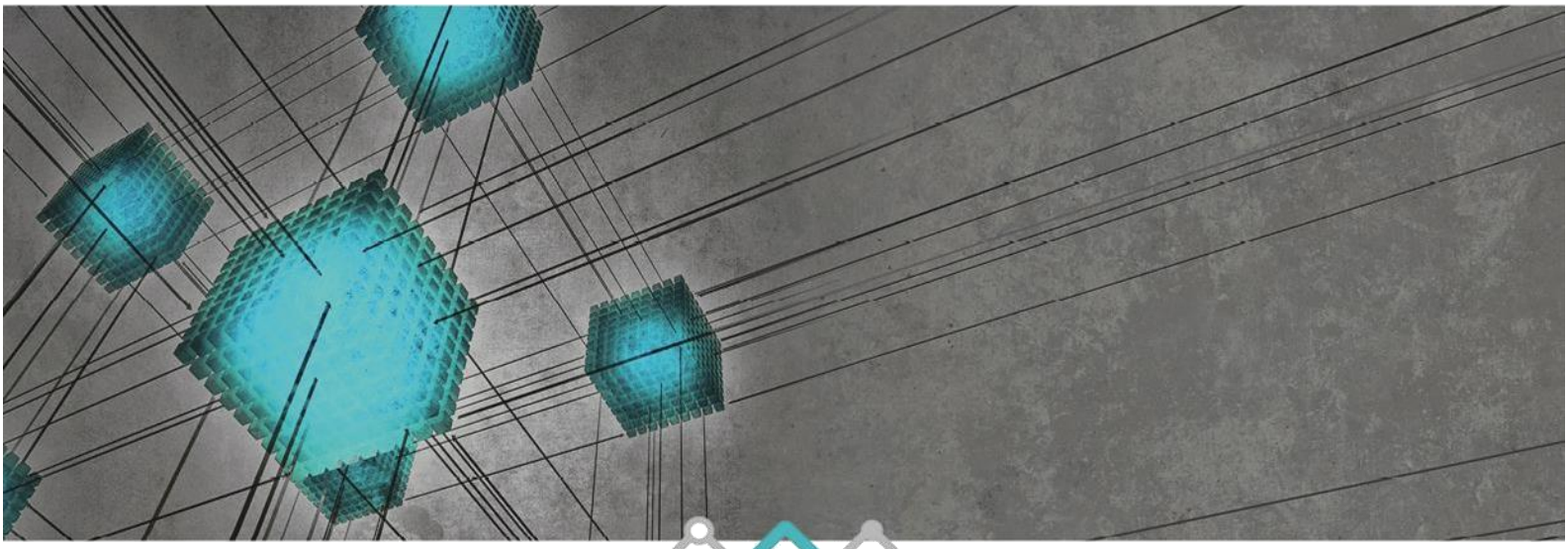

interiot

INTEROPERABILITY
OF HETEREOGENEUS
IOT PLATFORMS.

D3.1.

Methods for Interoperability and Integration.

December 2016

INTER-IoT

INTER-IoT aim is to design, implement and test interoperability tools, a framework and a methodology 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-layer approach to the integration of different IoT devices, networks, platforms, services and applications will allow a global continuum of data, infrastructures and services. Additionally, a reuse and integration of existing and future IoT systems will be facilitated, enabling the creation of a de facto global ecosystem of interoperable IoT platforms.

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

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

INTER-IoT

Methods for Interoperability and Integration.

Version: 1.0

Security: Public

December 31, 2016

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

Disclaimer

Executive Summary

The aim of Deliverable 3.1, entitled “Methods for Interoperability and Integration”, is to document the initial collection of the INTER-LAYER interoperability mechanisms. The deliverable is the first version of a series of three (i.e. D3.1 will be followed by D3.2 and D3.3). The deliverable reports the initial version of the interfaces, entities and block diagrams. It reports the technical work performed in five tasks, T3.1 (Definition and Analysis of Methods for Device Layer Interoperability and Integration, M5-M30); T3.2 (Definition and Analysis of Methods for Networking Layer Interoperability and Integration, M5-M30); T3.3 (Definition and Analysis of Methods for Middleware Layer Interoperability and Integration, M5-M30); T3.4 (Definition and Analysis of Methods for Application Service Layer Interoperability and Integration, M5-M30) and T3.5 (Definition and Analysis of Methods for Data and Semantics Layer Interoperability and Integration, M5-M30). Next versions of the deliverable will include also the work developed in T3.6 (Definition and Analysis of Methods for Cross-Layer Interoperability and Integration, M13-M30).

The methods have been based in the initial description of the layered architecture provided in the Description of the Action, and have been built based on the requirements gathered in the execution of T2.3 and delivered in D2.3 (INTER-IoT Requirements and Business Analysis, M9). The different components of INTER-LAYER described in the deliverable have been influenced by the stakeholders and the use cases and scenarios described in D2.4 (Use cases and scenarios, M12) in order to be in line with the proposed pilots.

INTER-IoT addresses a challenging objective to create an interoperable Internet of Things (IoT) ecosystem that will allow for the collaboration of vertical IoT platforms towards the creation of cross-domain applications. Thus, it designs an interoperable mediation component (i.e INTER-LAYER to enable the discovery and sharing of connected devices across existing and future IoT platforms for rapid development of cross-platform IoT applications. INTER-IoT allows flexible and voluntary interoperability at different layers. This layered approach can be achieved by introducing an incremental deployment of INTER-IoT functionality across the platform’s space, which will in effect influence the level of platform collaboration and cooperation with other platforms. INTER-IoT does not pretend to create a new IoT platform but an interoperability structure to interconnect different IoT platforms, devices, applications and other IoT artifacts.

Syntactic and semantic interoperability represent the essential interoperability mechanisms in the future INTER-IoT ecosystem, while organizational/enterprise interoperability has different structures/layers to enable platform providers to choose an adequate interoperability model for their business needs. It will be supported by INTER-FW, developed in WP4 that may allow the development of new applications and services atop INTER-LAYER and INTER-METH, developed in WP5, to provide a methodology in order to coordinate interoperability.

The document lists an initial collection of the interoperability mechanisms, building blocks and interfaces supported by sequence diagrams with a special focus in three of the layers: Device to Device layer with the virtual gateway; Middleware layer with the MW2MW component and the Data and Semantics layer with the Inter Platform Semantic Mediator (IPSM). The Network to Network layer provides an initial solution based on SDN and NFV, however it will require the final implementation of the virtual gateway to be completed. And also the Application and Services layer may require the results from the M2MW and IPSM in order to implement the existing solution.

The requirements and prioritization from the feedback gathered from stakeholders indicated that the mainly needed interoperability mechanisms are related with the device, middleware and semantics

layers, however the other two layers (i.e. Network and Application and Services) have also been addressed and mechanisms researched and proposed.

We report a comprehensive list of mechanisms and components and put them in relation to requirements and software components in the different layers. The main task of these components is to facilitate syntactic and semantic interoperability of IoT platforms at the identified layers. Each interoperability component may offer an API in order that the upper layers in an incremental approach or applications and services may use the interoperability mechanisms. In this document we include sequence diagrams depicting component interaction for interoperability.

The document also reports an initial view on components and envisioned functionality needed for the INTER-LogP and INTER-Health domains. The aim in those two domain-specific solutions is to offer dynamic access and configuration of devices in environments hosting a number of platforms. Additionally, we consider INTER-DOMAIN as the environment in which both domains and platforms from the open call will interoperate.

Finally, the document analyzes relevant work in the area of IoT interoperability, with focus on interoperability mechanisms; projects with similar goals as INTER-IoT, and platforms by INTER-IoT partners aiming to become part of the future INTER-IoT ecosystem. We can conclude that the proposed functional layered stack with five layers (D2D, N2N, MW2MW, AS2AS and DS2DS) is in accordance with the AIOTI proposed architecture, although INTER-IoT plans to extend different aspects, e. g. related with platform federation, roaming, or offloading.

List of Authors

Organisation	Authors	Main organisations' contributions
VPF	Miguel Llop, Pablo Giménez, Alexandre Sánchez, Mª Luisa Escamilla, Eduardo Olmeda	INTER-LogP influence in the layers Requirements review and link MW2MW, AS2AS and DS2DS contrib.
UPV	Carlos E. Palau, Benjamín Molina, Eneko Olivares, Regel González-Usach, Andreu Belsa, Jara Suárez de Puga, Blanca Terol	Coordination of the document D2D, N2N, MW2MW, AS2AS, DS2DS contribution AS2AS coordination
UNICAL	Giancarlo Fortino, Wilma Russo, Gianluca Aloi, Pasquale Pace, Raffaele Gravina	D2D, N2N, MW2MW contribution
PRO	Miguel Montesinos, Christophe Joubert, Amelia del Rey, Miguel A. Llorente	MW2MW, AS2AS, DS2DS contribution Link with INTER-FW Software architecture coordination
TU/e	George Exarchakos, Antonio Liotta, Tim van der Lee	D2D, N2N and AS2AS contribution
XLAB	Mariano Cecowski, Robert Plestenjak	D2D, MW2MW and DS2DS contrib. MW2MW coordination
SRIPAS	Katarzyna Wasielewska-Michniewska, Paweł Szmeja, Wiesław Pawłowski, Maria Ganzha, Marcin Paprzycki	D2D, N2N, MW2MW, AS2AS, DS2DS contribution DS2DS coordination
RINICOM	Garik Markarian Eric Carlson	D2D, N2N, MW2MW, AS2AS, DS2DS contribution N2N coordination
TI	Carlo Aldera, Alberto Delpiano, Fabio D'Ercoli, Giovanna Larini	D2D and N2N contribution INTER-Health influence in the layers
NEWAYS	Ron Schram, Roel Vossen, Johan Schabbink, Frans Gevers	D2D, N2N, MW2MW contribution D2D coordination
ABC	Alessandro Bassi, Jitka Slechtova	D2D, N2N, MW2MW contribution

Change control datasheet

Version	Changes	Chapters	Pages
0.1.0	Created base structure	-	20
0.2.0	Completed introduction and first version state of the art	1, 2	121
0.3.0	First version of specifications	3	163
0.3.1	Second version of state of the art. Completed conclusions.	2, 4	183
0.3.2	Second version of specifications. Completed annex.	3, 6	207
0.3.3	Completed missing parts. Format whole document.	All	215
1.0	Reviewed whole document. Minor changes. Ready for submission.	All	222

Contents

List of Figures

Acronyms

AIOTI	Alliance for Internet of Things Innovation
BIP	Best Ideas and Projects
EC	European Commission
IERC	European Research Cluster on the Internet of Things
EPI	European Platforms Initiative
INTER-LAYER	INTER-IoT Layer integration tools
INTER-FW	INTER-IoT Interoperable IoT Framework
INTER-METH	INTER-IoT Engineering Methodology
INTER-LogP	INTER-IoT Platform for Transport and Logistics
INTER-Health	INTER-IoT Platform for Health monitoring
INTER-META-ARCH	INTER-IoT Architectural meta-model for IoT interoperable platforms
INTER-META-DATA	INTER-IoT Metadata-model for IoT interoperable semantics
INTER-API	INTER-IoT Programming library
INTER-CASE	INTER-IoT Computer Aided Software Engineering tool for integration
INCOSE	The International Council on Systems Engineering
IoT	Internet of Things
ITU	International Communications Union
SDO	Standard Development Organisation
SDR	Software Defined Radio
IOT-A	Internet of Things - Architecture
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
SPEM	Software and Systems Process Engineering Meta-model
M2M	Machine to Machine
RFID	Radio Frequency IDentification
MAC	Media Access Control address
SDN	Software Defined Networking
W3C	World Wide Web Consortium
SSN	Semantic Sensor Network
SAREF	Smart Appliances REFerence
OGC	Open Geospatial Consortium
LTE	Long-Term Evolution networks
DSL	Digital Subscriber Lines

CAN	Controller Area Network
API	Application Programming Interface
CRUD	Create, Read, Update and Delete
SDO	Standards Developing Organization
GOIoTP	Generic Ontology for IoT Platforms
SDN	Software Defined Network
SDR	Software Defined Radio
NFV	Network Function Virtualization
AGC	Automatic Gain Control
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
MVC	Model, View, Controller
IoS	Internet of Services
QoS	Quality of Service
QoE	Quality of Experience

1 Introduction

Internet of things (IoT) covers a wide range of devices, protocols, technologies, networks, middleware, applications, systems and data that present vast heterogeneity. As a consequence of this heterogeneous nature and the lack of a global IoT standard, instead of the achievement of seamless integration among IoT systems, vertical silos proliferate. Interconnection and interoperability are critical aspects in IoT, as well as one of the most difficult challenges to face in IoT environments.

Whereas integration between heterogeneous elements is usually done at device or network level, and it is limited to data collection, INTER-IoT offers a layer-oriented solution to allow interoperability among IoT platforms and systems at different layers or levels. INTER-IoT will offer an open framework (INTER-FW) and methodology (INTER-METH) to guarantee a seamless integration of heterogeneous IoT technologies. This solution can be applied to any application domain and across domains in which there is a need for interconnection and/or interoperability.

INTER-IoT will facilitate the formation of interoperable IoT ecosystems, make the design of IoT devices, smart objects or services easier to companies and developers, and support launching them to the market quickly to a broader client base. In the long term, the ability for applications to connect to and interact with heterogeneous smart objects, will become a huge enabler for new products and services.

INTER-IoT benefits at different layers or levels will be:

- At the **Device level:** the seamless inclusion of new IoT devices and their interoperation with already existing heterogeneous ones, allowing a fast growth of smart objects ecosystems.
- At the **Networking level:** seamless support for smart objects mobility (roaming) and information routing. This will allow the design and implementation of fully connected ecosystems.
- At the **Middleware level:** a seamless resource discovery and management system for smart objects and their basic services, to allow the global exploitation of smart objects in large scale IoT systems.
- At the **Application and Services level:** the discovery, use, import, export and combination of heterogeneous services between different IoT platforms.
- At the **Data and Semantics level:** a common interpretation of data and information from different platforms and heterogeneous data sources, providing semantic interoperability.

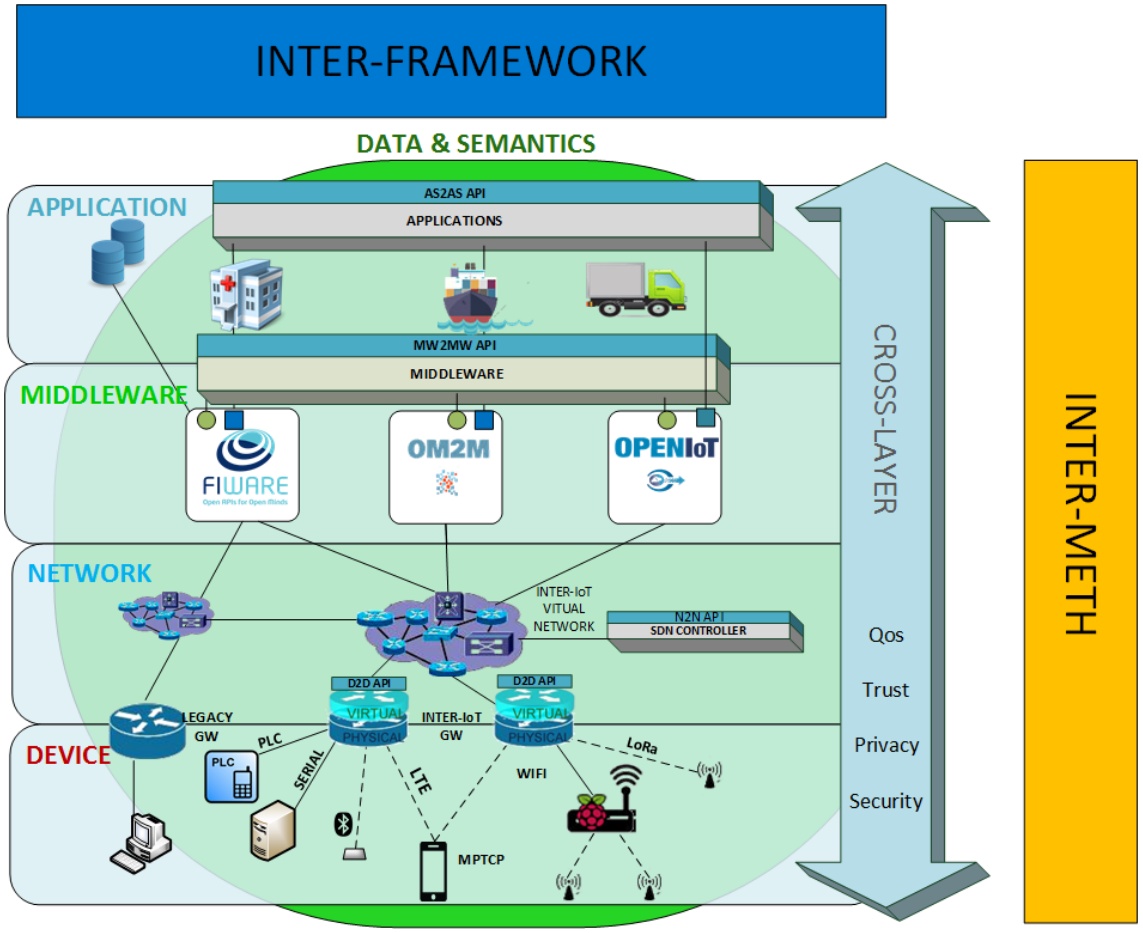

Figure 1: Layer structure of INTER-IoT layered-oriented approach (INTER-LAYER).

1.1 INTER-LAYER Overview

INTER-IoT presents a layer-oriented solution for interoperability, to provide interoperability at any layer and across layers among different IoT systems and platforms. Contrary to a more general global approach, an INTER-IoT approach has a higher potential in order to provide interoperability. It facilitates a tight bidirectional integration, higher performance, complete modularity, high adaptability and flexibility, and presents increased reliability.

This layer-oriented solution is achieved through INTER-LAYER. INTER-LAYER includes several interoperability solutions dedicated to specific layers (as depicted in Figure 1): Device-to-Device (D2D), Networking-to-Networking (N2N), Middleware-to-Middleware (MW2MW), Application & Services-to-Application & Services (AS2AS), Data & Semantics-to-Data & Semantics (DS2DS).

Each interoperability infrastructure layer has a strong coupling with adjacent layers and provides an interface. Interfaces will be controlled by a meta-level framework to provide global interoperability. Every interoperability mechanism can be accessed through an API. The interoperability infrastructure layers can communicate and interoperate through the interfaces. This cross-layering allows to achieve a deeper and more complete integration.

The different INTER-LAYER layers are detailed here:

Device layer (D2D): Currently applications and platforms are tightly coupled, preventing their interaction with other applications and platforms, sensors and actuators communicate only within one system, certain platforms do not implement some important services (i.e. discovery), or do so in an incompatible way. Roaming elements can be missing or inaccessible. IoT Device software is never platform independent as companies create proprietary software. These facts present enormous difficulties for the achievement of interoperability. At the device level, D2D solution will allow the seamless inclusion of novel IoT devices and their interoperation with already existing ones (legacy). D2D interoperability will allow a fast growth of smart objects ecosystems. As a potential solution INTER-IoT proposes a D2D gateway that allows any type of data forwarding, making the device layer flexible by decoupling the gateway into two independent parts: a physical part that only handles network access and communication protocols, and a virtual part that handles all other gateway operations and services. When connection is lost, the virtual part remains functional and will answer the API and Middleware requests. The gateway will follow a modular approach to allow the addition of optional service blocks, to adapt to the specific case.

Network layer (N2N): Currently the immense amount of traffic flows generated by smart devices is extremely hard to handle. The scalability of the IoT systems is difficult. Also creating the interconnections between gateways and platforms is a complex task. N2N solution aims to provide seamless support for smart objects mobility and information routing. It will also allow offloading and roaming, what implies the interconnection of gateways and platforms through the network. The approximation that INTER-IoT propose uses paradigms such as SDN and NFV, and achieves interoperability through the creation of a virtual network, with the support of the N2N API. The N2N solution will allow the design and implementation of fully interconnected ecosystems.

Middleware layer (MW2MW): At the middleware level INTER-IoT solution will enable seamless resource discovery and management system for the IoT devices in heterogeneous IoT platforms. Interoperability at the middleware layer is achieved through the establishment of an abstraction layer and the attachment of IoT platforms to it. Different modules included at this level will provide services to manage the virtual representation of the objects, creating the abstraction layer to access all their features and information. Among the offered services, we find the component-based interoperability solutions within the middleware based on communication using mediators, bridges and brokers, is found. Brokers are accessible through a general API. Interoperability at this layer will allow a global exploitation of smart objects in large scale multi-platform IoT systems.

Application & Services layer (AS2AS): INTER-IoT will enable the use of heterogeneous services among different IoT platforms. Our approach will allow discovery, catalogue and composition of services from different platforms. AS2AS will also provide an API as an integration toolbox to facilitate the development of new applications that integrate existing heterogeneous IoT services.

Semantics & Data layer (DS2DS): INTER-IoT solution for the DS2DS layer will allow a common interpretation of data and information among different IoT systems and heterogeneous data sources, achieving semantic interoperability. It will be based on semantic translation of IoT platforms' ontologies to/from a common IPSM modular ontology. The Inter Platform Semantic Mediator (IPSM) component will be responsible for performing ontology-to-ontology translations of the information using ontology alignments. It will be necessary to define explicit OWL-demarcated semantics for each IoT artifact that would like to interoperate, communicate and collaborate.

CROSS-LAYER will cover and guarantees non-functional aspects that must be present across all layers: trust, security, privacy, and quality of service (QoS).

1.2 Definitions and terminology

The following definitions will be explained in order to better understand contents of the next sections of this deliverable (alphabetically ordered).

CEP (Complex Event Processing)

Complex event processing is realized in analytic tools that enable processing and analysis of data on a real-time or a near-real-time basis, driving timely decision making and action.

CEP is relevant for the IoT in its ability to recognize patterns in massive data sets at low latency rates. A CEP tool identifies patterns by using a variety of techniques such as filtering, aggregation, and correlation to trigger automated action or flag the need for human intervention.

Constrained Devices

Small devices with limited CPU, memory, and power resources that can belong to a network, creating a constrained network that is characterized by unreliable or lossy communication channels, limited and unpredictable bandwidth and a highly dynamic topology.

Event

Event is something notable that happens. It can be represented as an object that encodes or records an event, generally for the purpose of computer processing. Events can contain data, are immutable, but more than one event may record the same activity. It should be noted that events are highly context dependent.

Interoperability

The characteristic of a network system or protocol whose interfaces are completely understood to work with other systems or protocols, present or future, in either implementation or access without any restrictions.

Mash-up

Composing a new service/application/data from existing applications/services (MACs; Mashable Application Components).

Ontology

Explicit specification of shared conceptualization. In the area of science and technology, this concept refers to a structure that provides a vocabulary for a domain of interest, in addition to the meaning of the entities that are present in that vocabulary. Typically, within an ontology the entities are grouped, organized into a hierarchy, related to other entities, and subdivided according to different notions of similarity.

Ontology alignment

The process of finding correspondences between two or more ontologies. The result of this process is an alignment i.e. a set of correspondences between entities (atomic alignment) or groups of entities and sub-structures (complex alignment) from different ontologies. A correspondence can be either a predicate about similarity, called a matching, or a logical axiom mapping.

Typically, the used mapping axioms are equivalence and subsumption. In practice, ontology alignment tools often state a degree of confidence for every correspondence in the mapping. An equivalence axiom with a degree of confidence is very close in meaning to a predicate about similarity (a matching). The terms mapping and matching are often not distinguished in the

terminology used by the alignment tools. A set of correspondences can be called alignment, matching, or mapping practically interchangeably.

Ontology merging

The process of combining two, or more, ontologies into one. Consequently, the resulting ontology stores knowledge from all merged ones. Merging often utilizes a set of alignments to create deep interconnections between ontologies and, in the end, merge them into one.

Semantics

The study of meaning and relationships between meanings.

Semantic interoperability

The ability of information systems and the business processes they support to exchange information with unambiguous, shared meaning. In the context of semantic interoperability, topics such as ontology alignment, matching, and merging need to be disambiguated. These terms are closely related and sometimes used interchangeably. For each there are many, sometimes overlapping, definitions.

Semantic model

Specifies the terms and concepts used to describe an area of knowledge. A semantic model usually includes concepts in the domain of interest, relationships between them, their properties, and their values. Usually this is expressed as an ontology.

Semantic Sensor Net Ontology

The Semantic Sensor Net Ontology (SSN) is a W3C standard describing sensors and observations, and related concepts, defined in OWL. It does not describe time, locations, etc.

The **Mihini** agent is an Eclipse additional that can be used as a mediator between an M2M service and applications running on an embedded gateway, while **M3DA** is a protocol optimized for the transport of binary IoT data and the management of devices by permitting the exchange of instructions and typed data between actors in a bandwidth friendly manner.

Semantic translation

Semantic translation is an application of ontology alignment. The goal is to enable one-way or two-way understanding between software artifacts that implement differing semantics. This is directly applicable to multiple domains such as IoT, bioinformatics and others, because there are competing ontologies that describe the same or a very similar area of knowledge.

Service

Loosely-coupled computing tasks communicating over the Internet that play a growing part in business-to-business interactions. Services are the mechanism by which needs and capabilities are brought together.

Service choreography

Service choreography captures collaborative processes involving multiple services and especially their interactions seen from a global perspective. While service orchestration relies on a centralized point of view of a single participant, the service choreography is based on a global perspective of the involved participants.

Service composition

A service composition is an aggregate of services collectively composed to automate a particular task. To qualify as a composition, at least two participating services plus one composition initiator need to be present.

Service orchestration

The orchestration of services is the task aiming at invoking a set of services in a structured way. It comprises a centralized entity, called the orchestration engine, that can be seen as a conductor driving his musicians.

Synchronization

Messages being sent between two parties can be exchanged in a synchronous or asynchronous way. In the synchronous scenario, the sender remains connected to the receiver until the latter confirms the successful exchange of the message. Before that confirmation is received, the sender process 'blocks' its execution abstaining from sending any further messages.

In the asynchronous scenario, the sender does not wait for the receiver to acknowledge the message but continues its execution and might send additional messages to the receiver. The receiver is then expected to send an acknowledgement to the sender once it makes sure the messages was successfully transmitted.

1.3 Constraints based on Requirements

During the first phase of the project numerous requirements have been collected either from stakeholders or from the pilot needs. These requirements became the foundation of our interoperability architecture solution. For that propose they had to be divided into several categories in order to assign them to and apply them in each of the INTER-LAYER solution tiers.

Additionally, an initial prioritization of the requirements took place, based on their importance for the achievement of the layer objectives.

In a later stage, another more specific prioritization took place based on the remarks of the stakeholders and several revisions of the requirements obtained from the first iteration. The latter prioritization was constructed under the MoSCoW methodology [1] that assigned one of four available priority values: Must, Should, Could and Not to have.

The structured set of requirements is being used for the design and composition of the solutions placed in each step of INTER-LAYER.

Prioritized and categorized requirements will be refered in the Section 3 of this document, and related with each of the components that constitute the INTER-IoT layered interoperability approach.

INTER-IoT has two pilots INTER-LogP, as an interoperable solution in the sea port scenario, for port management, and INTER-Health, associated with the domain of e-Health. These pilots will be extensively explained and detailed in the corresponding deliverables specifically devoted to them. It is important to note that a wide group of requirements had been identified specifically for each of the pilot use cases. Pilots performance and success is strongly dependent on the accomplishment of the aforementioned requirements.

The following two subsections are focused on the relation of pilot specific requirements and the proposed interoperability solution that is described in this deliverable. Below we explain how our interoperability approach intends to accomplish the relevant requirements for each pilot. In the first version of this deliverable the most critical requirements are presented, but a list of them is expected to be extended and modified in future iterations of the INTER-IoT solution.

1.3.1 Minimum requirements for the INTER-LogP pilot

The requirements of INTER-LogP pilot cover the functional and non-functional needs that our interoperability solution has to solve. Those requirements can be found in the annex organised according to the MoSCoW prioritization. In this section all requirements referenced in brackets refer to the Table 10 in section 6.1 (Annex)

Requirements under the categorization of ‘Must’ have to be implemented in the final pilot solution. It must be note that this group is essential for critical interoperability aspects [248,249], such as the access to services and resources from different IoT platforms, and the common semantic understanding between them. Other relevant aspects reflected into this categorization of maximum priority are the communication with Legacy Systems [193] and the need and importance of object virtualization [193].

All these requirements have been considered in the first iteration of the INTER-IoT interoperability solution. They had been considered in the solutions proposed in section 3, to allow the accomplishment of the most important needs. They had been considered in the proposed solutions of section 3, to allow to satisfy the most important necessities of the pilots. These needs will be solved with the interoperability solutions of each layer. As an example illustrating this statement, it can be noted that platform access is essential in D2D and N2N, that present the need for connecting with the different IoT platforms. As well MW2MW and AS2AS require to access to resources and services from other IoT platforms. Semantic support in all layers, and the integration of Semantic Mediators in MW2MW to perform semantic translations is necessary for the DS2DS solution. Virtualization is a crucial element in all layers.

INTER-LogP requirements under the Should categorization will be integrated in the final iteration of INTER-LogP, but taking into account that they can be dispensable, if a major reason justifies their exclusion. The majority of requirements from this categorization are related with interoperability [252,166,194,167], functional aspects such as alert management [84,168], communication and response time [54] and the creation and monitoring of geofences [195]. Those requirements are present in this document, although new ones may appear in further iterations of INTER-IoT solution.

Finally, it must be noted that some INTER-LogP requirements are desirable, but not strictly necessary for the development of the INTER-LogP solution. This group attain a lower priority: ‘Could’, such is the case of QoS [81]. They are referred in this deliverable as they have relevant role and significance for some important INTER-Layer aims, such as the interoperability at the network level.

1.3.2 Minimum requirements for the INTER-Health pilot

This subsection points out the most important requirements for the INTER-Health pilot and their relation with the INTER-IoT interoperability solution. Those requirements can be found in the annex organised according to the MoSCoW prioritization. In this section all requirements referenced in brackets refer to the Table 11 in section 6.1 (Annex).

'Must’ categorization correspond to requirements that have to be integrated in the final implementation of the pilot. INTER-Health requirements under this categorization are commented in

the following lines. In this group, it has been pointed out the necessity of a User Access Gateway [176], to solve interoperability needs among devices, IoT platforms and applications. As well as the need of semantic support [106], that will be solved in DS2DS layer.

Among the functional needs with maximum priority are those associated with response time and availability of sensors [71,127], as well as the ones regarding security and privacy [103,145,146,218]. These latter requirements will be applied in the first iterations of the solution, although they will be mostly developed at the CROSS-LAYER level, which is not covered here.

The requirements under the 'Should' categorization should be integrated into the final solution, but they could be dispensable and might even not be implemented if a strong reason justifies that decision. The majority of the requirements in this group are related with interoperability [107,101,102]. In particular, to the exchange of information among platforms and special measures between different platforms. They are addressed by MW2MW and AS2AS, with the semantic support of DS2DS if explicit semantic annotation of the analysed data is required.

Finally, standards for medical transfer of medical information [164] have to be taken into account, as well as, standards for handling and managing the user access [174, 177,172, 173]. It must be noted that the particular needs associated with medical environments require the existence of several strict requirements related to security and authentication, and this fact is also reflected within the set of requirements that are given above.

2 State of the Art

This section presents a state of the art in regard of interoperability. It is structured following the layer categorization of INTER-LAYER approach, as it is focused on the interoperability at the aforementioned different levels of IoT systems: Device, Network, Middleware, Application & Services, Data & Semantics.

Each subsection describes the problem, context, and currently available tools and solutions regarding interoperability for each particular layer, that may be considered in the INTER-IoT solution, together with a description of the layer. As well, a literature review is expounded. Additionally, for a quick overview of concepts referring to a particular layer, a summary table can be found at the end of each section.

2.1 Device Interoperability (D2D)

The D2D interoperability in this section is about:

- the ability to share information and services,
- the ability of two or more devices, systems or its components to exchange and use information,
- the ability of devices to provide and receive services from other devices.

In IoT this is typically achieved through a gateway. An IoT Gateway has the ability to allow different devices using the same or different access networks to communicate to other devices and through the northbound API to the middleware or other applications. These devices can be sensors, actuators or prosumers (sensor and actuator) and the gateway can provide other services such as device discovery, QoS, security, cache storage, and so on.

2.1.1 Common Approaches

In order to understand common approaches for device to device interoperability, first we have to understand the classification of devices in the IoT environment. Attending the RFC 7228 of IETF for the Terminology for Constrained-Node Networks.

Constrained devices might be gathering information in diverse configurations and sending the information to one or more servers. They might also act on information, by performing some physical action, including displaying it. Constrained devices may work under severe resource constraints such as limited battery and computing power, little memory, and insufficient wireless bandwidth and ability to communicate. Other entities on the network might have more computational and communication resources and could support the interaction between the constrained devices and applications in more traditional networks.

Despite the overwhelming variety of Internet-connected devices that can be envisioned, it may be worthwhile to have some succinct terminology for different classes of constrained devices. The class designations in Table 1 may be used as rough indications of device capabilities:

Name	Data Size (e.g RAM)	Code Size (e.g. Flash)
Class 0, C0	<< 10KB	**<< 100KB**
Class 1, C1	~10KB	**~100KB**
Class 2, C2	~50KB	**~250KB**

Table 1: classification of constrained devices. Source: http://www.rfc-base.org/txt/rfc-7228.txt

- Class 0 devices are very constrained sensor-like motes. Severely constrained in memory and processing capabilities most likely they will not have the resources required to communicate directly with the Internet in a secure manner. They will participate in Internet communications with the help of larger devices acting as proxies, gateways, or servers and will most likely be preconfigured with a very small dataset.
- Class 1 devices are quite constrained in code space and processing capabilities, such that they cannot easily talk to other Internet nodes employing a full protocol stack such as using HTTP, Transport Layer Security (TLS), and related security protocols and XML-based data representations. However, they are capable enough to use a protocol stack specifically designed for constrained nodes (such as the Constrained Application Protocol (CoAP) over UDP) and participate in meaningful conversations without the help of a gateway node. In particular, they can provide support for the security functions required on a large network. Therefore, they can be integrated as fully developed peers into an IP network, but they need to be parsimonious with state memory, code space, and often power expenditure for protocol and application usage.
- Class 2 devices are less constrained and fundamentally capable of supporting most of the same protocol stacks as used on notebooks or servers. However, even these devices can benefit from lightweight and energy-efficient protocols and from consuming less bandwidth. Furthermore, using fewer resources for networking leaves more resources available to applications. Thus, using the protocol stacks defined for more constrained devices on Class 2 devices might reduce development costs and increase the interoperability.

With respect to examining the capabilities of constrained nodes, particularly for Class 1 devices, it is important to understand what type of applications they are able to run and which protocol mechanisms would be most suitable. Because of memory and other limitations, each specific Class 1 device might be able to support only a few selected functions needed for its intended operation. In other words, the set of functions that can actually be supported is not static per device type: devices with similar constraints might choose to support different functions. Even though Class 2 devices have some more functionality available and may be able to provide a more complete set of functions, they still need to be assessed for the type of applications they will be running and the protocol functions they would need. To be able to derive any requirements, the operational scenarios need to be analyzed. Use cases may combine constrained devices of multiple classes as well as more traditional Internet nodes.

Having this constrains in mind at the device layer we can detect 2 different scenarios of interoperability:

The first scenario is the one in which the sensor and the actuator are connected to the same device and the actuator has to respond to triggers from the sensor. This kind of interoperability can be

routed in the device itself, only database values have to be synchronized with Cloud systems. The Device Manager will be responsible to handle this task and route the sensor signal to the actuator. The Device Manager will be the module that directs traffic to the appropriate port.

The other scenario is when the sensor is on one device and the actuator at another device, in this case we will always need to pass the Gateway. The middleware provides a bridge to allow different platforms to communicate to each other without interference from higher layers. Again, at device level the Device Manager will be responsible for routing the sensor signal to the northbound Gateway together with appropriate identification and other header information like sensor type, priority, etc. and maybe initiate security protocols.

2.1.2 Literature review

Some of the projects that have been working on the development of gateways with several connection interfaces and services as well as technologies for device connection are summarized in this section.

2.1.2.1 Implementation of Gateways

Eclipse Kura

Eclipse Kura[1] aims at offering a Java/OSGi-based container for M2M applications running in service gateways. Kura provides or, when available, aggregates open source implementations for the most common services needed by M2M applications. Kura components are designed as configurable OSGi Declarative Service exposing service API and raising events. While several Kura components are in pure Java, others are invoked through JNI and have a dependency on the Linux operating system.

OM2M Gateway

OM2M[2] provides an open source service platform for M2M interoperability based on the oneM2M standard. OM2M follows a RESTful approach with open interfaces to enable developing services and applications independently of the underlying network. It proposes a modular architecture running on top of an OSGi layer, making it highly extensible via plugins. 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.

2.1.2.2 Specifications and protocols

Open Mobile Alliance

OMA[3] was formed by the world's leading mobile operators, device and network suppliers, information technology companies and content providers as the industry focal point for the development of mobile service enabler specifications. OMA is a non-profit organization that delivers open specifications for creating interoperable services that work across all geographical boundaries, on

[1] http://www.eclipse.org/kura/
[2] http://www.eclipse.org/om2m/
[3] http://openmobilealliance.org/

any bearer network. OMA's specifications support the billions of new and existing terminals across a variety of wireless networks, including traditional cellular operator networks and emerging networks supporting machine-to-machine device communications for the Internet of Things (IoT).

Random phase multiple access

Random phase multiple access (RPMA)[4] is a low-power wide-area channel access method used exclusively for machine-to-machine (M2M) communication on the Internet of Things (IoT).

RPMA is a technology communication system employing direct-sequence spread spectrum (DSSS) with multiple accesses. RPMA technology employs tight transmit power control and high receiver sensitivity (-142dBm) with a total 172 dB link budget. RPMA utilizes the 2.4 GHz band, a globally available cost free spectrum. RPMA self modulates to find clear signal both on the network and device level. It is also optimized for maximum coverage and battery efficiency, as opposed to cellular which is designed for high throughput but requires a lot of power. To save battery life it has a special connection protocol in which access points ping the device, checks the device status receives any data, and then closes the connection. It is estimated that the majority of M2M and IoT connections need this kind of low data throughput, high battery life connectivity.

RPMA is currently used in 38 private networks worldwide. The 2.4 GHz spectrum is available worldwide and is cost-free to use. RPMA access points can cover 300 square miles. It would take 30 cellular towers to cover the same area. Ingenu[5], who owns RPMA, reportedly has access points covering 450 square miles each. RPMA is deployed in both the public Machine Network, as well as private networks.

RPMA's uplink is 624 kbit/s and downlink is 156 kbit/s, which is about 10 times the speed of dialup internet. When moving, RPMA's speeds drop, as is typical for wireless connections, to 2kbit/s. These speeds are adequate for the majority of IoT applications being faster than 2G and orders of magnitude faster than Sigfox.

SWAP/HomeRF

Initially called Shared Wireless Access Protocol (SWAP) and later just HomeRF[6], this open specification allowed PCs, peripherals, cordless phones and other consumer devices to share and communicate voice and data in and around the home without the complication and expense of running new wires. HomeRF combined several wireless technologies in the 2.4 GHz ISM band, including IEEE 802.11 FH (the frequency-hopping version of wireless data networking) and DECT (the most prevalent digital cordless telephony standard in the world) to meet the unique home networking requirements for security, quality of service (QoS) and interference immunity.

CoAP

Constrained Application Protocol (CoAP)[7] is a software protocol intended to be used in very simple electronics devices, allowing them to communicate interactively over the Internet. It is particularly targeted for small, low-power sensors, switches, valves and similar components that need to be controlled or supervised remotely, through standard Internet networks. CoAP is an application layer

[4] __, "Random Phase Multiple Access", available at https://goo.gl/sGZMQR, last revised 15 Dec. 2016
[5] http://www.ingenu.com/
[6] __, "HomeRF", available at https://en.wikipedia.org/wiki/HomeRF, last revised 15 Dec. 2016
[7] __, "Constrained Application Protocol", available at https://en.wikipedia.org/wiki/Constrained_Application_Protocol, last revised 15 Dec. 2016

protocol that is intended for use in resource-constrained internet devices, such as WSN nodes. CoAP is designed to easily translate to HTTP for simplified integration with the Web, while also meeting specialized requirements such as multicast support, very low overhead, and simplicity. Multicast, low overhead, and simplicity are extremely important for Internet of Things (IoT) and Machine-to-Machine (M2M) devices, which tend to be deeply embedded and have much less memory and power supply than traditional internet devices have. Therefore, efficiency is very important. CoAP can run on most devices that support UDP or a UDP analogue.

The Internet Engineering Task Force (IETF) Constrained RESTful environment (CoRE) Working Group has done the major standardization work for this protocol. In order to make the protocol suitable to IoT and M2M applications, various new functionalities have been added. The core of the protocol is specified in RFC 7252; important extensions are in various stages of the standardization process.

XMPP

Extensible Messaging and Presence Protocol (XMPP)[8] is a communications protocol for message-oriented middleware based on XML (Extensible Markup Language). It enables the near-real-time exchange of structured yet extensible data between any two or more network entities. Originally named Jabber, the protocol was developed by the Jabber open-source community in 1999 for near real-time instant messaging (IM), presence information, and contact list maintenance. Designed to be extensible, the protocol has been used also for publish-subscribe systems, signalling for VoIP, video, file transfer, gaming, the Internet of Things (IoT) applications such as the smart grid, and social networking services.

Unlike most instant messaging protocols, XMPP is defined in an open standard and uses an open systems approach of development and application, by which anyone may implement an XMPP service and interoperate with other organizations' implementations. Because XMPP is an open protocol, implementations can be developed using any software license; although many server, client, and library implementations are distributed as free and open-source software, numerous freeware and commercial software implementations also exist.

The Internet Engineering Task Force (IETF) formed an XMPP working group in 2002 to formalize the core protocols as an IETF instant messaging and presence technology. The XMPP Working group produced four specifications (RFC 3920, RFC 3921, RFC 3922, RFC 3923), which were approved as Proposed Standards in 2004. In 2011, RFC 3920 and RFC 3921 were superseded by RFC 6120 and RFC 6121 respectively, with RFC 6122 specifying the XMPP address format. In 2015, RFC 6122 was superseded by RFC 7622. In addition to these core protocols standardized at the IETF, the XMPP Standards Foundation (formerly the Jabber Software Foundation) is active in developing open XMPP extensions.

XMPP-based software is deployed widely across the Internet, and by 2003, was used by over ten million people worldwide, according to the XMPP Standards Foundation.

[8] __, “XMPP”, available at https://en.wikipedia.org/wiki/XMPP, last revised 15 Dec. 2016

DDS

The Data Distribution Service for Real-Time Systems (DDS)[9] is an Object Management Group (OMG) machine-to-machine middleware "m2m" standard that aims to enable scalable, real-time, dependable, high-performance and interoperable data exchanges between publishers and subscribers. DDS addresses the needs of applications like financial trading, air-traffic control, smart grid management, and other big data applications. The standard is used in applications such as smartphone operating systems, transportation systems and vehicles, software-defined radio, and by healthcare providers. DDS may also be used in certain implementations of the Internet of Things.

