/
Documentation at:	https://www.w3.org/Submission/OWL-S/
Description: OWL ontology for describing semantic web services. It was designed to enable automatic discovering, invoking, composing, and monitoring web resources offering services. OWL-S has three main parts: service role (service description), service model (how a client can interact with the service e.g. inputs, outputs) and service grounding (details needed to interact with the service e.g. communication protocols, message formats).	

6.3 Appendix 3 - Functional view study dataset

6.3.1 Applications

	Applications Specific application domains the IoT platform is designed or used for														
Platform	**Healthcare**	**Retail**	**Banking**	**Insurance**	**Gaming**	**Telcos**	**Smart Home**	**Smart Office**	**Smart City**	**Smart Energy**	**Smart Parking**	**Smart Transport**	**Smart Business**	**Application Entity**	**Others**
FIWare															
Open IoT							Y		Y	Y	Y	Y			
UniversAAL	Y						Y								
	Initially intended for eHealth and						It provides an ontology for Smart Home								
OneM2M	Y	Y					Y	Y	Y	Y	Y	Y	Y	Y	Y
Microsoft Azure IoT	Y							Y	Y	Y	Y	Y	Y	Y	
Amazon AWS IoT															
All-Joyn	Y			Y		Y	Y	P			Y	Y			
	Introspection, Configuration Service					Services in one app can discover and	Introspection	Via proxy object, established							
Butler	Y	Y					Y				Y	Y			
	Smart Health/Wellness	SmartShopping					SmartHome/Office		SmartCity			SmartTransport /Mobility			
i-Core	Y	Y					Y	Y	Y			Y	Y		Y
	Health Care Prison Security	Consumer Moving Patterns					Social Sensors	Smart Meeting	IJKdijk (dike integrity management)			Supply Chain Management and Logistics			Cross Domain: Crowded Event Management
Sofia 2	Y		Y	Y		Y			Y	Y	Y	Y	Y		
ThingSpeak														Y	Y
GE Predix									Y	Y	Y	Y	Y		Y
															Indstry
IBM Watson IoT	Y	Y	Y	Y	Y	Y									
WSO2	Y	Y	Y	Y			Y	Y	Y			Y	Y		Y
									Government				Architecture, Engineering and Construction		Education
Contiki							Y	Y	Y			Y			Y

6.3.2 Management

	Management Functionalities that are needed to govern an IoT system				
Platform	Configuration	Fault	Reporting	Member	State
FIWare	Y	Y	Y	Y	Y
	Backend Device Management - IDAS	Resource Usage Monit.	Data Visualization - SpagoBI	Backend Device Management - IDAS	Auxiliar enablers
Open IoT	Y	P	Y	Y	Y
	Extended GSN (X-GSN)	Application Runtime Monitoring (with Java Melody)	Application Runtime Monitoring (with Java Melody)	OpenIoT Security and Privacy module	Virtual Sensor Configuration & Monitoring
UniversAAL	Y	Y		P	P
	Middleware, profiling tool, LDDI	Context History, Log Monitor tool		User Profile Tool (it does not link devices and users) AAL Space Profiles	Only in supported technologies (potentially, ZigBee, Bluetooth Continua Alliance, KNX). It does not provies an API to globally manage this features.
OneM2M	Y				
Microsoft Azure IoT	Y	Y	Y	Y	Y
	Device Provisioning	Device Identity Store	Device State Store	Device State Store	Device State Store
Amazon AWS IoT	Y		Y	P	P
	Registry		AWS Console	Only devices	AWS monitoring tools + dashboard
All-Joyn					
Butler	Y	Y	Y	Y	
	System/Device Management Layer	SmartObject Management Portal at System/Device Management Layer	LogFile generated by SmartObject Management Portal	User&User/Device Directory at Communication Layer, Data/Context Management Layer and Service Layer	
i-Core		Y	Y	Y	
		An Alarm System monitors unexpected events	At CVO level	CVO Management Unit	
Sofia 2	Y	Y	Y	Y	Y
	Acquisition Layer	Auditoría	Dashboard	Acquisition Layer	Data Engines
ThingSpeak	Y				
	Basic channel definition				
GE Predix	Y	Y	Y	Y	Y
	Connectivity, Edge Manager	Anomaly Detection	Analytics Runtime	Edge Manager,	Analytics Runtime/ Catalog/ Framework
IBM Watson IoT	Y	Y	Y		X
	Watson Dashboard	CA Nimsoft (multiple levels)	CA Nimsoft (multiple levels)		Watson
WSO2	Y	Y	Y		Y
	App Manager Enter. Mobility Mgr	Application Server Governance Reg.	Application Mgr. Governance Reg.	App Manager Enter. Mobility Mgr	Application Mgr.
Contiki	P		P		P
	Configuration has to be done manually		Has to be done manually		has to be done manually