AMQP

The Advanced Message Queuing Protocol (AMQP)[10] is an open standard application layer protocol for message-oriented middleware. The defining features of AMQP are message orientation, queuing, routing (including point-to-point and publish-and-subscribe), reliability and security.

AMQP mandates the behaviour of the messaging provider and client to the extent that implementations from different vendors are interoperable, in the same way as SMTP, HTTP, FTP, etc. have created interoperable systems. AMQP is a wire-level protocol, a description of the format of the data that is sent across the network as a stream of octets. Consequently, any tool that can create and interpret messages that conform to this data format can interoperate with any other compliant tool irrespective of implementation language.

MQTT

MQTT[11] (formerly MQ Telemetry Transport) is an ISO standard (ISO/IEC PRF 20922) publish-subscribe-based "lightweight" messaging protocol for use on top of the TCP/IP protocol. It is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited. The publish-subscribe messaging pattern requires a message broker. The broker is responsible for distributing messages to interested clients based on the topic of a message. Andy Stanford-Clark and Arlen Nipper of Cirrus Link Solutions authored the first version of the protocol in 1999.

The specification does not specify the meaning of "small code footprint" or the meaning of "limited network bandwidth". Thus, the protocol's availability for use depends on the context. In 2013, IBM submitted MQTT v3.1 to the OASIS specification body with a charter that ensured only minor changes to the specification could be accepted. MQTT-SN is a variation of the main protocol aimed at embedded devices on non-TCP/IP networks, such as ZigBee.

Alternative protocols include the Advanced Message Queuing Protocol, the IETF Constrained Application Protocol and XMPP.

---

[9] __, “Data Distribution Service”, available at https://en.wikipedia.org/wiki/Data_Distribution_Service, last revised 15 Dec. 2016

[10] __, “Advanced Message Queuing Protocol”, available at https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol, last revised 15 Dec. 2016

[11] __, “MQTT”, available at https://en.wikipedia.org/wiki/MQTT, last revised 15 Dec. 2016

2.1.2.3 Existing Gateway access networks

WiFi (IEEE 802.11.x)

Wi-Fi or WiFi is a technology that allows electronic devices to connect to a Wireless Local Area Network (WLAN), mainly using the 2.4 gigahertz (12 cm) UHF and 5 gigahertz (6 cm) SHF ISM radio bands. It is usually password protected, but may be open, which allows any device within its range to access the resources of the WLAN network. This technology is based on the Institute of Electrical and Electronics Engineers' (IEEE) 802.11 standards.

A wide variety of devices can be connected by this including personal computers, video-game consoles, smartphones, digital cameras, tablet computers digital audio players and modern printers, and sensors and other any other smart object Wi-Fi compatible. Devices can connect to the Internet via a WLAN network and a wireless access point. Such an access point (or hotspot) has a range of about ~20 meters indoors and a greater range outdoors. Hotspot coverage can be as small as a single room with walls that block radio waves, or as large as many square kilometers achieved by using multiple overlapping access points.

Wi-Fi is less secure than wired connections, such as Ethernet, precisely because an intruder does not need a physical connection. Web pages that use TLS are secure, but unencrypted Internet access can easily be detected by intruders. Because of this, Wi-Fi has adopted various encryption technologies (WEP, WPA, WPA2, WPS). The Wi-Fi Alliance has updated its test plan and certification program to ensure all newly certified devices resist attacks.

IEEE 802.15.4

It is a standard which specifies the PHY and MAC layers for low-rate WPAN. It is maintained by the IEEE 802.15 working group, and define the basis for the ZigBee, ISA100.11a, WirelessHART, MiWi, and Thread specifications, each of which further extends the standard by developing the upper layers.

It is focused on low-cost, low-speed ubiquitous communication between devices and can be contrasted with other approaches, such as Wi-Fi, which offer more bandwidth and require more power. The basic framework conceives a 10-meter communications range with a transfer rate of 250 kbit/s.

The aim is to achieve extremely low manufacturing and operation costs and technological simplicity, without sacrificing flexibility or generality.

Important features include:

- Real-time suitability by reservation of guaranteed time slots.
- Collision avoidance through CSMA/CA.
- Integrated support for secure communications.
- Power management functions such as link quality and energy detection.
- Three possible frequency bands for operation (868/915/2450 MHz).

Other higher-level layers and interoperability sublayers are not defined in the standard. Specifications, such as 6LoWPAN and ZigBee are built on this standard. RIOT, TinyOS, Unison RTOS, DSPnano RTOS and Contiki operating systems also use a few items of IEEE 802.15.4 hardware and software.

DASH7

At the beginning was an international standard describing diverse RFID technology, each utilizing a unique frequency range. Later it becomes an open source Wireless Sensor and Actuator Network protocol, which operates in the 433 MHz, 868 MHz and 915 MHz unlicensed ISM band/SRD band.

DASH7[12] provides; multi-year battery life, range of up to 2 km, low latency for connecting with moving things, a very small open source protocol stack, AES 128-bit shared key encryption support, and data transfer of up to 167 kbit/s. The DASH7 Alliance Protocol is the name of the technology promoted by the non-profit consortium called the DASH7 Alliance.

The main pillar of this standard is the BLAST networking technology:

- **Bursty**: Data transfer is abrupt and does not include content such as video, audio, or other isochronous forms of data
- **Light**: For most applications, packet sizes are limited to 256 bytes. Transmission of multiple, consecutive packets may occur but is generally avoided if possible.
- **Asynchronous**: DASH7's main method of communication is by command-response, which by design requires no periodic network "hand--shaking" or synchronization between devices.
- **Stealth**: DASH7 does not use discovery beacons, end nodes can choose to respond only to pre-approved devices.
- **Transitional**: A DASH7 system of devices is inherently mobile or transitional. Unlike other wireless technologies DASH7 is upload--centric, not download--centric, thus devices do not need to be managed extensively by fixed infrastructure (i.e. base stations) to respond only to pre-approved devices.

EnOcean

EnOcean[13] is an energy harvesting wireless technology which covers the first three layers of the OSI protocol stack. The transmit range goes up to 30 meter in buildings and up to 300 meter outdoor. Even the main using is for building automation is also applied to other applications in industry, transportation, logistics and smart homes.

Modules based on EnOcean technology combine micro energy converters with ultra-low power electronics, and enable wireless communications between battery-less wireless sensors, switches, controllers and gateways. And it operates in 868 MHz in Europe generating a signal of astonishing range from an extremely small amount of energy.

INSTEON

Insteon[14] is a home automation and domotics technology that enables low-cost devices, light switches, lights, thermostats, leak sensors, remote controls, motion sensors, and other electrically powered devices to interoperate through power lines(PL), radio frequency (RF) communications, or both. It employs a dual-mesh networking topology in which all devices are peers and each device independently transmits, receives, and repeats messages.

[12] http://www.dash7-alliance.org/
[13] https://www.enocean.com/
[14] http://www.insteon.com/

NFC

The set of communication protocols known as Near-field communication[15] enable two electronic devices, one of them usually a portable, on-move device such as a smartphone, to establish link communication through the air by bringing them within about 4 cm of each other. So is a technology of proximity that allows these two devices to exchange data if they are in a small range, using electromagnetic induction between two loop antennas, operating within the globally available unlicensed radio frequency ISM band of 13.56MHz on ISO/IEC 18000-3 air interface at rates from 106 to 424 kbit/s.

NFC-enabled devices can be equipped with apps for future use of the data exchanged and full NFC device can work in three modes:

- NFC card emulation: devices act like smart cards, allowing users to perform payments or ticketing
- NFC reader/writer: for reading information stored on inexpensive NFC tags embedded in labels or smart posters
- NFC peer-to-peer: two devices communicate in an ad-hoc mode.

NFC standards cover communications protocols and data exchange formats, and are based on existing RFID standards including ISO/IEC 14443 and FeliCa.

Bluetooth BLE

Bluetooth[16] Smart technology operates in the same spectrum range (the 2.400 GHz-2.4835 GHz ISM band) as Classic Bluetooth technology, but uses a different set of channels. Instead of the Classic Bluetooth 79 1-MHz channels, Bluetooth Smart has 40 2-MHz channels. Within a channel, data is transmitted using Gaussian frequency shift modulation, similar to Classic Bluetooth's Basic Rate scheme. The bit rate is 1Mbit/s, and the maximum transmit power is 10 mW.

Bluetooth Smart uses frequency hopping to counteract narrowband interference problems. Classic Bluetooth also uses frequency hopping but the details are different; as a result, while both FCC and ETSI classify Bluetooth technology as an FHSS scheme, Bluetooth Smart is classified as a system using digital modulation techniques or a direct-sequence spread spectrum.

SigFox

SigFox[17] is a global IoT network operator. It uses differential binary phase-shift keying (DBPSK) in one direction and Gaussian frequency shift keying (GFSK) in the other direction. SigFox and their partners set up antennas on towers (like a cell phone company) and receives data transmissions from devices such as parking sensors or water meters. In Europe use the narrow frequency band around 868MHZ.

Its wireless systems send quite small amount of data, 12 bytes, and in a very slow mode, using the aforementioned standard radio transmission methods.

This technology fits perfectly for any application that needs to send small, lossy and infrequent bursts of data. Within the network the signal is typically sent a few times to ensure the reception but some

[15] http://nearfieldcommunication.org/
[16] https://www.bluetooth.com/
[17] https://www.sigfox.com/

drawbacks or limitations are given such as shorter battery life and inability to guarantee the receiving of a message by the tower.

A bi-directional network has not been designed yet, but SigFox said they will be working over this technology. If they are successful in deploying a two-way network, this will enable a wider variety of applications on their networks, though it will not have a symmetrical link because of the underlying technology they have chosen.

ZigBee

ZigBee[18] is a low-cost/low-power wireless mesh network standard targeted at the wide development of long battery life devices in wireless control and monitoring applications. Zigbee devices have low latency, which further reduces average current. ZigBee operates in the industrial, scientific and medical (ISM) radio bands: 2.4 GHz in most jurisdictions worldwide; 784 MHz in China, 868 MHz in Europe and 915 MHz in the USA and Australia. Data rates vary from 20 kbit/s (868 MHz band) to 250 kbit/s (2.4 GHz band).

The ZigBee network layer natively supports both star and tree networks, and generic mesh networking. Every network must have one coordinator device, tasked with its creation, the control of its parameters and basic maintenance. Within star networks, the coordinator must be the central node. Both trees and meshes allow the use of ZigBee routers to extend communication at the network level.

ZigBee builds on the physical layer and media access control defined in IEEE standard 802.15.4 for low-rate WPANs. The specification includes four additional key components: network layer, application layer, ZigBee device objects (ZDOs) and manufacturer-defined application objects which allow for customization and favor total integration. ZDOs are responsible for some tasks, including keeping track of device roles, managing requests to join a network, as well as device discovery and security.

LTE/LTE-A/LTE-M

Generation of cellular standards for mainly smartphone data exchange. LTE is standard for high-speed wireless communication for mobile phones and data terminals. It is based on the GSM/EDGE and UMTS/HSPA network technologies, increasing the capacity and speed using a different radio interface together with core network improvements. The LTE-A is a major enhancement of the standard before. Those two works perfectly with device with enough resources but, to include them in the IoT paradigm they develop a parallel standard focused in M2M communication.

LTE-M, which is an abbreviated version of LTE-MTC (or “machine-type communications”), is a part of 3GPP’s release 12 and 13, and it is still under consideration. The LTE channel is made up of resource blocks of about 230 kHz of spectrum, and LTE-M is part of the 1.4 mHz block, comprised of six resource blocks. LTE-M is more energy efficient because of its extended discontinuous repetition cycle (DRX), which means the endpoint can communicate with the tower or the network on how often it will wake up to listen for the downlink. LTE-PSM from Rel 12 (power-saving mode) had a similar feature, but extended DRX was created specifically for LTE-M in Rel 13.

The advantage of LTE-MTC for M2M communications is that it works within the normal construct of LTE networks. In other words, a cellular carrier like AT&T only has to upload new baseband software onto its base stations to turn on LTE-M and won’t have to spend any money on new antennas. It’s

[18] http://www.zigbee.org/

also five times simpler than a category 4 receiver—like that found in user equipment like a cell phone—because it needs only to understand and digitize 1.4 mHz of the channel instead of 20 mHz.

LTE-M has a little higher data rate than NB-LTE-M and NB-IoT, but it is able to transmit fairly large chunks of data. Thus, it can be used for applications such as tracking objects, wearables, energy management, utility metering, and city infrastructure.

LoRa

LoRa[19] is a Low Power Wide Area Network (LPWAN) specification focused on wireless devices with battery restrictions. This specification tries to provide seamless interoperability among smart *Things* without the need of complex local installations and gives back the freedom to the user, developer, businesses enabling the roll out of Internet of Things. LoRaWAN network architecture is typically laid out in a star-of-stars topology in which gateways is a transparent bridge relaying messages between end-devices and a central network server in the backend. Gateways are connected to the network server via standard IP connections while end-devices use single-hop wireless communication to one or many gateways. All end-point communication is generally bi-directional, but also supports operation such as multicast enabling software upgrade over the air or other mass distribution messages to reduce the on air communication time. Communication between end-devices and gateways is spread out on different frequency channels and data rates. The selection of the data rate is a trade-off between communication range and message duration. Due to the spread spectrum technology, communications with different data rates do not interfere with each other and create a set of "virtual" channels increasing the capacity of the gateway. LoRaWAN data rates range from 0.3 kbps to 50 kbps. To maximize both battery life of the end-devices and overall network capacity, the LoRaWAN network server is managing the data rate and RF output for each end-device individually by means of an adaptive data rate (ADR) scheme. The result is a network ideal for Internet of Things (IoT), metering, security, asset tracking, and machine-to-machine (M2M) applications.

ANT

ANT[20] is a proprietary wireless communication protocol stack that operates on the 2.4 GHz ISM band. ANT is oriented towards the usage with sensors and health applications. The range of ANT is 30 meters at 0 dBm.

Each ANT node can transmit, receive or be set up as bi-directional channels. ANT accommodates three types of messaging, Broadcast, acknowledgements and bursts (max 20 kbit/s). Possible topologies with ANT are point to point, tree, and mesh. A maximum of 65533 nodes set in one network.

Each transmission occurs in an interference free time slot within the defined frequency band. The radio transmits for less than 150 µs allowing a single channel to be divided into multiple time slots.

6LowPan (IETF)

IPv6 over Low power Wireless Personal Area Networks (6LowPan)[21] allows IPv6 packets to be sent and received over IEEE 802.15.4 based networks. Some of the functions included; adapting the packet sizes of the two networks, address resolution in hierarchical manner, differing device designs,

[19] https://www.lora-alliance.org
[20] https://www.thisisant.com/
[21] https://datatracker.ietf.org/wg/6lowpan/charter/

differing focus on parameter optimization, adaptation layer for interoperability and packet formats, addressing management mechanism, routing consideration for mesh topologies (LOADng and RPL), device and services discovery and other security issues.

Thread (Thread Group)

Thread[22] is an effort of over 50 companies to standardize on a closed-documentation, royalty-free protocol running over 6LoWPAN to enable home automation. Thread however is IP-addressable, with cloud access and AES encryption. It supports over 250 devices on a network with mesh topology.

Z-Wave (Z-Wave Alliance)

Z-Wave[23] is a wireless communication protocol for home automation, oriented to the residential control and automation market and intended to provide simple and reliable method to wirelessly control lighting, HVAC, security systems, home cinema, etc.

2.1.3 Summary Table

Protocol	COAP	MQTT	XMPP	AMQP	DDS
Transport	UDP	TCP	TCP	TCP	TCP UDP
Publish/Subscribe	☑	☑	☑	☑	☑
Request/Response	☑	☒	☑	☒	☒
Security	DTLS	SSL	SSL	SSL	SSL DTLS
QoS	☑	☑	☒	☑	☑
Header Size	4	2	-	8	-
Low Power and Lossy	Excellent	Good	Fair	Fair	Poor
Architecture	Tree	Tree	Client-server	P2P	Bus

Table 2: Comparison between device IoT protocols.

2.2 Network Interoperability (N2N)

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 protocol stack. At this level we have to contemplate pervasiveness and ubiquitous networks aspects as this is not as a traditional network. The particularity of the IoT network is the treatment of many different types of data flows as well as

[22] http://threadgroup.org/
[23] http://z-wavealliance.org/

protocols to support communication. Also, the integration of new devices on the scheme, that need to communicate with each other, make the interoperation on the N2N layer more difficult.

In this particular case we describe the interoperability between networks or parts of the network that belong to an IoT deployment. To that end, together with the aspects considered before we have to take into account particular characteristics to be achieved on this layer, such as the mobility of objects through different access networks, secure seamless mobility and the backing of real time data among the network. The operation in highly constrained environment is also an important issue to analyze. And finally, the use of really heterogeneous protocols (6LowPAN, RPL, LoRa, SIGFox, etc) and mechanisms (tunneling mechanisms over IP, GRE and 6LoWPAN, etc) on IoT network level are problems we desire to solve with a proposed interoperability solution.

The interoperability solution will be based on software defined paradigms but mainly on two approaches: SDR for interoperability on access network and SDN/NFV for the core network.

2.2.1 Common Approaches

To achieve network to network interoperability, the solution we propose is based on the philosophy of SDN, and extension of its capabilities as:

- Decoupling of data plane from logical plane using the well-studied protocol OpenFlow.
- Virtualizing network services at the top of the architecture.
- Implementation of techniques for traffic engineering to handle different flows of data generated by sensors depending on its priority.

Additionally, we wish to provide interoperability at the network access, using SDR. Finally, together with this vision, we will handle the seamless mobility and the offloading of information with an extension of these aforementioned approaches.

However, before getting into detail of the proposed interoperability solution, we will describe and explain how these technologies operate and make a literature review of the projects which main function is the implementation of them.

2.2.2 Literature review

In this section we provide a brief summary about concepts, technologies or projects, needed for the proper understanding of the interoperability solution.

2.2.2.1 SDN

Regarding the communication between different networks or even with the communication of different elements within a network, we present new paradigms in networking to be used in the N2N layer of the INTER-IoT project: the concepts of Network Functions Virtualization (NFV) and Software Defined Networks (SDN).

The NFV idea is to virtualize network functions and components that we can find within a network. Such real elements as firewalls, routers, switches, load balancers, etc can be located in central or distributed equipment and control, in a virtual manner, all the underlined network [2]. To that end,

there is no need to deploy the physical elements and the same functions can be performed also unfolding new network functions and services software based.

The hardware that contains all these functions can be on one or several servers, with a processor, capable enough to manage several virtual machines or virtual elements which represent the aforementioned physical entities.

On the other hand, we have the software defined network concept that presents a new network architecture where the data plane and the control plane are decoupled (so they work separately). The intelligence and network state, as well as the management and routing algorithm are centralized and the underlying network infrastructure is abstracted from the applications and services. This separation allows the virtualization of the different components of the network, being NFV and SDN highly compatible and complementary [3].

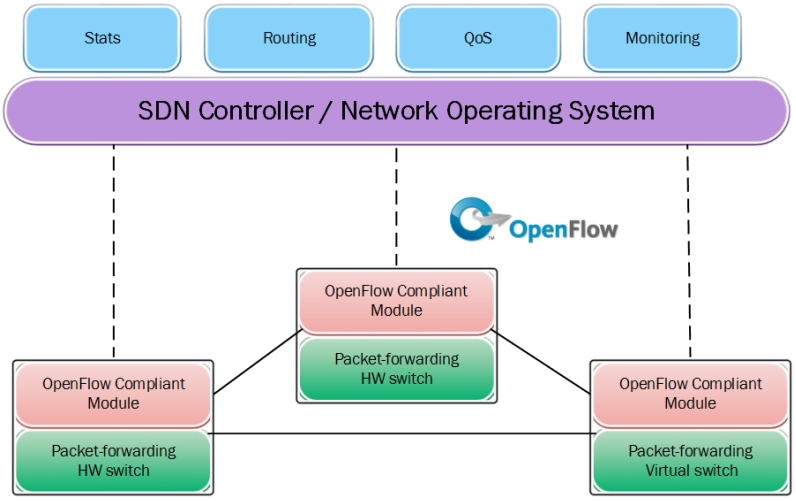

Figure 2: SDN network architecture example.

During the first steps of the communication networks, those were thought as a circuit switching approach and, as data traffic were introduced and increased, the attempts to adapt the infrastructure led to a new packet switching paradigm in order to improve the system, enhancing the experiences of the users and reducing the delays arisen by the network. However, the data traffic keeps growing, within new technologies as streaming and big data traffic. The amount of data carried by the networks is rushing levels never seen before and demands a better management to provide more reliability, robustness security and quality of service for users and other smart objects connected to them. Additionally, this growth of the traffic implies the inclusion of more network elements making more complex networks, less manageable and hardly scalable [4].

With this picture in mind the Stanford University [5] created a new network paradigm which aims to solve these limitations, firstly breaking the complexity of traditional network, simplifying it by crumbling the vertical integration of the network elements, and secondly centralizing the logic that those traditional network elements possess.

The main principle of these networks is to decouple the data and control plane so that, on one side, the elements that compose them, as switches, routers, etc., become mere forwarders which route the information in a nimbly an efficient manner. Forwarding decisions are then flow based. And, on the other side, the whole logic of routing, algorithms and other services previously provided by firewalls, middleboxes, IPS, etc. are transferred to a single point of control which is the decision-maker. An example of this architecture can be observed in Figure 2.

Even an amount of the approaches of SDN have been designed following these basic guidelines, the most remarkable is the provided by ONF (Open networking foundation), founded as a result of thin paradigm, and following the architecture proposed by OpenFlow. This view of the SDN is introduced in what follows.

According to OpenFlow on the data plane all the elements of the network, whether real or virtualized, become mere OF-enable switches, whose main function is to receive the flow of packets through one of its interfaces, check the flow tables stored inside and resend the packets following an order. Inside the switch there must be, at least, one flow table and it should have, at least, one flow entry. The flow entries have three main fields; statistics, rule and action. When a packet arrives to the switch a specific field is matched against the rule and, if there is a match, the determined action is performed. This could be; resend by a selected interface, send to the controller or drop, among others. Additionally, the third parameter is used to collect statistics and information about the number and type of packets that has been forwarding. With this, the controller has a better overview about the state of the network and the flows that are being carried [5].

For the logic plane we could find the controller where all switches are connected to, this is the engine of the system, and more complex to develop. In case no matches were found in the table switch by the packet, the switch sends the register number and contain of the packet to the controller to decide which action should be performed. For this purpose, all switches are connected to one or more controllers through a secure channel, TCP/TLS, and communicate each other by OpenFlow protocol, to exchange the packets, tables and orders that have to be executed by the switches.

However, this controller or NOS (Network Operating System) has more services inside, as network operative software that can be divided in three main parts:

- **Southbound Interface**: the connection with the forwarding devices through a secure channel is implemented by the southbound API which allows the entrance of OpenFlow packets into the controller and formalizes the way the control and data planes interact.
- **Controller core**: we can see this part as the brain or engine of the controller, where all logic rest.
- **Northbound Interface**: the NOS can offer an API to the application developers.

The following sections summarize all the technologies until these days related with SDN.

Network Operating Systems

PicOS: A SDN OS for white box switches Layer-2/3 feature set with support for OpenFlow, OVSDB, and other protocols.

OpenNetworkLinux: A Linux distribution for "bare metal" switches, that is, network forwarding devices built from commodity components.

OpenSwitch: A linux network operating system from Hewlett-Packard. OpenSwitch provides a fully-featured L2/L3 control plane stack, traditional and programmatic, declarative control plane.

Installation Environment

ONIE Open Network Install Environment (ONIE) defines an open "install environment" for bare metal network switches, such as existing ODM switches and the upcoming OCP Network Switch design. ONIE enables a bare metal network switch ecosystem where end users have a choice among different network operating systems.

Virtual Switches

Open vSwitch[24] is a production quality, multilayer virtual switch designed to enable massive network automation through programmatic extension, supporting standard management interfaces and protocols. Between its features we can find:

- Security: VLAN isolation and traffic filtering among others.
- Monitoring: Netflow, sFlow, SPAN, RSPAN, among others.
- QoS: traffic queuing and shaping.
- Automated Control: OpenFlow/OVSDB management protocol among others.

Indigo is an open source project aimed at enabling support for OpenFlow on physical and hypervisor switches. This is the most used and extended implementation but we can find **CPqD** that is an OpenFlow 1.3 compatible user-space software switch implementation, **Lagopus** is high-performance software OpenFlow 1.3 switch and, also, **LINC-Switch** a pure OpenFlow software switch written in Erlang and **Snabbswitch** an open source virtualized Ethernet networking stack.

Network Virtualization

In case of virtualization of the network we have tools as **FlowVisor**, an OpenFlow controller that acts as a hypervisor/proxy between a switch and multiple controllers that can slice multiple switches in parallel, effectively slicing a network, or **OpenVirtex** also a network hypervisor that can create multiple virtual and programmable networks on top of a single physical infrastructure.

Protocols

The main protocol used for communication between planes is **OpenFlow**[25], a communications protocol that gives access to the forwarding plane and its remote programming of a network switch or router over the network. The OF standard is the first SDN standard and vital element of an open SDN architecture [5], with its variation for management and configuration **OF-Config.** Also we can find **OVSDB** communication protocol which is used to manage the OpenvSwitch database.

Controllers

A list of most used open protocols is below:

- **NOX -** An open source development platform for C++-based software-defined networking (*SDN*) control applications.
- **NodeFlow** - An OpenFlow Controller for Node.js.
- **ONOS** - Is defined as a network operating system designed for high availability, performance, scale-out and rich abstractions. Additionally, ONOS has useful Northbound abstractions and APIs to enable easier application development and Southbound abstractions and interfaces to allow for control OpenFlow-ready and legacy devices.
- **OpenDaylight**- Is an Open Source project with a modular, pluggable, and flexible controller platform at its core. This controller is implemented mainly in software and is contained within its own JVM. As such, it can be deployed on any hardware and operating system platform that supports Java.

[24] http://www.openvswitch.org/
[25] https://www.opennetworking.org/

- **POX** - A networking software platform written in Python.
- **Ryu** - Component-based software defined networking framework.
- **Floodlight** - A java-based OpenFlow controller.
- **Vyatta** - The first commercial Controller built directly from OpenDaylight.
- **OpenContrail** - A SDN project that utilizes SDN & NFV and provides all the necessary components for network virtualization.
- **IRIS** - A Recursive SDN Openflow Controller created by SDN Research Section, ETRI.
- **Open MUL** - A lightweight SDN/Openflow controller written almost entirely in C from scratch.
- **Beehive Network Controller** Is a distributed SDN controller built on top of Beehive. It supports OpenFlow but can be easily extended for other southbound protocols.
- **Ravel** A software-defined networking (SDN) controller that uses a standard SQL database to represent the network.

Simulators/Emulators

The main tool used for simulation of virtual SDN network is **Mininet**[26]. Mininet is a network emulation orchestration system. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. Other options include **OpenNet** a simulator for software-defined wireless local area network, **EstiNet** a world-renowned software tool for network planning and **Ns-3** a discrete-event network simulator that supports openflow environment.

Languages

The most extended one to configure and include rules in an OpenFlow enabled switch is **Pyretic.** Also we have **Frenetic** a programming language and Runtime System derived from the one named before and, finally, **P4** a declarative language for expressing how packets are processed by the pipeline of a network forwarding element such as a switch, NIC, router or network function appliance.

Libraries

There are different libraries focused on extending and simplifying the functionalities of Open Flow, such as:

- **Loxigen**: a tool that generates OpenFlow protocol libraries for a number of languages,
- **Openfaucet**: a pure Python implementation of the OpenFlow 1.0.0 protocol, based on Twisted,
- **Oflib-node** - an OpenFlow protocol library for NodeJS. providing conversion between OpenFlow wire protocol messages and JavaScript objects,
- **OpenFlowJ** a Java implementation of low-level OpenFlow packet marshalling/unmarshalling and IO operations,
- **Nettle** a Haskell library for working with the OpenFlow protocol and finally,

[26] http://mininet.org/

- **OCaml OpenFlow** a serialization and protocol library for OpenFlow.

Test

Among the available testing tools for the SDN deployments are:

- **Oftest:** a simple OpenFlow Testing Framework,
- **STS** a SDN Troubleshooting System, simulates network devices, allowing programmatically test cases generation,
- **Nice-of** a tool to test OpenFlow controller application for the NOX controller platform,
- **OpenSDNCore** a virtualization Testbed for NFV/SDN Environment.

NFV

Finally, we present the most used frameworks to virtualize the different functions of a network performed usually for physical devices:

- **Neutron-OpenStack**[27] - Neutron is an OpenStack module project to provide "networking as a service" (NaaS) between interface devices managed by other OpenStack services (e.g. Nova), focused on layers L2-L3 of the OSI stack and technology agnostic.
- **MANO** - Open Source Mano is an ETSI-hosted project to develop an Open Source NFV Management and Orchestration software stack aligned with ETSI NFV.
- **OPNFV** - It is focused on building NFV Infrastructure (NFVI) and Virtualized Infrastructure Management (VIM) creates a reference NFV platform to accelerate the transformation of enterprise and service provider networks. OPNFV works upstream with other open source communities to bring both contributions and learnings from its work directly to those communities in the form of blueprints, patches, and new code. More recently, OPNFV has extended its portfolio of forwarding solutions to include FD.io and ODP, is able to run on both Intel and ARM commercial and white-box hardware, and includes Management and Network Orchestration (MANO) components primarily for application composition and management.

2.2.2.2 SDR

Software defined radio (SDR) holds a promising future due to its inherent flexibility, which is the natural outcome of replacing discrete analog radio components by digital signal processing (DSP) in software. Some of the key advantages obtained by this functional transition from the analog to digital domains are that these digitized sections become impervious to temperature changes, component manufacturing variation, allow for re-configurability / re-programmability and, in the case of SDR, allow linear phase filters design to improve performance. These capabilities have several very interesting implications as they may be considered a building block for other technological concepts, such as the cognitive radio, advanced spread spectrum and software defined antennas including digital beamforming.

A truly idealized SDR scheme would, on the transmitter side, attach a DAC (digital to analog converter) under software control directly to an antenna and hence produce the RF signal directly. On the receiver side an ADC would connect directly to the antenna and sample the RF signal directly;

[27] http://www.openstack.org/

the samples would then be processed to extract the data. However due the practical limitations in DAC/ADC components this idealized SDR is not possible at the frequencies commonly used for modern everyday communication.

Therefore, we need to consider a more generalized and practical SDR scheme, depicted in Figure 3, which consists of three main units: (i) a reconfigurable digital radio (ii) a software tunable analog radio and (iii) a software tunable antenna system.

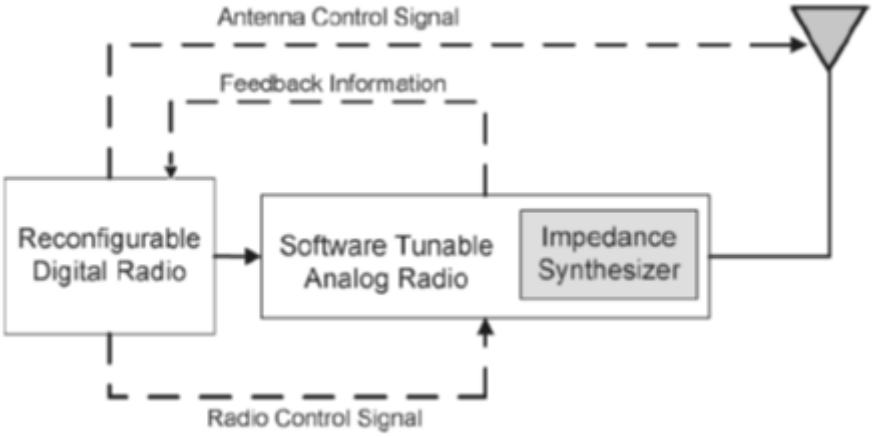

Figure 3: Ideal SDR components.

The precise division of functional responsibilities associated with the reconfigurable digital radio and tunable analog radio, shown in Figure 3 may be dependent on many implementation details such as transmission data rates and technology available. However, broadly speaking, the reconfigurable digital radio will be dealing with both the transmitter and receiver baseband signal processing. Transmitter functions that may be performed by this block include modulation, symbol mapping, precoding, error control encoding, scrambling, digital filtering and possibly up-conversion to an IF (intermediate frequency). The computationally more demanding digital receiver processing functions may include demodulation, symbol de-mapping, error control decoding, equalization, descrambling, digital filtering, fine/course carrier synchronization, symbol synchronization and digital down-conversion, among others.

The function responsibilities for the software tunable analog radio, also known as the frequency agile RF frontend, are RF down-conversion, analog filtering, impedance matching, amplification, automatic gain control (AGC), programmable frequency synthesis and ADC/DAC control.

From the above description, we can see that the division of radio functionality between the tunable analog radio section and the digital radio section can conceptually occur in several places e.g. this partition may occur at the baseband, intermediate frequency or RF stages. This process of the digital realm gradually taking on more of the roles of the analog is, in this context, sometimes called the ‘SDR paradigm’.

The last point concerning Figure 3 is that the software tunable antenna is there to allow the antenna to efficiently radiate RF over a wide range of frequencies. However, it is worth mentioning that this tunable antenna may be either replaced with a switched antenna system or, omitted and a standard broadband antenna used instead which satisfies the design requirements.

From the above discussion, we can see that implementing an SDR, if we assume a simple antenna, requires a two key blocks: a processor and an analog radio. One question that now arises is; what implementation platform options exist for these blocks and how do we chose between them? Well,

that it depends what the purpose of the SDR radio is. For instance, a hobbyist may want to build an SDR for listening to aircraft ADS-B data and this could be achieved with a Raspberry-Pi and a Stratux 1090ES $35 dongle, alternatively, at the other end of the complexity scale a software defined military radar may require specialised hardware, a large team of people and many millions of dollars. In each of these cases the key blocks mentioned would be implemented on completely different hardware and software platforms. We briefly assess some typical options in a little more detail.

2.2.2.3 Other Issues

QoS

Quality of Service (QoS) has always been a domain of extensive analysis and study by the moment different devices got connected. Due to the layered architecture of networks, applications do not have the power to modify the underlying protocols. They have to rely on the capability of the underlying networks to satisfy the requirements of the generated traffic. These requirements are quantifiable and could be e.g. low latency, high packet delivery ratio, low packet loss etc. QoS is the term to represent this capability of networks.

QoS allows for an explosion and coexistence of new applications with different requirements over the same networks. It helps network schedulers prioritize traffic flows according to their requirements. Though many technologies i.e. (G)MPLS emerged to supported guaranteed quality of service for certain applications, e.g. video conferencing, they are not extensively used. Progress in L1&L2 network engineering allowed overprovisioning of resources which made those technologies in many cases obsolete. Moreover, as network diversity was increasing, ensuring end to end latency across different protocol stacks and devices was becoming more and more difficult.

Yet, QoS is coming back with higher intensity as Industrial Internet of Things introduces new possibilities and challenges. IoT has the capability to penetrate deeper in human and business activities with small battery powered devices giving access to more critical processes. Despite the business opportunities this may entail, new challenges also emerge. For instance, remote control of life-critical equipment has stringent latency and PDR requirements and is usually implemented over best effort networks like IP. The traffic volume of those applications is low but every packet's delivery is critical.

If IoT is to be used with cyber-physical and safety-critical infrastructure, the interoperability of various edge IoT networks/platforms should also incorporate quality of service over best-effort networks. IoT edge networks may be either managed or unmanaged. Managed networks can provide certain QoS or an estimate of it, whereas unmanaged networks lack this predictability. Overprovisioning, though an option is not a desirable feature as it translates to more energy consumption, which the low-power small devices used in critical applications, cannot afford.

Therefore, INTER-IoT should work on gateway and network level to address the issue of QoS. Two interoperable managed IoT networks serve different types of traffic from different applications. Cross network traffic flows should then be treated in different ways by the two networks as long as the end-to-end QoS is achieved. Even more complicated, interoperability by a mixture of managed and unmanaged networks requires new techniques like partial overprovisioning to the managed part in order to compensate for the losses in the unmanaged part of the traffic flow. To sustain a certain level of reliability across a traffic flow, new mechanisms are needed to ensure QoS via:

- Monitoring and detecting bottlenecks and sources of other turbulences in edge networks.

- In mixed inter-networks (managed and unmanaged), nullify impediments and turbulences introduced by unmanaged networks via on-demand and at-runtime optimization of their managed counterparts.
- Trigger SDR and network offloading components in case adapting resources of managed networks cannot alleviate the QoS degradation.

An INTER-IoT gateway has to implement the monitoring and bottleneck detection modules in order to compute the remedy a managed network has to deploy. That is, the gateway receives monitoring data from the underlying IoT nodes and QoS requirements from the applications. The two are combined by a QoS estimator module which is then feeding a QoS controller to reconfigure the managed parts of a network flow for a specific application. At the network level, reconfiguration is possible at the managed networks only. However, monitoring is a feature that any underlying networks should implement but modifying any parameter in L2-4 is possible for managed networks only.

Managed low-power lossy networks (LLN) are using the IEEE802.15.4e-2012 [6] amendment of the IEEE802.15.4 [7] protocol to schedule transmissions on planned timeslots and channels. Nodes in Time Slotted Channel Hopping (TSCH) networks are synchronized on a time reference set by the PAN coordinator and trickled down to all nodes by a tree directed acyclic graph (DAG) topology. A new node may join the network by hearing for Enhanced Beacons from other already joined nodes. The Enhanced Beacons carry synchronization information for the new node to use and synchronize while joining. The time progresses in time-slots, the minimum time unit usually equal to 10ms. Timeslots are packaged into repetitive slot-frames. Each timeslot is sufficient for a node to transmit a packet and wait for the acknowledgement. Hence, a schedule is effective if communicating nodes are both awake at that period, one set for transmission and the other for reception. Non active nodes at a time-slot go to sleep.

Time is the one dimension of slot-frames measured in number of slot offsets; channel offsets are the second. Nodes in a TSCH network use one of 16 frequencies available at 2.4GHz at a given timeslot. Each frequency is mapped to a channel offset in the slotframe matrix. The channels shift by one position at every timeslot so that same slot offset is not likely to be allocated the same physical frequency when the slotframe is repeated. The slotframe matrix, composed of slot and channel offsets, is the scheduled template that all nodes adhere and repeat. There might be multiple slotframes running in parallel.

Efforts on scheduling TSCH networks have shifted to decentralized schedulers. Tinka et. al. [8] presented the first decentralized algorithm which allocates one frequency to facilitate new nodes joining the network and one dedicated slot for targeted communication. DeTAS [9] was another decentralized scheduling technique which relies on exchanging traffic information between RPL [10] children and their parent. That allows the parent to allocate enough resources to serve the predicted traffic. All nodes follow a common macro-schedule which consists of micro-schedules (one per subtree) calculated in a distributed way. Wave [11] is a distributed scheduling algorithm that splits the slotframe into regions (waves). All nodes with at least one packet to be sent are allocated a cell in the first wave and as long as they have more packets, they are allocated extra cells in subsequent waves. Finally, orchestra [12] is an autonomous distributed scheduler which schedules one upstream and one downstream cell per neighbor without communication between neighbors.

Though the advantages of decentralized schedulers are well documented, their limitation lies on the fact that they cannot have a complete view of the conditions in the network making optimization harder. Decentralized schedulers are mostly using local information at every node. As explained in

various studies [13] [14] [15] [16] [17] local node metrics like link quality estimation are not as effective as time-based metrics for scheduling and QoS optimization purposes. Time-based metrics usually consider end-to-end traffic conditions that a centralized scheduler would be easier to identify.

TASA [18] was designed to rely on traffic patterns in the network known a priori. Based on the queue length of the nodes and using graph coloring and matching techniques, it generates conflict-free schedules. For completeness, TMCP [19], JFTSS [20] and MODESA [21] are other paradigms of centralized schedulers.

Offloading

TCP Offloading Engine (TOE) is a mechanism that enables offloading of network TCP/IP stack from the CPU to specific hardware on the network controller. Recent increases in Ethernet speeds, that has been faster than that of than processor speed, have resulted on I/O bottlenecks, in which the CPU is unable to cope with ever increasing network data flow, which can take more than half of the processing power. Even one powerful modern CPU's would not be fast enough to handle the full speed of a 10 Gbit network, let alone 40 Gbit or more.

To cope with this problem, special purpose computational hardware in the network controllers have been used to 'offload' that work into a specific implementation.

Since the TCP/IP communication consists of transmitting and receiving of packages, offloading technologies were developed to address different aspects of the problem.

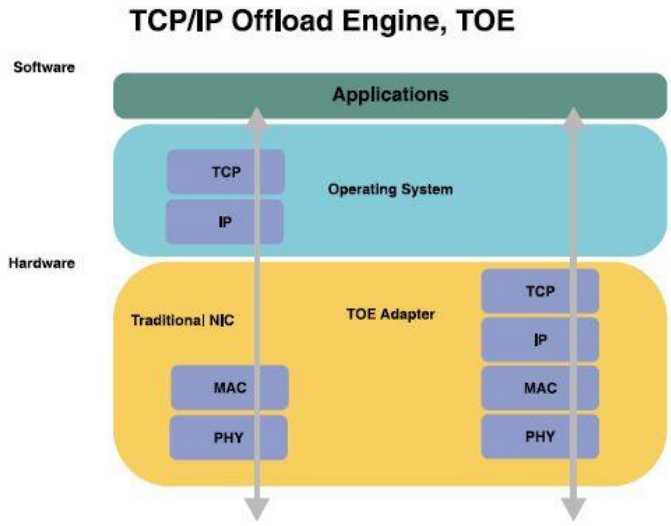

Figure 4: 10GEA scheme comparison of traditional and TEO NIC.

The size of the Standard MTU, or Maximum Transmission Unit, is set to 1500 bytes. The majority of modern communication devices such as Ethernet controllers, switches and routers support sizes of 9000 bytes, or more. Increasing this size is technically not offloading, but it can drastically reduce CPU load by reducing the number of packages and consequent space and computing overhead. Nevertheless, increasing MTU is not always possible, since the majority of internet devices are still configured to run on with 1500 bytes, but it can be effectively used in Local Area Networks.

For offloading transmitted packages, the most common technique is that of LSO or Large Segment Offload, which is also called TSO or TCP Segment Offload when applied to TCP/IP, and GSO or Generic Segment Offload. This method segments a large package of increased MTU into 1500 bytes or smaller segments before they are transmitted. In IPv4, the largest possible package size is 64 Kb

or 65535 bytes. With a little help from the network controller, a CPU can simply pass large package to the network stack, which will then segment it into 46 packages of 1448 bytes in length each.

On the other end of the communication the LRO or Large Receive Offload and GRO or Generic Receive offload, combines these small packages into one big package to then pass it forward to the process. Because of this, the LRO should not be used on devices that act as routers, as it breaks the end-to-end principle.

GSO and GRO are software only solutions in the Linux kernel, and they are applied if network hardware is unable to perform hardware offload.

Hardware support for TOE is proprietary and is implemented differently by each hardware vendor. Because of this, there is no hardware support in Linux kernel. To enable hardware TOE support in Linux, one must install proprietary vendor-provided network interface (NIC) drivers.

Whenever a component on the network layer makes use, or expects the use of Offloading such as LSO, hardware or software integration of such messages must be provided for those components that do not support it out of the box, or turn off the offloading from the source elements.

Roaming

Roaming ensures that a traveling wireless device is kept connected to a network without breaking the connection. In wireless telecommunications, traditional Roaming is a general term referring to the ability for a wireless device to automatically send and receive data, or access other services when travelling outside the geographical coverage area of the home network, by means of using a visited network.

Roaming encompasses roaming between networks of different network standards, e.g. WLAN (Wireless Local Area Network) or GSM. Device equipment and functionality, such as SIM card capability, antenna and network interfaces, and power management, determine the access possibilities.

2.2.3 Summary table

Approach	Brief summary on how they facilitate interoperability	What can be found in this chapter	Technologies
SDN	To connect two or more devices of a network provided by different vendors as well as to interact with two or more totally different network deployment, a new layer of abstraction is needed. This layer is, also, programmable in order to manage the deployment and	A description of the paradigm and its implementation the different parts that compose it, and the deployment and visualization technologies.	• NOS(OpenSwitch, PicOS, OpenNetworkLinux) • Virtual Switches (**OpenvSwtich**) • Environments(ONIE) • Network Virtualization(FlowVisot, OpenVirtex) • Protocols(OpenFLowOVSDB) • Controllers(NOX,POX,RYU,ODL,ONOS, etc)

	adapt it to specific needs.		• Emulators(Mininet, OpenNet, EstiNet) • Languages(Pyretic, Frenetic, P4) • Test(oftest, STS, Nice-of, OpenSDNCore) • NFV(Neutron-OpenStak, MANO, OPNFV)
SDR	There are some devices that use non-standard RF communication links. To provide access to these devices it is needed a software configuration of the antenna, adaptable and dynamic in order to perform the translation into a standard IP based protocol.	A description of what SDR is, the methodology to implement it and some technologies being used for this purpose.	• DSP • Components(SDR Transmitter, SDR receiver) • Projects (GNU Radio, HPSDR)
QoS/ Offloading/ Roaming	Not just to provide interoperability but a minimum quality in the communication between different network. Also, the possibility of using more than one access network to take the information from a device and the mobility of a device between networks that cannot belong to the same IoT deployment.	What these concepts means, why they have to be addressed in the interoperability network solution, and what kind of technologies can implement them.	• QoS (MPLS,TSCH) • Offloading(TCP Offloading Engine (TOE), MPTCP) • Roaming(CAPWAP)

Table 3: Network interoperability approaches comparison.

2.3 Middleware Interoperability (MW2MW)

Middleware refers to the software and hardware infrastructure that enables the communication layer between the different components of a system, generally in either request/response fashion or a sustained connection communication e.g. for data streaming.

The middleware thus abstracts several aspects of an end-to-end communication including the name/address/location of a service, the message transport, service instance, interoperability, etc. For example, a client can issue a request to a service without knowing which instance of that service it will communicate with, thus hiding some of the complexities of scaled services.

The middleware is also a convenient layer to place additional system-wide meta-services such as security, anonymization, auditing, monitoring.

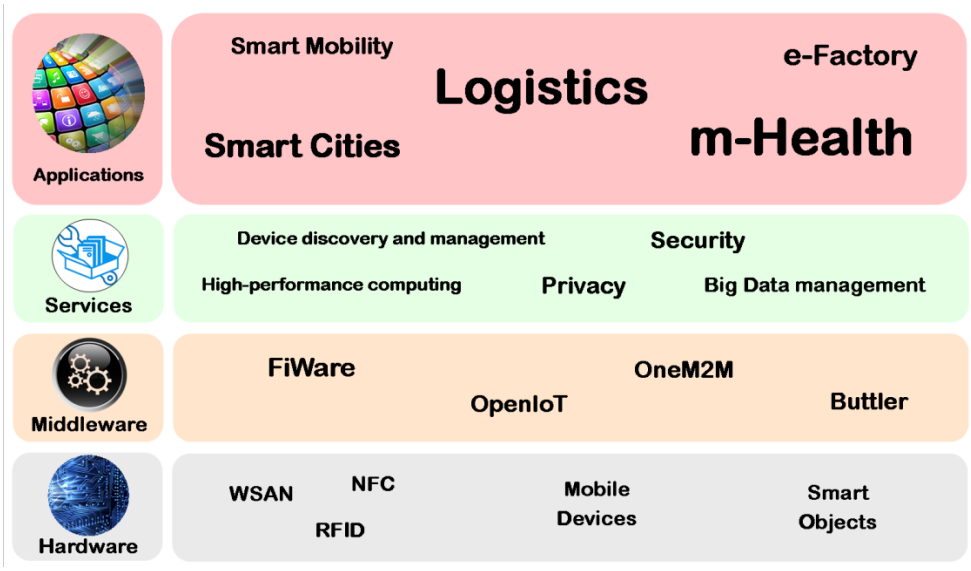

Figure 5: Multi-layer diagram of IoT use-cases and technologies.

In the specific domain of the IoT, a software platform defined as middleware has the critical objective to connect the heterogeneous ecosystem of applications communicating over heterogeneous interfaces using and operating on diversified technologies.

Due to the intrinsic difficulty to define and enforce a common standard among all this jeopardized scenarios, IoT middlewares have to provide an abstraction and adaptation layer to applications from the things and offer multiple services by means of easy-to-use, yet powerful APIs.

Development of middleware in the domain of IoT is an active area of scientific and industrial research, and a number of interesting solutions have been developed so far.

Interoperability, context awareness, device discovery and management, data collection/storage/processing/visualization, scalability, privacy, and security are among the most significant aspects that are being tackled by current IoT middlewares.

2.3.1 Common Approaches

There are a number of approaches to middleware, mostly depending on the type of applications and services that it will be connecting.

Historically, the Object Request Broker (ORB) model was used, in which a process was able to make a remote procedure call (RPC) to a process in a foreign system through the computer network. Even though there were mechanisms to facilitate this model, the communication was purely point-to-point between the two processes, and provided no middle entity that could facilitate interoperability problems and other related issues.

A common problem with direct service access, such as remote procedure calls (RPC), is that the requests are pushed to the service provider, regardless of its current state or load level. This can create load problems in which a busy server is trying to process several requests that are currently making use of most of its resources, but must attend to the reception of the new request, loading the server even more, and potentially preventing it from successfully complete any jobs while at the same time losing requests for services.

To avoid this kind of problems, a Middleware Queue approach is used, in which a message queue is deployed to its own infrastructure to receive the requests to one or several services, and the service providers go to the queue to fetch a new request whenever they judge to have the necessary resources to perform it. This Pull method prevents overloading the server, allows for transparently proxying several service instances into one single queue (or queue topic), makes it much simpler to move or update services, and can be scaled by means of new queue instances.

Architecture of middlewares varies greatly, but can be generally divided into Message Oriented Middleware (MoM) and Service Oriented Middleware (SoM), each with two types of implementation: server implementations that correspond to server-client architecture and decentralized implementations present in peer-to-peer systems.

A message-oriented server implementation proposes that core functionality of a middleware lies in one or more central servers called message brokers. This software artifact is responsible for consistent, reliable and fault-tolerant delivery of messages to any willing client. Message brokers can queue, delay, store, broadcast, translates, (periodically) retransmit or retry messages for delivery, along with other message oriented functionalities. MoMs, generally speaking, fall into one of two categories: Enterprise Messaging Systems (EMS) or Enterprise Service Buses (ESB).

While EMS are basically handling the communication between service providers and consumers using existing defined protocols that can be based in XML, SOAP or other similar standards, an Enterprise Service Bus can include more complex functionalities such as orchestration, validation, QoS monitoring, transformations and conversions, etc. ESBs also offer services, thus acting as both Message Oriented and Service Oriented middleware. A service oriented server middleware realizes a very popular model of a central entity that offers services. This model is very popular in software that is not always thought of as middleware e.g. Web servers.

A decentralized model implements all functionalities in a client by means of a client app, or a library. A middleware server is usually simpler to deploy and monitor, while a decentralized model allows for easier distribution and flexibility of network topology. It also lacks any global state of communication, such as a central message log.

2.3.2 Literature review

2.3.2.1 Message Oriented Middlewares

Whether as part of an Enterprise Messaging Systems or Enterprise Service Bus, message queues are very commonly used in communication middlewares due to their scalability, modularity and flexibility.

There are several commercial and open-source implementations of Message Queues that provide the basic, and more advanced features. Practically all of them allow for processes to subscribe (listen to) a queue name or topic, and to publish a message into a queue topic. Additionally, topics can be set to deliver a received message to one and only one subscribed party, or all of them (broadcast).

Other features that some queues implement include: Receipt acknowledge, in which a publisher is informed that the Queue received the request, Delivery policy (e.g. at least once, at most once), Persistence (accepted messages are never lost, even on Middleware failure), or Synchronicity (whether a second message is made available before the first one is responded to) and so on.

Kafka

Apache Kafka[28] is an open source, high-throughput, low-latency platform for handling real-time data feeds. Written in Scala, it is a massively scalable message queue architected as a distributed transaction log, making it highly valuable for processing streaming data in a persistent manner. It can handle massive instance losses without message losses, and while it makes use of its own binary message format, it results in much greater performance than other queue implementations.

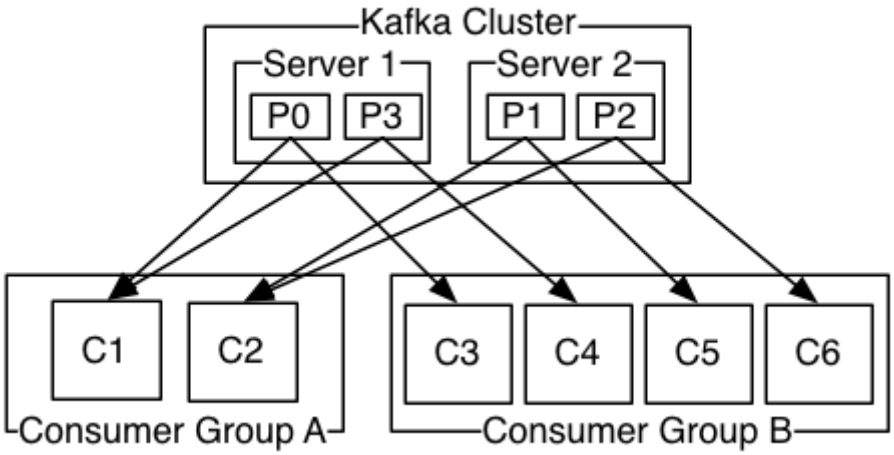

Figure 6: A two-server Kafka cluster with two partitions each, and two consumer groups with 2 and 4 consumers respectively. Source: https://kafka.apache.org/

Kafka's main strengths are:

- scalability,
- persistence,
- fault-tolerance,
- performance,
- ordering warranty,
- streaming-friendly (Apache Streams, Storm, Samza).

Its downsides are:

[28] https://kafka.apache.org/

- added complexity from managing cluster components.

RabbitMQ

RabbitMQ[29] is a messaging broker (an intermediary for messaging). It gives applications a common platform to send and receive messages, and messages a safe place to live until received.

RabbitMQ is open source message broker software that implements the Advanced Message Queuing Protocol (AMQP). RabbitMQ main strengths are:

- Reliability.
- Flexible Routing: Messages are routed through exchanges before arriving at queues. RabbitMQ features several built-in exchange types for typical routing logic.
- Clustering: Several RabbitMQ servers on a local network can be clustered together, forming a single logical broker.
- Federation: For servers that need to be more loosely and unreliably connected than clustering allows, RabbitMQ offers a federation model.
- Highly Available Queues: Queues can be mirrored across several machines in a cluster, ensuring that even in the event of hardware failure your messages are safe.
- Multi-protocol: RabbitMQ supports messaging over a variety of messaging protocols: AMQP 0-9-1, 0-9 and 0-8, and extensions, STOMP, MQTT, AMQP 1.0, HTTP.
- Many Clients: There are RabbitMQ clients for various languages: Java,.NET, Ruby, Python, PHP, JavaScript, more...
- Management UI: RabbitMQ ships with an easy-to use management UI that allows monitoring and controlling every aspect of your message broker.
- Tracing.
- Plugin System: RabbitMQ ships with a variety of plugins extending it in different ways.
- Commercial Support.
- Large Community.

ZeroMQ

ZeroMQ[30] (ØMQ) is a cross-platform asynchronous messaging library, developed by a community of contributors as an open source project. It provides a message queue for a distributed environment, without the need for a message broker. It is written in C++, but provides bindings in many languages including C#, Java, Python, Scala, PHP and others.

Initially, ZeroMQ was designed to support online stock-trading, where speed was crucial. Later, it was expanded with support for distributed applications, in an effort to provide a fast, decentralized, general use business messaging library.

The basic building blocks of a ZeroMQ implementation are sockets (very similar to TCP or UDP sockets) that represent many-to-many connections between endpoints. Messages are transported

[29] https://www.rabbitmq.com/
[30] http://zeromq.org/

in accordance with one of several message patterns that include publish-subscribe, push-pull, and request-reply.

Architectural elements are organized in a tree structure, where each one has an “owner” - a parent element responsible for graceful shutdown of every child. ZeroMQ defines communication management objects, such as listeners that dynamically create message passing elements. The latter have two parts: a “session” that directly interacts with a ZeroMQ socket, and an “engine” that is responsible for network communication. There are many kinds of engines available (e.g. TCP, IPC), and new engines can be implemented.

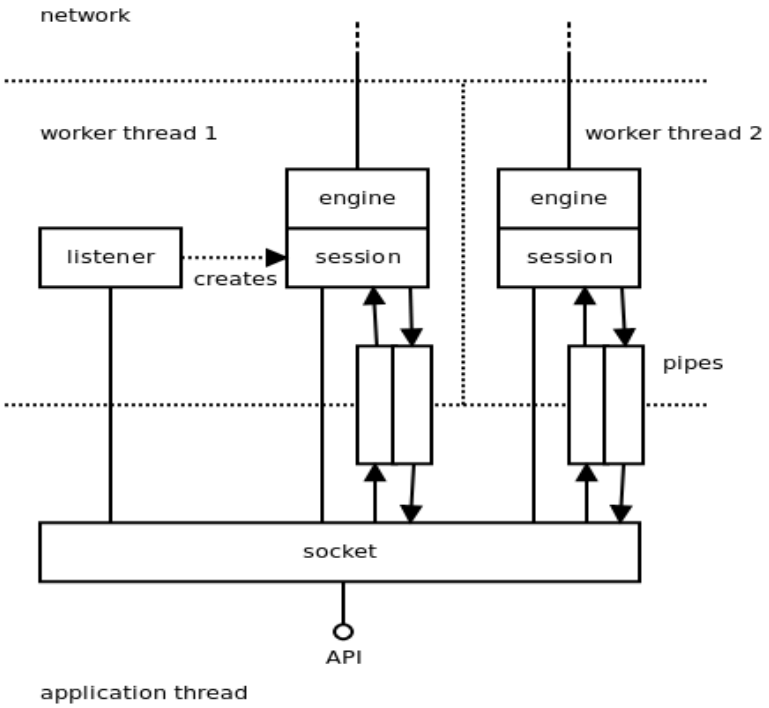

Figure 7: ZeroMQ general schema. Source: http://www.aosabook.org/en/zeromq.html

Features:

- Decentralized: No messaging server is a core feature. It is a library, not an application.
- No global state: It does not store its own internal state. It is up to the user to provide (if needed) a global state and properly handle cases of concurrent access to resources.
- High performance: Optimization of throughput and latency was the focus for ZeroMQ since the beginning of implementation. It is regularly benchmarked and maintained.
- Persistent communication: Focused on long-lived connections and optimization of most frequent operations i.e. passing of messages. Unfortunately, it also means that it is not optimized with respect to error handling and memory allocation.
- Concurrency: Internally implements an actor model to provide scalability, avoid locking and solve other multi-threading problems.
- Familiar API: Based on BSD socket API.
- Multiple messaging patterns: Offers a set of predefined, non-extendable, orthogonal messaging patterns simplifying configuration and a clear messaging scheme.

- No fixed architecture: Many different architectures (both distributed and centralized) can be implemented on top of it. While this provides flexibility and wide range of possibilities, it also requires more design and implementation work from the developer.

ActiveMQ

ActiveMQ[31] is an Apache project written in Java that implements the Java Message Service Client/Server pair. The messaging broker can be clustered, and uses a database for its persistence. It supports most of the current messaging protocols, includes several Enterprise Bus features, and provides with clients for several programming languages.

Extended features such as message groups, virtual destinations and composite destinations, and other such high-level functionalities.

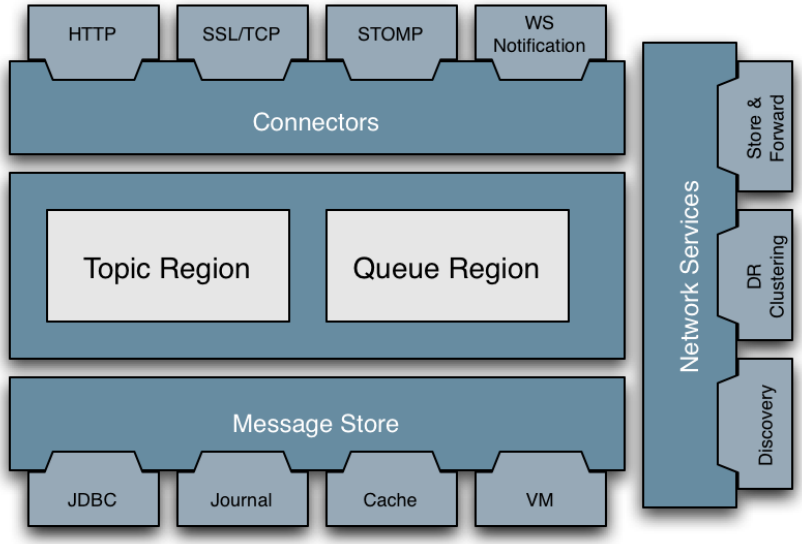

Figure 8: Main parts of Apache ActiveMQ. Source: http://activemq.apache.org/

The main features are:

- persistence,
- ordering warranty,
- fault-tolerance,
- synchronous,
- a rich set of features,
- multiple-protocol,
- REST API accessible.

Downsides:

- non-stellar performance,
- dependent on database for persistence.

[31] http://activemq.apache.org/

Mosquitto

Eclipse Mosquitto[32][33] provides a lightweight server implementation of the MQTT protocol that is suitable for situations from full power machines to embedded and low power machines. Sensors and actuators, which are often the sources and destinations of MQTT messages, can be very small and lacking in power. This also applies to the embedded machines to which they are connected, which is where Mosquitto could be run.

Typically, the current implementation of Mosquitto has an executable in the order of 120kB that consumes around 3MB RAM with 1000 clients connected. There have been reports of successful tests with 100,000 connected clients at modest message rates.

As well as accepting connections from MQTT client applications, Mosquitto has a bridge which allows it to connect to other MQTT servers, including other Mosquitto instances. This allows networks of MQTT servers to be constructed, passing MQTT messages from any location in the network to any other, depending on the configuration of the bridges.

The main features of Mosquitto are:

- It is an open source (EPL/EDL licensed).
- Mosquitto implements the MQTT protocol versions 3.1 and 3.1.1.
- MQTT provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for Internet of Things “messaging” such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers like the Arduino.
- It works on different operating systems like Windows, Mac OS X, Linux, etc
- Mosquitto is an iot.eclipse.org project. The Mosquitto broker code is also being contributed to Eclipse as part of a new project.

It also comes with two clients, *mosquitto_pub* and *mosquitto_sub*. *mosquitto_pub* client is used for publishing simple messages, while *mosquito_sub* is for subscribing to a topic and printing the message that it received.

[32] https://mosquitto.org/

[33] http://projects.eclipse.org/projects/technology.mosquitto

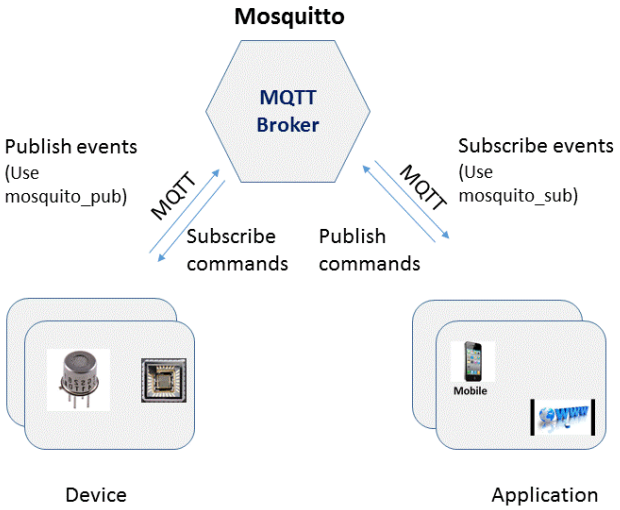

Figure 9: Example of Mosquitto broker events. Source: https://goo.gl/SNGCBe/

OpenDDS

OpenDDS[34] is an open source implementation of DDS specification (reference to section 6.a) programmed in C++ promoted by the company Object Computing Inc (OCI).

OpenDDS uses the IDL interfaces defined by the DDS specification to initialize and control service usage. Data transmission is accomplished via an OpenDDS-specific transport framework that allows the service to be used with a variety of transport protocols. This is referred to as pluggable transports and makes the extensibility of OpenDDS an important part of its architecture. OpenDDS currently supports TCP/IP, UDP/IP, IP multicast, shared memory, and RTPS_UDP transport protocols as shown in the figure. Transports are typically specified via configuration files and are attached to various entities in the publisher and subscriber processes.

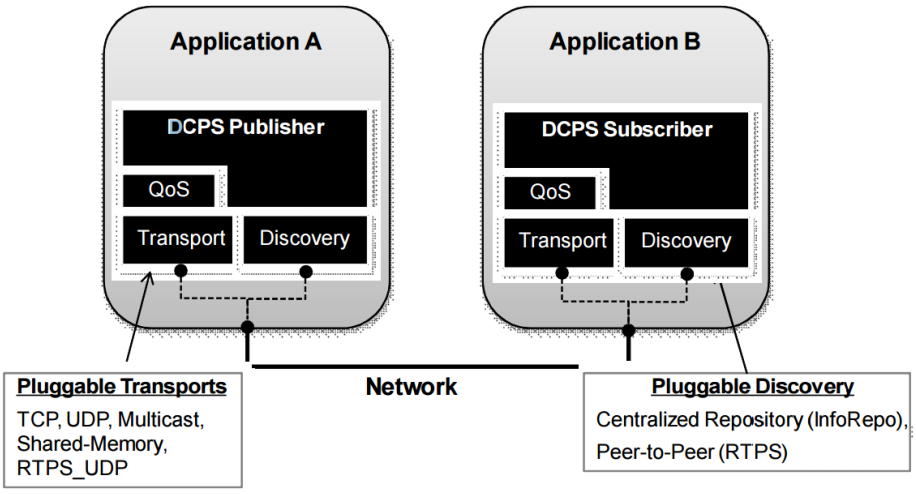

Figure 10: OpenDDS Extensible Transport Framework. Source: http://opendds.org/

OpenDDS supports the full set of DCPS Quality-of-Service (QoS) policies, including:

[34] http://opendds.org/

- Liveliness: Controls liveliness checks to make sure expected entities are still alive.
- Reliability: Determines whether the service is allowed to drop samples.
- History: Controls instance whose value changes before it is communicated to all Subscribers.
- Resource Limits: Controls resources that the service can use to meet other QoS requirements.

OpenDDS support C++ and provides a full-compliant Java interface with access to all the features. It is licensed for free usage and modification without permission, maintaining the license to the OpenDDS module but not forcing to distribute the code using OpenDDS or licensing it in any form.

OpenFire

Openfire[35] is a real time collaboration (RTC) server licensed under the Open Source Apache License. It uses the only widely adopted open protocol for instant messaging, XMPP. It provides reliable XMPP group chats and instant messaging and also is a backend for chat clients, including database registry and consultation. The server management is almost completely done by using a Web interface, so the usability and maintainability are notable. Openfire supports the following features:

- Web-based administration panel.
- plugin interface.
- customizable SSL/TLS support.
- user-friendly Web interface and guided installation.
- database connectivity (i.e. embedded HSQLDB or other DBMS with JDBC 3 driver) for storing messages and user details.
- LDAP connectivity Platform independent, pure Java.
- full integration with Spark (XMPP client).
- can support more than 50,000 concurrent users.

Openfire has a very active community (Ignite Realtime[36]) which supports and maintains this and other real time communication projects. Therefore, a great amount of plugins for extending the OpenFire server are available, making this server highly customizable and adaptable for specific needs. OpenFire community also develop and support messaging clients based on XMPP and fully compatible with OpenFire. Examples are Smack or Spark.

Others

IBM's WebSphere MQ, formerly IBM MQ, is a proprietary message-oriented middleware solution and Enterprise service bus launched in the 1990s, and available to a wide spectrum of architectures such as zOS mainframes, AIX, HP-UX, Linux or Windows.

Oracle Advanced Queuing is another proprietary message asynchronous queueing system, which is available on several platforms, and makes uses of the Oracle database for persistence.

[35] https://www.igniterealtime.org/projects/openfire/
[36] http://www.igniterealtime.org/

Mule is a commercial Enterprise service bus written in Java, with a community version offered under the CPAL free license. Besides connectors to several languages and an IDEA based on Eclipse, Mule also provides a toolkit focused on eHealth, providing support for HL7 standard messages. It supports AMQP, KMS and WMQ protocols, and the commercial provides features such as high-availability and scalability, and a management console.

2.3.2.2 Existing Platform middlewares

FIWARE

The FI-WARE[37] platform, supported by the European Commission, provides a number of middleware services for distributed applications, and a support framework for Internet of Things. The main Enabler is the IoT Broker, which manages the communication between applications and devices (or gateways).

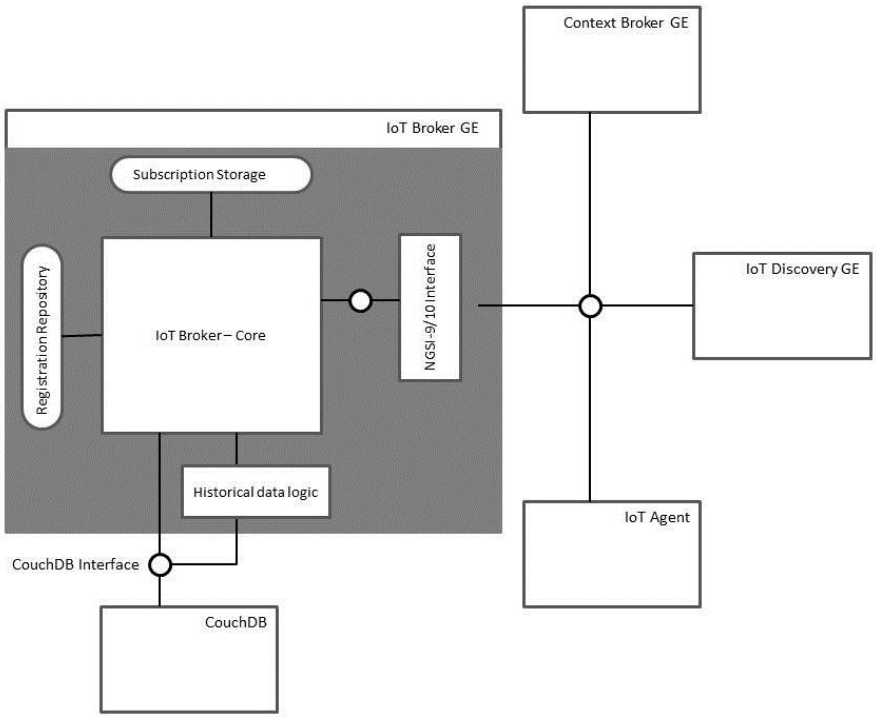

Figure 11: FI-WARE IoT Broker internal scheme. Source: https://forge.fiware.org/plugins/mediawiki/wiki/fiware

The IoT Broker provides with message subscription and registration, with persistence provided by CouchDB. Additional General Enabler provides further features that can be associated with the IoT Broker:

- The IoT Discovery supports device registry and resource discovery.
- The IoT Data Edge Consolidation that allows for real-time data filtering, aggregation and other processes.

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

- The Backend Device Management provides Agent definition and managements.
- The Protocol Adapter for support of CoAP's 6LoWPAN protocol.

Other General Enabler can be related directly or indirectly to this infrastructure. For example, the Data Edge Consolidation can use the Complex Event Processing to preprocess data streams to feed particular applications.

FI-WARE provides with open API definitions and mostly, but not exclusively, open-source implementations.

OpenIoT

OpenIoT[38] is building a novel open source platform for the IoT, which includes unique functionalities such as the capability to compose (dynamically and on-demand) non-trivial IoT services, following a cloud/utility based paradigm.

The OpenIoT architecture consists of seven main elements that belong to three different logical planes, as illustrated in Figure 12.

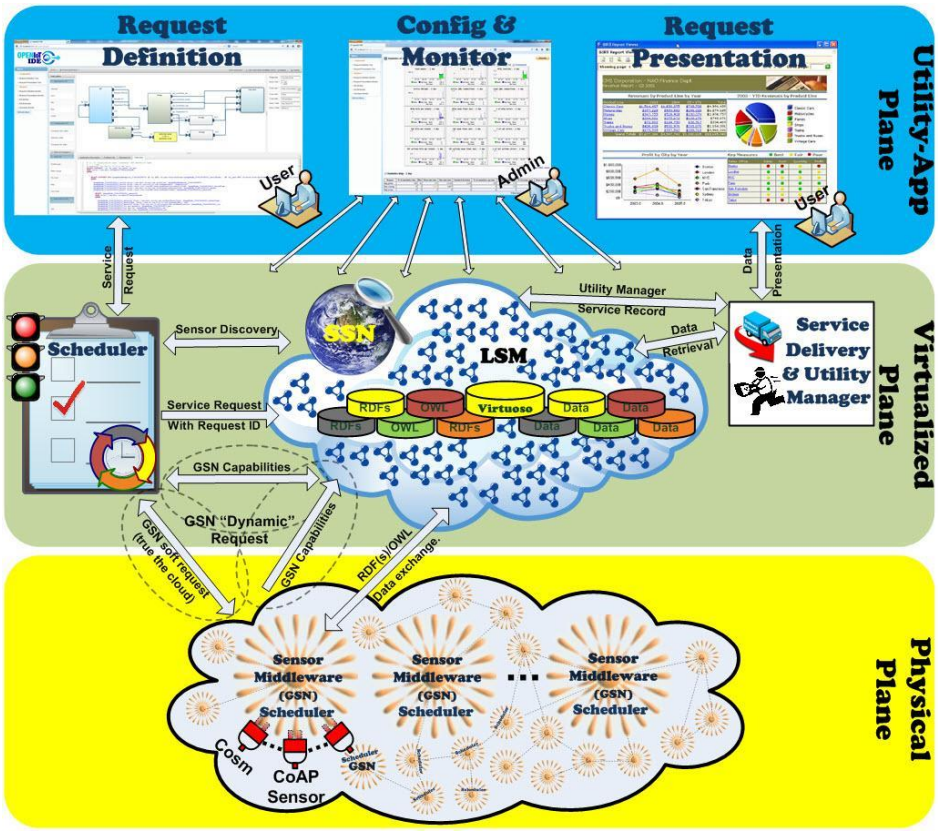

Figure 12: OpenIoT architecture. Source: https://goo.gl/HTF4In

OpenIoT is developing a blueprint middleware infrastructure for implementing/integrating Internet of Things solutions. The OpenIoT infrastructure will provide the means for:

- Collecting and processing data from virtually any sensor in the world, including physical devices, sensor processing algorithms, social media processing algorithms and more.

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

- Semantically annotating sensor data, according to the OpenIoT ontology, based on 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.

Butler

Some general considerations about BUTLER[39] platform:

- It is a set of enablers and services that provide means for building context-aware applications on top of smart connected objects.
- It provides not only generic APIs to access resources provided by IoT devices, but also additional services such as:
    - security service,
    - localization services,
    - behaviour prediction services,
    - context management services.
- Applications can be reused to enhance the user experience and security.
- It is specifically conceived for IoT devices and applications. It integrates different IoT devices and communication technologies in order to provide a homogeneous access to the underlying heterogeneous networks.

The main advantage of BUTLER platform is simplicity of use and its support of existing IoT protocols.

[39] http://www.iot-butler.eu/

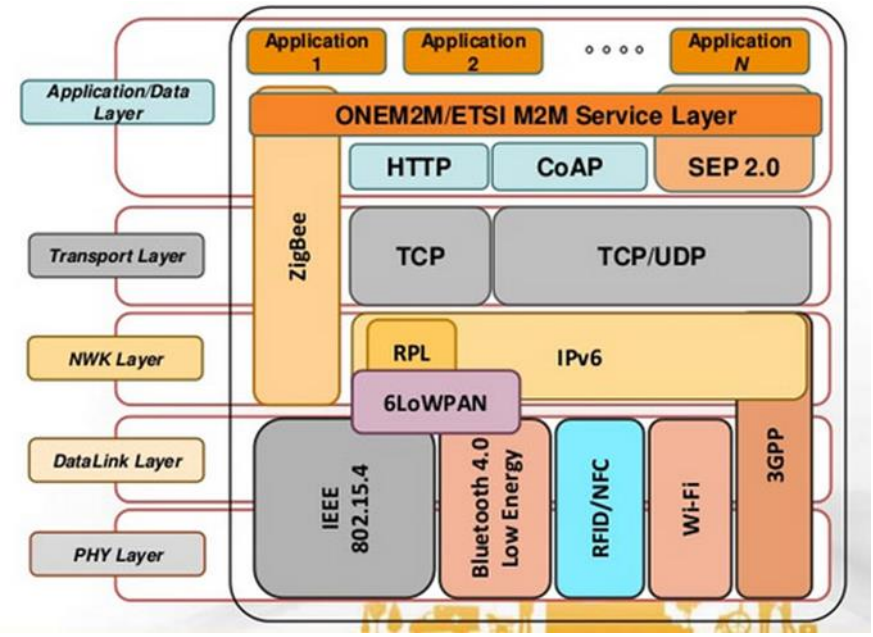

Figure 13: Buttler's layers technologies. Source: D3.1 Architectures of BUTLER Platforms and Initial Proofs of Concept.

The core of the BUTLER platform is the Gateway which provides:

- A unifying platform that bridges the communication between the physical and virtual worlds.
- An abstraction layer in order to access to IoT devices from various manufacturers using different protocols. It is based on a service oriented approach that allows better management of the dynamicity of the environments, easier and faster application development and other additional features such as service discovery, lookup, run-time binding and management.

SOFIA2

SOFIA is a middleware architecture allowing for the interoperability of several systems and devices. It allows making real information available for intelligent applications (Internet of Things).

Some of its main advantages are:

- open-source,
- multi-platform: available for MS Windows, Android, Linux, iOS and others,
- multi-language: portings to Java, JavaScript, C++, Arduino and others,
- communication agnostic: with implementations for TCP, MQTT, HTTP (REST and WebServices), Ajax Push, and others.

Its goal is to achieve interoperability among different applications that share semantic concepts.

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[40].

SOFIA2 is focused on the following areas:

[40] http://sofia2.com/

- Adaptation to the enterprise environment: High availability operation with distributed data centers.
- Simplified work with the Platform, 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 with 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 and others.

The SOFIA2 Platform can be conceptualized through these four concepts:

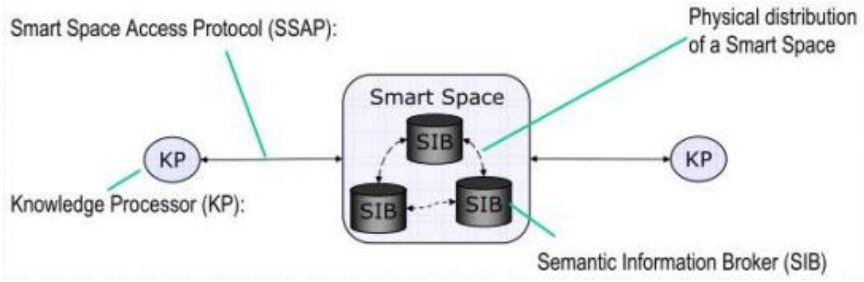

Figure 14: SOFIA2's conceptual blocks. Source: http://sofia2.com/

- **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.

OneM2M

The aim of oneM2M[41] is to develop technical specifications which address the need for a common M2M Service Layer that can be readily embedded within various hardware and software, and relied upon to connect the myriad of devices in the field with M2M application servers worldwide.

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

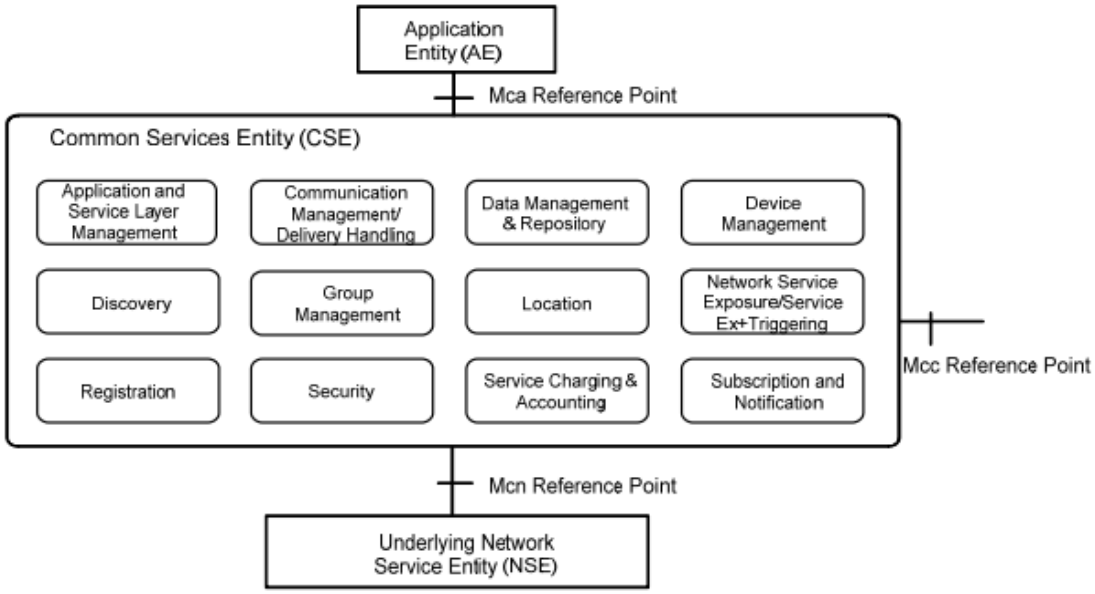

Figure 15: OneM2M Services Set. Source: OneM2M TS-0001: Functional Architecture http://www.onem2m.org/

In Figure 15 you can see the main services offered by OneM2M:

- Communication Management and Delivery Handling: provide communications with other CSEs (Common Service Entities), AEs (Application Entities) and NSEs (Network Service Exposures). It decides at what time to use which communication connection for delivering communications, and to buffer communication requests when needed.
- Data Management and Repository: is responsible for providing data storage and mediation functions (collecting data for Big Data aggregation, data conversion, etc.).
- Device Management: provides management of device capabilities on MNs (e.g. Gateways), ASNs and ADNs (e.g. Devices), as well as devices that reside within a network.
- Discovery: searches information on applications and services per attributes and resources.
- Group Management: is responsible for handling group related requests.
- Location: allows AEs to obtain geographical location information of Nodes.
- Network Service Exposure, Service Execution and Triggering: manages communication with the Underlying Networks to use network services over the Mcn reference point.
- Registration: processes a request from an AE or another CSE to register with a CSE in order to allow the registered entities to use the services offered by the CSE.
- Security: sensitive data handling, security administration, security association establishment, access control, authentication, authorization; and identity management.
- Service Charging and Accounting: provides charging functions for the Service Layer. It supports different charging models which also include online real time credit control.
- Subscription and Notification: provides notifications for event changes on a resource.

AllJoyn

AllJoyn™[42] is a collaborative open-source software framework that makes it easy for developers to write applications that can discover nearby devices, and communicate with each other directly regardless of brands, categories, transports, and OSes without the need of the cloud. The AllJoyn framework is extremely flexible with many features to help make the vision of the Internet of Things come to life.

It abstracts out the details of the physical transports and provides a simple-to-use API. Multiple connection session topologies are supported, including point-to-point and group sessions. The security framework is flexible, supporting many mechanisms and trust models. The types of data transferred are also flexible, supporting raw sockets or abstracted objects with well-defined interfaces, methods, properties, and signals.

AllJoyn is a distributed software bus in which:

- Each device runs a bus daemon.
- Applications communicate directly with a daemon.
- Daemons on each device communicate with daemons on other devices.
- Daemons do message routing and namespace management.

Conceptually peers are applications, not devices. Applications communicate with a local daemon. Daemons handle routing between devices.

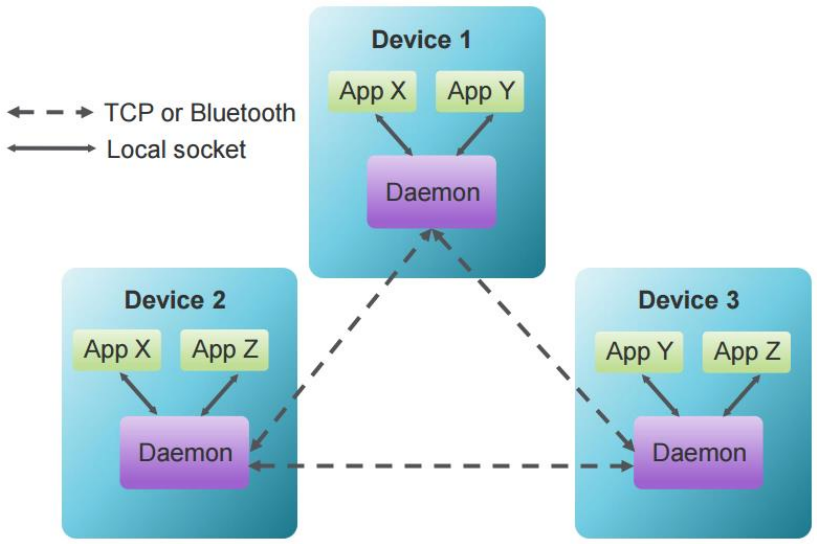

Figure 16: 3 devices communicating through the virtual channel through p2p links. Source: Lioy, M. (2011), Peer-to-Peer Technology Driving Innovative User Experiences in Mobile.

Its Bus formation is ad hoc, based on proximal discovery, while abstracting multiple discovery mechanisms. The protocol is transport independent, using a ground-up implementation of the D-Bus wire-protocol with extensions that support several networks including Wi-Fi and Bluetooth currently.

[42] https://allseenalliance.org/

Eclipse OM2M

Eclipse IoT is an ecosystem of companies and individuals that are working together to establish an Internet of Things based on open technologies. It provides open source implementations of the standards, services and frameworks that enable an Open Internet of Things.

Among the different projects, the Eclipse OM2M project[43], initiated by the French lab LAAS-CNRS, is an open source implementation of oneM2M and SmartM2M standard. It provides a horizontal M2M service platform for developing services independently of the underlying network.

OM2M provides an open source service platform for M2M interoperability based on the oneM2M standard. OM2M follows a RESTful approach with open interfaces to enable developing services and applications independently of the underlying network. It proposes a modular architecture running on top of an OSGi layer, making it highly extensible via plugins. 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.