6.3.3 Service Organization

	Service Organization Used for composing and orchestrating Services of different levels of abstraction		
Platform	Service Composition	Service Orchestration	Service Choreography
FIWare	Y	Y	
	Mashup	Mashup	
Open IoT	Y	Y	Y
	Request definitio	Scheduler	Service Delivery & Utility Manager
UniversAAL	Y	Y	Y
	Middleware has a dedicated service bus.	Middleware service bus. It does not provide a tool for orchestrating, but composition is easy in the lifecycle.	Services can subscribe to other services profiles, so that the middleware automatically resolves the publication to the interested services.
OneM2M			Y
Microsoft Azure IoT		Y	Y
		Stream Processors	Stream Processors
Amazon AWS IoT	Y	Y	
	AWS marketplace like AppSymphony	AWS Data Pipeline + marketplace	
All-Joyn			
Butler	Y	P	Y
	SmartObject Gateway enables the (not-autonomic) service composition at the ServiceLayer (specifically at Data&ServiceDiretory) exploiting	Service Orchestration foreseen but not detailed (inherited by IoT-A)	MQTT broker in ContextManagement Portal at Data/Context Management Layer
i-Core	Y	Y	Y
	CVO Level	At VO and CVO Level	MQTT broker at VO and CVO levels
Sofia 2			
ThingSpeak		Y	P
			Offers channels that acts as brokers.
GE Predix	Y	Y	
	Analytics Fwk, App Composer, workflow	Analytics Fwk, Workflow	
IBM Watson IoT	Y	Y	Y
	IBM Watson IoT Platform	IBM Watson IoT Platform	message hub
WSO2	Y	Y	Y
	Application Mgr. Data Services Server	WSO2 ESB	Message Broker
Contiki			

6.3.4 IoT Process Management

	IoT Process Management To provide the functional concepts necessary to conceptually integrate the idiosyncrasies of the IoT world into traditional (business) processes. The different roles of the business objects and users will be defined here	
Platform	Process Modelling	Process Execution
FIWare		
Open IoT	Y	Y
	Scheduler	Service Delivery & Utility Manager
UniversAAL	P	
	It has tools to model domain-specific(health & telecare) businessess. Tools are vert contrained and the support is weak.	
OneM2M	Y	
Microsoft Azure IoT	Y	Y
	Service Assisted Communication	Service Assisted Communication
Amazon AWS IoT		
All-Joyn		
Butler	P	P
	Business Process Modeling foreseen but not detailed (inherited by IoT-A)	Business Process Execution foreseen but not detailed (inherited by IoT-A)
i-Core	Y	Y
	Agile-based IoT process modeling	
Sofia 2		
ThingSpeak	Y	Y
GE Predix	Y	Y
	Analytics Fwk, Workflow, enterprise connect	Analytics Fwk, Workflow, enterprise connect
IBM Watson IoT		
WSO2	Y	Y
	Business Process Server WSO2 Developer Studio	Business Process Server Data Services Server
Contiki		