OM2M includes several primitive procedures to enable machines authentication, resources discovery, applications registration, containers management, synchronous and asynchronous communications, access rights authorization, groups' organization, re-targeting, etc.

Bluemix

IBM Bluemix[44] is a cloud platform as a service (PaaS) powered by open source projects and developed by IBM. It supports several programming languages and 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. Bluemix supports several programming languages including Java, Node.js, Go, PHP, Python, Ruby Sinatra, Ruby on Rails and can be extended to support other languages such as Scala through the use of buildpacks.

Besides common IoT services, BlueMix provides with 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.

Others

Dioptase[45] is an IoT middleware oriented to data stream processing with a design focused on heterogeneous and distributed computing.

Kaa IoT[46] is another enterprise providing an IoT in a dual open-source and commercial license, promising to provide hardware agnostic integration, connectivity model independent and replicable, based on Zookeeper.

Other such platforms and platform expansions include **LinkSmart, Synapt-IoT** and **RoboMQ**.

[43] http://www.eclipse.org/om2m/
[44] https://www.ibm.com/cloud-computing/bluemix/
[45] https://mimove.inria.fr/dioptase/
[46] http://www.kaaproject.org/

2.3.2.3 Existing Cloud platform services

The number of current platform solutions to link the IoT domain with the cloud is huge. Some sources (Saverio Romeo, Beecham Research) talk about 300, while others (Maurizio Griva, Reply) go as high as 360. Usually, these platforms are focused on IoT services that are provided not as deployable self-hosted solutions, but that are given as a Platform as a Services (often associated to an Iaas). For example, infrastructure provider AWS has a platform solution for IoT called AWS IoT, while the Microsoft Azure Iaas and Paas includes the IoT Suite. Service providers such as IBM offers Watson IoT and other big companies such as Oracle and Cisco are offering solutions for IoT business. There are also other smaller companies are offering integrated IoT solutions, often following an open-source approach to their online services, as seen in the previous section.

Whereas any precise data on market share of the different solutions are not available, a few of them appear to enjoy much more popularity than others.

Amazon Web Services (AWS) IoT

The Amazon[47], like other services oriented to provide a generic IoT platform, connects to the devices or objects that report on their 'state' through a message broker using the MQTT protocol, to then make the data available to the Amazon Web Services, such as DynamoDB, AWS Lambda, Kinesis stream processing or S3, through a configurable rules engine, and finally to the IoT Applications than consume that information. The platform also includes a registry that can be managed through the AWS command line interface.

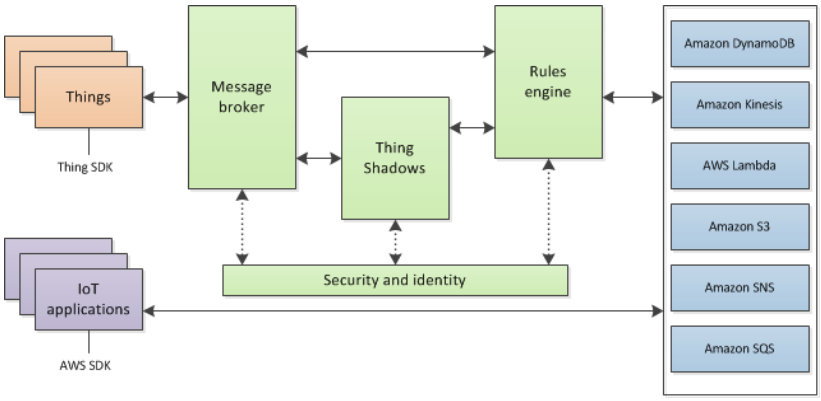

Figure 17: AWS IoT internal structure. Source: https://aws.amazon.com

Amazon also provides open sources SDKs, one for embedded C that can be used in most POSIX operating systems, an Arduino SDK, and as well as a node.js version for more powerful objects and gateways. It also provides an HTTP API for Web applications.

The communication between devices and AWS can be encrypted using TLS, together with X.509 authentication, and AWS also provides support for Identity Access Management of groups and roles, and the Amazon Cognito identification service.

Microsoft Azure IoT Suite

Overall, Azure[48] IoT Suite is the part of Azure Cloud Services that provides IoT device communication and data processing service. Together with services that Azure offers (including databases, Hadoop,

[47] https://aws.amazon.com/

[48] https://azure.microsoft.com

machine learning, etc.), it is an enterprise-scale solution on which a very wide range of IoT systems can be implemented.

IoT specific features include data collection and stream analytics in which streams can be pre-processes using SQL-like language, which allows definition of rules, and piped into other Azure products.

Azure IoT Gateway SDK implements the AllJoyn protocol for Windows and Linux. It supports communication via HTTP, AMQP and MQTT. The role of the gateway is to send or receive data to and from Azure, where the core of any IoT suite solution lies. On the cloud server side this functionality i.e. secure bi-directional communication, is offered by the Azure IoT Hub service.

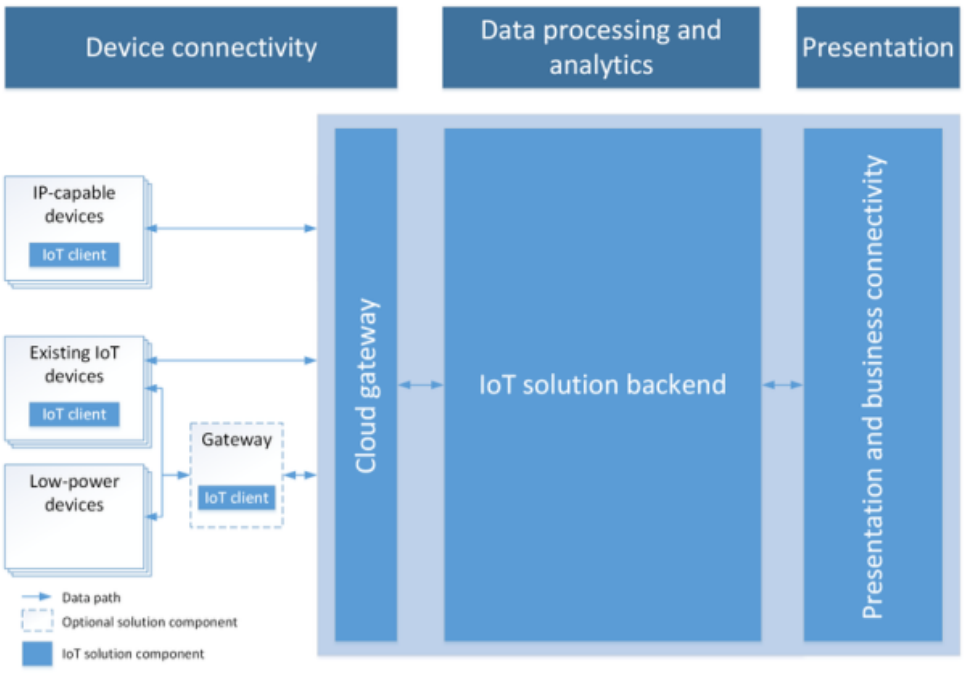

Figure 18: Generic IoT solution reference architecture used by Azure. Source: https://docs.microsoft.com

Features:

- implements AllJoyn protocol,
- offers high availability via the Azure cloud platform,
- deep integration with Azure means easy access to thousands of tools from Azure Marketplace,
- requires high amount of configuration,
- developers can buy pre-configured product packages,
- advanced tools are all paid, but basic solutions can be implemented on a free account,
- customized (negotiable) pricing for big solutions.

IBM Watson IoT

Watson[49] is a cognitive computing algorithm and system. Watson can analyze high volumes of data and by means of machine learning and artificial intelligent processes information in a new way more similar to the way a human can process data. The system is able to process natural language and translate a question into an action. This action results in a hypotheses based on evidence (statistical similarities). The system is able to continue to learn when it is working.

Watson is designed to make more data-driven decisions. Since most information and data is unstructured (like news, internet, databases) Watson can scan this data and find relations and similarities to the question that was asked, and filter which one is the best fit. Watson started as a supercomputer platform that was fed with lots of data. IBM entered Watson into the American TV-show Jeopardy in which it had to answer questions about common and specific knowledge. It was able to beat its human competitors. The new IoT version of Watson is cloud-based and made available to the public and companies. The main fields in which IBM states Watson will be working are: health care, financial services, retail and education markets.

The goal of Watson is to surface new insights in these working fields by analyzing the massive amounts of data available.

Cisco IoT Cloud Connect

Cisco Internet of Things (IoT) Cloud Connect[50] is the Cisco new mobility-cloud-based software suite. It offers a complete solution for service providers and mobile operators to provide better IoT products.

Cisco IoT Cloud Connect is a hosted managed cloud IoT platform that manages and automates IoT service delivery for mobile network operators. It is a holistic approach to help mobile operators address IoT and machine-to-machine (M2M) go-to-market needs. Its main features are:

- global IoT connectivity and management,
- lower point of entry and cost structure, for higher profitability,
- quicker time to market,
- capability for mobile network operators to win business at lower Average Revenue Per Connection (ARPC),
- complete cloud-based solution and joint Cisco and mobile operator partnership, not just point products.

[49] http://www.ibm.com/watson/

[50] http://www.cisco.com/c/en/us/solutions/service-provider/iot-cloud-connect/index.html

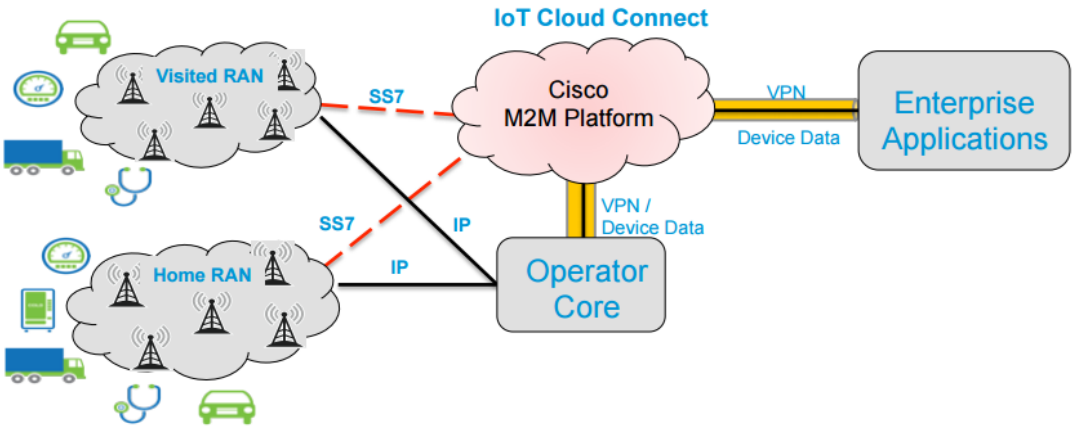

Figure 19: IoT Cloud Connect Architecture Overview. Source: https://www.ciscoknowledgenetwork.com/

2.3.2.4 Others

ThingWorx

ThingWorx[51] is an end-to-end enterprise-ready platform for ThingWorx components. It aims at providing a simplified, seamless approach for developers to create comprehensive machine-to-machine/IoT solutions.

ThingWorxAnalytics enables real-time pattern and anomaly detection, predictive analytics, and contextualized recommendations to IoT solutions, without the need for expertise in mathematics, statistics, or machine learning.

ThingWorx platform integrates with Amazon Web Services (AWS) IoT, and is also going to support Microsoft Azure IoT Hub. For enterprise customers, integrations with multiple cloud offerings enable several anticipated benefits, including such as faster development cycles within an open platform.

Salesforce IoT

Salesforce 'IoT Cloud'[52] is relying on Amazon Web Services (AWS) and aims at supporting IoT-related data analysis/storage/processing on top of every public or private cloud. The service is said to be both data format and product-agnostic, allowing external communication by means of connectors that interface Salesforce IoT cloud cooperates with third-party services (e.g. IoT Cloud connects data from websites, social interactions, and applications to Salesforce). Its internal component deputed to real-time event processing is 'Thunder' and it is a software stack built out of open source big data well-known products (Spark, Storm, Kafka, Apache Cassandra, and Salesforce's own Heroku PaaS). However, like other Salesforce services before it, IoT Cloud is designed to be used by business users rather than developers or engineers, meaning marketing managers and other non-technical employees can create rules about how the system treats the data it ingests.

[51] https://www.thingworx.com/

[52] http://www.salesforce.com/iot-cloud/

Oracle IoT

Oracle Internet of Things (IoT) Cloud Service[53] is a managed Platform as a Service (PaaS) cloud-based offering that supports critical business decisions and strategies by allowing to securely connect devices to the cloud, analyze data from those devices in real time, and integrate data with enterprise applications, Web services, or with other Oracle Cloud Services, such as Oracle Business Intelligence Cloud Service.

More specifically, it allows connecting existing sensors and devices with Oracle IoT Cloud Service Client Libraries and Gateways, available for a wide range of platforms and programming environments. It includes Device Virtualization, High Speed Messaging, and Endpoint Management. Oracle IoT also features powerful analysis tools for (i) real-time Stream Processing of incoming data streams with event aggregation, filtering, and correlation, (ii) Data stream enrichment with contextual information, and (iii) Event storage with querying and visualization support with integrated Oracle BI Cloud Service support and enable Big Data analysis.

Google Cloud IoT

Cloud Platform offers a full spectrum of cloud products and services including compute, storage, networking, big data, machine learning, authentication and end-to-end security. One of its vertical solutions indeed supports the IoT[54]. Thanks to the global Google infrastructure, Google Cloud IoT supports data and events streaming to the Cloud at massive scale. Taking advantage of Google Cloud Pub/Sub, real-time, reliable processing, and querying of IoT data are provided. Another interesting feature is Cloud Dataflow, a unified programming model for both batch and streaming data sources. Security is also a major concern: Google Cloud Platform APIs are secure by default with full encryption, backed by integrated and pervasive security across the entire infrastructure.

An important advantage of adopting Google Cloud is its global fiber network that allows very low latency delivery of data to and from IoT devices. In addition, reliability and security are enhanced because IoT data do not travel the public Internet through the majority of its time in transit.

2.3.3 Summary table

Though most middleware solutions being currently used are based on Message Queues or other similar Message Oriented Middleware, some take a distributed peer-to-peer approach without a single point of fracture, but at the same time require a lot more logic to be implemented in the applications, impacting negatively in their interoperability.

On the other hand, IoT platforms, either deployable or platform services, mostly differ in the additional services they provide on top of a similar architecture, though implemented with various technologies.

To compare more easily the different implementation of Message Queues that we have studied, we look to the following table:

[53] https://cloud.oracle.com/iot

[54] https://cloud.google.com/solutions/iot/

Queue	Scale	Federation	Ack	Policy	Persist	Sync	Licence	Performance
Kafka	Great	-	No	at least	Good (file)	Asyn	Apache	Great
RabbitMQ	Good	Good	Yes	either	Good	Asyn	Mozilla	Good
ZeroMQ	Good	Good	-	-	Good (db)	Asyn	Apache	Very Good
ActiveMQ	Good	Good	Yes	exactly	Good (db)	either	Apache	Good
Mosquitto	Good	Good	Yes	exactly	Good	Asyn	Eclipse	Very Good
OpenDDS	Great	distributed	Yes	At least	Average (file)	Asyn	Own ~ MIT	Very Good
ejabberd	Great	Good	Yes	exactly	Poor* (mem)	Asyn	GNU	Great
OpenFire	Good	Great	Yes	exactly	Good (db)	Asyn	Apache	Average
WebSphere	Very Good	Good	Yes	exactly	Good (db)	either	Comm	Good
Oracle Advanced Queuing	Very Good	Good	Yes	exactly	Good (db)	either	Comm	Good

Table 4: Message queues overview.

From Table 4 it seems clear that Performance, Persistence and Policy are features that are traded off in MQ implementations, and thus different uses cases, with different requirements, might need different message queue implementations to work as expected.

2.4 Application & Services Interoperability (AS2AS)

The main problems to achieve interoperability between applications and services are intrinsic to the nature of these. IoT services and applications are:

- Multiple and heterogeneous: There are many services and applications in the IOT platforms (e.g. CEP, Historical DB, Big Data Processing, Visualization, Analytics, etc.).
- Distributed: Some difficulties exist in order to connect their capabilities.
- Dynamic: The demand of new functionalities in the short term, speeds up changes and evolution of those services.

The IoT platforms do not have the capability to interact between each other at the application/service layer. For that reason, we are going to describe several approaches to make interoperable the applications and services provided by heterogeneous IoT platforms.

2.4.1 Common Approaches

We have analyzed different approaches that are useful to achieve interoperability between services and applications. Service composition is the main element supporting this goal. Consequently, in this state of the art we will put the focus on this technique, as well as on tools and technologies that are needed to implement it.

Native access should be considered as well as a method to access IoT platforms applications and services. Almost all IoT platforms provide a public API to access their services. The APIs are usually based on RESTful principles, and allow common operations such as PUT, GET, PUSH or DELETE. However, there are other IoT Platforms (e.g. Open-IoT) that don't include a REST API for easing the development of Web services, but used different interaction means.

Application, data and device catalogues dedicated to the IoT are generally missing. With a solution based on a Service Catalogue, we will be able to register applications to make them discoverable. Furthermore, it will offer a description or detailed information about services/applications. Utilizing Service Discovery will make it possible to discover information about IoT services. It is also desirable to consider the possibilities provided by the virtualization of services and applications.

We have also analyzed the advantages that wrappers can provide to IoT. By the term wrapper we mean a specific program able to extract data from Internet sites or services and convert the information into a structured format.

Finally, service composition encompasses all those processes that create added-value services, called composite services, from existing services in the IoT platforms.

2.4.2 Literature review

2.4.2.1 Service Virtualization

Services can provide an IoT platform with powerful functional features by presenting an API to communicate with other components, and while these could be deployed in a particular fixed infrastructure, it is common practice to utilize them in flexible virtual environments that offer a number of benefits.

For example, since IoT services loads can greatly vary in time, service virtualization provides us with flexible mechanisms of scaling such services by creating additional instances of that service whenever needed, in order to handle the additional load while maintaining the quality of the service. This approach also greatly improves resource efficiency, since unused resources can be released.

Virtual services can be instantiated in different geographical areas, in order to accommodate load that is not necessary evenly distributed while maintaining a low latency. It can even allow for hybrid deployments, where instances of the service can be deployed to different infrastructure providers, for example both a private cloud (to handle the expected constant load) and one or several public cloud providers to cover for the potential bursts of usage in different locations, reducing the overall total costs of ownership.

Virtualization of services can also simplify the monitoring of the infrastructure, network issues and security incidents. Mechanisms for auditing and other such features are often available as part of the IaaS, or as a parallel PaaS, simplifying the implementation of such measures.

Service deployment methods

MaaS or Metal as a Service is an IaaS approach that is actually not virtualized, but it provides mechanisms for quick provision of a vast amount of resources over the infrastructure. The advantage over virtualized infrastructure is that the reserved resources are exclusively used by the owner, avoiding any conflict between instances that can happen in virtualized environment related to security or performance, especially if making heavy use of over-provisioning. It is also for this reason that the reserved resources tend to be more expensive, since an idle reserved CPU will not be used by another instance, whereas in an over-provisioned virtual environment these idle resources could be used by other instances.

Virtualized IaaS, on the other hand, is the cornerstone concept in a modern cloud enabled workflow. They are cheaper and more resource efficient than MaaS, but provide a lower quality service with less warranties for performance, stability and security.

Some of the most commonly used IaaS implementations are OpenStack[55], VMWare[56], Microsoft Hyper-V[57] and Citrix Xen[58].

An emerging technology within virtualization is the usage of Containers. The idea behind them is that resources are wasted by creating so many different virtual machines, each with a complete running copy of the full-fledged operating system and several libraries needed. To that end, many Containers can be run within a single operating system instance, using the same library pool, but each container inside an independent virtual environment. Currently the most popular container implementation is Docker[59], which wraps a deployable service in a complete virtualized file system that contains everything needed to run: the code, runtime, system tools, system libraries; that is, anything that can be installed on a server.

Simulate a service behaviour in a production environment

In software engineering, service virtualization is a method to emulate the behaviour of specific components in heterogeneous component-based applications such as API-driven applications, cloud-based applications and service-oriented architectures. In general, in current emerging Internet of Things (IoT) platforms, service virtualization can now be used in cloud-based applications as well. The main objective of service virtualization is to provide development and testing environments access to dependent system components, which are currently not available but which are necessary to test an application.

Service virtualization is useful when the dependent components of the application are:

- being developed,
- controlled by external sources and have limited availability,
- being used by multiple stakeholders and are not conveniently available,
- difficult to provision or configure in a test environment,

[55] https://www.openstack.org/
[56] https://www.vmware.com/
[57] http://technet.microsoft.com/library/cc753637
[58] https://www.citrix.com/products/xenserver/
[59] https://www.docker.com

- costly to use. e.g. accessing data from the cloud.

The development of IoT applications has many dependencies on other components and services. It is difficult for these to be available at just the right time when a component is being tested. It is here that service virtualization comes in handy and becomes an effective tool to speed up development and testing work, without having to wait for a dependent component to be completed or become available.

2.4.2.2 Service Catalogue and Service Discovery

IT service catalogue can be defined as a list of resources and offerings within a given domain or organization. There are a few applications, data and device catalogues dedicated specifically to the IoT. However, there exist general-purpose tools and standards for cataloguing and managing services that can be considered applicable also in the IoT domain. Desired functionalities that we have identified in our search are:

- register the applications to make them discoverable,
- offer a description or detailed information about the services/applications,
- use the same metadata annotations, and then create a point of interoperability,
- allow to publish linked-data descriptions of resources,
- unify data catalogue with semantics,
- discover information about IoT services.

Below we list selected tools and standards that have been developed so far and have some of the desired features.

UDDI

The Universal Description, Discovery, and Integration (UDDI)[60] protocol is a central element of the group of related standards that comprise the Web services stack. Its development was led by the UDDI consortium of enterprise software vendors and customers - Organization for the Advancement of Structured Information Standards (OASIS). The UDDI specification defines a standard method for publishing and discovering the network-based software components of a service-oriented architecture. It was designed to be requested by SOAP messages and to provide access to Web Services Description Language (WSDL) documents. Unfortunately, UDDI was not adopted on a scale that was expected by its creators i.e. major public node were closed and in 2007 OASIS decided to complete its work. It is mentioned here for the reason of completeness of overview as a noteworthy historic initiative.

WSDL

WSDL[61] is an XML format for describing network services, which was used for service description in UDDI, as a set of endpoints operating on messages containing either document-oriented or

[60] http://uddi.xml.org/

[61] https://www.w3.org/TR/wsdl20/

procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. Aspects regarding semantics expressed in WSDL are discussed in 2.5.2.2. In a nutshell, WSDL operates at the syntactic level providing functional description, however it cannot unambiguously determine what the service does (the syntax is specified but not the meaning). The extensions allow introducing semantics but it is work considered only for WSDL as a description language, and not full UDDI solution. More on adding semantics to WSDL is available in 2.5.2.2.

HyperCat

HyperCat[62] standard is a hypermedia catalogue format designed for exposing information about the Internet of Things assets over the Web. Whereas UDDI was inspired by the idea of public UDDI nodes with service catalogues expressed in WSDL, HyperCat extends this idea by proposing catalogues format allowing horizontal scalability and interoperable IoT discovery on the Web.

HyperCat allows a server to provide a set of resources to a client, each with a set of semantic annotations. Specifically, each HyperCat catalogue is an array of any number of Uniform Resource Identifiers (URIs) representing resources annotated with RDF-like triples. Implementers are free to choose or invent any set of annotations to suit their needs. A set of best practices and tools are currently under development. Using similar or overlapping semantics in multiple catalogues increases the possibilities for interoperability.

HyperCat is an open, lightweight JSON-based hypermedia catalogue format for exposing collections of URIs over HTTP/HTTPS. HyperCat is simple to work with and allows developers to publish linked-data descriptions of resources.

iServe

iServe[63] is a platform for service publication, analysis, and discovery. Besides providing the typical features of service registries, it offers additional functionality that exploits service descriptions, service annotations and further data gathered and derived from the analysis of these descriptions, data crawled from the Web, periodic monitoring and user activities.

A crucial feature of iServe is that it publishes service descriptions as Linked Data, no matter their original format (underlying formalism originally used for service description). These services, which are referred to as Linked Services, help application developers to locate Web services and Web APIs they can use to process or enrich their data.

iServe uses a minimal set of vocabulary, called Minimal Service Model (MSM), for describing both WSDL services and Web APIs in RDF that can be used for service matchmaking based on SPARQL. This model abstracts from original approaches to annotating the services e.g. SAWSDL, OWL-S (service semantic annotations are discussed in 2.5.2.2). However, original services' semantic annotations are used to automatically generate the appropriate RDF statements according to the Minimal Service Model, and then expose them as Linked Data. Whereas HyperCat does not impose any particular semantic annotations, iServe is based on MSM. MSM is a simple RDF ontology that

[62] http://www.hypercat.io/

[63] http://iserve.kmi.open.ac.uk/

provides common model for Web service and API description and additionally uses the SAWSDL, WSMO-Lite and hRESTS.

iServe provides three interfaces to interact with published linked data about services:

- Web-based application, called iServe Browser, allowing users to browse, query, and upload services to iServe.
- A SPARQL endpoint where all the data hosted in iServe can be accessed and queried.
- A RESTful API that enables creating, retrieving and querying for services directly from applications.

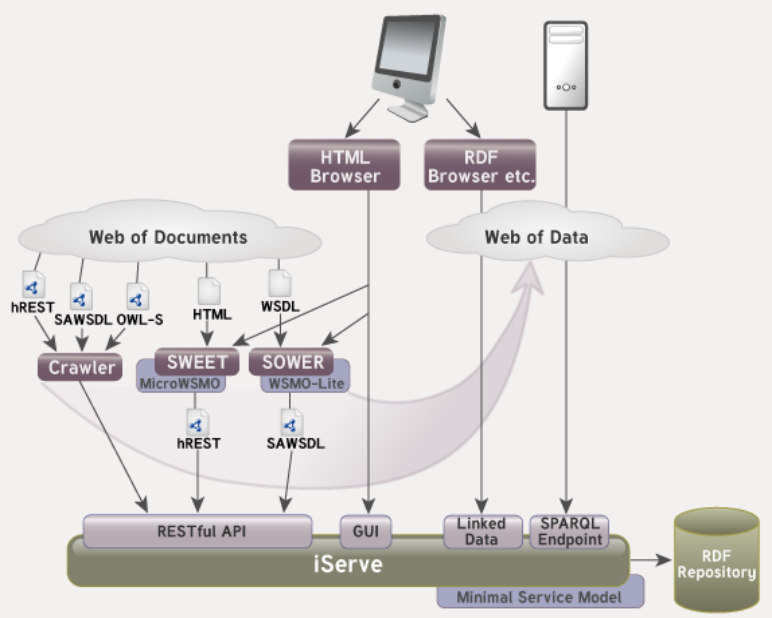

Figure 20: iServe architecture. Source: http://iserve.kmi.open.ac.uk/

2.4.2.3 Wrapping Technologies and the IoT

The goal of this section is to make a survey of the existing wrapping technologies needed to discover and incorporate existing Web/IoT services to facilitate interoperability. The survey presents a first explanation of what a wrapper or Web wrapper is, the kinds of wrappers and existing wrappers on the market and finally how it is possible to wrap existing IoT services by using a specific technology.

Wrappers are specific programs able to extract data from Internet web sites or services and convert the information into a structured format. More specifically, wrappers have three main functions: retrieval that allows to access Web through HTTP protocol, extraction that allows to identify specific information, and conversion that returns the information in a particular format. A piece of software that facilitates the generation of web wrappers through its engine that usually specifies the techniques of extraction, extraction rules, accessible data types, data transformation algorithm and so on is called a web wrapping toolkit. Once the wrappers were written manually by expert programmers using a specific language like Java. However, this meant writing a different wrapper for each web site which made the manual approach both time consuming and error prone. Later, using methods of artificial intelligence and machine learning the wrappers became both semi-automatic and automatic. In the semi-automatic approach, users are lead through a graphical user interface step by step to teach them what information to extract and where from. For automatic

approach, the wrappers are generated using machine learning. Usually, a set of rules is generated by using specific training examples. However, this type of wrappers generally returns varying result(s) depending on the quality of the given training examples and the complexity of the web sites that are wrapped. In the literature there is a different classification about the wrapper. For example, Kuhlins et al. [22] simply divided the toolkits into two general categories: commercial and non-commercial. Laender et al. [23], categorized these toolkits according to ways in which the wrappers are generated. Firat [24] developed a 3-dimensional matrix in classifying academic toolkits, based primarily on how mechanism works behind wrappers generation.

Laender and Ribeiro-Neto [23] proposed a taxonomy to classify web wrapping toolkits based on toolkits' respective extraction techniques. They divided web wrapping toolkits into six categories:

Language for wrapper development:

Specially designed programming languages were developed to assist programmers in creating web wrappers more efficiently. Examples include Jedi [25], TSIMMIS[64] and WebQL[65].

HTML-Aware tools:

HTML-aware toolkits treat web pages as a document tree, and rely on the underlying HTML hierarchical structure of the web pages for data extraction. Examples include XWRAP Elite[66] and Lixto[67].

NLP-based tools:

Natural language processing (NLP) is a technique for extracting data from plain text documents where there is no structured presentation or layout. Examples include RAPIER [26] and WHISK [27].

Wrapper Induction tools:

Typically, wrappers are automatically generated by feeding training examples to wrapper induction toolkits, for which delimiter-based extraction rules are derived. Examples include WIEN [28] and STALKER [29].

Modeling-based tools:

Toolkits in this category create wrappers based on a pre-defined domain model, which specify a certain desired structure. Examples include NoDoSe [30] and Kapow[68].

Ontology-based tools:

Ontology-based web wrapping technologies make use of application ontology to locate data contents in the document. These constants are then used to construct objects. The pioneer effort of this group of toolkits comes from the Data Extraction Group[69] in Brigham Young University.

[64] http://infolab.stanford.edu/tsimmis/
[65] http://www.ql2.com/products-services/ql2-webql/
[66] http://www.cc.gatech.edu/projects/disl/XWRAPElite/
[67] http://www.lixto.com/
[68] http://www.kofax.com/data-integration-extraction
[69] http://www.deg.byu.edu/

Examples of Web Wrapping toolkits are ContentMaster[70], DB2 Information Integrator for Content[71], WebMethods Integration Platform[72], Network Query Language (NQL) [31], Kapow, Visual Web Task[73], Visual Wrapper[74], Connotate Platform[75], Webinator[76], WebQL, WIEN, WinTask[77], XWRAP Elite.

Application wrapping techniques to IoT devices

Producers of electronic devices provide also application services to collect data coming from the devices themselves. A very easy approach to support users in building customized applications on demand, is discussed in Fujima and Jantke [32]. Interoperability for IoT devices is very important, and different middleware components have been developed. Many of these components are developed as Web services and IoT devices send or fetch information through them. Starting from these components to build new services, users need to know a programming language, web technology, framework, library perfectly and sometimes this is very hard. To facilitate end users to reuse these components it is possible to wrap Web services through visual components and provide a media platform to build and distribute custom applications with the wrapper software component such as depicted in the following figure.

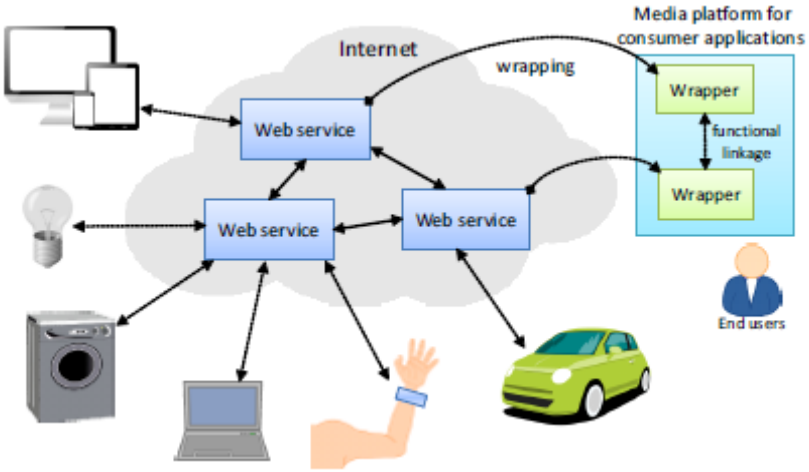

Figure 21: Wrapping of Internet of Things with Webble Technology.

As an example, Webble Technology [33] is a middleware based on a meme media technology. The word “meme” indicates a unit carrying knowledge, ideas, arts as the gene that characterizes the human, cultural and biologic evolution. “Meme media” indicates a media that accelerate the distribution memes among people through the Internet and application services that use it. In meme media platform, each functional component is wrapped as a visual component that provides a standard interface. It uses MVC architecture applicable to various domains. It is possible to change the behaviour of every component and make a connection between them with an easy drag and

[70] http://www.ebizq.net/news/7075.html
[71] https://www.ibm.com/analytics/us/en/technology/db2/
[72] http://www2.softwareag.com/corporate/products/webmethods_integration/integration/
[73] http://visual-web-task.software.informer.com/
[74] http://www.lixto.com/
[75] http://www.connotate.com/platform/
[76] https://www.thunderstone.com/texis/site/pages/webinator.html
[77] http://www.wintask.com/

drop operations. A real case study was addressed by using a fitbit device and implementing specific functionality for retrieving and visualizing data, modifying and manipulating queries and providing interactive ways to compare results of different query. To realize these functionalities a framework has been implemented.

2.4.2.4 Service Composition

Services can greatly benefit from the usage of other existing services in order to simplify their development and allow for better focus. Whether a complex system is broken into separate decoupled services to improve their scalability, or a simple proof of concept relies on existing services to create a minimum viable product without much effort, making use of one or several services is a convenient way to create complex yet reliable systems. The composition of such services can be as simple as one service (i.e. an ecommerce site) making use of a second service (i.e. a payment gateway), to very complex and flexible schemas of interconnection that need a coordinator to manage the mesh of requests and responses.

In particular, in the context of Internet of Things, service composition can be understood as allowing routes for the data to be treated before reaching an application or end-user, or agents requiring information or processes from other such agents. Such compositions can help provide more valuable information and actions than plain raw data, tailored to a particular receiver or purpose.

2.4.2.4.1 Mash-up

Mash-up is a way to compose a new service/application/data from existing applications/services called Mashable Application Components (MACs). MAC is a functional entity that can be executed and combined that provides access to resource or functionality.

Mash-up composition can be considered from the front-end perspective in which components interface both other components and the human end-user through a user interface. In contrast to back-end service composition, front-end mash-up has a direct impact on application's look and feel. Front-end components can be widgets that provide information presentation and handle user interaction while back-end components are responsible for information processing. Note, however, that mash-up is often defined as simply as a composition of services with less complex constructs than service composition.

FIWARE

In FIWARE project [34], which follows the aforementioned distinction, mash-up execution engines are characterized by:

- relying on Web standards including standards on mash-up description languages,
- openness, extensibility,
- ability of integration in multiple channels,
- users executing their mashups from their favourite browser,
- persistence of the state of the mash-up.

Web application mash-up engines provide methods for integration of heterogeneous data, application logic and user interface components in order to create new composite application. Note that this kind of applications could be developed with a traditional programming approach, however

mash-up promotes reusability, shareability and do-it-yourself applications from off-the-shelf components.

According to FIWARE project, *the Application and Services Ecosystem and Delivery Framework in FI-WARE comprises a set of generic enablers (i.e. reusable and commonly shared functional building blocks serving a multiplicity of usage areas across various sectors) for creation, composition, delivery, monetization, and usage of applications and services on the Future Internet.*

FIWARE proposes an Application Mashup GE that addresses the composition of Web applications from the front-end perspective by not only service providers but also end users. Application Mashup GE specifies tools that do not require programming skills i.e. they enable rapid prototyping of mash-ups from available components.

In the architecture, there are two main components regarding application mash-up and service composition: aggregator and mediator. The aggregator allows the creation and execution of mash-up applications (or composed services). The mediator is needed to provide communication between different components.

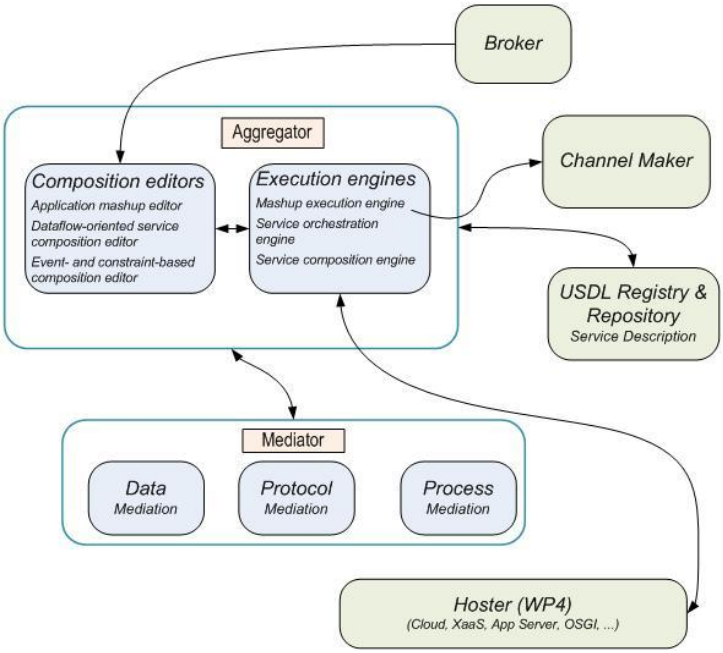

Figure 22: High-level architecture of FIWARE mediator and aggregator. Source: https://goo.gl/Ggd9ZI

Collaborative Open Market to Place Objects at your Service (COMPOSE)

The COMPOSE project[78] aims to integrate the IoT with IoS through an open marketplace, in which data from Internet-connected objects can be easily published, shared, and integrated into services and applications. One of the issues addressed by the project is the ad-hoc creation, composition, and maintenance of service objects and services. COMPOSE partially funded the development of Glue.things[79].

[78] http://www.compose-project.eu/
[79] http://www.gluethings.com/

Glue.things

It is a Platform as a Service (PaaS) designed for application and services in the Internet of Things domain. Glue.things offers the development tools to wire data of Web-enabled IoT devices to the Web of Services by connecting them to the platform and designing mash-ups in Glue.things dashbord. Glue.things offers a mash-up toolkit (client library, web dashboard, REST APIs) to connect devices and dashboard that supports integration, real-time communication (Web Sockets, MQTT and CoAP-based on real-time data stream networks such as MeshBlu, PubNub and servIoTicy), and data stream mash-ups, triggers / actions and finally distributed deployment of these mash-ups. Mash-up editor is built on Node-RED and applications are deployed as Node-RED applications. Service description is based on: WSDL, SAWSDL, OWL-S and RDF.

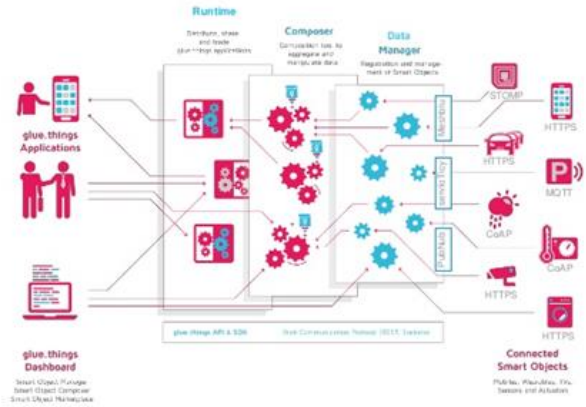

Figure 23: Glue.things overview. Source: http://www.gluethings.com/

Node-RED

Is a visual tool for wiring IoT i.e. hardware devices, APIs and online services[80]. Node-RED is also discussed in section 2.4.2.5.

Kofax Kapow

Is a commercial platform that provides mash-up functionality[81]. It enables to connect and integrate a variety of data sources and applications e.g. dynamic sites built with JavaScript and AJAX techniques, databases, data from Excel, XML, XLS, RSS feeds and from APIs based on SOAP, REST, XML and JSON. The visual IDE allows for building data integration flows and deploying to management console from where flows can be scheduled or published as lightweight applications.

Health Mash-up system -

One example of a health mashup is Mobile Health Mashups [35]. It is a service that collects data from varies health sensors and mobile applications. It produces various graphs and answers to general questions about correlation in health state and various activity or contextual data.

The system has a server running custom algorithms, rest APIs for various sensors and a mobile widget for expressing findings from the system.

[80] http://nodered.org/

[81] http://www.kofax.com/data-integration-extraction

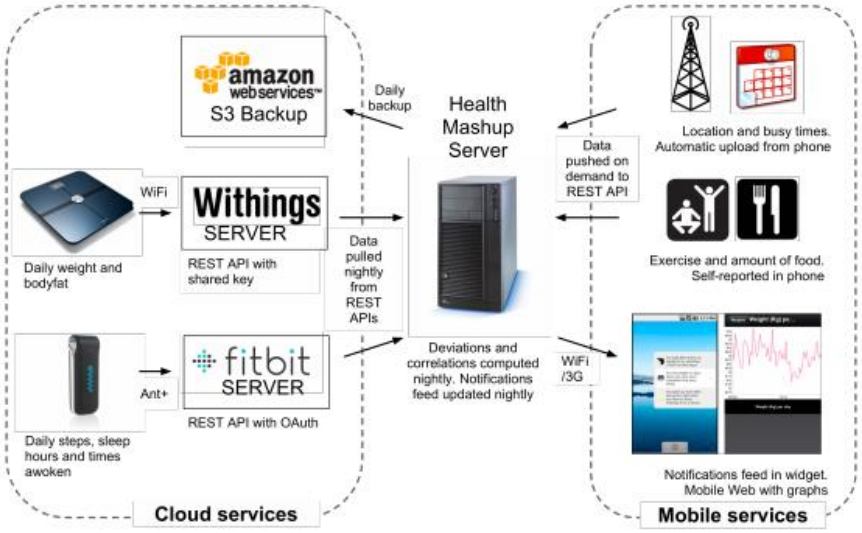

Figure 24: Health Mashup system overview. Source: [35]

2.4.2.4.2 Service Orchestration

Service orchestration is a service composition strategy based on the use of a central module that controls all inputs and outputs of the atomic services (components) and performs the service composition logic. This component is called the orchestrator and needs to have control over all the services composing the business processes. This is also called centralized service composition approach [36].

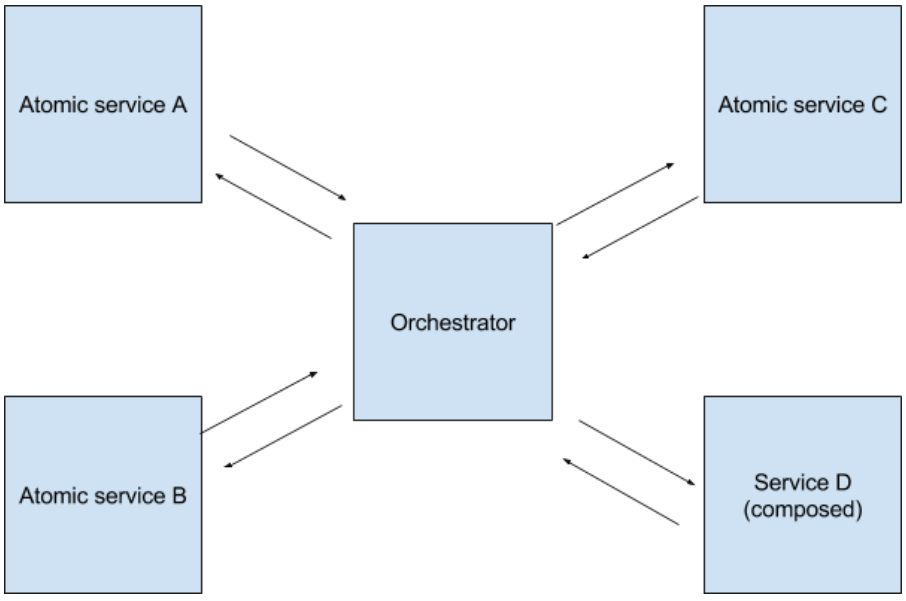

Figure 25: Orchestrator scheme.

Orchestration in relevant IoT platforms

Service composition through orchestration is a basic feature in most of the IoT platforms analyzed. Since the considered IoT platforms are intended to cover a full IoT ecosystem, natural solution for service composition is the use of orchestration.

FIWARE

It has an orchestration service inside, the PaaS generic enabler. This is an orchestration engine to launch composite cloud applications based on text templates.

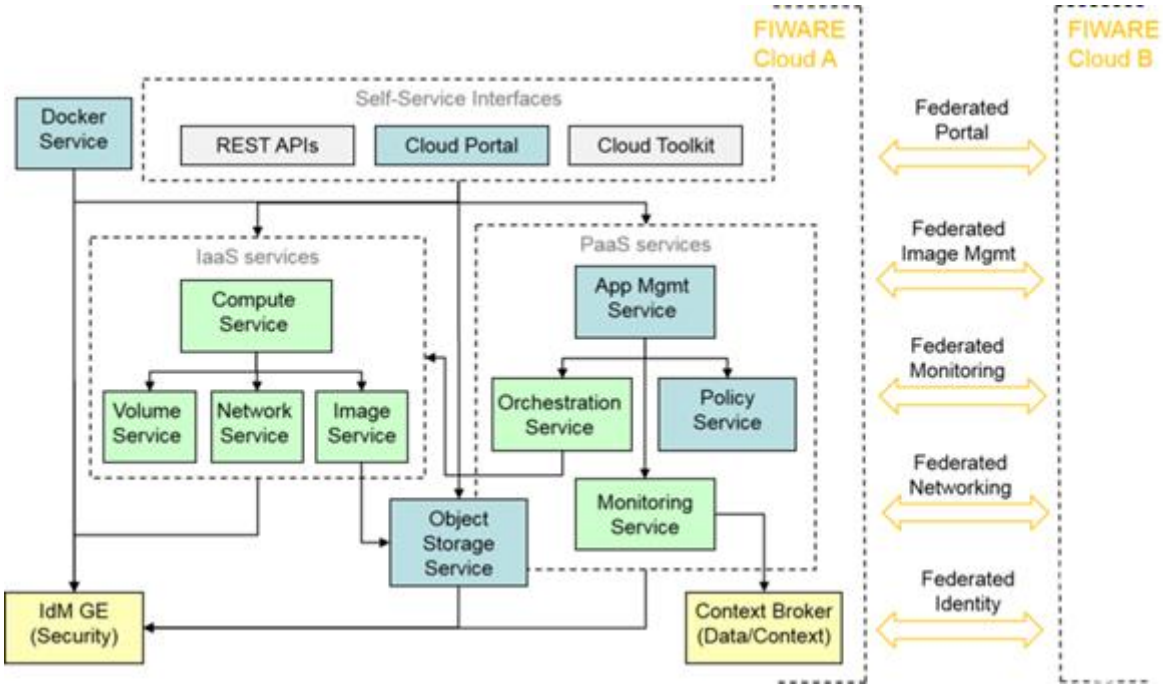

Figure 26: FIWARE orchestrator. Source: D2.2.2 FI-WARE High-level Description

SOFIA2

There is little information about SOFIA2, but the release 2.12 declared to have an API orchestrator within the API Manager[82].

Sensinact

Provides a composition and orchestration function for easing the development of custom business logic.

[82] https://about.sofia2.com/2014/09/26/sofia2-release-2-12-0-published/

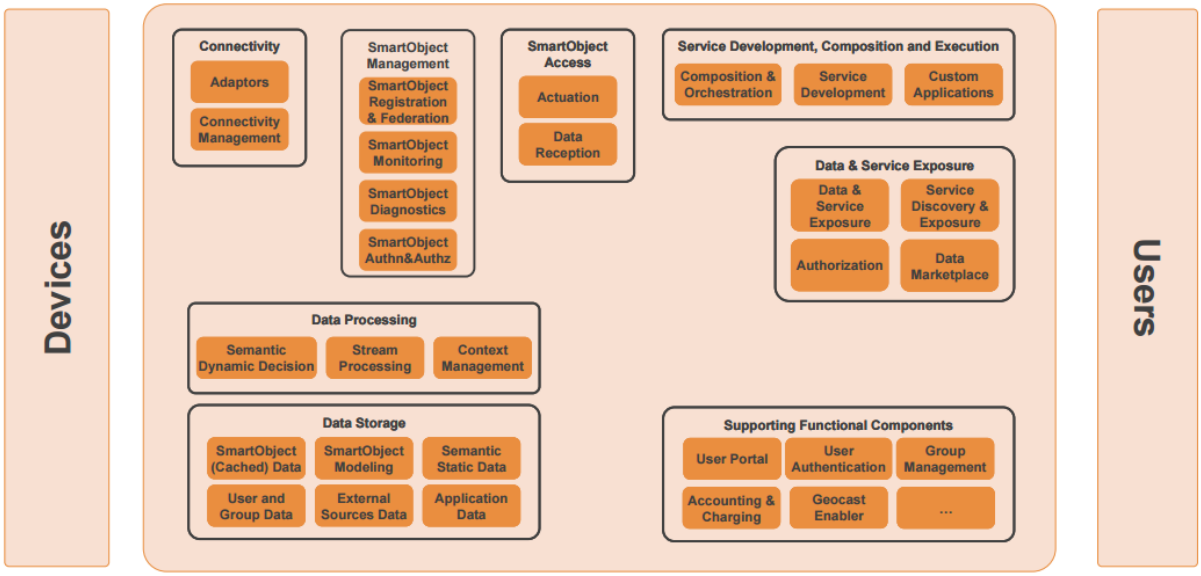

Figure 27: Sensinact orchestrator. Source: https://goo.gl/dquUM2

UniversAAL

It has a service composition tool using orchestration. There are two versions of the component, one using OWL-S and other based on JavaScript [37].

The universAAL component for service orchestration is ASOR (AAL Spaces Orchestrator) and follows a script based approach for composition definition.

OneM2M

Does not provide a service orchestration specification in its standard, alike, the main open implementation of the OpenM2M standard, Eclipse OM2M, does not have any implementation for service orchestration.

OpenIoT

Does not define a component for service orchestration itself, relying on applications to develop their own service composition, if needed.

Relevant research projects using orchestration

Beacon[83]

The main goal of this project is to define and implement a federated cloud network framework that enables the provision of federated cloud infrastructures, with special emphasis on intercloud networking and security issues, to support the automated deployment of applications and services across different clouds and datacenters.

[83] http://www.beacon-project.eu/

INPUT[84]

The INPUT Project aims to contribute to the evolution of the Internet "brain" beyond current limitations due to obsolete IP network paradigms, by moving cloud services much closer to end-users and smart-devices.

MCN (Mobile Cloud Networking)[85]

The aim of the Mobile Cloud Networking project is to extend the concept of Cloud Computing beyond data centres towards the Mobile End-User by redefining infrastructure and networks according the cloud-centric approach.

Arcadia[86]

This project is intended to provide a novel reconfigurable by design Highly Distributed Applications' development paradigm over programmable Infrastructure. To do so, ARCADIA Framework will rely on the development of an extensible Context Model which will be used by developers directly at the source-code level.

Switch[87]

SWITCH aims at improving the existing development and execution model of time critical applications by introducing a novel conceptual model (application-infrastructure co-programming and control model), in which application QoS/QoE, together with the programmability and controllability of the Cloud environments, can all be included in the complete lifecycle of applications.

Cloud Lightning[88]

Proposes a new way of provisioning heterogeneous cloud resources to deliver services, specified by the user, using a bespoke service description language. Due to the evolving complexity of modern heterogeneous Clouds, it proposes to build the system based on principles of self-management and self-organization.

Sonata[89]

The project addresses the significant challenges associated with both the development and deployment of the complex services envisioned for 5G networks and empowered by these technologies. It orchestrates complex services to connectivity, computing and storage resources, and automatically re-configures running services.

Sensoria[90]

The objective of this project was the development of a novel comprehensive approach to the engineering of software systems for service-oriented architectures where foundational theories, techniques and methods are fully integrated in a pragmatic software engineering approach.

[84] http://www.input-project.eu/
[85] http://www.mobile-cloud-networking.eu/
[86] http://www.arcadia-framework.eu/
[87] http://www.switchproject.eu/
[88] http://cloudlightning.eu/
[89] http://www.sonata-nfv.eu/
[90] http://www.sensoria-ist.eu/

Definition Languages

The most accepted and extended language for service orchestration is Web Service Business Process Execution Language (WS-BPEL or BPEL) 2.0, which is a public OASIS standard that leveraged former languages of big actors such as WSFL from IBM or XLang from Microsoft. It is considered a de facto standard for process orchestration execution.

Other important specification standard from OASIS Alliance is Topology and Orchestration Specification for Cloud Applications (TOSCA)[91], which enables the interoperable description of application and infrastructure cloud services, the relationships between parts of the service, and the operational behaviour of these services (e.g., deploy, patch, shutdown)--independent of the supplier creating the service, and any particular cloud provider or hosting technology.

BPEL engines

Following are listed some of the most important or representative engines and framework for business process management and orchestration as a part of it. Please, note that the following list (and all the others in this section) is not exhaustive and only shows a subset of the noteworthy solutions for BPM.

- Orchestra[92]
- RiftSaw[93]
- WSO2[94]
- Activiti[95]
- jBPM[96]

Other projects

Here, other orchestration related projects are listed. Some of them are modules or components that can be used to implement a part of the orchestration and some are complete solution. By no means this list shall be considered exhaustive, but only a sample of interesting or noteworthy projects for INTER-IoT.

- Open Stack Orchestration Program:
    - Heat[97]
    - Mistral[98]
- Hurtle[99]

[91] https://www.oasis-open.org/committees/tosca
[92] http://orchestra.ow2.org/
[93] http://riftsaw.jboss.org/
[94] http://wso2.com/products/business-process-server/
[95] http://www.activiti.org/
[96] http://www.jbpm.org/
[97] https://wiki.openstack.org/wiki/Heat
[98] https://github.com/openstack/mistral
[99] http://hurtle.it/

- Apache ODE[100]
- Cloudify[101]
- Juju[102] from Canonical

2.4.2.4.3 Service Choreography

Choreography is a service composition paradigm opposite to orchestration. Unlike the latter, choreography does not rely on a centralized service/element to manage all the service invocations and the results composition. Instead, choreography defines a set of global rules known by all the services that let them act in a defined way to compose services. This approach is completely decentralized and avoids the overhead of managing the service, as well as the performance burden that the orchestrator could introduce. However, in this approach, there is less control over the business process and a degree of tolerance to failures is needed.

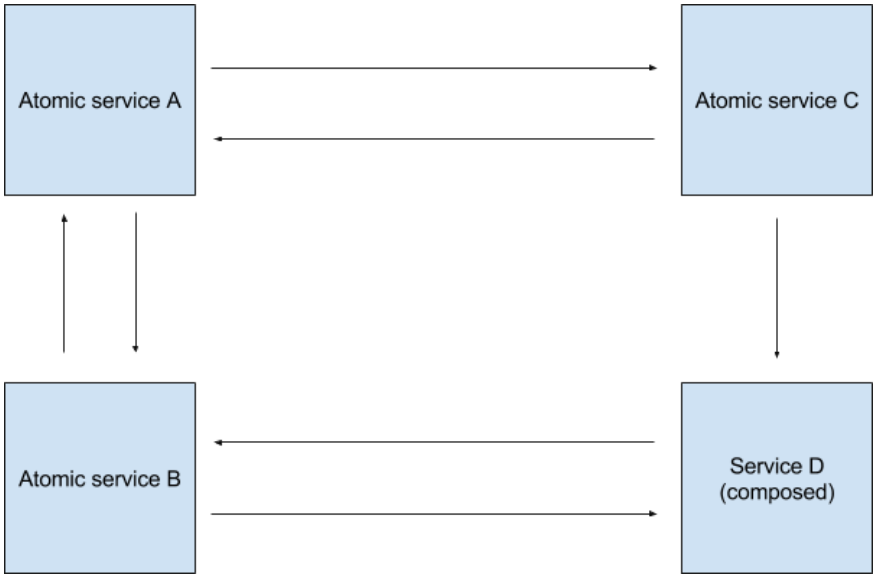

Figure 28: Choreography scheme.

Choreography languages

While orchestration implies the control of services from an element in an execution environment, choreography is not intended to be executed sequentially, but to describe or define an interaction framework between cross-domain or cross-organizational services. The most extended language for choreography relations description is WS-CDL (Web Service Choreography Description Language) from W3C. Other languages are WSCI (Web Service Choreography Interface), also from W3C and Ontology Web Language for Services (OWL-S). Choreographies can be modelled and annotated by using the widely extended standard Business Process Management and Notation (BPMN) 2.0 from Object Management Group (OMG) [38] [39].

Choreography in IoT platforms

[100] http://ode.apache.org/
[101] http://docs.getcloudify.org/3.5.0/intro/what-is-cloudify/#application-orchestration
[102] https://jujucharms.com/

Although most platforms offer service orchestration, currently also the possibility of having service choreography is raising. The service choreography is a conceptually attractive approach of dealing with the changing contexts of the augmented entities, for which centralized service orchestration has its limitations.

Service choreography is more appropriate as a means of coordination among different organizations. Furthermore, it offers advantages in terms of scalability and resiliency.

The IoT-A [103]project establishes the basis to include choreography services on new IoT platforms.

Thanks to the pervasive availability of connectivity of standalone devices, the growth of the device oriented operating systems and middlewares available, and the gradual adoption of communication standards and reference implementations, the number of application scenarios where choreography become appropriate and even preferred is increasing. Examples of high-level choreography can be found in very favorable scenarios such as smart cities (automatic traffic signals set up, advanced HVAC systems…) or intelligent transport infrastructures (e-toll, v2i communications [40] and others).

Research projects using choreography: Baile[104], Choreos[105], Chorevolution[106].

2.4.2.5 Tools

Node-RED

Node-RED[107] is a tool for wiring together hardware devices, APIs and online services to carry out tasks. It uses a visual programming approach that allows developers to connect predefined code blocks, known as ‘nodes’, together to perform a task. The connected nodes, usually a combination of input nodes, processing nodes and output nodes, when wired together, make up ‘flows’.

Node-RED uses a Flow Based Programming (FBP) model. FBP describes a graph of nodes, which exchange messages containing data via the edges. The edges are defined outside the nodes, in others words nodes have no control on where the data comes from and where it goes to.

Different categories of nodes are defined:

- Input: to process incoming events.
- Output: to serve or send outgoing events.
- Functions: to manipulate messages and message payloads.

[103] http://www.iot-a.eu/public
[104] http://ccsl.ime.usp.br/baile/
[105] http://www.choreos.eu/
[106] http://www.chorevolution.eu/
[107] https://nodered.org/

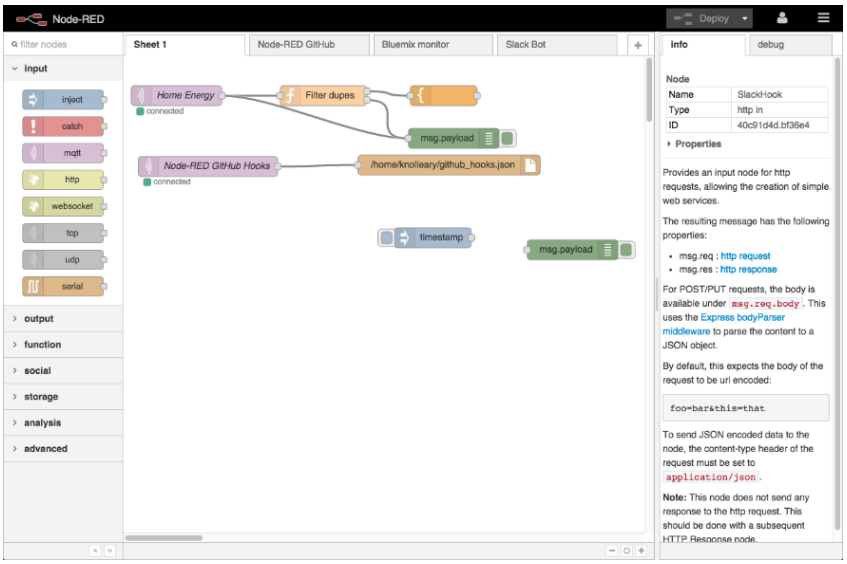

Figure 29: Node-RED dashboard. Source: http://nodered.org/

Using Node-RED provides several advantages for INTER-IoT:

- Node-RED is an application/service composition tool.
- There is already an active community regularly producing new nodes and the Node-RED platform is an open-source project hosted on GitHub.
- Node-RED flows are represented in JavaScript Object Notation or JSON and can be easily exported to the clipboard to be imported into Node-RED or shared online.
- IBM is also considering making it simpler to build on the work of others in Node-RED by introducing sub-flows. Sub-flows would allow users to collapse flows of multiple linked nodes into a single node, allowing more complex logic to be abstracted into a single node.
- Provides a lightweight proof of concept runtime.
- It's easy to use for simple tasks.
- It's simple to extend and to add some new capabilities and types of integration.
- It's capable of creating the back-end glue between IoT applications.
- Extensively used in the domain of IoT. A large number of nodes is developed to interoperate with the main IoT platforms.
- It has been used in IoT interoperability projects, such as COMPOSE. Furthermore, it is one of the most used tools in the EPI projects.
- There are many ways to run Node-RED e.g. under Docker.

Node-RED has the following disadvantages:

- Node-RED is not an Enterprise strength application runtime.
- It doest not include a graphical tool to visualize data (although there are already third party tools to do that).
- Other languages than JavaScript over Node might have better performance handling large amounts of data.

In the section 3.4.4 of this document will explain in more detail the advantages offered by Node-RED in the AS2AS layer.

Apache NiFi

NiFi[108] was built to automate the flow of data between systems. In Apache Nifi, the term dataflow is used to mean an automated and managed flow of information between systems. This problem space has been around ever since enterprises had more than one system, where some of the systems created data and some of the systems consumed it, and Apache Nifi addressed the problem with a flow based programming solution.

Apache NiFi advantages for INTER-IoT:

- Powerful and reliable system for processing and distributing data.
- Directed graphs of data routing and transformation.
- Web-based User Interface for creating, monitoring, & controlling data flows.
- Highly configurable - modify data flow at runtime, dynamically prioritize data.
- Data Provenance tracks data through the entire system.
- Easily extensible through development of custom components.

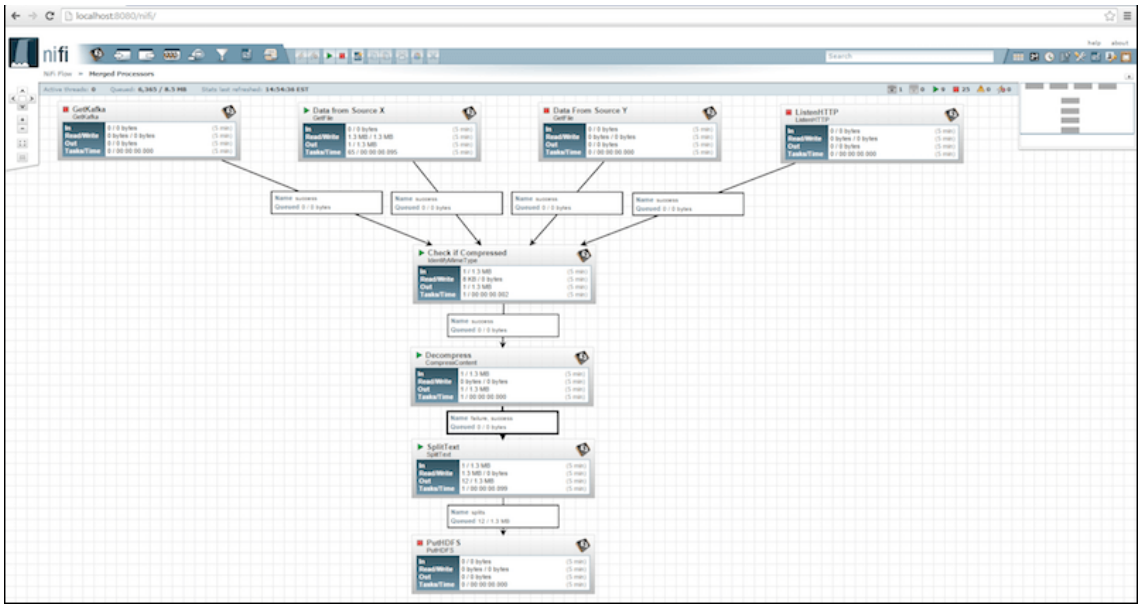

Figure 30: Apache NiFi. Source: https://blogs.apache.org/nifi/

But Nifi has the following disadvantages:

- It is a tool that can offer solutions for IoT, but it is not mainly focused on IoT.
- NiFi does not have a community working on interoperability solutions with the main IoT platforms as big as other tools (Node-RED).

[108] https://nifi.apache.org/

- Although NiFi is not a complicated tool, other tools like Node-RED, have the capacity to provide a simpler solution and in which the first results are obtained more quickly.
- The interoperability concepts with which this tool works, is not close to the one we want to apply in AS2AS. The reason is that NiFi does not really work with IoT concepts such as IoT platforms, IoT services, devices, sensors etc.

Even though, it is not a tool that meets all the desired characteristics for the interoperability pursued in AS2AS, its use in the future is not completely discarded as a complement to other tools (for example Node-RED). It's is possible take advantage of NiFi's powerful and reliable system to process and distribute data.

Flogo

According to the official documentation of its website, project Flogo is an Open Source Framework for IoT Edge Apps & Integration. It can be used to build IoT applications that run on edge devices and integrate them with IoT gateways and cloud services.

Flogo is a process engine with the following goals:

- Wire together hardware devices, APIs and online services.
- Integrate and orchestrate stateless devices and microservices.
- Transform/Filter/Route/Aggregate/Enrich State Management via State Service and Flow Service
- Activate and diagnose devices, manage their performance, etc.
- Recover from faults, continue where a device crashed.
- Provide synchronous and asynchronous communication.

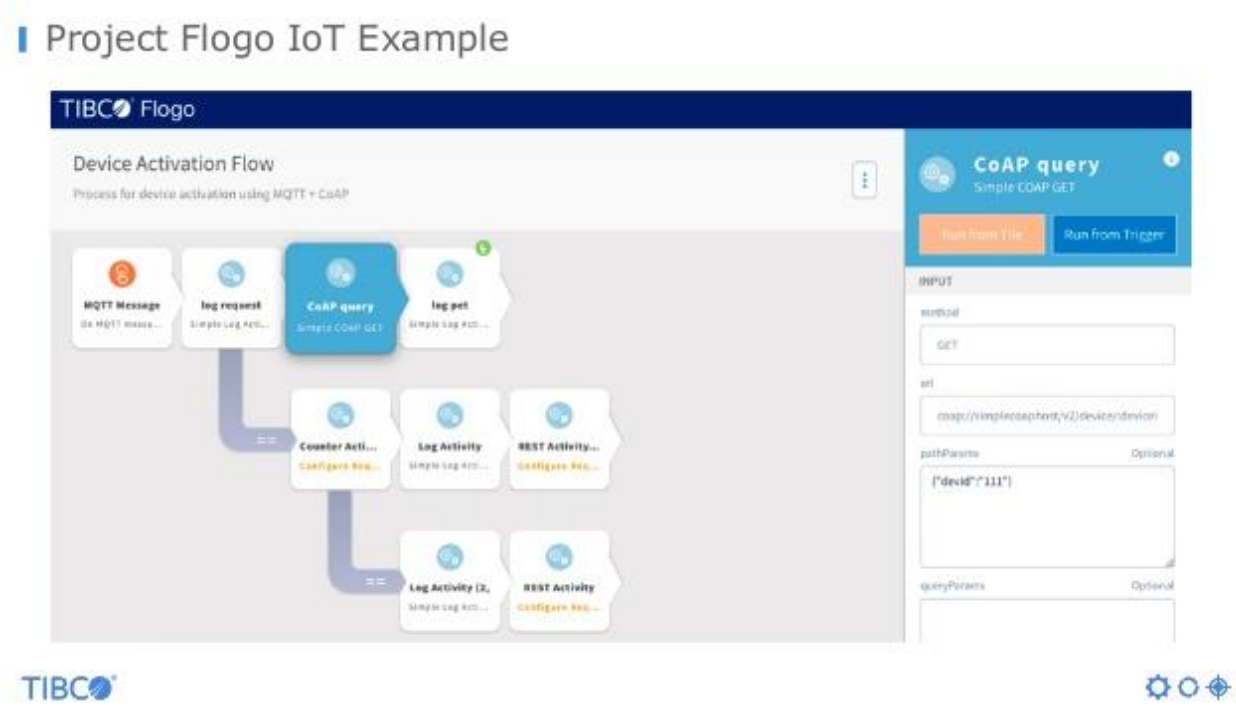

Figure 31: Project Flogo. Source: http://www.flogo.io/

Flogo has the following features:

- Ultra lightweight: in their documentation they explain that it has the advantage of being 20 to 50 times lighter than Java or Node-RED.
- Open Source.

However, Flogo has the following main disadvantages:

- It is currently in Developer Preview and primarily addresses the needs of Flogo Extensions developers and IoT Solutions Developers. For that reason, right now it is “for developers, by developers”.
- In comparison to Node-RED, it has not developed so many solutions to interact with the main IoT platforms and does not have a community which would be as big and active.
- There are not enough examples, documentation and implementation available for IoT interoperability solutions.

NoFlo

NoFlo, similarly to Node-RED, is a JavaScript implementation of Flow-Based Programming (FBP) model. The logic of this software is defined as a graph. The nodes of the graph are instances of NoFlo components, and the edges define the connections between them. NoFlo components react to incoming messages, or packets. There is no shared state, and the only way to communicate between components is by sending packets.

There are two ways to run your flow-based programs with NoFlo:

If the application is based on flows, then a user can simply have NoFlo execute and run it. Flow-based programs done in this way are called independent graphs.

The other option is to embed NoFlo graphs into an existing JavaScript application by using it as a regular Node.js library. This is useful when a user already has an existing system where you want to automate some parts as their own flows, or to add new functionality.

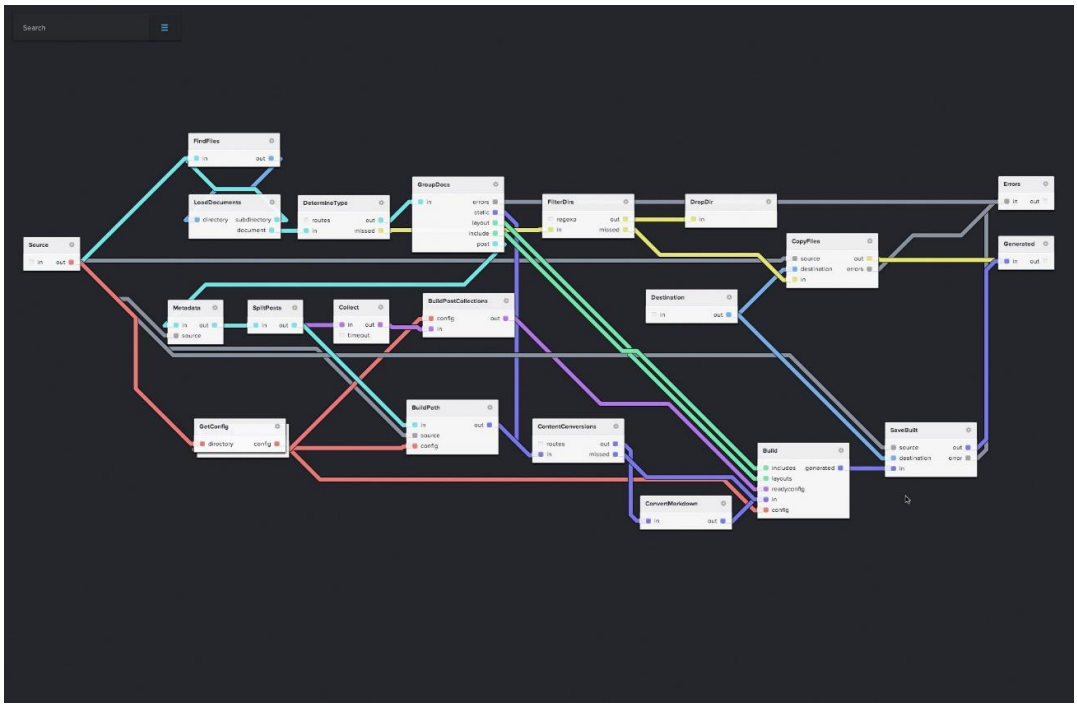

Figure 32: NoFlo. Source: http://noflojs.org/

There is an ecosystem of tools around NoFlo that make it more powerful:

- Browser-based visual programming IDE for NoFlo.
- Command-line interface for running NoFlo programs on Node.js.
- Tool for running NoFlo and other FBP runtimes as a distributed system.
- Data-driven tests for NoFlo and other FBP environments.
- Tools for debugging.

However, NoFlo has the following disadvantages:

- NoFlo is harder to use than Node-RED. The tool is divided into several components which makes difficult the first steps and the graphical environment is not so friendly.
- There are not enough examples, documentation and implementation available about IoT interoperability solutions.

Intel(r) IoT Services Orchestration Layer[109]

This is a solution that provides a visual graphical programming interface for developing IoT applications. It offers the following components:

- HTML5 IDE running inside browser. To create the IoT application, including its internal logic (e.g. workflows) and its HTML5 based end user interface, through drag-and-drop.
- Distributed middleware running on top of Node.js to host and execute the IoT applications created by the IDE.
- An Orchestration Center which runs the workflow engine to execute the logic, and web servers to host the HTML5 IDE for developers and HTML5 UI for end users.
- One or multiple Service Hubs which actually manage the devices and cloud services using various protocols. The Service Hub gathers this information about services to Orchestration Center to let the developers create applications based on these services. The Service Hub also receives commands from Orchestration Center to actually invoke the services managed by it, according to the logic defined by workflows.

[109] https://github.com/01org/intel-iot-services-orchestration-layer-dev

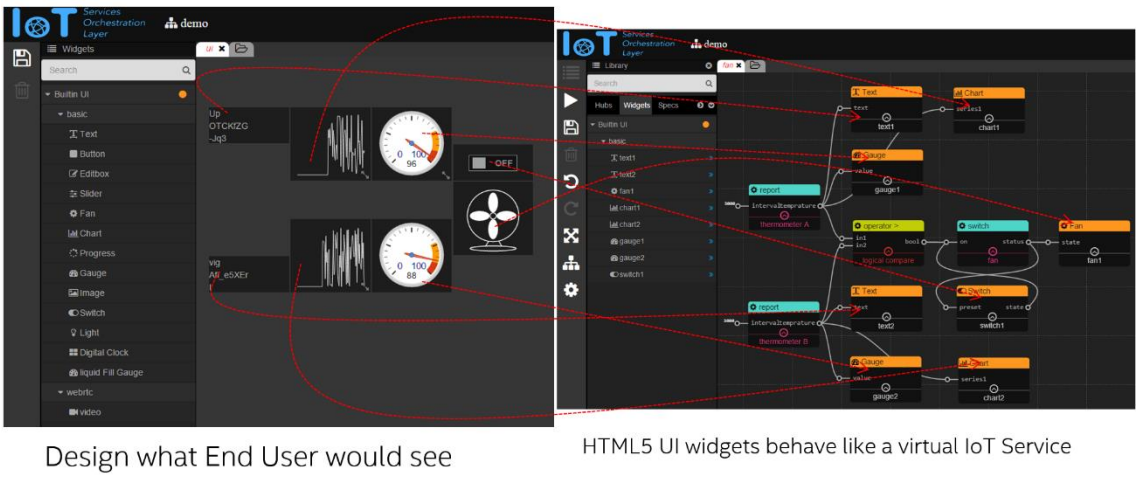

Figure 33: Intel(r) IoT Services Orchestration Layer Source: http://01org.github.io/intel-iot-services-orchestration-layer/

However, Intel(r) IoT Services Orchestration Layer has the following disadvantages:

- The features it offers now are far from the purpose of the AS2AS layer. There are no examples working with the APIs provided by the main IoT platforms.
- It does not have a community as big as the other tools of this sections.
- Most of the examples work with devices and not with applications and services.

2.4.3 Summary table

Application & Services Interoperability		
Approach	**Brief summary on how they facilitate interoperability**	**What can be found in this chapter**
Service Virtualization	IoT services loads can greatly vary in time. Service virtualization provides flexible mechanisms of scaling creating additional instances of a service whenever needed, in order to handle the additional load while maintaining the quality of the service.	The benefits of the virtualization, service deployment methods and a brief summary of the widely used technology within virtualization: the usage of Containers.
Service Catalog and Service Discovery	A catalogue will be able to register the applications to make them discoverable. Furthermore, it will offer a description or detailed information about the services/applications.	A brief explanation of the following protocols / standards, UDDI/WSDL/Hypercat. iServe as an example of a platform working with Service Catalogue.
Wrapping Technologies	Specific programs able to extract data from Internet sites or services and convert the information into a structured format.	An example of how to wrap web services through visual components and provide a media platform to build and distribute custom applications.

Service Composition	Encompasses all those processes that create added-value services,(composite or aggregated), from existing services.	The relationship between the techniques of service composition (mash-up, orchestration, choreography) and the IoT Platforms and their services.
Tools	The tools presented in this section provide solutions to facilitate the interoperability of services that come from different IoT platforms.	The explanation of the following tools: NodeRed,Apache NiFi, NoFlo, Flogo and Intel(r) IoT Services Orchestration Layer.

Table 5: Application and Services interoperability summary table.

2.5 Data & Semantics Interoperability (DS2DS)

2.5.1 Introduction

In every Internet of Things platform or system it is essential to take into account the production, collection, transmission, and processing of vast amounts of data. Any application consuming those data needs to understand its structure and meaning. Both aspects can be represented by suitable metadata, which in order to be useful should be machine readable. The metadata provides a semantic description of the data and can be utilized for many purposes, such as resource discovery, management, and access control. Of course, the more expressive is the language used for representing the metadata, the more accurate the description might become, although decidability and computability put some obvious barriers. As it turns out, the concept of metadata can be seen as a special case of the notion of ontology, which by definition represents explicit specification of shared conceptualization [41].

The term ontology originates from the branch of philosophy that studies the nature and structure of beings. In recent years, mostly due to the fast growth of various types of data available through the Internet, ontologies became important in information science and technology. In this area, the concept ontology refers a structure that provides a vocabulary for a domain of interest, together with the meaning of entities present in that vocabulary. Typically, within an ontology the entities may be grouped, put into a hierarchy, related with each other, and subdivided according to different notions of similarity. In the last two decades, the development of the Semantic Web resulted in the creation of many ontology-related languages, standards, and tools. Among the most important standards, the following should be highlighted: Resource Description Framework (RDF) and RDF Schema (RDFS) [42] [43], OWL Web Ontology Language [44], and the RDF query language SPARQL [45], which enables retrieval and manipulation of data stored in the RDF format.

The importance of semantics, and ontologies in particular, becomes even more evident and relevant in the context of interoperability. In the IoT context, there is a large number of platforms available on the market. Unfortunately, usually they are not able to cooperate or even communicate with each other, even if they belong to the same domain. Ontologies give the possibility to share a common understanding of the domain, to make its assumptions explicit, and to analyse and reuse the domain knowledge. In other words, they provide the basis for achieving semantic interoperability, i.e., the ability of computer systems to exchange data with unambiguous, shared meaning.

Of course, in order to achieve shared meaning of data, the platforms or systems have to use a common ontology either explicitly, or implicitly (via a semantic mediator). In either case, we have to be able to combine or merge different ontologies, and structurally manage such combinations.

In what follows, this state of the art of semantics is focused on ontologies for the IoT, semantics of services, and the discussion of semantic aspects of selected IoT platforms. The last section will be devoted to the tools supporting semantics and ontology manipulation.

2.5.2 Literature review

2.5.2.1 IoT semantics

Regarding more general issues concerning interoperability in IoT systems, this problem has been, and still is, addressed by researchers on many levels, including mainly device [46], middleware [47] [48], and service [49], whereas the semantic layer has received considerably less attention. The integration of IoT data into the Web with semantic modeling and linked data approach was discussed in [50]. The early stage of adoption of semantic methods in the IoT becomes evident after searching for available ontologies. It must be noted that the practical use of semantic methods and tools requires the existence of explicitly expressed ontologies, represented by using one of the ontology languages (currently RDF(S) or OWL). Therefore, let us discuss what is actually available for practitioners that would like to use semantic technologies in IoT environments.

The general observation is as follows. Most existing ontologies, capturing the IoT domain, were developed within individual research projects and, as a consequence, they typically are in a prototype stage, often incomplete and sometimes abandoned upon project completion. A notable exception is the W3C SSN ontology, which was developed as a joint effort of several research organizations and became the standard ontology for the semantic sensor networks.

For all practical purposes, this is the only ontology explicitly mentioned in [51]; if it is not counted the OpenIoT ontology, which is a recent effort, based on the W3C SSN (see, below). However, while this ontology captures the domain of WSN, it would require further elaboration of the details of the problem at hand to be used in IoT applications. This is to be done in more specific sensor network ontologies that attempt at capturing further information about sensor capabilities, performance, usage conditions, and should enable contextual data discovery.

Among ontologies that have been developed in recent years, the following ones are worth to mentioning in the context of the INTER-IoT project. Additionally, [52] and [53] should be consulted for further references. Let us start from a short description of ontologies that, as far as we were able to establish, are no longer under active development. Observe that some of them are more generic, while others are focused on more domain-specific aspects of sensors and sensor networks.

CSIRO Sensor Ontology [54]. It was an early attempt of development of a generic ontology for describing functional, physical and measurement aspects of sensors. It was created at the Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia. Its main classes include sensors, features, operations, results, processes, inputs and outputs, accuracy, resolution, abstract and physical properties, and metadata links.

SWAMO Ontology [55]. The aim of the SWAMO project [56] was to use a collaborative and distributed set of intelligent agents for supervising and conducting autonomous mission operations. SWAMO ontology enables automated decision making and responses to the sensor Web environment. One of its advantages was its compatibility with the Open Geospatial Consortium (OGC) standards, enabling geo-data consumption and exchange.

MMI Device Ontology [57]. An extensible ontology of marine devices (hence, an ontology that is slightly more domain-specific than others) that integrates with models of sensor descriptions. Its main classes include component, system, process, platform, device, sensor, and sampler.

SEEK Extensible Observation Ontology (OBOE; [58]) is a suite of ontologies for modeling and representing scientific observations. It can express a wide range of measurement types, includes a mechanism for specifying measurement context, and has the ability to specify the type of entity being measured. In this way it is focused more on the results produced by sensors than sensors themselves.

Machine-to-Machine Measurement (M3) ontology is a semantic model developed for the Semantic Web of Things (SWoT) project[110]. It is a comprehensive taxonomy that spans wide range of devices and concepts from IoT domain. It is the semantic basis of the SWoT framework that, was designed to enable interoperability between both domain-specific and cross-domain applications. A smaller version of this ontology – the M3 Lite[111] is used and developed for the FIESTA-IOT project.

All these ontologies, as well as the **SemSOS observation-centric** ontology suite [59], the **stimuli-centered** ontology design pattern [60], as well as the **OGC SensorML** standard [61] contributed to the development of the, mentioned above, **W3C Semantic Sensor Network ontology** (SSN) [62]. The **W3C SSN** [63] [64] ontology is actually a suite of general purpose ontologies for describing sensors, their accuracy and capabilities, observations and methods used for sensing.

Further information, concerning deployment and use of sensors is also captured.

More specifically, the SSN consists of 10 conceptual modules (Deployment, System, OperatingRestriction, PlatformSite, Device, Process, Data, SSOPlatform, MeasuringCapability, ConstraintBlock) which contains 41 concepts and 39 object properties. It directly inherits 11 concepts and 14 object properties from the top-level DOLCE-UltraLite ontology [65].

The W3C SSN ontology has been widely used, and both extended and specialized. Among the notable extensions are the wireless sensor networks ontology; **WSSN** [66], and sensor cloud ontology **SCO** [67]. The specializations include the **AEMET** meteorological ontology [68], atmosphere observation ontology **SWROAO** [69], flood prediction ontology **SemSorGrid4Env** [70], and (data) stream annotation ontology **SAO** [71] [72]. A more recent **IoT-Lite** [73] ontology is a lightweight instantiation of the SSN that provides a general IoT knowledge model intended to limit processing time of ontologically demarcated resources.

Another noteworthy IoT ontology is the **Smart Appliances Reference Ontology** [74] (**SAREF**). It describes a top-level perspective on IoT home appliances along with their functions and services. The model is generic enough to be used outside of the home environment, and includes concepts such as device, sensor, actuator, service, state and function. SAREF is most naturally applied at home, office or any limited public space.

The **oneM2M** Base Ontology [46] is a core IoT ontology with very general and basic concepts. Its latest version (2.0) can be found at [75]. As the ontology prepared for the second release of the IoT Platform oneM2M it is designed to satisfy special semantic requirements for IoT. Its purpose is to provide an ontology that other systems can align with and match with their own ontologies in order to achieve semantic interoperability.

[110] http://sensormeasurement.appspot.com/
[111] http://lov.okfn.org/dataset/lov/vocabs/m3lite

It can be claimed that (similarly to the way that the OpenIoT project proceeded) any project planning to fuse Internet of Things and semantic technologies should definitely start by taking full advantage of the W3C SSN ontology. Only then, it should extend it by adding concepts necessary to deal with intended application areas. These concepts may be needed either on the sensor level or to represent concepts formalizing knowledge concerning application areas of interest themselves. Note that the need for adding concepts concerning sensors and sensing is not very likely, as the W3C SSN is quite comprehensive. Nevertheless, it may turn out that it does not capture some unique concepts related to use of sensors in a selected application area.

The **W3C SSN** ontology has seen the strongest uptake and has influenced several projects, most notably the OpenIoT. The methodology used in engineering of the OpenIoT ontology is a promising approach for development of IoT interoperable solutions. This method starts with the W3C SSN ontology and extends it by a chain (or network) of ontologies, in order to capture domain specific concepts, either by linking and modifying existing domain ontologies or producing new low-level ontologies with the SSN at their core.