6.3.5 Virtual Entity

	Virtual Entity Functions for interacting with the IoT System on the basis of VEs, as well as functionalities for discovering and looking up services about VEs.			
Platform	VE Resolution	VE & IoT Service Monitoring	VE Service	
FIWare	Y	Y	Y	Y
	IoT Discovery + Backend Dev. Mgmt.	Auto-register	Through Orion + IoT Broker	IoT Broker
Open IoT	Y	Y	Y	Y
	Service Delivery & Utility Manager	GSN service, monitoring	Cloud	W3C Semantic Sensor Networks (SSN)
UniversAAL	Y		P	Y
	Services are declared in a semantic way, so they always are bound to virtual representation of devices or services. They can be bound to instances of them.		Universaal is fundamentally build on the virtual domain, so that creating, reading and updating a virtual entity are easy operations. However, the binding with the physical devices strongly depends on the physical layer, which is not so developed.	Universaal is semantic native
OneM2M				Y
Microsoft Azure IoT	Y	Y	Y	
	Device Entity Store	Device Entity Store	Device Entity Store	
Amazon AWS IoT			Y	
			Things Shadow	
All-Joyn	Y	Y	Y	Y
Butler	Y	Y	Y	Y
	follows IoT-A specifications	follows IoT-A specifications	follows IoT-A specifications	W3C SSN ontology
i-Core	Y	Y	Y	Y
	At VO and CVO Level	At VO and CVO Level		The Web Ontology Language (OWL) and Resource Description Framework (RDF) for describing VOs
Sofia 2	Y		Y	Y
	Acquisition Layer		Acquisition Layer	KP's
ThingSpeak			P	Y
			Define the sensors ans its values but not as an entity but as a channel	Existing + custom fields
GE Predix	Y		Y	Y
	Asset, Edge Manager		Digital Twins	Usr Defined Domain Objs.
IBM Watson IoT	Y		Y	Y
	Watson		Watson	Device Type
WSO2	Partially	Partially	Partially	Partially
	Data Analytics Server Data Services Server	Data Analytics Server Data Services Server	Data Analytics Server Data Services Server	Data Analytics Server Data Services Server
Contiki				

6.3.6 IoT Service

	IoT Service Discovery, look-up, and name resolution of IoT Services. IoT Services can be used to get information provided by a resource retrieved from a sensor device															
	IoT Service											IoT Service Resolution.				
	Query information	**Update information**	**Use resource operation/service**	**Subscribe to information**	**Subscription with filters**	**Registration**	**Historic data access**	**CEP**	**Big data storage**	**Others**	**IoT Client**	**Discovery**	**Lookup**	**Service Id. Resolution**	**Service Descr. Mgmt.**	**Others**
FIWare	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	Y	Y	
	Orion/IoT Broker	Orion + IoT Agents/Broker	Orion + IoT Agents/Broker	Orion	Orion	Orion + IoT Agents/Broker	Cosmos	Proton, with basic geo capabilities	Cosmos	Data visualization, marketplace,...	Examples exist	IoT Discovery	IoT Discovery	IoT Discovery + IoT Broker	Repository + IoT Discovery	
Open IoT	Y	Y	?	Y	Y	Y	P		P			Y		?	Y	
UniversAAL	Y	Y	?	Y	Y	Y	Y	Y		Y	P	Y	P	Y	Y	
										UI, Security, Marketplace...	Technology-specific: ZigBee, Continua, Z-Wave, KNX	Strongly bound to physical interface	In the supported protocols			
OneM2M	Y	Y		Y		Y						Y				
Microsoft Azure IoT	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y
	Stream Analytics + Apache *	Stream Analytics + Apache *	Stream Analytics + Apache *	Stream Analytics + Apache *	Stream Analytics + Apache *	Stream Analytics + Apache *	Stream Analytics + Apache *	Stream Analytics + Apache *	Stream Analytics + Apache *		IoT Client	IoT Client	IoT Client	IoT Client	IoT Client	IoT Client
Amazon AWS IoT	Y	Y		Y	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
					Not attributes		AWS services	Rules engine	AWS services	All AWS Services	SDK					
All-Joyn		Y	Y	Y		Y		Y		Y						
		AllJoyn Core Permission Management	AllJoyn Core Permission Management	AllJoyn Security 2.0		AllJoyn Core Permission Management		With some AllJoyn Key Exchange		AllJoyn Security 2.0						
Butler	P	Y	Y	Y	P	Y		Y	Y			Y	Y	P	Y	
	SmartObject data can be queried	ServiceDirectory at Service Layer			datastream can be filtered by CEP engine	Data and Service Directory at Service Layer		FIWARE-like CEP components (CEP task, CEP applications, CEP engine) feeded by data coming from SOs and distributed among the BUTLER architecture	Persistent Storage (a functional component implementing a NoSQL database) at Data / Context Management Layer			Data and Service Directory at Service Layer	BUTLER SmartObject Gateway uses the OSGi registry at Service Discovery within Service Layer	see IoT-A	service descriptions managed into the Service Directory at Service layer	
i-Core	Y	Y	Y	Y	Y	Y		Y				Y	Y	Y	Y	
								Esper CEP engine for the SmartOffice, WebMethods CEP engine for the SmartBusiness, both inro CVO Container				VO and CVO Registry	VO and CVO Registry	VO and CVO Registry	VO and CVO Registry	
Sofia 2	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
ThingSpeak	Y	Y		Y		Y	Y	P	Y	Y			Y			
				IoT Analytics				React App					Basic			
GE Predix	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	Asset/ Time Series	Asset/ Time Series	Asset/ Time Series	EventHub/ RabbitMQ	EventHub/ RabbitMQ	Asset	Time Series		Redis	Lots of services	Machine	Asset/ Time Series	Asset/ Time Series	Asset/ Time Series	Asset/ Time Series	PaaS
IBM Watson IoT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
	Watson	Watson	Watson	Watson	Watson	Watson	Watson	Node-Red IBM streams	OS NEXUS		Watson	Service Discovery	Service Discovery	Service Discovery + Bluemix	Service Discovery + registry	
WSO2	Y	Y	Y	Y	Y	Y	Y	Y	Y			Y	Y			
	Data Services Server	Data Services Server	Data Services Server	Message Broker	Message Broker	Message Broker	Data Services Server	Complex Event Processor	Data Services Server; Data Analytics			Governance Reg.	Governance Reg.	Governance Reg.	Governance Reg.	
Contiki																