In principle, ontologies for IoT might also consider some special aspects and features of the IoT environment. Semantic descriptions of IoT devices or smart objects should include identity, type, physical characteristics, location, embedded devices and provided services. Ontologies might include support for smart objects that are not sensors, but act as smart devices, such as virtual devices, human interfaces or algorithms. Moreover, since many sensors and smart devices have low resources (e.g. low battery, connectivity, etc), it should be possible to semantically annotate also these kind of special characteristics and resource limitations. The location of smart objects may be a critical information in order to analyse data from them, especially in the case of mobile sensors and devices. Therefore, it should be as well taken into account.

Note that, even though standards for description of various subdomains of IoT exist, there is no single selected conceptual model of the IoT domain. Interoperability is obviously achieved when platforms use the same ontology or standard. Otherwise the mapping needs to be prepared, or platforms need to conform to selected standards. The challenge is to propose interoperability mechanisms that do not require changes in the original platforms' semantics. These can be based on ontology alignments with the central ontology, and semantic translations. Naturally, this problem can be extended to domain-specific ontologies (e.g. transportation, m-Health) that are commonly referenced in the messages exchanged in IoT ecosystem. And different semantics can be used to model a given domain depending on platform perspective.

2.5.2.2 Service description semantics

Following a discussion in 2.4.2.2, in this subsection a variety of service description standards are briefly discussed. Those standards have been proposed and implemented starting from standards operating on the syntactic level, which can be later annotated with semantics, or fully semantic service description. Note that discussed standards are mentioned in the context of Service Catalogue and Discovery tools, that can use them directly, or they can serve as an inspiration for proposing a set of semantic annotations for service description. Semantics can be used by e.g. the service consumer to describe the service requirements, so that matchmaking techniques can be later used to find the semantic similarity between the service description and the requirements. Semantics can provide mechanisms for achieving more advanced interoperability in which two syntactically different services (e.g. different input and output parameters) can be recognised, and they are providing the same functionality.

Web Service Description Language (WSDL)[112] is an XML-based interface definition language that specifies, in a machine-readable format, functionality offered by described Web service i.e. how it can be called, what are the input parameters and what is returned by the service. A client program connecting to a Web service can read the WSDL file to determine what operations are available on the network endpoint and how to interact with the service. WSDL 2.0 describes a service from two perspectives: abstract (functionality) and concrete (how and where that functionality is offered). WSDL operates at the syntactic level providing functional description, however it cannot unambiguously determine what the service does (the syntax is specified but not the meaning or impact on the environment). WSDL is one of the basic standards in Web service stack, and serves as a basis to many extensions.

Web Service Semantics (WSDL-S)[113] defines a mechanism to associate semantic annotations with Web services that are described using WSDL. It is assumed that external semantic model relevant to services is available and can be referenced from WSDL via extensibility elements. This approach is agnostic to ontology language, as it is not assumed that semantics is expressed in OWL. WSDL-S allows annotating inputs, outputs and operations, and specifying preconditions and effects of the service.

Semantic Annotations for WSDL and XML Schema definition language (SAWSDL)[114] is a set of extensional attributes that allow description of additional semantics. SAWSDL uses WSDL-S as its primary input and became W3C candidate recommendation. SAWSDL allows referencing external semantic models e.g. ontologies from WSDL and XML Schema components with annotations, without imposing restrictions on semantic language. The service provider can explicitly add semantics by annotating the appropriate parts of the Web service description with concepts from a richer semantic model. It allows annotating with categorization information useful for publishing service in a registry specially for service discovery and composition. Additionally, SAWSDL allows mapping types in XML Schema to and from ontology UEDI.

United Service Description Language (USDL/Linked-USDL)[115] is proposed as a data model for describing various types of services. Besides technical aspects, it puts much focus on business aspects of a service e.g. provisioning, pricing, composition. The Linked USDL initiative is a remodelled USDL with respect to linked data and Web of data principles. New specification is modelled as an RDF(S) (with reuse of existing ontologies) and can better support automated processing and online service trading. The work on Linked USDL has been partially funded by the following projects: FAST, RESERVOIR, MASTER, SERVFACE, SHAPE, SLA@SOI, SOA4All, FI-WARE, and COMPOSE.

The ESSI WSMO working group is responsible for developing Web Service Modelling Language (WSML)[116] that formalizes the Web Service Modelling Ontology (WSMO)[117]. 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 behavioural aspects), goals (user

[112] https://www.w3.org/TR/wsdl20
[113] https://www.w3.org/Submission/WSDL-S
[114] https://www.w3.org/TR/sawsdl/
[115] https://linked-usdl.org/
[116] https://www.w3.org/Submission/WSML/
[117] https://www.w3.org/Submission/WSMO/

desires) and mediators (interoperability between different WSMO elements). WSMO can be referenced from e.g. WSDL-S and SAWSDL semantic annotations. Lightweight Semantic Descriptions for Services on the Web (WSMO-Lite)[118] is a lightweight set of semantic service descriptions in RDFS that can be used for annotations of WSDL elements using the SAWSDL annotation mechanism (bottom-up modelling of semantic Web services). WSMO-Lite addresses the following issues: service ontology, annotation mechanism for WSDL using service ontology, provides bridge between WSDL, SAWSDL and domain ontologies. Semantic Markup for Web Services (OWL-S)[119] is a first major 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 prole (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). Concepts from OWL-S can be referenced with e.g. WSDL-S and SAWSDL.

hRESTS[120] is a microformat for describing RESTful Web services. It identifies service definition, operations, inputs, outputs inside HTML page which describes RESTful service. MicroWSMO is an extension that adds semantic annotations referencing WSMO-Lite ontology. hRESTS forms is equivalent to WSDL for RESTful services, and MicroWSMO is analogous to SAWSDL. The WSMO-Lite service semantics ontology is directly applicable in MicroWSMO and hRESTS annotations.

2.5.2.3 Semantic aspects of relevant IoT platforms

FIWARE

FIWARE is a middleware platform and open community which resulted from the EU driven FP7 project Future Internet Core Platform. Various functionalities offered by the platform, called Generic Enablers (GEs), are grouped in the form of chapters following FIWARE Reference Architecture model.

Semantics and metadata are used mainly within the Data/Context Management and Internet of Things Services Enablement chapters. In particular, the IoT Discovery GE[121] provides the Sense2Web linked-data platform semantic repository for registering and managing descriptions in RDF/OWL, as well as querying the data via SPARQL. Although within FIWARE platform itself no specific ontologies are explicitly mentioned or used, the FIWARE community is one of the main contributors of the IoT-Lite ontology, which is a lightweight instantiation of the W3C SSN.

OneM2M

The popular IoT platform oneM2M [46] has an ontology, but it has not been finalized (as of the time of writing of this document). Latest version (0.9) can be found at [75]. The purpose of this oneM2M Base Ontology, as explicitly stated by authors, is to provide an ontology that other systems can align with (i.e. match it with their own ontologies) and, in this way, achieve interoperability. Interestingly, one of the side-goals of the Fiesta-IoT project [76] is to fix the interoperability problem between the

[118] https://www.w3.org/Submission/WSMO-Lite/
[119] https://www.w3.org/Submission/OWL-S/
[120] http://dl.acm.org/citation.cfm?id=1486962
[121] http://catalogue.fiware.org/enablers/iot-discovery

oneM2M ontology and the FIWARE platform. However, the developed solution [77] has not been made public.

OpenIoT

When considering semantic technologies applied to the IoT in general, it is crucial to mention as well the results of the recently completed EU-funded OpenIoT project. The OpenIoT open source platform [78] utilizes both cloud computing and semantic methods and focuses on interoperable IoT deployments.

At the sensor level, the OpenIoT utilizes the XGSN [79], an extension of the GSN middleware [80], which enables semantic annotation of virtual sensors.

The OpenIoT ontology uses the W3C SSN ontology as the starting point. It has been combined with several well-known vocabularies and ontologies at the time when it was being developed (e.g. PROV-O provenance ontology, LinkedGeoData [81] and WGS84 geo-ontologies [82], LSM linked sensor middleware ontology [83], etc.). It was also augmented with cloud-related concepts.

By combining cloud-computing and sensing capabilities, the OpenIoT platform supported on-demand cloud-based access to the IoT resources, which was needed in the context of the OpenIoT project.

UniversAAL

UniversAAL[122] is a semantic and distributed software platform designed to ease the development of integrated Ambient Assisted Living applications. The project took the PERSONA project (FP6) which was also semantic focused, as its base. "*The semantic nature of universAAL makes it ideal for highly heterogeneous environments, it's power making it suitable for IoT (Internet of Things), wearables, Big Data, and many more domains*" -it can be read on the main page of the project, currently maintained by some of the partners that were part of it.

In universAAL, each and every component is modeled semantically, in addition to the real world or the services, which are also represented as ontologies.

Most of the ontologies have been completely built from scratch, as they were discussed and agreed among the partners, but not based on any existing standard. The exceptions to this are: (I) Personal information ontology - based on vCard, (II) Devices ontologies - inspired by several standards (ISO, IEEE) but not following strictly any of them, and (III) ISO 11073 -a standard for medical devices and data that was mapped into a set of ontologies.

The Java classes depicting the universAAL ontologies can currently be found on GitHub, while the ontologies are available from the project repository.

In the first case, ontologies are represented as POJOs, the common model representation for Java applications. In the latter, they are described in OWL and Turtle (Terse RDF Triple Language).

SOFIA2

SOFIA2[123] is a middleware that allows the interoperability of multiple systems and devices with key concepts including: Smart Space -a collaborative virtual environment in which devices and application interoperate to deliver a complex functionality, KB (Knowledge Processor)-, a Smart

[122] http://www.universaal.info/

[123] http://sofia2.com/home_en.html

Space client, producing and consuming information, and SIB (Semantic Information Broker) -the Smart Space core, integrating the exchanged semantic information and storing data acting as the interoperability bus. In SOFIA2 a semantic information is defined as the set of classes and attributes that will be shared by various applications that interoperate within the Smart Space. Semantic information reflecting all existing concepts in the domain are defined in JSON according to a JSON Schema. These JSON schemas are added to the platform where they can be searched and subscribed to. Even though JSON Schema is not exactly an ontology language, it allows to validate whether the semantic information sent by the KP satisfies the semantics.

2.5.2.4 Semantic and ontological tools

Let us now look into tools supporting semantic (ontologies) manipulation that can be useful for achieving semantic interoperability in IoT. Before proceeding, let us stress that we are interested only in operations performed on ontologies. Terminology for operation on ontologies was introduced in 1.2.

We assume that either (which is unlikely) IoT platforms that are to interoperate use ontologies represented in RDF/OWL, or extraction of semantics (e.g. from XML, JSON, etc.) has been performed, and RDF/OWL ontologies created as a result. Now, ontology aligning has to take place.

The following classification of tools is not done on the basis of the underlying algorithms or methods - this type of classification has been already done (i.e. [84]). Instead, we propose very pragmatic criteria that are essential when selecting methods for application in real-life use cases:

- availability of the website and the date of the last update, presence on the web site and recent date of the last update show vitality of the tool. As a matter of fact, tools that have not been updated for more than two years are very suspicious from the point of view of being lock-down to a dead-end software,
- number of related publications and date of last publication: a larger number of publications indicates that the method is better established (as it has been reviewed more often), while date of last publication indicates the vitality,
- availability of the source code and documentation crucial for actual use,
- used technology, and I/O data format indicate what levels of expressiveness can be handled by the method, and what input/output data can be processed; here we also consider the interfaces (GUI and/or command line) are also considered,
- known academic and commercial utilization. It is very valuable a method that has been applied outside of a purely academic environment,
- scalability -usability in the IoT requires tools that are scalable and efficient.

Overall, we are interested in tools that are mature (went through a number of development cycles and resulted in multiple publications), actively maintained and systematically developed, which preferably have been applied in real-life scenarios, and bring some promise of scalability.

We have investigated all methods and tools mentioned in [84] [85] [86], as well as tools found as a result of Internet search, a total of 97, taking into account the criteria listed above. As a result, we have reached the following conclusions:

- Numerous tools implementing ontology matching appear in the literature, but most of them are non-functional. We have identified only nine that are still alive and more or less correspond to our needs.
- For 60% of the tools, we have not found an active website. In many cases, if the website was available, it was very basic and not recently updated.
- We have observed a tendency to present mostly OAEI contest results. While the OAEI initiative is very useful when comparing and evaluating methods, lack of follow-up or other publications suggests that the method or tool was developed primarily to participate in the contest.
- Besides few cases, we have not found information about the use of the tool in projects or commercial applications. Almost all documented use cases came from the OAEI contests.
- Scalability can be deduced only from the results of the OAEI contests. We have not found other results explicitly benchmarking scalability of the methods.
- For 85% of the tools, we could not find either source code or executables. For the remaining 15%, significant part had no technical documentation, or user manual. Instead, only tools/matching methods were described in publications.
- In 85% of cases, explicitly stated description of what are the input/output ontologies formats and languages was missing.
- Almost no tool seriously considered the situation in which the semantic input is not explicitly represented in one of the core ontology languages (RFD/OWL). However, lack of explicit formal RDF/OWL ontology is a typical situation for the ICT systems of today (e.g. use cases of the INTER-IoT project).

Let us now look into more details of tools that met our criteria. It must be noted that, in addition to the seven listed below, there are two more active tools that could have been listed in this section: YAM++ [87] and LODE. However, YAM++ was omitted because there is no source code available (only executables, that cannot be modified, if needed); while LODE is accessible only as a Web application (it lacks of a command line interface, or an API).

LogMap

LogMap [88] [89] is an open-source tool, developed at the University of Oxford. It can match very large ontologies, such as FMA and SNOMED. Since 2011, LogMap takes part in the OAEI contests, constantly achieving very good results. In 2015, it was the only tool taking part in all OAEI tracks.

LogMap was written in Java, and can be used both from the command-line and via a Web-based Ajax interface. The command-line version is available as a stand-alone distribution, as well as in the form of the OAEI packages. As input, the tool accepts any of the OWL API formats, and produces alignments between classes, properties, and instances. As one of very few ontology matching tools, LogMap has capabilities for repairing inconsistencies on-the-fly. For consistency checking, it utilizes a method based on propositional Horn-clause satisfiability (Dowling-Gallier algorithm [90]). The source code (last updated in May 2016) is freely available from the GitHub. Pre-build packages can be downloaded from the SourceForge.

A relative weakness of LogMap lies in the way of computation of the candidate mappings and matches. The algorithm finds similarities between concepts, utilizing vocabularies of the input

ontologies. As a consequence, the result may not be satisfactory if the ontologies are lexically despaired, or do not provide enough lexical information.

The Website of LogMap lists eleven publications devoted to various aspects of the tool, with the most recent from 2016.

COMA

COMA 3.0 [91] (previously called GOMA or COMA++) is a framework that supports several matching algorithms and is highly customizable. It is an open source project, with the last update of the code in January 2013, that evolved from the work done at the University of Leipzig. The tool performs matching and merging on XSD (XML Schema), OWL (OWL-Lite), XDR (XML Data Reduced) and relational database schemas. Internally, any supported data format is transformed into a generic model of a directed acyclic graph, which enables processing of schemas and ontologies distributed among multiple namespaces and files. COMA has full GUI support for all its operations.

COMA implements an iterative algorithm based on a collection of matching algorithms (matchers). Selection of matchers, as well as decisions, which matching axioms are correct, is made by the user. Specifically, the user assigns a confidence value to each matching axiom, and can manually create and delete them. Any number of iterations (computing and refining matching axioms) can be performed, each building on the result of previous one. The end result can be saved to a file in a COMA specific format. COMA can use the resulting matching to create, among others, a merged ontology, the intersection of ontologies, etc. Merging of ontologies and schemas is limited to the, paid, Business Edition of COMA.

Because of its architecture, COMA is a good candidate for a framework for implementation and testing of new matchers. Lack of support for RDF or more expressive proles of OWL, are a limiting factor.

AgreementMaker

AgreementMakerLight [92] (AML, a continuation of AgreementMaker and part of the SOMER project) is an automated matching system that acts as an extensible framework that implements many matchers. It is open source and actively updated. Initially, the AgreementMaker was specialized to work with biomedical ontologies but, currently, it can be applied to any ontology in OWL, OBO or SKOS format. It has performed very well in 2014 [93] and 2015 [94] editions of the OAEI competition. It is claimed that the AML can efficiently (i.e. within several minutes) compute alignments on very large ontologies (e.g. WordNet), although it requires large amounts of RAM (e.g. 8GB for ontologies with less than 100 000 classes).

Currently, AML implements 6 matchers that range from simple (label similarity) to complex (so-called, structural matcher), as well as filters (e.g. cardinality filter). Each matcher is configurable, e.g. the string matcher has a choice of four similarity measures. Background knowledge matcher can calculate similarity scores by using an external knowledge source, like WordNet.

However, it supports matchings between classes and properties, but not individuals. Alignment can be reviewed and each axiom is explained on a graph.

Alignment axioms may also be added or removed manually. The results are in the Alignment API [95] format.

The AML can work both as a GUI and as a command line application. The possibility to extend the framework with new matchers is very valuable. Unfortunately, the AML does not natively perform ontology merging.

Alignment API

Alignment API[124] [95] is a definition of format and schema for storing alignments in RDF, as well as a set of tools that operate on them. It is designed to be tool-agnostic and to enable storing, exchanging, and sharing alignments. The API itself, outside of simple reference implementations, does not define any matchers, nor does it provide matching or merging services for ontologies or schemas. Instead, it defines a set of standard operations and interfaces for working with alignments. The Alignment API specification and tools are actively updated and open source.

An Alignment Server (part of the Alignment API) can store, compare and manage alignments. It can be accessed via pluggable interfaces that currently include: HTTP, SOAP and REST Web services and FIPA ACL22. The server allows information about the alignment computation process (e.g. program/matcher name, processing time) to be stored in the alignment file. The format is extensible, so any kind of additional information can be added and the schema itself can be extended.

The API defines interfaces for matching algorithms, query translation, finding existing alignments, manipulating alignments, rendering them in a different language, etc. Alignment Server provides a reference implementation of those operations, but for specific problems, own implementations are encouraged.

The official webpage lists around a hundred tools that are compatible with the Alignment API. Some of them are listed in this article.

Silk Framework

Silk Framework [96] is an open source tool for discovery of links between datasets in the context of the Open Linked Data. It generates links between sources, based on user-provided link specifications. The supported formats include RDF, CSV and XML, with strong focus on RDF. Querying of data is done through a user-specified SPARQL endpoint. Link specifications can be written manually in Silk-LSL (Link Specification Language), or constructed in the Silk Workbench -a Java Web application. They can be exported and incorporated into original data sets. Results produced by the Silk can be stored in an Alignment API compatible format.

As opposed to the schema matching systems, Silk discovers and verifies links between data values and nodes. Support for any SPARQL endpoint means that large amounts of data, spread among SPARQL datasets, can be queried, with full interlinking between different graphs and namespaces. Furthermore, Silk allows defining complex data transformations that go well beyond simple links. This can be useful in translation of data between semantics.

The manual input of link specifications means that links cannot be discovered entirely automatically as it is necessary to specify what kind of linkage pattern has to look for. In this way, Silk is more of a tool to interlink data, rather than to discover alignments. Note that Silk does not perform automatic schema matching or ontology merging.

S-Match

S-Match [97] is an open source semantic matching framework that transforms tree-like structures such as catalogues, conceptual models, etc., into lightweight ontologies to then determine the semantic correspondences between them. The project has an up-to-date website with information, including documentation, and tutorials. There are over 20 papers (last from 2011) devoted to various

[124] http://alignapi.gforge.inria.fr/

aspects of the project (e.g. algorithms implementation). S-Match is a Java application that can be run from GUI or from command line. The inputs to the method are text files, in which tree like structures are defined. The use of a native input format is one disadvantage of the tool. As a consequence, input ontologies have to be transformed before running the tool.

The source code is available from GitHub (last update in January, 2015). Ready to use pre-build packages (most recent from 2013), it can be downloaded from SourceForge.

S-Match was utilized in 10 documented projects that are referenced on the website.

OntoBuilder

OntoBuilder project provides an open source set of tools to extract ontologies from Web pages and map ontologies from similar domains, generating an ever-improved single ontology, with which a domain can be queried.

OntoBuilder services for schema matching provide several algorithms e.g. similarity flooding, combined algorithm, precedence algorithm, term and value combined algorithm, graph algorithm, value algorithm, term algorithm. The Top K Framework graphical tool allows to view and save best mappings (based on a user-defined threshold). OntoBuilder is written in Java and it can be used as a graphical tool, as a jar package, or as a command line tool. The source code and documentation are freely available from the Bitbucket repository.

Even though the last publication is from 2010, and the last update to the website with downloadable OntoBuilder was done in June 2011, the tool is well documented with 15 publications linked from the website. We suspect that the project is not actively developed (making it the weakest of the seven). However, the deliverables produced in the past can provide useful input for our work.

2.5.3 Semantics in INTER-IoT layers

Interoperability on D2DS layer can be considered separately from other INTER-IoT layers but the proposed solution can be as well incorporated into other layers architecture. In the former case the aim is to provide semantic translation functionalities between artifacts communicating with messages. In the latter case, on MW2MW layer, solution from DS2DS layer can be used as a component that utilizes the communication infrastructure to communicate (receive input messages, publish output messages) with other MW2MW components. Messages exchanged with IoT platforms can be translated syntactically in MW2MW and semantically in DS2DS component. On AS2AS layer, DS2DS solution can be used for semantic translation of data received e.g. from IoT applications and services. Note that, on AS2AS layer service catalogue with semantic annotations can be used to enable more efficient service description and discovery.

2.5.4 Summary table

IOT SEMANTICS		
Artifact	**Description**	**Comment**
CSIRO Sensor Ontology	a generic ontology for describing functional, physical and measurement aspects of sensors	Contributed to the development of W3C SSN ontology
SWAMO Ontology	semantic description of	

	the fundamental concepts in the Sensor Web system including description of autonomous agents for system-wide resource sharing, distributed decision making, autonomic operations	
MMI Device Ontology	an extensible ontology of marine that integrates with models of sensor descriptions	
SEEK Extensible Observation Ontology	a suite of ontologies for modeling and representing scientific observations	
SemSOS	observation-centric ontology suite	
W3C SSN	a suite of general purpose ontologies for describing sensors, their accuracy and capabilities, observations and methods used for sensing	Inspired by aforementioned ontologies, the stimuli-centered ontology design pattern and OGC SensorML standard
IoT-Lite	a lightweight instantiation of the SSN	
WSSN	wireless sensor networks ontology	Extensions (specializations) of W3C SSN
SCO	sensor cloud ontology	
AEMET	meteorological ontology	
SWROAO	atmosphere observation ontology	
SAO	stream annotation ontology	
SemSorGrid4Env	flood prediction ontology	
OpenIoT ontology	Developed within OpenIoT project, based on W3C SSN	
SAREF	Smart Appliances Reference Ontology	
oneM2M	ontology of the IoT Platform oneM2M	not yet fully completed
SERVICE DESCRIPTION SEMANTICS		
Artifact	**Description**	**Comment**
WSDL	Web Service Description Language	format for describing services
WSDL-S	Web Service Semantics	semantic annotations for WSDL
SAWSDL	Semantic Annotations for WSDL and XML Schema definition language	
USDL/Linked-USDL	United Service Description Language - a data model for describing various types of services and its linked data variant	service description ontologies
WSMO, WSMO-Lite	Web Service Modeling Ontology and its lightweight version	

OWL-S	Semantic Markup for Web Services	
hRESTS	Microformat for describing RESTful services	format for describing RESTful services
microWSMO	semantic annotations standard for hRESTS	Semantic annotations for hRESTS
SEMANTICS IN IOT PLATFORMS		
Artifact	**Description**	**Comment**
FIWARE	the IoT Discovery GE provides the Sense2Web linked-data platform semantic repository for registering and managing descriptions in RDF/OWL, as well as querying the data via SPARQL	
OneM2M	oneM2M Base Ontology shall provide an ontology that other systems can align wit	
OpenIoT	the OpenIoT ontology uses the W3C SSN ontology combined with other vocabularies / ontologies	
UniversAAL	All components semantically annotated; most ontologies build from scratch	
Sofia 2	An ontology as the set of classes and attributes that will be shared within the Smart Space	
SEMANTIC TOOLS		
Artifact	**Description**	**Comment**
LogMap	ontology matching tool	
COMA	customizable framework for matching and merging algorithms	
AgreementMaker	extensible framework implementing many matchers	
AlignmentAPI	alignment persistence format and tools for its manipulation	
SilkFramework	tool for discovery of links between datasets in the context of the Open Linked Data	
S-Match	semantic matching framework that transforms tree-like structures into lightweight ontologies	
OntoBuilder	tools to extract ontologies from Web pages and map ontologies from similar domains, generating an ever-improved single ontology	

Table 6: Summary of relevant semantic tools and resources.

3 INTER-LAYER Specifications

INTER-LAYER is the layer-oriented approach of INTER-IoT for providing interoperability solutions at different layers of technology stack. For INTER-LAYER description and overview refer to section 1.1.

INTER-LAYER is composed from a set of solutions addressing interoperability at each layer of an IoT platform or system: Device-to-Device (D2D), Network-to-Network (N2N), Middleware-to-Middleware (MW2MW), Application Services-to-Application Services (AS2AS) and Data & Semantics-to-Data & Semantics (DS2DS).

In the following subsections, the architecture solution dedicated to each layer, components and corresponding use cases are described. Furthermore, if it is considered relevant, the technologies applied on each layer are as well described and explained.

3.1 D2D proposed solution

The following factors illustrate the need to implement an interoperability solution at the device layer:

- Applications and platforms are tightly coupled, preventing them from interacting with other applications/platforms.
- Sensors and actuators communicate only within one system.
- Certain platforms do not implement some important services (i.e. discovery), or do so in an incompatible way.
- Roaming elements can be lost or inaccessible.
- IoT Device software is never platform independent, since companies produce proprietary/closed solutions for economical reasons. This makes interoperability hard or impossible.

Interoperability at the device level implies that heterogeneous IoT devices are able to interact with each other. IoT devices can be accessed/controlled through a unifying interface and integrated into any IoT platform.

This interoperability solution at device level is achieved through a Device to Device Gateway (D2D Gateway, or sometimes simplified as Gateway or GW in this document). There are two approaches for the implementation of this Gateway: physical and virtual.

The physical implementation of the Gateway is oriented towards hardware with medium computational capacities and storage. The virtual implementation is meant for hardware with low computational capacities and storage. In this implementation only the south part of the Gateway (network and protocol capabilities) is processed in hardware while all the other Gateway functionalities are shifted to the virtual Gateway with more computational power.

For this reason, the Gateway will be developed in Java using the OSGi framework. All components of the Gateway will be packaged as OSGi bundles and the OSGi framework will be utilized to control the bundles that will be needed depending on the chosen implementation mode.

3.1.1 Architecture

The gateway architecture is shown in Figure 34. To understand the content first we need to define the concept of a device. For our purposes a device is a platform or hardware that is able to run a gateway as stated in Figure 34. It has sufficient processing power, storage facilities and two or more connections which are omni-directional.

The gateway at the device level is designed in a way that modularity in protocols and access networks is always considered. Any access network (AN) can be inserted into the structure as long as it is interfacing accordingly with the Controller. The same is true for the protocols and middleware modules.

The device is build up in a way that once the system structure is functional a split-up can be realized. Part of the device gateway can be placed in the Cloud to allow functionalities that a physical gateway is not able to perform in an efficient way. The device Dispatcher will take care of connecting or simulating the actual platform. When connection is lost, the virtual part remains functional and will answer to requests of API and MW.

At the lowest level there are the sensors and actuators. These are connected to the AN modules. These modules take care of connectivity with the (wireless) sensors and actuators. The AN Controller will not interfere in this connectivity.

The Device Manager takes care of coupling of the AN Module to active or needed protocol. The Controllers only handle traffic routing to and from the modules.

Once a sensor/actuator is registered the Device Manager will store this information in the Registry.

When the device receives new measurement data it will be passed to the Dispatcher that will store it in the measurement storage. Any data update for the MW (or higher levels) will come from the Measurement Storage.

The Dispatcher will pass the information to the MW Controller that sets up the connection with the MW Module through which the data is sent to the MW layer (Figure 48).

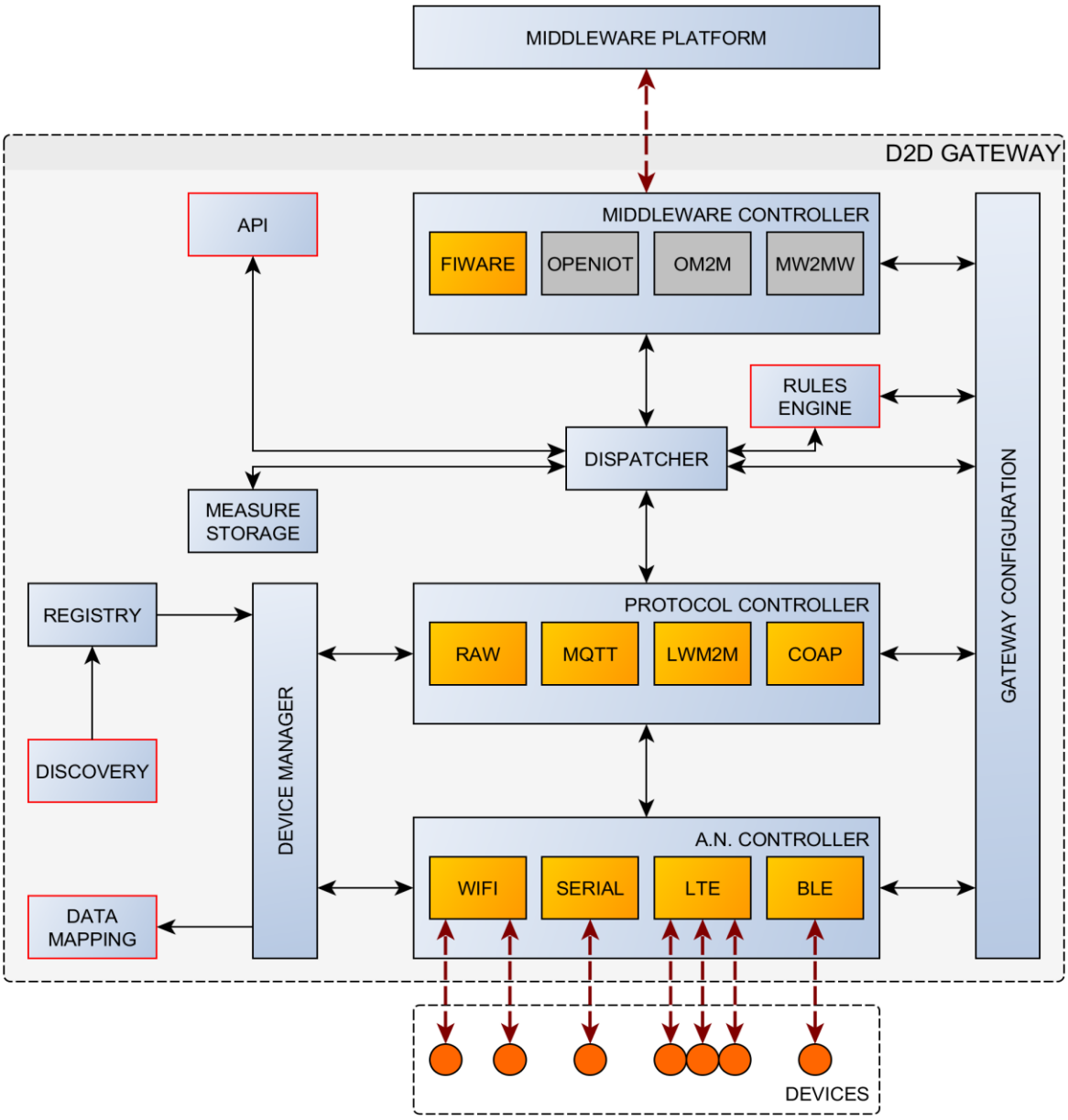

Figure 34: Gateway architecture overview (in yellow: components that can be implemented but at least one is needed for a functional system, blocks with red border are optional and can be implemented when needed).

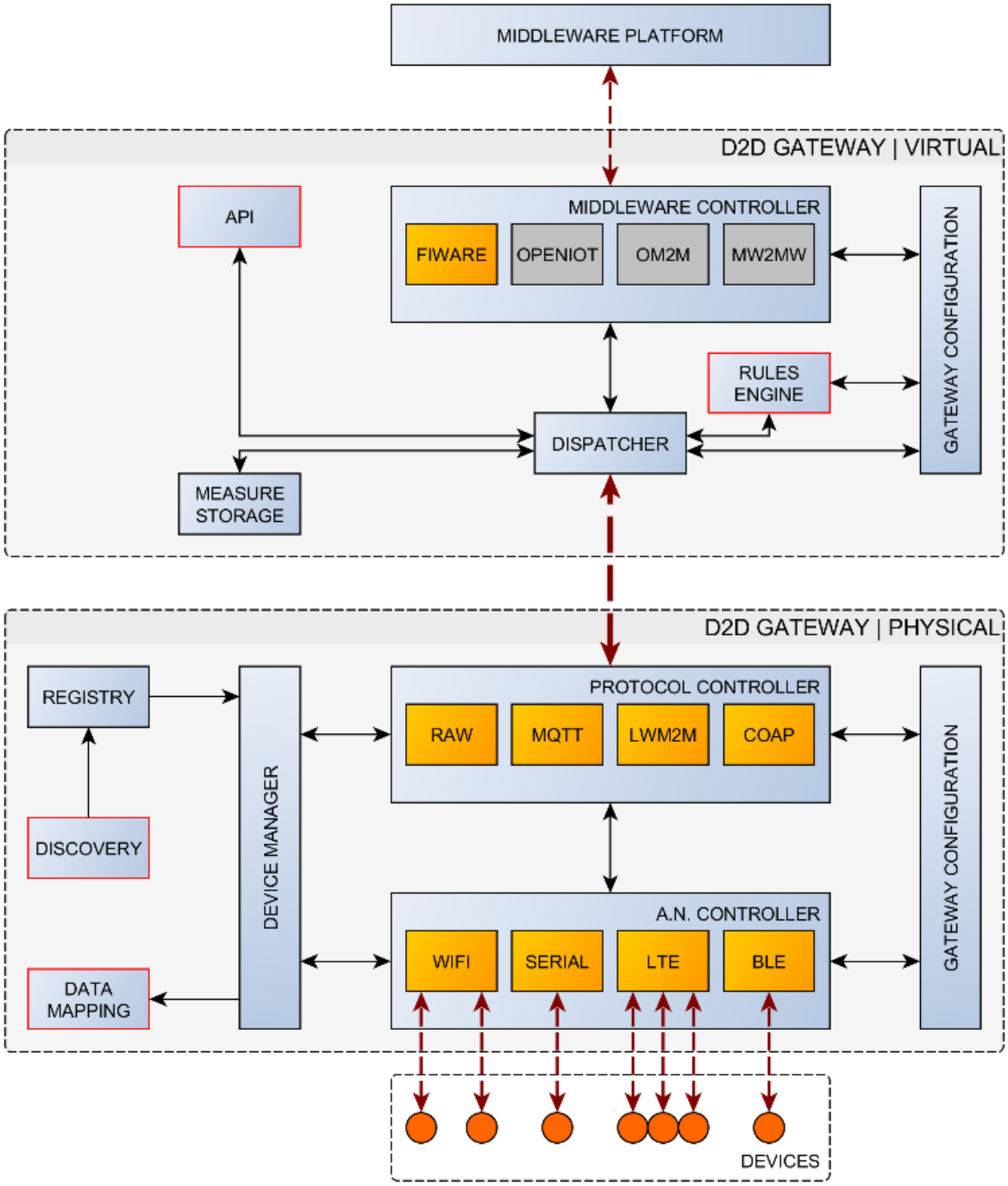

Figure 35: Gateway architecture split into two parts, the physical part for the embedded device and the part that can be executed in a virtual container.

3.1.2 Components

Component	Registry
Description	This component is responsible of registering all the devices with its multiple sensors and actuators in the gateway.
Functionalities	Reads from a file (later could be a UI) the configuration of a sensor/actuator (a device could have more than one sensor/actuator or just one) in JSON or XML format that includes the name of the device, the access network IF supported, the communication protocol supported, and an array with the name of the value measured, the value, and the type of data, sending later on this information to the device manager and to create a Unique ID for each sensor/actuator.
Relation with other component	The Registry module will add an entry in the Device Manager with the information about each sensor and actuator.
Use Cases Involved	[60]
Requirements Involved	[245], [242], [138], [93], [57], [45], [39], [60], [22], [15]

Component	Device Manager
Description	The Device Manager is a component that will be accessible to every other component that needs information of any sensor/actuator.
Functionalities	It will store in memory (and persisted in a local database) a map containing an entry for each sensor/actuator. This entry will be identified by an unique ID generated automatically and will contain information about the type of sensor/actuator, the protocol used for communication and the physical address of the device of the access network.
Relation with other component	The protocol and access network modules will call the Device Manager in order to resolve the metadata for each sensor/actuator.
Use Cases Involved	[60], [62], [64], [65]
Requirements Involved	[138], [93], [45], [39], [23], [22], [21], [15]

Component	AN Controller
Description	Allows access to the devices. Interfaces between the devices and the protocol modules.
Functionalities	Operations supported: • Device - AN Module binding, AN Module mounting • Device power off/on • Data reading from device • Data writing to device This interface is to be used by Protocol Controller and to be implemented by AN Modules.
Relation with other component	This interface is to be used by Protocol Controller and to be implemented by AN Modules. The Device Manager configures the access network module according to the registry.
Use Cases Involved	[60], [62], [64]
Requirements Involved	[138], [93], [45], [39], [23], [22], [21], [15]

Component	AN Modules
Description	The Access Network (AN) modules provide the INTER-IoT GW access to the following communication channels: • 802.15.4, namely the ZigBee specification (other specs within the 802.15.4 standard may be included in the future) • WiFi • Serial communication via USB • Other proprietary RF links accessible via SDR based solutions compatible with the GW device
Functionalities	Each of the AN Modules will perform the following tasks using the respective Access Network: • Establish/terminate a communication channel to the sensor/actuator

	• Send data to the sensor/actuator • Receive data produced by the sensor/actuator and forward it to the AN controller for due processing • Trigger connection status notifications
Relation with other component	The AN Controller will use the AN Module interface to perform the following tasks: • Establish/terminate a connection with the sensor/actuator • Request data from the sensor/actuator • Handle the data pushed by the sensor/actuator to the GW • Send commands to sensors/actuators • Test the sensor/actuator connectivity upon registration • Enforcement of the sensor/actuator reconnection policy
Use Cases Involved	[60], [62], [63], [64]
Requirements Involved	[138], [93], [45], [39], [23], [22], [21], [15]

Component	**Protocol Controller**
Description	This component is located within the real part of the gateway architecture and it contains all the communication protocols supported by the Gateway also implementing the common interfaces between those protocols and the other components such as the PHY GW Configuration, the AN Controller, the Device Manager and the Dispatcher.
Functionalities	This component provides support to the messages exchanged during the registration phase and the triggering action of each device. Also during the communication between the MW Platform and the devices (through the Dispatcher) when they act as actuators and during the internal communications with its specific Protocol Modules (CoAP, MQTT, LWM2M, etc…).
Relation with other component	During the registration of each device, the Protocol Controller: • Receives from the Device Manager the request about the protocol supported/used by a specific device • Sends the previous request to the specific Protocol Module (CoAP)

	• Sends back an acknowledgement to the Device Manager once it has received the requested information from the Protocol Controller During the triggering action of each device, the Protocol Controller: • Receives protocol information for each specific device from the AN Controller • Sends the information to the relevant protocol Module with raw data (byte []) During the platform sends information to a device (actuator), the Protocol Controller: • Receives an actuator common message from the Dispatcher • Sends a request info about the, i) Protocol ii) the Access Network iii) the physical address to the Device Manager and receives the answer • Sends the Actuator common message format to the Protocol Module and gets a parsed message • Sends the parsed message with the actuator instructions to the AN Controller
Use Cases Involved	[60], [62], [63], [64]
Requirements Involved	[93], [57], [45], [39], [256], [15], [23], [21], [283], [153], [72], [56], [26], [25]

Component	**Protocol Module**
Description	This component is located within the Protocol Controller and it implements the specific features of any supported protocol (CoAP, MQTT, LWM2M, etc.) throughout standard interfaces towards the Protocol Controller and the Dispatcher.
Functionalities	This component provides specific support to messages exchanged during the registration phase and the triggering action of each device. Also during the communication with the MW Platform through the Dispatcher in both directions (from the devices to the MW Platform to collect data and from the MW Platform to the devices when they act as actuators) and during the communication between the Dispatcher and the Rules Engine.
Relation with other component	During the registration of each device, the Protocol Module: • Notifies its existence to the Protocol Controller

	During the triggering action of each device, the Protocol Module: • Sends the structured Data (UID, Info, etc…) to the Dispatcher During the information exchange between the Dispatcher and the MW Platform, the Protocol Module: • Sends the structured Data (UID, Info, etc…) to the Dispatcher During the information exchange between the Dispatcher and the Rules Engine, the Protocol Module: • Sends the structured Data (UID, Info, etc…) to the Dispatcher During the platform sends information to a device (actuator), the Protocol Module: • Receives an actuator common message from the Protocol Controller • Sends back a protocol parsed message to the Protocol Controller
Use Cases Involved	[62], [63], [64]
Requirements Involved	[93], [45], [39], [23], [256], [21], [15], [283], [153], [72], [56], [26], [25]

Component	**Gateway Configuration**
Description	This component will be duplicated in the virtual and physical part. It will be a simple module that will read the gateway configuration from a configuration file.
Functionalities	It will read a properties file with a key/value format and store it in memory. Every component can access the gateway configuration component to get the configuration value for a given key.
Relation with other component	Every other component can use this component to access the gateway configuration.
Use Cases Involved	[61], [64], [65], [62], [47]
Requirements Involved	[283], [153], [93], [72], [45], [39], [22], [21], [15]

Component	Dispatcher
Description	The Dispatcher handles all traffic between the Protocols layer (physical device) and the Middleware Controller (virtual device). The device (Protocol Controller) will sent a trigger to the Dispatcher whenever a new data sample is available, the Dispatcher stores the new measurement data from the device into the measurement storage. Any update request or data request from upper layers (MW or API) will be handled by the Dispatcher. It will get the latest data sample from the Measurement Storage and sends it to the middleware.
Functionalities	Routing of data and messages between MW Controller and Protocol Controller (physical and virtual)
Relation with other component	Measurement storage, GW Configuration, API, MW Controller, Protocol Controller, Rules Engine
Use Cases Involved	[65], [61], [62], [64], [49], [48], [47], [27], [10], [9], [6], [46], [20], [15], [36], [19]
Requirements Involved	[283], [153], [93], [72], [45], [39], [22], [21], [15]

Component	Measurement Storage
Description	The Measurement Storage (MS) handles specific requests coming from the Dispatcher, which forwards a request form a platform. It returns the data to the Dispatcher that forwards them towards the platform.
Functionalities	As a cache in the gateway it stores the information about the devices connected and the last available value, in case of polling, of these devices. If a platform requests the value, and the one contained in MS is practically new, or it is the last one obtained in case of disconnection the value is returned in a faster way. This helps to improve the performance and speediness of the data gathering. Additionally, if the data from a device stays the same during a long period of time there is no need to update MS every time the device sends a new value, the last value is maintained, saving resources.
Relation with other component	Dispatcher External API

Use Cases Involved	[62], [64]
Requirements Involved	[283], [153], [138], [93], [72], [45], [39], [23], [22], [21], [15]

Component	MW Controller
Description	The MW Controller is the module that acts as a mediator between the MW Module and the rest of the gateway.
Functionalities	It will wrap the active MW Module in order to have a common interface for the gateway. It will check through thee Gateway Configuration that only one of the MW Modules is active. It will create the connection to the MW platform and will handle the messages interchanged between the module and the platform, as well as the messages sent to the Dispatcher.
Relation with other component	It will be the communication interface in the north of the gateway to the middleware platform. This component will use the Dispatcher in order to deliver the messages sent by the MW Module to the rest of the gateway.
Use Cases Involved	[64], [65], [62], [61]
Requirements Involved	[283], [153], [138], [93], [72], [45], [39], [56], [26], [25], [23], [22], [21], [15]

Component	MW Module
Description	The MW Module will be specific to a IoT Middleware platform and will handle the communication of the gateway with the platform.
Functionalities	This component will be activated by the Middleware Controller and will implement the function of registering the sensors and actuators to the middleware platform as well as processing the requests and responses exchanged with the platform.
Relation with other component	This component will be placed inside the Middleware Controller, and will interact with the Middleware Controller to access the rest of the gateway.