6.3.7 Security

	Security Responsible for ensuring the security and privacy																	
	Authorisation				Key Exchange & Management						Trust & Reputation			Identity Management		Authentication		
Platform	**Authorization**	**Policy Mgmt. (RBAC)**	**Specific technology**	**Others**	**Encription (SSL,...)**	**TLS**	**X.509**	**Specific Sw**	**Key Provision**	**Others**	**Request trust**	**Provide Trust**	**Others**	**Identify Management**	**Others**	**Authentication**	**Specific technology**	**Others**
FIWare	Y	Y	Y		Y	Y	Y		Y		Y	Y	Y	Y	Y	Y		Y
	AuthZForce	PDP, PEP	XACML 3.0		In access control	In access control	In access control				Software trustworthy	Software trustworthy	Certification workflow	KeyRock + Privacy	SSO		SCIM 2.0, Oauth 2.0	IoT Agents / IoT Broker
Open IoT	Y	Y												Y		Y		
UniversAAL	Y	Y	Y											Y		Y	Y	
																	Multi-factor authentication and session management	
OneM2M	Y				Y	Y			Y		Y	Y				Y		
						TLS-PSK												
Microsoft Azure IoT	Y					Y	Y							Y		Y		
	X.509 Compliant															AzureActive Directory		
Amazon AWS IoT	Y	Y	Y		Y	Y	Y		Y					Y		Y	Y	
			Amazon Cognito														IAM	
All-Joyn	Y								Y	Y	Y	Y						
Butler	Y	Y			Y	Y	Y		Y		P	P		P		Y		
	Authorization Server with OAuth2.0	SmartObject Management Portal			communications between SmartMobile and core BUTLER SmartServers	messages between Sos and Authorization Server are transported over TLS session	The TLS session performs Server Authentication based on an X509 server certificate		access-token and security material are generated by the Authorization Server		Foreseen in the IoT-A architecture at Security Level	Foreseen in the IoT-A architecture at Security Level		BUTLER User Profile Manager provides not-anonymous UserID		AuthenticationServer (AS)		
i-Core	Y	Y			Y	Y					Y	Y		Y		Y		
	Model-based Security Toolkit (SecKit) at Security Management Layer	Model-based Security Toolkit (SecKit) at Security Management Layer			At VO and CVO Level	At VO and CVO Level					Model-based Security Toolkit (SecKit) at Security Management Layer	Model-based Security Toolkit (SecKit) at Security Management Layer		Model-based Security Toolkit (SecKit) at Security Management Layer		Model-based Security Toolkit (SecKit) at Security Management Layer		
Sofia 2	y	y			Y	Y	Y			Y				Y		Y	Y	
										XXTEA							KP	
ThingSpeak																Y	Y	
																	API Key	
GE Predix	Y	Y			Y	Y	Y		Y	Y				Y		Y		Y
	Access control	Access control							Credential Store	Data Integrity				UAA		SAML/OOAuth 2.0/penId		Credential Store
IBM Watson IoT	Y	Y			Y	Y	Y	Y	Y							Y		
	Cloud Foundry	Single Sign On			dashDB DB2	secure gateway	examples	dashDB DB2	Beta (key protect)							Single Sign On		
WSO2	Y	Y	Y	Y	Y	Y	Y	Y	Y					Y	Y	Y	Y	Y
	WSO2-IS	RBAC ABAC (PAP, PDP, PIP)	XACML 2.0 & 3.0 Delegated	IDToken Signature			Key Stores	Carbon Server Secure	Identity Server					Service and Identity Providers	Facebook Google Yahoo	SAML2 OAuth2 OpenId	Inbound & Oubound Identity	Workflow Engine Analytics
Contiki																		

6.3.8 Communication

	Communication It's an abstraction, modelling the variety of interaction schemes derived from the many technologies belonging to IoT systems and providing a common interface to the IoT Service FG																			
	Hop To Hop Communication	Network Communication	End To End Communication																	
Platform			**HTTP**	**MQTT**	**AMQP**	**WiFi**	**LAN**	**PLC**	**Bluetooth**	**CoAP**	**ZigBee**	**UPnP**	**KNX**	**SunSpot**	**TST**	**ZIGPOS**	**OMS**	**M2M**	**WebSocket**	**LWM2M**
FIWare	Y	Y	Y	Y	P					Y										Y
		OFNIC																		
Open IoT			Y	Y						Y										
UniversAAL	Y	P	Y							P	Y		Y							
	LDDI	Theoretically it allows in particular conditions																		
OneM2M		P	Y	Y						Y									Y	Y
		Has a routing module but not form multicast, etc																		
Microsoft Azure IoT			Y	Y	Y															
Amazon AWS IoT			Y	Y															Y	
All-Joyn						Y	Y	Y	Y											
Butler	Y	P	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y		
	IoT Protocol Adapters	Foreseen in the IoT-A architecture at Communication Level																		
i-Core			Y	Y		Y			Y	Y	Y	Y					Y	Y		
Sofia 2	Y		Y	Y		Y	Y	Y	Y										Y	
	IoT Gateway																			
ThingSpeak	Y		Y	P																
	Very basic		For create and update	Just for update the channel																
GE Predix	Y	Y	Y	Y	Y	Y	Y													
	Asset/Edge Manager	Connectivity																		
IBM Watson IoT			Y	Y																
WSO2		Y	Y	Y	Y	Y													Y	
		WSO2 ESB																		
Contiki	Y	Y	Y	P		P	P	Y	P	Y	Y	Y								
						not recommended	not recommended		some implementation required											