Use Cases Involved	[64], [65], [62], [61]
Requirements Involved	[283], [153], [138], [93], [72], [45], [39], [56], [26], [25], [23], [22], [21], [15]

Component	Commons
Description	Even if it does not appear in the architecture, is a basic component that includes several methods and tools to be used by the rest bundles/components.
Functionalities	It will be a collection of classes that implements the common message formats used by the rest of the components as well as utility classes.
Relation with other component	Every other component can use this component/library to access the common classes and interfaces implemented.
Use Cases Involved	[64], [65], [62], [61], [60]
Requirements Involved	[283], [153], [138], [93], [72], [45], [39], [56], [26], [25], [23], [22], [21], [15]

3.1.3 Use Cases

Use case	Device Registry
Use Case ID	#60
Description	A device is registered within the gateway by a descriptive method with basic parameters needed for its addressability and understanding of data.
Objectives	To include the information about a device, sensor or actuator, in order to receive or send information from the device and to the gateway or to another system connected to the gateway.
Components Involved	The Registry Module. The Device Manager. The AN specific Module. The AN controller. The Protocol specific Module.

	The Protocol Controller. The GW Configuration Module.
Requirements Involved	[245], [242], [138], [93], [57], [45], [39], [60], [22], [15]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-834

GW01 Device Registry

View online: http://tinyurl.com/gw01v01

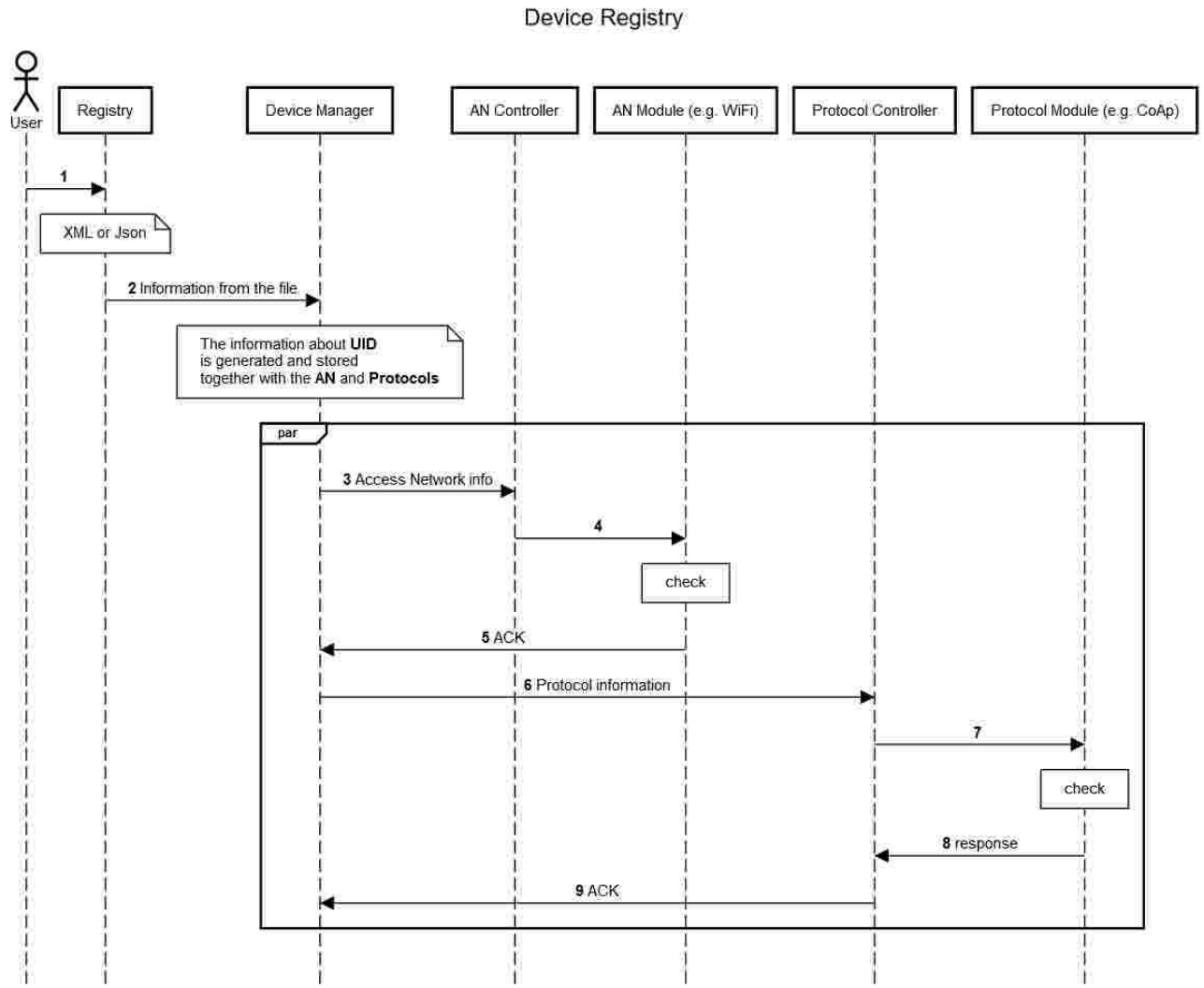

Figure 36: Device Registry sequence diagram.

Step 1: A user wants to register a device to be used. Then, he writes in the configuration file the parameters needed to register and connect the device.

Step 2: The registry module reads from this file and includes this information in the system

Step 3: The information about access network is sent to the AN Controller to be checked.

Step 4: The AN Module starts the bundle of the respective module to test the connection.

Step 5: A confirmation (ACK) or non-confirmation (NACK) is returned to the Device Manager.

Step 6: At the same time, the information about the protocol is also checked, connecting with the Protocol Controller.

Step 7: The Controller starts the adequate Protocol Module to check its availability.

Step 8: Information about the protocol is returned to the Controller, in case of more than one protocol supported, this step takes place several times.

Step 9: Finally, a confirmation by the Controller with all the protocols supported by the device is sent to the Device Manager.

Use case	A Platform is configured on the Gateway
Use Case ID	#61
Description	The configuration of a platform that will be connected to our gateway and will receive/send all the information from/to the devices. Initially this configuration could be done by a configuration file. Later a simple GUI can be created for inserting the information about the platform in order to create the connection.
Objectives	To configure correctly the platform so this one can exchange information with the gateway and so with the devices.
Components Involved	The MW Controller Module. The MW specific platform Module. The GW Configuration Module.
Requirements Involved	[39], [20], [15]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-835

GW02 Platform Registry

View online: http://tinyurl.com/gw02v01

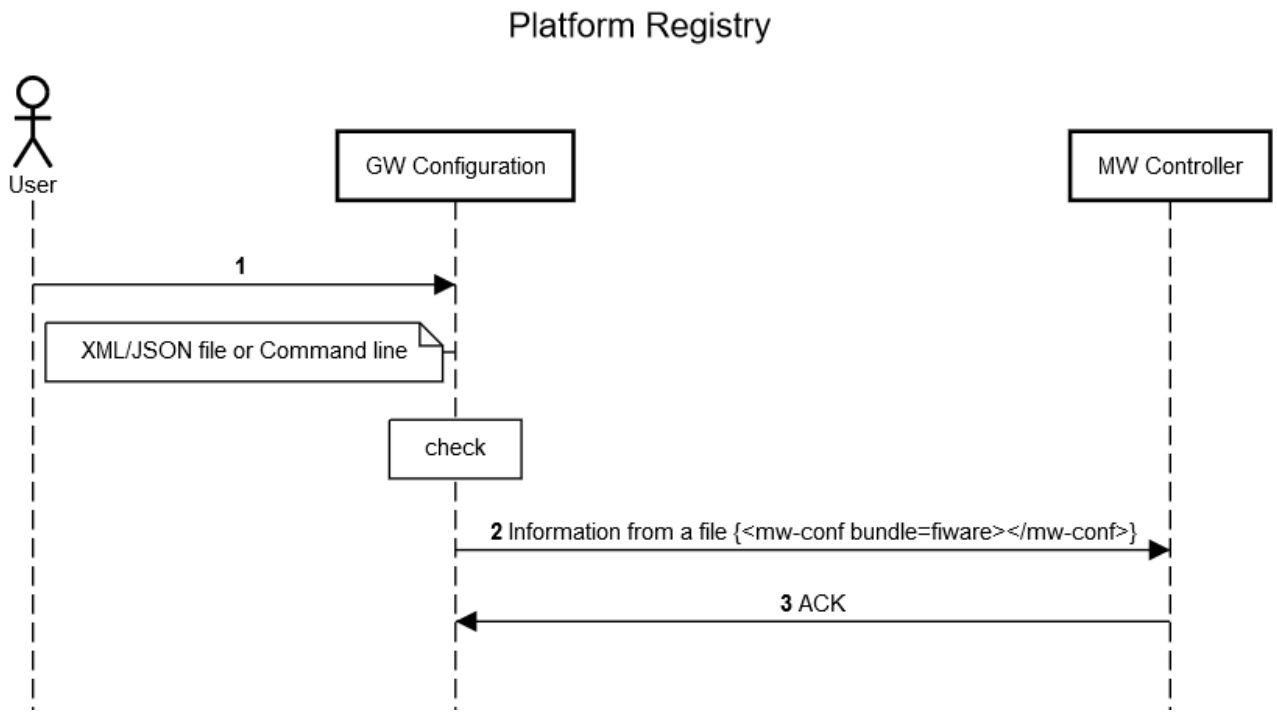

Figure 37: Platform Registry sequence diagram.

Step 1: A user starts the gateway framework.

Step 2: The GW Configuration module is activated and reads from a file all the gateway configuration entries.

Step 3: The MW Controller is activated and gets from the Configuration Module all the information related with the configuration of the MW platform.

Step 4: The MW Controller performs a test of communication with the MW platform and throws an Exception if there is a problem.

Use case	Device (sensor) triggers information
Use Case ID	#62
Description	A device, typically a sensor, triggers an event sending determined information to the gateway in order to be stored on the IoT Middleware Platform or in order to generate a response for an actuator (being handled by the rules engine).
Objectives	To send data from the device side through the gateway to reach its destination (local platform, cloud or other device) in an efficient way.
Components Involved	The AN Module. The AN Controller.

	The Device Manager. The Protocol Specific Module. The Protocol Controller. The Dispatcher. The MW Specific Module. The MW Controller.
Requirements Involved	[138], [93], [45], [39], [23], [22], [21], [15]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-838

GW03 Device Trigger

View online: http://tinyurl.com/gw03v01

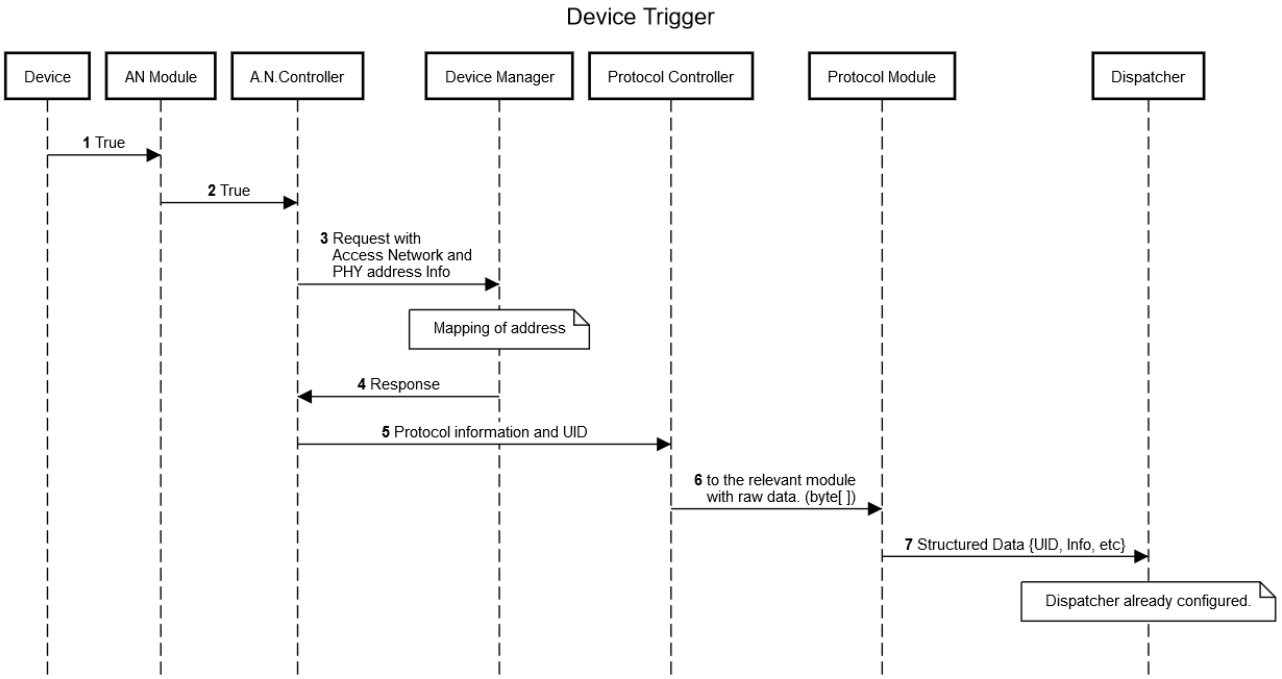

Figure 38: Device Trigger sequence diagram.

Step 1: Device sends data to the AN Module.

Step 2: AN Controller sends request to Device Manager with AN and PHY address.

Step 3: Device Manager looks up UID and protocol information based on AN and PHY address.

Step 4: Device Manager sends response to AN Controller with UID and protocol information.

Step 5: AN Controller sends data, protocol information and UID to Protocol Controller.

Step 6: Protocol Controller sends raw data with UID to the correct Protocol Module.

Step 7: Structured data is sent to the Dispatcher.

GW04 Dispatcher to MW platform

View online: http://tinyurl.com/gw04v01

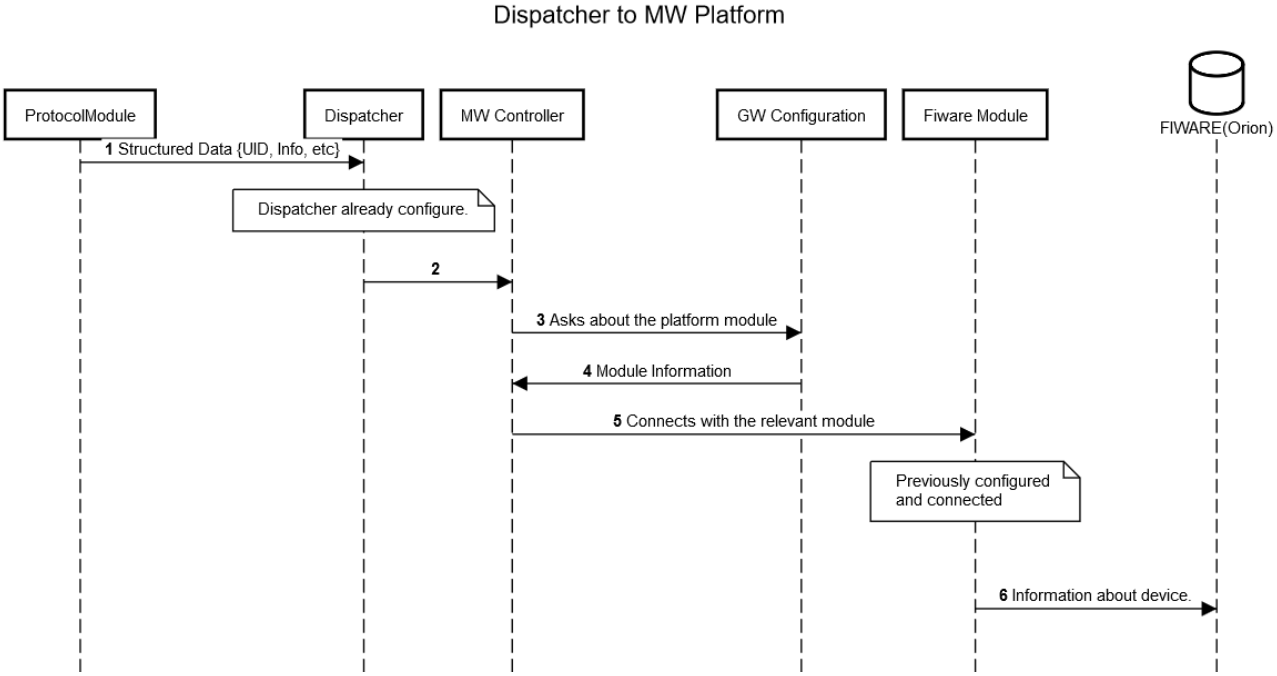

Figure 39: Dispatcher to MW platform sequence diagram.

Step 1: The Protocol Module sends the structured data, following the message common format, to the Dispatcher with metadata information about internal UID etc.

Step 2: The Dispatcher, previously configured, takes this messages and sends it to the Middleware Controller.

Step 3: The Middleware Controller asks to the Gateway Configuration about the module that is running connected to the platform.

Step 4: Gateway Configuration response about the platform is sent.

Step 5: The Controller then connects with the relevant Module, or even start it if it was not running already (for energy saving reasons), and sends the information.

Step 6: The adequate Module takes the message and encapsulate in the format that the platform supports.

Step 7: The information is finally send to the specific component of the platform.

GW05 Dispatcher to Rules Engine

View online: http://tinyurl.com/gw05v01

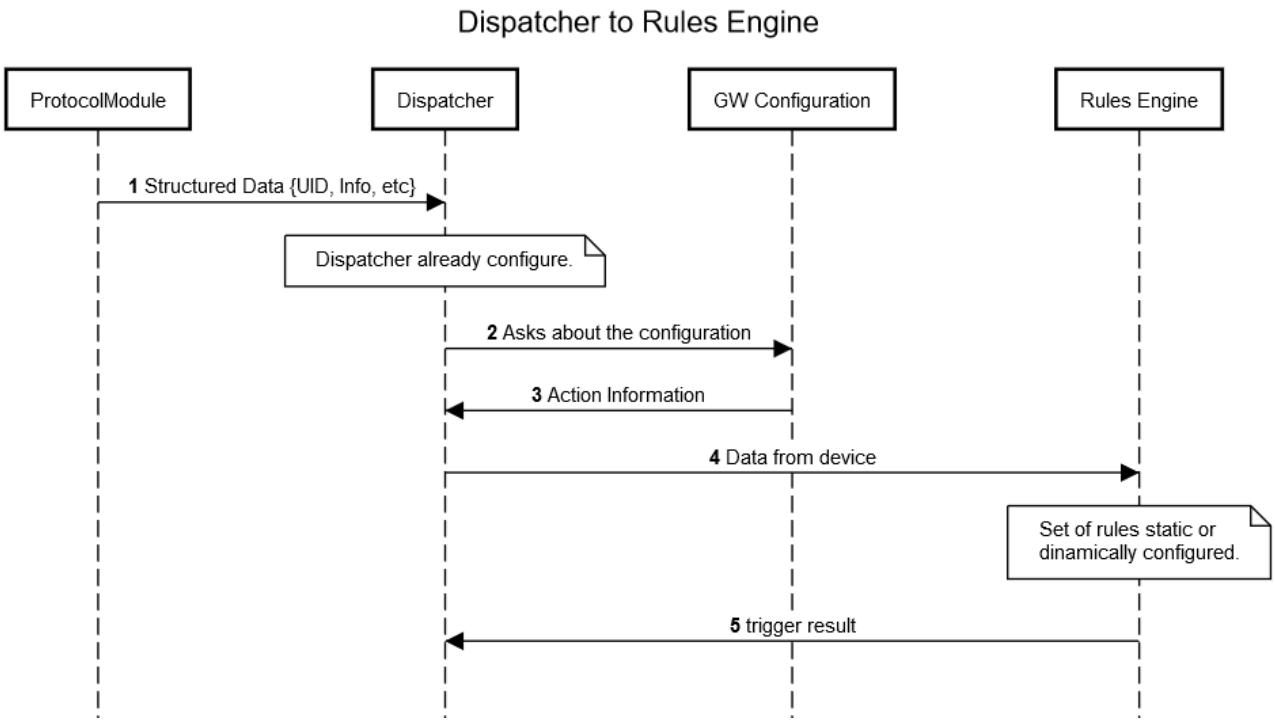

Figure 40: Dispatcher to Rules Engine sequence diagram.

Step1: The Protocol Module sends a structured data to the Dispatcher Module.

Step2: The Dispatcher is already configured, and sends an information request to the GW Configuration.

Step3: The GW Configuration Module sends configuration information back to the Dispatcher.

Step4: The Dispatcher sends the configuration data to Rules Engine Module.

Step5: The Rules Engine sets the configuration by statically or dynamically configuring the specific rules.

Step6: The Rules Engine sends the trigger result to the Dispatcher.

Use case	Platform requests information from a device (sensor)
Use Case ID	#63
Description	The Gateway receives the request from the Platform and re-directs it to the Device, to obtain specific information. If no change in the value has been performed in a short period, the response will be provided directly from the Measurement Storage.
Objectives	To obtain a data requested by the Platform from a concrete Device.

Components Involved	The platform module connected to the IoT platform. The Platform Controller. The Dispatcher. The Measurement Storage. The Protocol Module and Controller. The AN Controller and Module.
Requirements Involved	[283], [153], [93], [72], [45], [39], [22], [21], [15]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-839

GW06 Platform request

View online: http://tinyurl.com/gw06v01

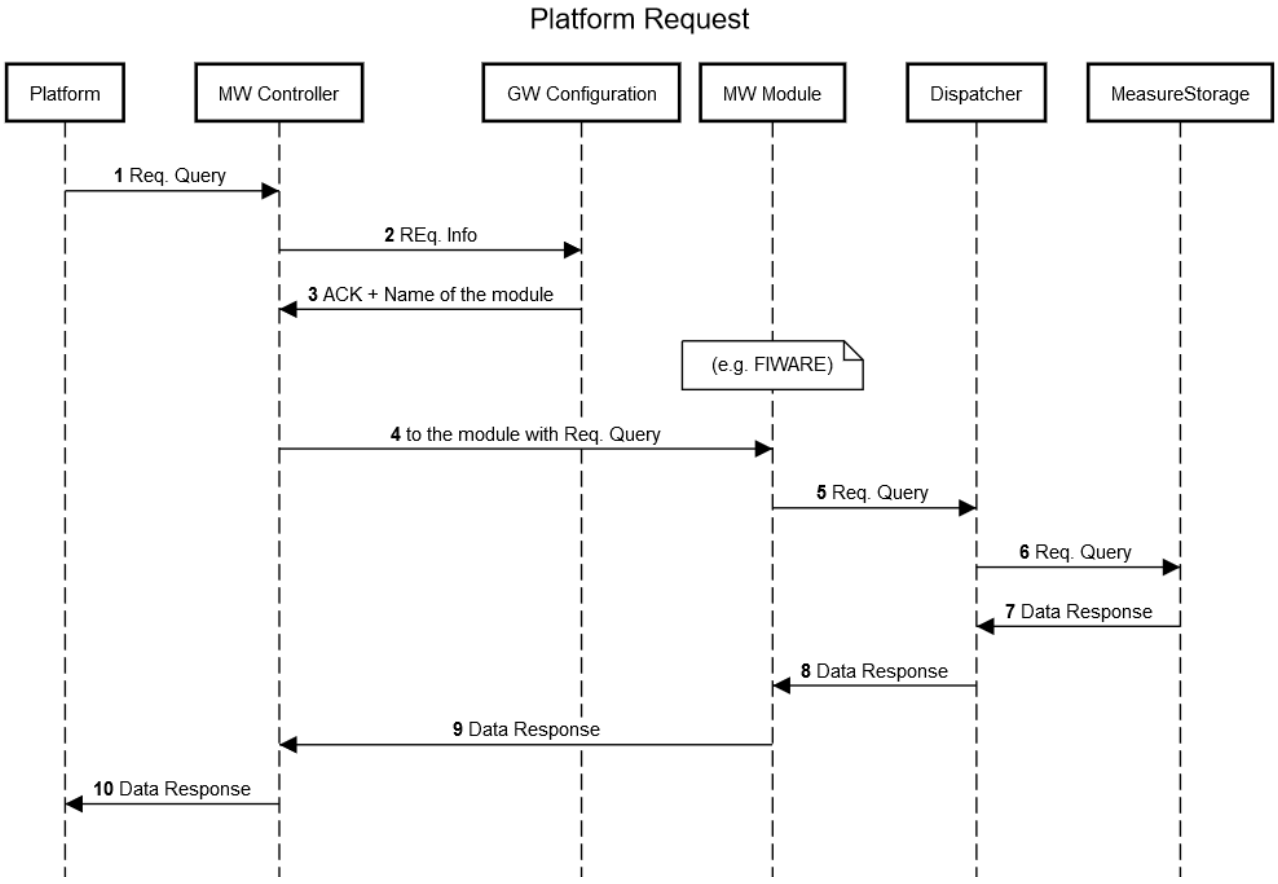

Figure 41: Platform request sequence diagram.

Step 1: The platform sends a query requesting determined information from a device.

Step 2: The request is handled by the Middleware Controller.

Step 3: The MW Controller consults in the Gateway Configuration Module about the bundle in charge of manage and parse the request message.

Step 4: The Gateway Configuration Module informs the Controller about with bundle is adequate.

Step 5: The request message arrives to the MW Module and is parsed to a common format message to be sent latter to the Dispatcher.

Step 6: The Dispatcher takes the message and reads the information from the Message Storage, if this one has been recently included.

Step 7: The information requested if it is the last value is returned to the Dispatcher.

Step 8: The Dispatcher forwards this response message to the pertinent module to be parsed.

Step 9: the MW Module parses the response and sends it to the platform.

Use case	Platform sends information to device (actuator)
Use Case ID	#64
Description	The platform sends information, normally a change of state, to the device, typically an actuator.
Objectives	To change the state of an actuator connected to the gateway.
Components Involved	The MW Module connected to the platform. The MW Controller. The Dispatcher. The Protocol Controller. The relevant Protocol Module. The Device Manager. The relevant AN Module. The AN Controller.
Requirements Involved	[283], [56], [45], [39], [26], [25], [22], [21], [15]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-840

GW07 Platform response to Actuator

View online: http://tinyurl.com/gw07v02

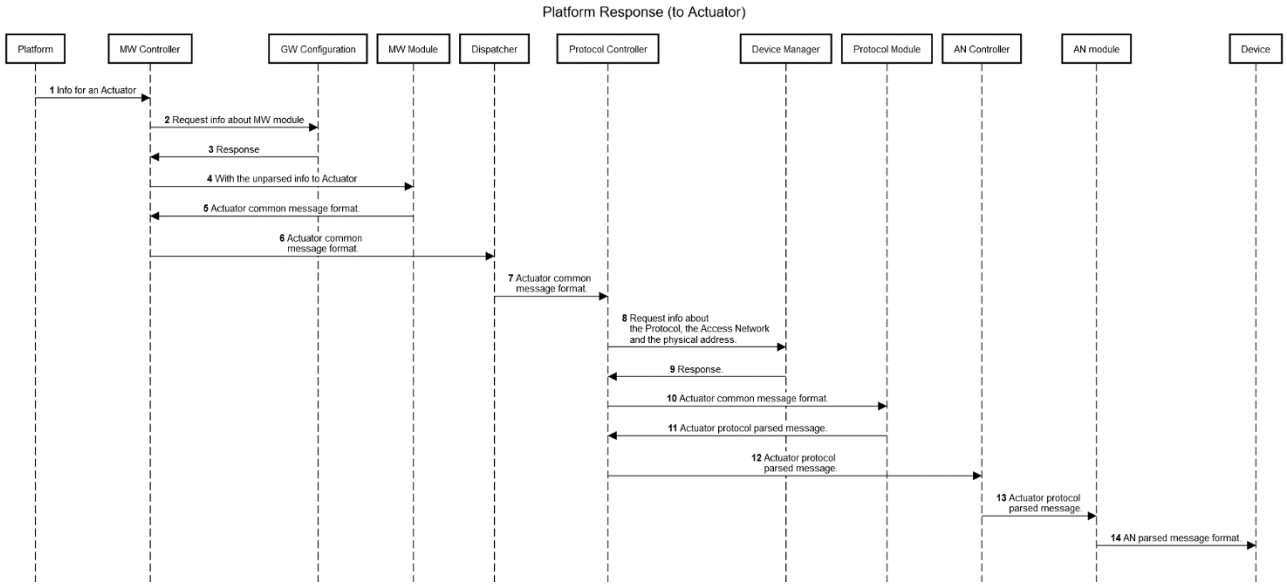

Figure 42: Platform response to Actuator sequence diagram.

In this case, the platform sends data to a device (actuator):

Step 1: The information to be sent to the actuator arrives from the platform to the Middleware Controller.

Step 2: The MW Controller requests information from the Gateway Configuration Module to know which MW Module is used.

Step 3: The answer is sent back to the MW Controller.

Step 4: The MW Controller sends the data to the correct MW Module in order to format it correctly.

Step 5: The actuator message correctly formatted is sent back to the MW Controller.

Step 6: The MW Controller transfers this data to the Dispatcher.

Step 7: The Dispatcher sends the message to the Protocol Controller.

Step 8: The Protocol Controller sends a request to the Device Manager to know the protocol, AN Module, and physical address of the actuator.

Step 9: The answer is sent back to the Protocol Controller.

Step 10: The Protocol Controller sends the actuator message in common format to the corresponding Protocol Module.

Step 11: The Protocol Module answers with the parsed actuator message ready to be sent.

Step 12: The Protocol Controller can now send the parsed message to the AN Controller.

Step 13: The AN Controller sends the message to the corresponding AN Module, which transmits the message to the device.

Use case	Support to Non-Standard access networks
Use Case ID	#40
Description	There is a class of sensors/actuators that use non-standard RF communication links. The reasons why this class of devices use non-standard RF links include the following: • to improve the link range; • to allow interoperability with legacy systems; • to enhance the resilience to jamming and other kinds or interference; • to enhance link security. The INTER-IoT platform provides access to this class of devices, thus allowing the INTER-IoT user community to take advantage of the services provided by these sensors/actuators. To accomplish this the INTER-IoT GW comprises a bridge facility to perform the translation of the proprietary link into a standard IP based protocol usable by the INTER-IoT GW (e.g. Ethernet, WiFi).
Objectives	Allow sensors/actuators that use proprietary RF links to be accessible to the INTER-IoT community.
Components Involved	The Non-Standard AN Module that implements the bridging functionality. The AN Controller itself that invokes the Non-Standard AN Module functionality.
Requirements Involved	[17], [18], [170], [204]
Use case link	INTERIOT-779

GW08 Non Standard AN initialisation

View online: http://tinyurl.com/gw08v01

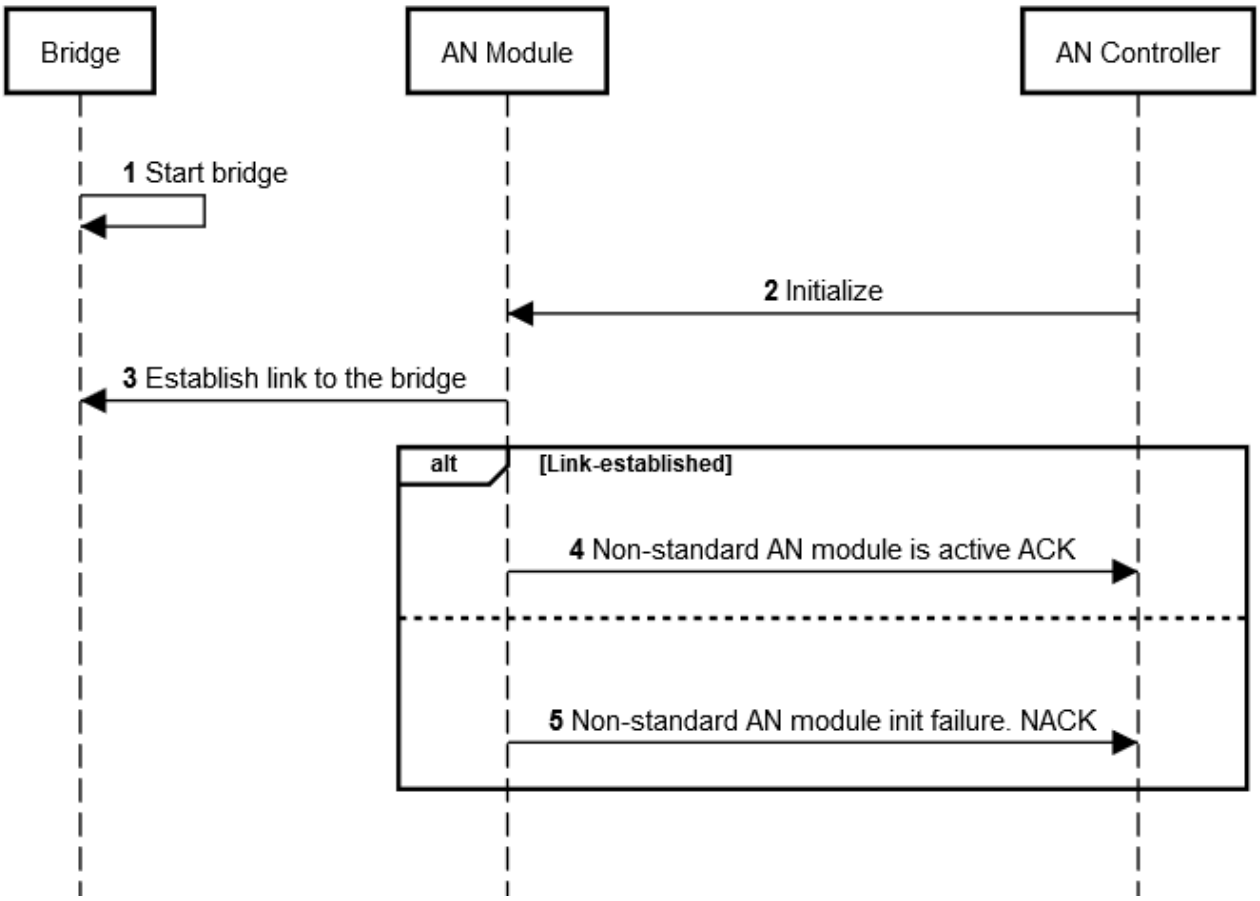

Figure 43: Non-Standard AN initialization sequence diagram.

This use case focus solely in the need to initialize the AN Module that is capable of connecting to the non-standard Access Networks through the specific external hardware element that handles the communication (called here “Bridge”). Once the initialization procedure is done, sensor/actuator devices that rely on the AN should behave as other sensor/actuator devices that use standard AN (e.g. Wifi, BLE, etc…). The following is a detailed description of the initialization procedure depicted in the diagram above:

Step 1: The Bridge is physically connected to the INTER-IoT platform and is running;

Step 2: During its own initialization procedure the AN Controller requests the Non-Standard AN Module to initialize itself;

Step 3: The Non-Standard AN Module will then try to connect to the Non-Standard ANs Bridge. If successful, the AN Module reports to the AN Controller that the GW is now able to connect to devices that use any (supported) non-standard Access Network; otherwise, the AN Module reports an initialization failure.

The Bridge element may be implemented using an SDR based solution. This approach offers obvious advantages because it allows the configuration of multiple proprietary links via a plug-in mechanism, thus removing the need to use one specific HW based bridge solutions for each non-standard link supported by the INTER-IoT framework.

3.2 N2N proposed solution

The great challenge which interoperability in the network layer must face is caused by the following problems:

- It is hard to manage big amount of traffic flows generated by smart devices.
- Poor scalability of systems, with difficulties to integrate new nodes.
- Difficulties in interconnecting gateways and platforms via networks used by different systems.
- Several devices with totally different radio network access have to be accessed from a single gateway as an AP.

Facing these problems and including new capabilities into the system INTER-IoT has created a solution based on two main paradigms; SDN and SDR.

Within these main applied paradigms some sub-solutions have been implemented to achieve the interoperability:

- Decoupling of data plane from logical plane with communication between both tiers via OpenFlow protocol.
- Virtualizing network services at the top of the architecture as routing, host tracking, topology discovery, and statistics.
- Implementation of techniques for traffic engineering to handle different flows of data generated by sensors depending on their priority.
- Implementation of a Software Defined Radio access network to dynamically choose the adequate connection channel for specific devices.

In the following sections, we will describe both solutions in terms of architecture, technologies, and components, and how them are related with the requirements extracted from the use cases and the stakeholders.

Additionally, other issues have been contemplated, such as, secure and seamless mobility of devices across networks (roaming) and the fast discharge of data from more than one connection (offloading). These two characteristics are addressed in some parts of the gateway and in the functions virtualized within the network.

We provide our interoperability solution at network layer and expose an open API. It will allow future developers and partners to utilize it and make their systems interoperate.

3.2.1 Architecture

Software Defined Network

As we have already defined in section 2.2.2.1, the SDN paradigm decouples the data plane (switches) and the control plane (SDN controller)in order to better manage the network from a central point of view, having a wide overview of this one. In our approach using the SDN paradigm and the well-known protocol OpenFlow, we will virtualize network services at the top of this architecture as routing and statistics, and implement techniques for traffic engineering to handle different flows of data generated by sensors depending on its priority.

The services offered at this layer will include routing algorithms based on typical routing protocols (OSPF, RIP, etc.) and variations for IoT traffic, storage of the network topology and state of the elements that compose the network, topology discovery and management, host tracking, packets statistics to know information about the type of traffic and security.

On the Southbound of the controller we will have a module to connect with the different virtual switches that will compose our system. Later on, we will implement (at the Northbound) the aforementioned virtual services within our controller using the APIs that this one provides.

The next figure shows the architectural overview of the whole network, consisting of two main parts.

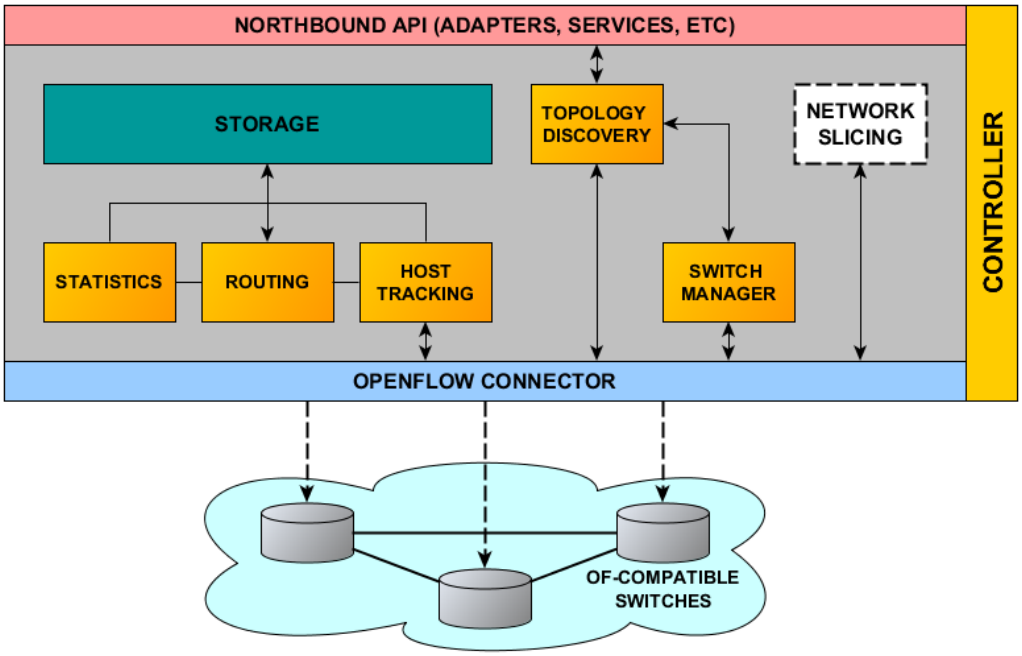

Figure 44: N2N proposed solution architecture for SDN.

The first part is the data plane, composed of virtual switches using the OpenVSwitch technology. They are connected to each other in a determined topology and all of them securely connected to the controller through TCP/SSL using OpenFlow. The other part is the logical plane where the controller is located, provided with an OpenFlow connector to parse all packets coming from the network to the different services running on it.

The information about the nodes of the network, the number of switches, its configuration and state, etc will be managed and stored within the Switch Manager. Additionally, the Topology Discovery will obtain this information to create a graph representing the state of the network with more information about the state of the links.

From the services perspective, a module to gather statistics about packet is running as well, taking the information from the other modules. The IoT Routing and Host Tracking Modules are in charge of creating and managing the routes that each packet has to take to reach its destination, and information about the packet origin.

Software Defined Radio

This INTER-IoT SDR component will be developed to provide an additional entry point in the Access Network Controller Modules section of the physical plane of the INTER-IoT gateway. The flexibility of this technology means that the applications to utilizing this feature are still to be defined. It is envisioned that as the technology develops and becomes less expensive, specific use cases will become more apparent.

The goal of this development will be to demonstrate a point-to-point communication link in the form of a TCP/IP wireless transparent bridge, to supply data via the SDR to be utilized by other systems connected to INTER-IoT. This will involve transferring data over IP from a platform to an SDR external to the INTER-IoT system. This SDR will then communicate the information wirelessly to the SDR included in the access network controller section of the INTER-IoT gateway. The data will be output over IP to the Protocol Controller be made available to other platforms served by INTER-IoT.