6.3.9 Devices

	Devices				
Platform	**IP Capable**	**Constrained**	**Gateway**	**HTTP enabled**	**CoAP enabled**
FIWare	Y	Y		Y	Y
Open IoT				Y	Y
UniversAAL			Y	Y	
OneM2M				Y	Y
Microsoft Azure IoT	Y	Y			
Amazon AWS IoT	Y		Y	Y	
All-Joyn	Y				
Butler	Y	Y	Y	Y	Y
i-Core	Y	Y	Y	Y	Y
Sofia 2	Y		Y	Y	
			KP		
ThingSpeak				Y	
GE Predix	Y	Y	Y	Y	
IBM Watson IoT	Y		Y		
WSO2				Y	
Contiki	Y	Y	Y	Y	Y

7 References

[1] MacKenzie et al. Reference Model for Service Oriented Architecture 1.0 http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf OASIS Standard, 12 October 2006

[2] Bass, Len. Software architecture in practice. Pearson Education India, 2007

[3] Nick Rozanski, Eoin Woods. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley, 2005.

[4] Shames, P. and Yamada, T. Reference architecture for space data systems. s.l.: DSpace at Jet Propulsion Laboratory [http://trsnew.jpl.nasa.gov/dspace-oai/request] (United States), 2004

[5] Marek Obitko (advisor Vladimir Marik): Translations between Ontologies in Multi-Agent Systems, Ph.D. dissertation, Faculty of Electrical Engineering, Czech Technical University in Prague, 2007.

[6] Muller, G. A reference architecture primer, (2008). Available at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.5696&rep=rep1&type=pdf

[7] Haller S. The Things in the Internet of Things. Tokyo: s.n, 2010. Available at http://www.iot-a.eu/public/news/resources/TheThingsintheInternetofThings_SH.pdf

[8] Basic Geo (WGS84 lat/long) vocabulary (https://www.w3.org/2003/01/geo/)

[9] INTER-IoT Project (http://www.inter-iot-project.eu)

[10] oneM2Mstandards for M2M and the Internet of Things (http://www.onem2m.org/)

[11] Semantic Sensor Network XG nal report (2011) (https://goo.gl/aaTcSf)

[12] SmartM2M; Smart Appliances; Reference Ontology and oneM2M mapping. Technicalspecification 103 264, European Telecommunications Standards Institute (2015)

[13] Ben Alaya, M., Medjiah, S., Monteil, T., Drira, K.: Towards Semantic Data Interoperability in oneM2M Standard. IEEE Communications Magazine 53(12), pp. 35-41 (Dec 2015) (https://hal.archives-ouvertes.fr/hal-01228327)

[14] Bermudez-Edo, M., Elsaleh, T., Barnaghi, P., Taylo, K.: IoT-Lite: A lightweight semantic model for the Internet of Things. In: Proc. of the IEEE Conferences on Ubiquitous Intelligence & Computing, July 2016, Toulouse, France

[15] Compton, M., Barnaghi, P., Bermudez, L., Garcia-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W.D., Phuoc, D.L., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., Taylor, K.: The SSN ontology of the W3C semantic sensor network incubator group. Web Semantics: Science, Services and Agents on the World Wide Web 17, pp. 25-32 (2012) (https://goo.gl/urwO7g)

[16] Daniele, L., den Hartog, F., Roes, J.: Created in close interaction with the industry: The Smart Appliances REFerence (SAREF) ontology. In: Cuel, R., Young, R. (eds.) Formal Ontologies Meet Industry: Proc. of the 7th Int. Workshop, FOMI 2015, Berlin, Germany, August 5, 2015. pp. 100-112. Springer (2015)

[17] Ganzha, M., Paprzycki, M., Pawłowski, W., Szmeja, P., Wasielewska, K.: Semantic interoperability in the Internet of Things: an overview from the INTER-IoT perspective (in press). Journal of Network and Computer Applications (2016)

[18] Ganzha, M., Paprzycki, M., Pawłowski, W., Szmeja, P., Wasielewska, K.: Towards semantic interoperability between Internet of Things platforms (submitted for publication). Springer (2016)

[19] Ganzha, M., Paprzycki, M., Pawłowski, W., Szmeja, P., Wasielewska, K., Fortino, G.: Tools for ontology matching—practical considerations from INTER-IoT perspective. In: Proc. of the 8th Int. Conference on Internet and Distributed Computing Systems. LNCS, vol. 9864, pp. 296-307. Springer (2016)

[20] Ganzha, M., Paprzycki, M., Pawłowski, W., Szmeja, P., Wasielewska, K., Palau, C.E.: From implicit semantics towards ontologies—practical considerations from the INTER-IoT perspective (submitted for publication). In: Proc. of 1st edition of Globe-IoT 2017: Towards Global Interoperability among IoT Systems (2017)

[21] Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing? International journal of human-computer studies 43(5), pp. 907-928 (1995)

[22] Jayaraman, P.P., Calbimonte, J.P., Quoc, H.N.M.: The schema editor of OpenIoT for semantic sensor networks. In: Kyzirakos, K., Henson, C.A., Perry, M., Varanka, D., Grütter, R., Calbimonte, J.P., Celino, I., Valle, E.D., Dell'Aglio, D., Krötzsch, M., Schlobach, S. (eds.) Proc. of the 1st Joint Int. Workshop on Semantic Sensor Networks and Terra Cognita (SSN-TC 2015) and the 4th Int. Workshop on Ordering and Reasoning (OrdRing 2015) co-located with the 14th Int. Semantic Web Conference (ISWC 2015), Bethlehem, PA, United States, October 11-12th, 2015. CEUR Workshop Proceedings, vol. 1488, pp. 25-30. CEUR-WS.org (2015)

[23] Soldatos, J., Kefalakis, N., Hauswirth, M., Serrano, M., Calbimonte, J.P., Riahi, M., Aberer, K., Jayaraman, P.P., Zaslavsky, A., Podnar Žš arko, I., Skorin-Kapov, L., Herzog, R.: OpenIoT: Open source Internet-of-Things in the Cloud. In: Podnar Žarko, I., Pripužic, K., Serrano, M. (eds.) Interoperability and Open-Source Solutions for the Internet of Things. LNCS, vol. 9001, pp. 13-35. Springer-Verlag (2015)

[24] Vrandečić, D.: Ontology Evaluation, pp. 293-313. Springer Berlin, Heidelberg (2009) (http://dx.doi.org/10.1007/978-3-540-92673-3_13)

[25] Bassi, A., Bauer, M., Fiedler, M., Kramp, T., van Kranenburg, R., Lange, S., Meissner, S., eds.: Enabling Things to Talk—Designing IoT solutions with the IoT Architectural Reference Model, Springer-Verlag (2013)

[26] Maria Ganzha, Marcin Paprzycki, Wiesław Pawłowski, Paweł Szmeja, Katarzyna Wasielewska, and Carlos E. Palau. From implicit semantics towards ontologies—practical considerations from the INTER-IoT perspective. In Proceedings of 1st edition of Globe-IoT 2017: Towards Global Interoperability among IoT Systems. Accepted for publication, 2017.

[27] Eclipse OneM2M site https://wiki.eclipse.org/OM2M/one

[28] Sofia2 site http://sofia2.com/

[29] ThingSpeak Help site https://es.mathworks.com/help/thingspeak/

[30] IOT-A D1.5 Final Architectural Reference Model for the IoT http://www.iot-a.eu/public/public-documents/d1.5/at_download/file

[31] McGuinness, Deborah L., and Frank Van Harmelen. "OWL web ontology language overview." W3C recommendation 10.10 (2004): 2004.

[32] Ronak Sutaria and Raghunath Govindachari from Mindtree Labs in "Making sense of interoperability:Protocols and Standardization initiatives in IOT

[33] The IoT ARM reference manual, Fraunhofer, Martin Bauer et al. http://publica.fraunhofer.de/dokumente/N-276076.html

[34] Pras, A. Network Management Architectures. ISSN 1381-3617. PhD Thesis, University of Twente.