SDR Transmitter

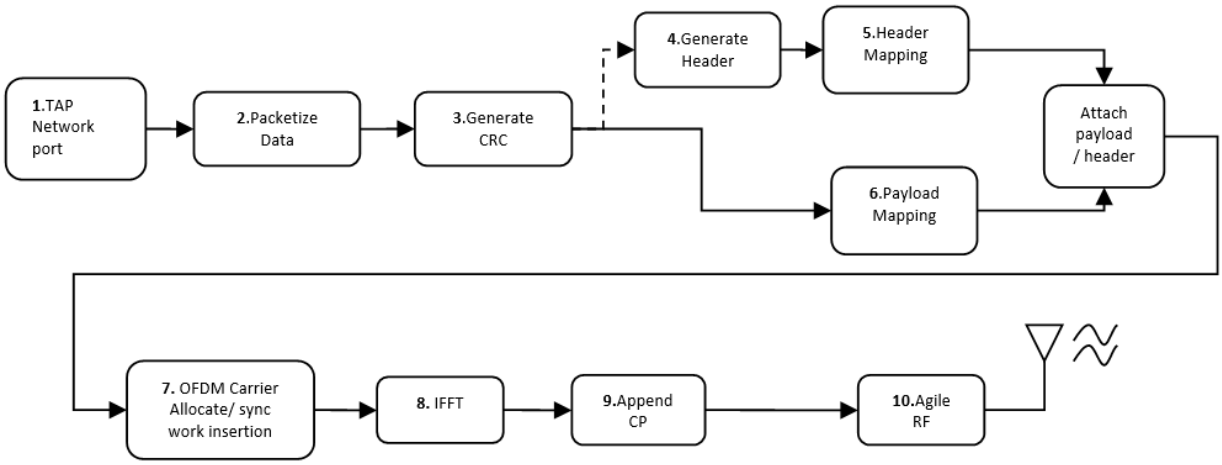

Figure 45: SDR Transmitter Dataflow.

Transmitter Dataflow notes:

1. Virtual Network Port (TUN/TAP) – interface to host OS.
2. Packetize data.
3. Generate CRC for error detection.
4. Generate header.
5. Map header bits to required header I/Q constellation symbols.
6. Map payload bits to required payload I/Q constellation symbols – this may or may not be the same symbol set as the header.
7. OFDM carrier allocation – allocate data to sub-carriers, insert pilot carriers, append pilot symbols and sync word(s).
8. Inverse Fast Fourier Transform (IFFT) to translate from frequency domain to time domain
9. Append the cyclic prefix – typically an additional FFT_length/4 time domain samples.

10. Agile tuneable software radio takes in digital complex baseband samples and outputs up-converted RF waveform.

SDR Receiver

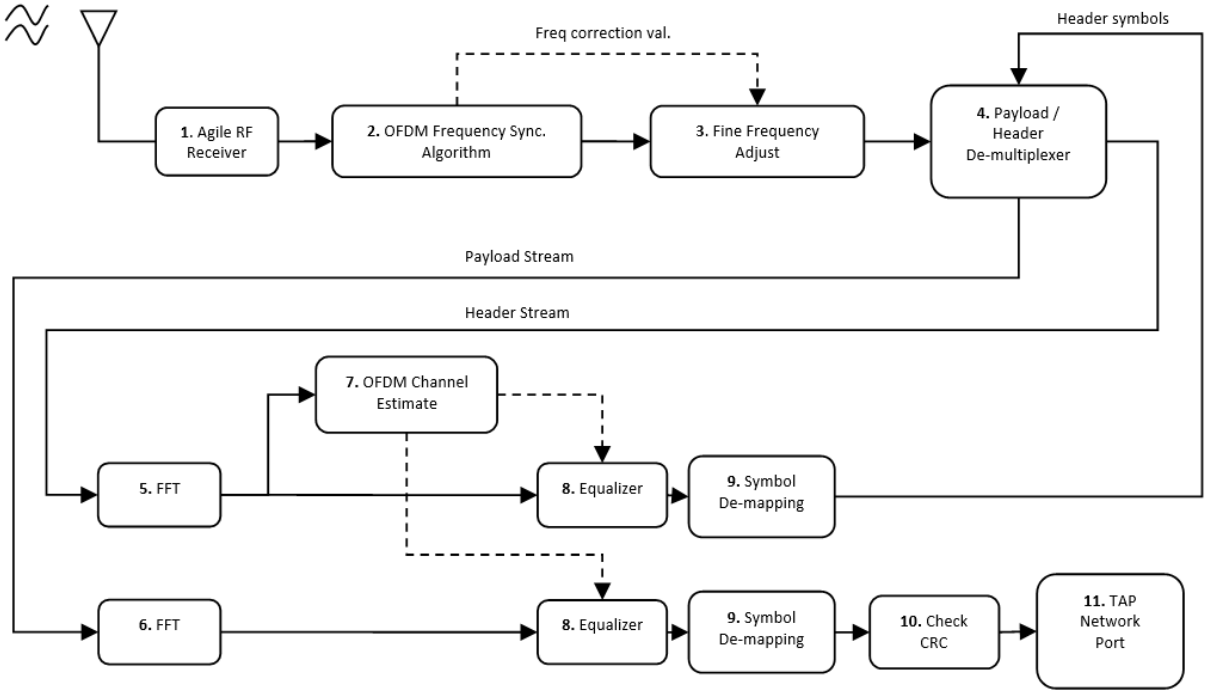

Figure 46: SDR Receiver Dataflow.

Receiver Dataflow Notes:

1. Agile tuneable software radio receiver takes in band-limited RF waveform and outputs down-converted complex baseband digital samples.
2. OFDM two-stage frequency synchronization algorithm – e.g. Schmidl and Cox.
3. Fine frequency correction.
4. Create two streams of data. Firstly, the header is extracted and channel estimation made before a buffered payload samples can be forwarded to the FFT for processing.
5. The header's Fast Fourier Transform (FFT) to convert from time domain to frequency domain.
6. The payload's Fast Fourier Transform (FFT) to convert from time domain to frequency domain.
7. Obtain channel estimates based on received pilot symbols / carriers.
8. Single-tap Frequency Domain Equalization (FDE) and course frequency correction.
9. Symbol de-mapping – hard decision decoding. Output is stream of bits.
10. Check CRC and accept / discard packets
11. Virtual Network Port (TUN/TAP) – interface to host OS.

3.2.2 Technologies

SDN Protocol: OpenFlow

As we already know is the protocol that aims to separate the intelligence required to route a packet from the act of moving it forward through the correct interface of the router, switch or network component and enables remote programming of the forwarding plane. This is achieved by inserting flow tables, designed by the protocol, inside the switches managed by the controller.

This protocol has several stable releases; from 0.8 to 1.3, that is the last one, but always the most used have been versions 1.1 and 1.3. In this case, the controller we have chosen to communicate through this protocol will support both versions in order to connect with legacy switches that maybe have implemented one of them.

The flow entries that compose the flow table inserted in the virtual switches and managed by this protocol only need three fields that are: Match Fields, Counters and Instructions.

Where the Match Field is a specific field from the packet carried by the network, to match against them. This field used to be; ingress port, packet header or even optionally metadata previously specified.

The Counters is used to update the number of packet matched against the Match Field.

And the Instructions, is the action to take when a match is found, to change the pipeline processing.

Additionally, the OpenFlow protocol enables to represent additional methods of forwarding using group entries to classify and manage groups of flows. These entries are: Group Identifier, Group Type, Counters and Action Buckets.

With the help of this protocol and the programmability of the switches, different policies have been applied in order to manage the flows coming from the devices and through the gateway. Additionally, with the information provided by the headers of the protocol, informative statistics have been obtained in order to extract an overview of the state of the network. All of this is managed by different modules within the controller as we could see in the following sections.

SDN Virtual Switches: OpenVSwitch

OpenVSwitch is the virtual switch selected to deploy and managed our virtual network. Its programmability and virtualization are key point for this choice. Also supports important features such as: IP-tunneling, VLAN creation, link monitoring, remote configuration, fine-grained QoS control, VM interface traffic policing, Multi-table forwarding pipeline with flow-caching engine, among others.

This switch supports both aforementioned versions of the OpenFlow protocol so can be programed to make specific actions with flow packets, following the OpenFlow pipeline as shown in Figure 47.

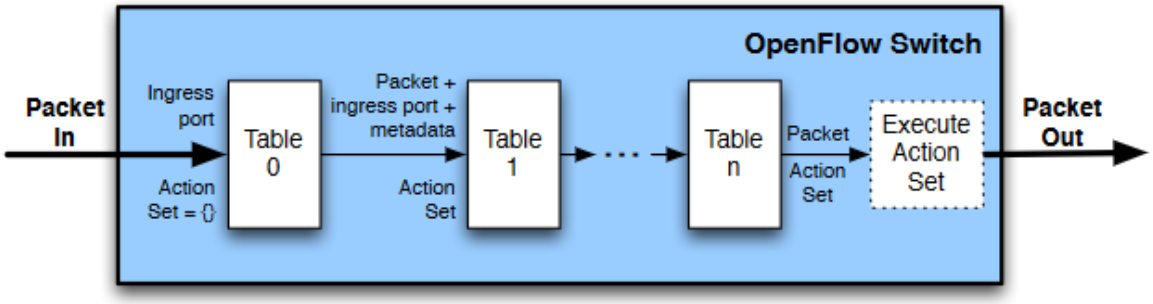

Figure 47: Packet Flow through the processing pipeline inside the virtual switch. Source: https://goo.gl/SoIPHl

This way, after the processing and take of decision carried out in the specific modules of the Controller, the adequate entry of the table is inserted in the switch so that when the packet of a flow arrives, the switch is already prepared to carry out the action.

SDN Controller: OpenDayLight

The OpenDaylight Controller is pure software and as a JVM it can be run on any OS and Metal as long as it supports Java.

On the Southbound it can support multiple protocols (as plugins), e.g. OpenFlow 1.0, OpenFlow 1.3, BGP-LS, etc. Of course, the one we use is OpenFLow but some other switches can be connected even if they use other protocols. These modules are linked dynamically into a Service Abstraction Layer (SAL). The SAL exposes services to which the modules of the northbound par are written on. The SAL figures out how to fulfill the requested service irrespective of the underlying protocol used between the Controller and the network devices. This provides investment protection to the Applications as the OpenFlow protocol can evolve over time. For the Controller to control devices in its domain it needs to know about the devices, their capabilities, reachability, etc. This information is stored and managed by the Topology Manager. The other components like ARP handler, Host Tracker, Device Manager and Switch Manager help in generating the topology database for the Topology Manager. All of these modules have been adapted to work with information from an IoT environment and with the specific IoT devices and platform communicating by the network and, additionally new features have been included to improve the performance (QoS) and offer new application at the top of the network.

The Controller exposes open Northbound APIs which are used by these applications. It supports the OSGi framework and bidirectional REST for the Northbound API. OSGi framework is used for applications that will run in the same address space as the Controller while the REST (Web based) API is used for apps that do not run in the same address space (or even the same machine) as the Controller. The business logic and algorithms reside in the apps. These apps use the Controller to gather network intelligence, runs its algorithm to do analytics and then use the Controller to orchestrate the new rules throughout the network.

The Controller has a built in GUI. The GUI is implemented as an application using the same Northbound API as would be available for any other user application.

Software Defined Radio

At the moment of writing of the deliverable, for the N2N solution, we have decided to use the following technology.

The Avnet PicoZed SDR is being utilized. It has a processor component, a Xilinx Z7035 Zynq®-7000 All Programmable SoC which combines a Kintex-7 FPGA with dual-core ARM Cortex-A9 processor. The software tunable analog radio element is provided by the Analog Devices AD9361 integrated RF Agile Transceiver. This is provided in a handheld form-factor and provides frequency-agile wideband receive and transmit paths in the 70 MHz to 6.0 GHz range, making it ideal for a broad range of fixed and mobile SDR applications.

QoS

From the QoS point of view, the Network Layer will provide an autonomous de-centralized resource reservation scheme for managed IEEE802.15.4e-TSCH networks. The Application Service Layer will provide an API to monitor (un)managed networks and coordinate reserved resources of the managed IEEE802.15.4e-TSCH networks. This API will be available for applications to reconfigure networks to their needs.

Network Layer: To achieve high levels of reliability on low-power lossy networks, IEEE has generated an amendment to the IEEE802.15.4 protocol allowing time scheduling and channel hoping over 2.4GHz band. Smart scheduling algorithms are to be devised and orchestrate all links of the managed network to prevent interference, collisions and reduce latency. In this way, latency and packet delivery ration can be improved alleviating bottleneck costs introduced by unmanaged parts of data packet paths.

RPL will be used as the chosen routing protocol of the managed network as the predominant choice in low-power lossy networks. It will also allow easier scheduling orchestration from a centralized entity. Any tree-based routing protocol could also be used e.g. CTP.

Application Service Layer: For interoperability purposes, CoAP and 6LoWPAN will be used for interaction between the coordinator & monitor and the underlying networks. The centralized monitor and coordinator should be able to:

- capture as accurate as possible the current status of the networks i.e. connectivity, routing topology, network performance metrics,
- correlate the current state of the network to the application performance metrics,
- devise adaptation/reconfiguration strategy for the managed networks,
- deploy reconfiguration strategy with CoAP commands.

3.2.3 Components

Software Defined Radio

The components necessary to connect the SDR modules to INTER-IoT are described in the D2D proposed solution, section 3.1. The SDR transmitter and receiver are described below.

Component	SDR Transmitter
Description	The SDR Transmitter will gather information from a device and output an RF waveform to transmit this data. It uses the following steps to achieve this goal.
Functionalities	• Virtual Network Port (TUN/TAP) – interface to host OS. • Packetize data. • Generate CRC for error detection. • Generate header. • Map header bits to required header I/Q constellation symbols. • Map payload bits to required payload I/Q constellation symbols – this may or may not be the same symbol set as the header. • OFDM carrier allocation – allocate data to sub-carriers, insert pilot carriers, append pilot symbols and sync word(s). • Inverse Fast Fourier Transform (IFFT) to translate from frequency domain to time domain. • Append the cyclic prefix – typically an additional FFT_length/4 time domain samples. • Agile tuneable software radio takes in digital complex baseband samples and outputs up-converted RF waveform.
Relation with other component	The SDR Transmitter will interact with the SDR Receiver.
Use Cases Involved	[40]
Requirements Involved	[17], [18], [170], [204]

Component	SDR Receiver
Description	The SDR Receiver will gather information from the SDR Transmitter and output an IP stream into the AN Module in the INTER-IoT gateway. It uses the following steps to achieve this goal.
Functionalities	• Agile tuneable software radio receiver takes in band-limited RF waveform and outputs down-converted complex baseband digital samples.

	• OFDM two-stage frequency synchronization algorithm – e.g. Schmidl and Cox. • Fine frequency correction. • Create two streams of data. Firstly, the header is extracted and channel estimation made before a buffered payload samples can be forwarded to the FFT for processing. • The header's Fast Fourier Transform (FFT) to convert from time domain to frequency domain. • The payload's Fast Fourier Transform (FFT) to convert from time domain to frequency domain. • Obtain channel estimates based on received pilot symbols / carriers. • Single-tap Frequency Domain Equalization (FDE) and course frequency correction. • Symbol de-mapping – hard decision decoding. Output is stream of bits. • Check CRC and accept/discard packets. • Virtual Network Port (TUN/TAP) – interface to host OS.
Relation with other component	The SDR Receiver will interact with the SDR Transmitter and the gateway via an AN Module.
Use Cases Involved	[40]
Requirements Involved	[17], [18], [170], [204]

Software Defined Network

Component	OF Connector
Description	It is an OpenFlow understanding plugin that communicates, by OF protocol, with all the switches that conforms our virtual network. Is the Bridge between the Controller and the nodes of the network.
Functionalities	Translates the messages coming down from the core of the Controller to OpenFlow encapsulated messages that can be understood by the switches. Also, several improvements can be added to this plugin as: Notifications

	Testing Collect Packets The versions supported of the OF protocol are the 1.0 and 1.3.
Relation with other component	All, as is the Bridge from which all packets from the network has to pass.
Use Cases Involved	[43], [55], [41]
Requirements Involved	[233], [231], [229], [207], [80], [93], [72], [57], [17]

Component	**Switch Manager**
Description	The Switch Manager API holds the details of the network element. As a network element is discovered, its attributes (e.g. what switch/router it is, SW version, capabilities, etc.) are stored in the database by the Switch Manager.
Functionalities	This Switch Manager stores information about each node of the network and its status, so you can use it later for other component.
Relation with other component	All
Use Cases Involved	[43], [55], [41]
Requirements Involved	[233], [231], [229], [207], [80], [93], [72], [57], [17], [16]

Component	**IoT Routing**
Description	This module is inside the Controller and performs the routing adequate algorithm to carry the packets.
Functionalities	In this module some headers of the packet are introduce to perform a routing algorithm previously configured and resolve the next hop in the network.
Relation with other component	OF Connector Switch Manager

	Topology Host Tracking Network Slicing
Use Cases Involved	[43], [55], [41]
Requirements Involved	[17], [233], [231], [229], [95], [89]

Component	**IoT Host Tracking**
Description	Module in charge of handle the information from a host, including the address, the position in the network, etc.
Functionalities	Track the location of the host relatively to the SDN network. Including: Host address Attachment point (link) to a node/switch
Relation with other component	OF Connector Switch Manager Topology Storage
Use Cases Involved	[43], [55], [41]
Requirements Involved	[233], [231], [229], [80], [57], [11], [45], [18], [17]

Component	**Statistics**
Description	This module storage and provides information about the number of packets analyzed through the Controller.
Functionalities	Depending of the field can give the number of packets attending at some filters. Statistics service will export API to be able to collect statistics at least per: Flow Node Connector (port)

	Queue
Relation with other component	OF Connector Topology Switch Manager
Use Cases Involved	[55], [41]
Requirements Involved	[233], [231], [226], [57]

Component	**Storage**
Description	Additionally to the storage of the switch state, this is a module to save the information about statistics, topologies, direction, and other data related with the network.
Functionalities	Keeps the information of the network updated and stored for the other modules to access them.
Relation with other component	Topology Host Tracking Routing
Use Cases Involved	[43], [55], [41]
Requirements Involved	[17], [28], [57], [95], [229], [231], [233], [232]

Component	**Topology Discovery**
Description	Topology Service is a set of services that allow conveying topology information like a new node a new link has been discovered and so on.
Functionalities	Keeps tracking of the distribution of the nodes in the network and its links to each other, also some additional information as the bandwidth.
Relation with other component	OF Connector Switch Manager Host Tracking

Use Cases Involved	[43], [55], [41]
Requirements Involved	[233], [231], [229], [226], [207], [204], [80], [57], [17], [16]

3.2.4 Use Cases

Software Defined Radio

Use case	Support to non-standard access networks
Use Case ID	#40
Description	There is a class of sensors/actuators that use non-standard RF communication links. The reasons why this class of devices use non-standard RF links include the following: • to improve the link range; • to allow interoperability with legacy systems; • to enhance the resilience to jamming and other kinds or interference; • to enhance link security. The INTER-IoT platform provides access to this class of devices, thus allowing the INTER-IoT user community to take advantage of the services provided by these sensors/actuators. To accomplish this the INTER-IoT GW comprises a Bridge facility to perform the translation of the proprietary link into a standard IP based protocol usable by the INTER-IoT GW (e.g. Ethernet, WiFi).
Objectives	Allow sensors/actuators that use proprietary RF links to be accessible to the INTER-IoT community.
Components Involved	SDR Transmitter. SDR Receiver.
Requirements Involved	[39], [93], [11], [16]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-779

Software Defined Network

Use case	SDN communications: functions virtualization and central management
Use Case ID	#41
Description	The implementation and use of the SDN paradigm to speed up IoT connections and centralize the management.
Objectives	The virtual network could be manage from a central point, using the API access to request topologies, statistics, historical, etc.
Components Involved	The Virtual Switches The OF Connector The Statistics Module. The Host Tracking Module. The Topology Discovery Module.
Requirements Involved	[231], [233], [229], [226], [57]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-772

Use case	SDN communications: traffic routing
Use Case ID	#55
Description	The implementation and use of the SDN paradigm to prioritize data flows using traffic engineering, having a general overview of the whole network at any time.
Objectives	The data flows will travel through the software defined network from the gateway to the IoT platform in a secure manner following the defined policies.
Components Involved	The Virtual Switches The OF Connector. The Host Tracking module. The Topology Discovery Module. The IoT Routing Module.
Requirements Involved	[233], [207], [204], [80], [78], [89], [219], [17], [16]

Use case link	http://jira.inter-iot.eu/browse/INTERIOT-773

Use case	**Offloading workflow management**
Use Case ID	#43
Description	Discharging data traffic from more than one access network, simultaneously.
Objectives	Improve the speed of uploading or discharging data traffic from the device using more than one access network technology.
Components Involved	The Virtual Switches The OF Connector. The Host Tracking Module. The Topology Discovery Module. The IoT Routing Module. The Switch Management Module.
Requirements Involved	[17], [57], [228], [227],[233]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-774

3.3 MW2MW proposed solution

Interoperability at the middleware layer is achieved through establishment of an abstraction layer and subsequent attachment of all platforms to it. These attachments are established using bridges abstraction layer. This way we avoid the need to interconnect all platforms among themselves, instead connecting them directly to the abstraction layer and providing a mechanism for their communication within this layer.

Communication at the middleware layer is based on a message broker, which is accessed in every communication performed in MW2MW. This allows both complete decoupling between components and isolation of the communication responsibility in a single element, which in turn makes profiling, scaling and adaptation to enterprise infrastructures easier. Furthermore, the broker is accessed through a general API that exposes basic common operations (message pub/sub, topic creation, basic resources management...), enabling interchangeability of the actual implementation of the message broker.

Data model, used in MW2MW, is based on the ontological reference model of meta-data developed in INTER-IoT. It includes core concepts, shared between IoT platforms, that have been identified and standardized in ontologies, such as SSN (Semantic Sensor Network ontology). Using one

common model for all internal MW2MW components improves efficiency of internal data transfer, as well as allows components to make assumptions about structure and content of data, so that rich functionalities specific to IoT domain can be implemented and offered in one common data model.

There is, however, no requirement of compliance with the MW2MW data model placed upon platforms that use it. The models of platforms participating in communication through MW2MW are lifted to ontologies and semantically translated in IPSM (see DS2DS section). As a result, the commonalities between data models of IoT platforms can be expressed in an unified way (i.e. in a common model), despite the possibility of having different semantics.

3.3.1 Architecture

In designing the architecture for the Middleware integration, as shown in Figure 48, we focused on the extensibility and scalability of capabilities and features. To that end, we logically separated components that make up the communication and flow control, middleware services and bridges to middleware platforms. A message broker component is not shown on the figure for sake of simplicity, but it is represented by communication streams, shown in bold green. Although the system contains an abstraction layer for the queuing mechanism, we are using Apache Kafka for our deployments thus supporting a distributed streaming platform that implements the reactive streams approach.

In the communication part, the API Request Manager is responsible for handling requests received from the API proxy, which includes: bookkeeping of active sessions and their respective callbacks; forwarding requests to the Platform Request Manager for further processing; and providing feedback to the caller. The Platform Request Manager prepares and sends requests to specific platforms through bridges, using already established permanent data streams, which it creates during startup with the help of Data Flow Manager, or it creates new data streams. All data streams that go south, from the Platform Request Manager to bridges, go through permanent data streams, which can be either routed through IPSM (when needing ontological and/or semantical translation, as decided by consulting Platform Registry & Capabilities), or bypassing it and connecting directly to the bridges (thus eliminating the overhead). All data streams that go north, from the bridges to the Platform Request Manager, need to be created as needed.

During request pre-processing the Platform Request Manager is potentially assisted by some middleware services, such as routing or the device registry. It sends requests to underlying platforms as/when needed. The Data Flow Manager acts as orchestrator of data flows from the platforms (bridges) to the original caller, utilizing already established permanent data streams or creating new ones and ensuring that all intermediaries are included in the path. Finally, The Message Queue, not shown in the diagram, only receives and provides the messages to the corresponding components, including ad-hoc temporary topics for single requests, and fixed platform channels.

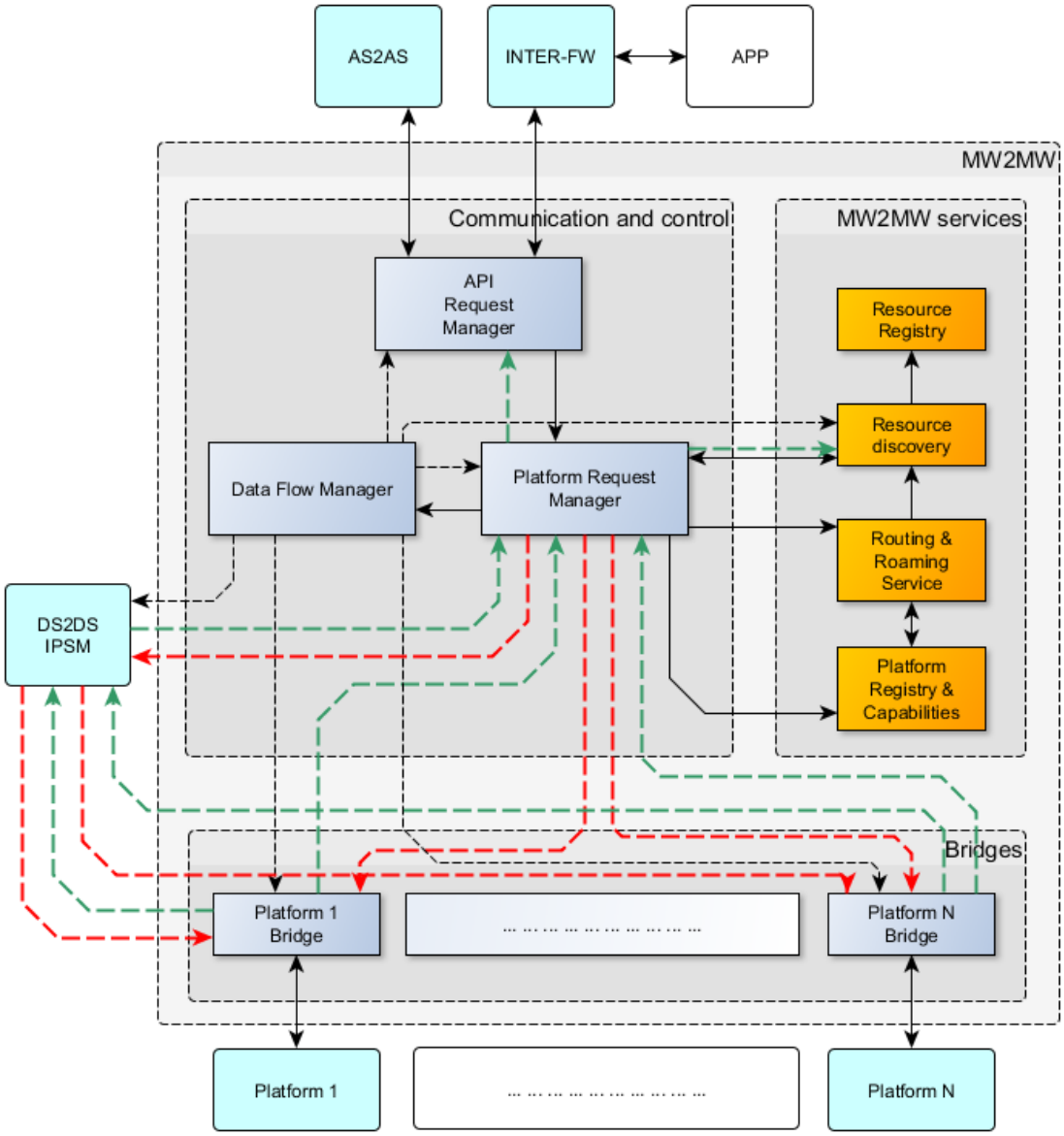

Figure 48: MW2MW architecture overview (green arrows: data streams from platform to upper layers; red arrows: permanent data streams from the Platform Request Manager to the Bridges - used for platform reqests; dashed black arrows: stream orchestration; black arrows: API calls – may be implemented through streams as well).

South from the Communication and Control block, the Bridges manage the communication with the underlying platforms by translating requests and answers from and into Messages for the queue. Different bridges might need to use HTTP, REST, sockets or other technologies to talk to the platforms, but these will be translated northwards into messages. They also pass the message content to the Semantic Mediator (a service external to MW2MW), which will allow for ontological and format translation between the platforms and a common language.

In the services group of components, the most important are the Platform Registry and Capabilities, that contains the information of all connected Platforms including their type and service capabilities, the Resource Discovery that creates requests to obtain the necessary information from the platforms, and the Resource Registry, that contains a list of resources (e.g. devices) and their properties that can be quickly consulted. In the second phase, the Routing and Roaming Service will be expected

to allow the communication with a particular device independently of the platform it is currently connected to, while Authentication and Accountability (not shown) would provide services for the security and monitoring of all the actions.

3.3.2 Components

Component	API Request Manager
Description	It handles requests, received from the API proxy.
Functionalities	Bookkeeping of active sessions and their respective callbacks through usage of unique call IDs, forwarding requests to the Platform Request Manager for further processing, providing feedback to the API caller.
Relation with other component	Platform Request Manager, Data Flow Manager
Use Cases Involved	[26], [25], [23], [65]
Requirements Involved	[2], [6], [13], [25], [43], [57], [72], [75], [89], [179], [201], [234], [235], [236], [237], [238], [255], [281], [282], [283]

Component	Data Flow Manager
Description	It orchestrates data flows from the platforms (bridges) to the original caller.
Functionalities	Creation of a dataflow, associated with a unique call ID. Creation of permanent data flows between bridges and the Platform Request Manager, going south, at middleware startup, routing either through IPSM or bypassing it. Creation of data flows, which go north, when needed.
Relation with other component	Platform Request Manager, API Request Manager, Bridges
Use Cases Involved	[26], [25], [23], [65]
Requirements Involved	[2], [6], [13], [25], [43], [57], [72], [75], [89], [179], [201], [234], [235], [236], [237], [238], [255], [281], [282], [283]

Component	Platform Request Manager

Description	It arranges and manages flow of requests to underlying platforms.
Functionalities	Obtainment of the list of available platforms, creation of a unique flow ID, routing of the request flow to the underlying platforms. Creation of permanent data streams going south, routed either through DS2DS IPSM or bypassing it. Creation of data streams going north when needed.
Relation with other component	Data Flow Manager, API Requests Manager, Resource Discovery, Device Registry, Routing & Roaming Service**,** Platform Registry and Capabilities, DS2DS IPSM
Use Cases Involved	[26], [25], [23], [65]
Requirements Involved	[2], [6], [13], [25], [43], [57], [72], [75], [89], [179], [201], [234], [235], [236], [237], [238], [255], [281], [282], [283]

Component	**Resource Discovery**
Description	Resource discovery is a module to find resources based on a query that specifies the desired results. It creates requests to obtain the necessary information from the IoT platforms or it looks up the information from the Resource Registry.
Functionalities	The component includes methods to allow users to send a query to obtain the list of devices throughout the integrated platform it has access to, which comply with a search query or filter, or to consult the Resource Registry. It forwards the request to all relevant IoT Platforms.
Relation with other component	Platform Request Manager, Device Registry
Use Cases Involved	[25]
Requirements Involved	[2], [6], [13], [17], [43], [57], [72], [179], [234], [235], [236], [237], [238], [255]

Component	Resource Registry
Description	It contains a list of devices and their properties that can be quickly consulted when needed.
Functionalities	Any new device can be added to the list of registered devices and it should have a unique identification. However, it is not guaranteed to be complete and should thus be complemented with queries from Resource Discovery to actual platforms.
Relation with other component	[25]
Use Cases Involved	[60] Device Registry
Requirements Involved	[2], [6], [13], [17], [43], [57], [72], [179], [234], [235], [236], [237], [238], [255]

Component	Routing and Roaming Service
Description	It allows the communication with a particular device independently of the platform it is currently connected to. When a device access to the facilities of a different company, it is connected transparently to the platform.
Functionalities	Method to automatically register a device in different platforms.
Relation with other component	Platform Request Manager, Platform Registry and Capabilities, Device Registry
Use Cases Involved	N/A
Requirements Involved	[2], [6], [13], [17], [18], [43], [57], [72], [179], [234], [235], [236], [237], [238], [255]

Component	Platform Registry and Capabilities
Description	It contains the information of all connected Platforms including their type and service capabilities. A unique ID is assigned to each registered platform

Functionalities	The component includes methods to (1) add (register) a platform (along with its supported services) to the registry, (2) update supported services of a given platform, (3) remove (unregister) a platform from the registry, (4) get the information (supported services) of a given platform, (5) generate a unique ID upon the first registration of a new connected platform, (6) return the list of all the connected (registered) platforms
Relation with other component	Platform Request Manager, Routing & Roaming Service
Use Cases Involved	[26], [25], [23], [65]
Requirements Involved	[2], [6], [13], [25], [43], [57], [72], [75], [89], [179], [201], [234], [235], [236], [237], [238], [255], [281], [282], [283]

Component	**Bridge (generic interface)**
Description	Bridge manages the communication with the underlying platforms by translating requests and answers from and into Messages for the queue. The generic interface provides a structured template to easily develop new bridges.
Functionalities	Translation of requests from and into messages for the queue. Code structure and abstraction layer for platform-specific implementations.
Relation with other component	Data Flow Manager, Platform-specific implementations(Bridge - D2D, Bridge - FIWARE, Bridge - Azure, Bridge – OpenIoT, Bridge - oM2M)
Use Cases Involved	[26], [25], [23], [65]
Requirements Involved	[2], [6], [13], [25], [43], [57], [72], [75], [89], [179], [201], [234], [235], [236], [237], [238], [255], [281], [282], [283]

Component	**Bridge - D2D**
Description	Implementation of a bridge for the D2D platform.
Functionalities	This component provides the functionalities to manage the communication with the D2D layer

Relation with other component	DS2DS IPSM, Data Flow Manager
Use Cases Involved	[26], [25], [23], [65]
Requirements Involved	Same as the Bridge component

Component	**Bridge - FI WARE**
Description	Implementation of bridge for the FIWARE platform.
Functionalities	This component provides the functionalities to manage the communication with the FI WARE platform
Relation with other component	Bridge - D2D, Data Flow Manager
Use Cases Involved	[26], [25], [23], [65]
Requirements Involved	Same as the Bridge component

Component	**Bridge - Azure**
Description	Implementation of bridge for the Azure platform.
Functionalities	This component provides the functionalities to manage the communication with the Azure platform
Relation with other component	Bridge - D2D, Data Flow Manager
Use Cases Involved	[26], [25], [23], [65]
Requirements Involved	Same as the Bridge component

Component	**Bridge - OpenIoT**

Description	Implementation of a bridge for the OpenIoT platform.
Functionalities	This component provides the functionalities to manage the communication with the Open IoT platform
Relation with other component	Bridge - D2D, Data Flow Manager
Use Cases Involved	[26], [25], [23], [65]
Requirements Involved	Same as the Bridge component

Component	**Bridge - oM2M**
Description	Implementation of a bridge for the oM2M platform.
Functionalities	This component provides the functionalities to manage the communication with the Open M2M platform
Relation with other component	Bridge - D2D, Data Flow Manager
Use Cases Involved	[26], [25], [23], [65]
Requirements Involved	Same as the Bridge component

Component	**Queue/Streams Abstraction**
Description	It receives and provides the messages to the corresponding components.
Functionalities	Implementations vary from ad-hoc temporary topics for single requests to fixed platform channels.
Relation with other component	Bridge, Data Flow Manager, DS2DS - IPSM, Platform Request Manager
Use Cases Involved	[26], [25], [23], [65]
Requirements Involved	Same as the Bridge component

3.3.3 Use cases

In the following figures green arrows represent messaging through Message Broker.

Use case	Subscribe to MW2MW event messages
Use Case ID	#26
Description	Subscribers shall be able to subscribe to topics, in order to be informed of any new information (reading, device update, etc.) related to that defined topic.
Objectives	A subscriber will be able to create a subscription through the system, which will allow it to receive as soon as possible news from the publisher about any event relevant to the desired topic.
Components Involved	API Request Manager, Platform Request Manager
Requirements Involved	[2], [6], [13], [72], [75], [179], [201], [234], [235], [236], [237], [255], [281], [282]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-804

MW01 MW2MW Subscription request to topic

View online: http://tinyurl.com/mw01v01

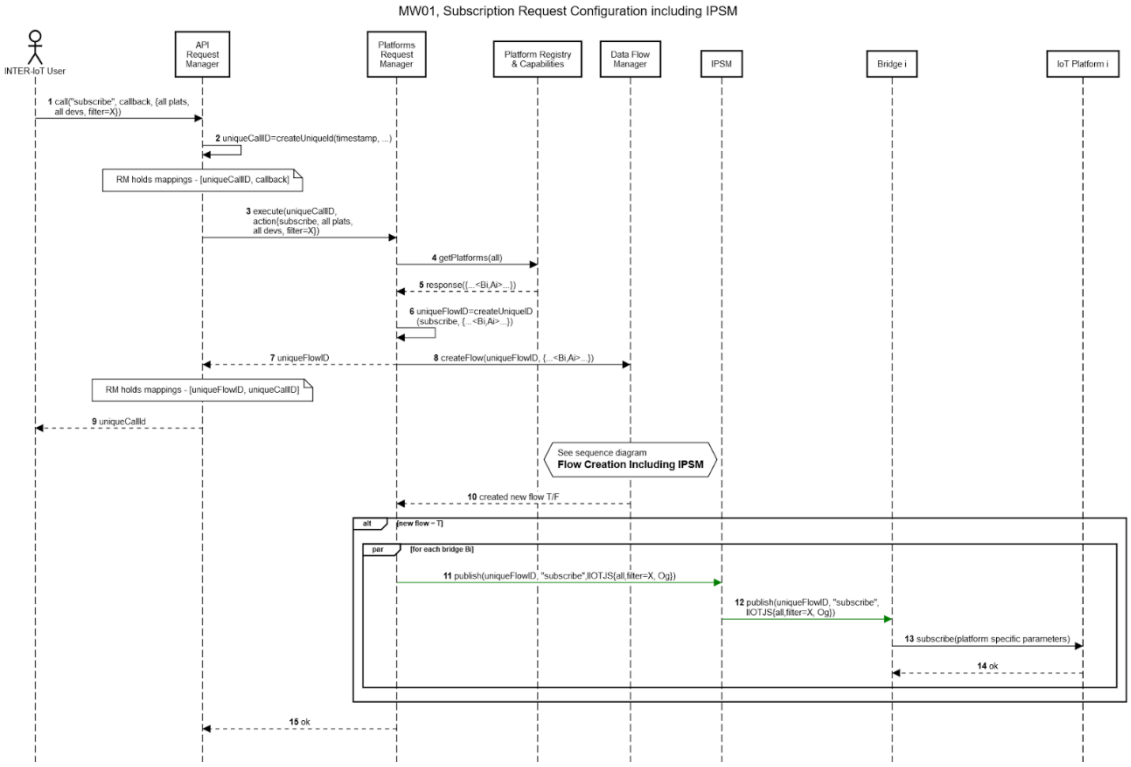

Figure 49: MW01, Subscription Request Configuration including IPSM.

Step 1: The API Request Manager (RM) receives a request for subscription from an external actor.

Step 2-3: The RM creates a unique ID for the call (we can think of it as “session”) and keeps the mapping between unique IDs and callbacks. It then sends a subscribe request to the Platforms Request Manager (PRM). The request contains all platforms, the list of all devices and a filter in the filtering format of INTER-IoT.

Step 4: The PRM asks all platforms by contacting Platform Registry and Capabilities (PRC) component.

Step 5: The list of authorized platforms is returned. The answer is composed by a list of references to bridges and corresponding semantic descriptions.

Step 6: A unique identifier for the data flow is created at this point. The identifier is a function of platforms and filters. This means, that for a repeated identical request from other clients we do not duplicate communications streams and requests to platforms.

Step 7-10: The data flow is created, if needed (see sequence diagram “Flow Creation Including IPSM”)

Step 11-15: For each bridge Bi, if the Data Flow Manager (DFM) created a new flow, then PRM sends a “startTopicFlow” request to Bi with the flow ID, all devices and query filter as parameters. Bi creates a publisher for Fi. Bi translates from the common filter to the specific filtering syntax of the platform and subscribes with the indicated conditions to the platform Pi.

MW02 New info pushed to topic (with semantic mediation)

View online: http://tinyurl.com/mw02v02

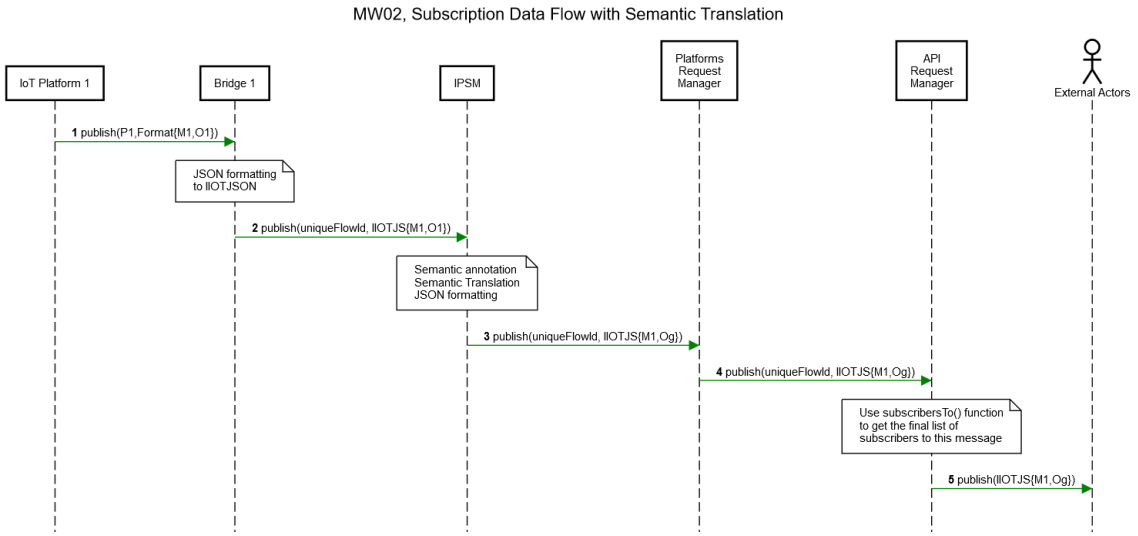

Figure 50: MW02, Subscription Data Flow with Semantic Translation.

A resource in platform 1 (P1) produces some data to which an external actor (an INTER-IoT user, actually) is subscribed. An existing subscription in a platform means that in its corresponding bridge there is, at least, two topics, one for receiving the notifications from the platform and one to publish them to IPSM. At the same time, the IPSM must have, at least, one topic to publish back, once the data is semantically *translated* to the PRM. Finally, it exists another topic exclusive of that subscription for the communication PRM <=> RM. The RM stores a list of callback for that topic ID.

Step 1: P1 sends a notification to the callback mechanism provided by Bridge 1 (B1) with the data M1. Bridge 1 gets the the ID of the device which has generated the trigger. The data M1 observed by B1 is in Ontology 1 (O1).

Bridge 1 formats M1 in JSON as it came from the platform and puts it as a payload in a new json file. B1 adds identification and metadata to that file: platform id (*P1*), device ID, timestamp of the current operation (which will be the reference timestamp for the event in INTER-IoT realm) and a unique ID for the message (for storage and processing purposes).

Step 2: B1 publishes the message to the IPSM through the topic existing for that purpose. IPSM is subscribed to that topic.

IPSM performs the semantic annotation and the semantic translation of the message. It also formats the result in a JSON file with the INTER-IoT metadata.

Step 3: IPSM sends the message to PRM through the topic for that purpose.

Step 4: PRM forwards the message to the API Request Manager (RM).

Step 5: API RM gets the list of callback for the topic and send them the notification.

MW08 New info pushed to topic (without semantic mediation)

View online: http://tinyurl.com/mw08v01

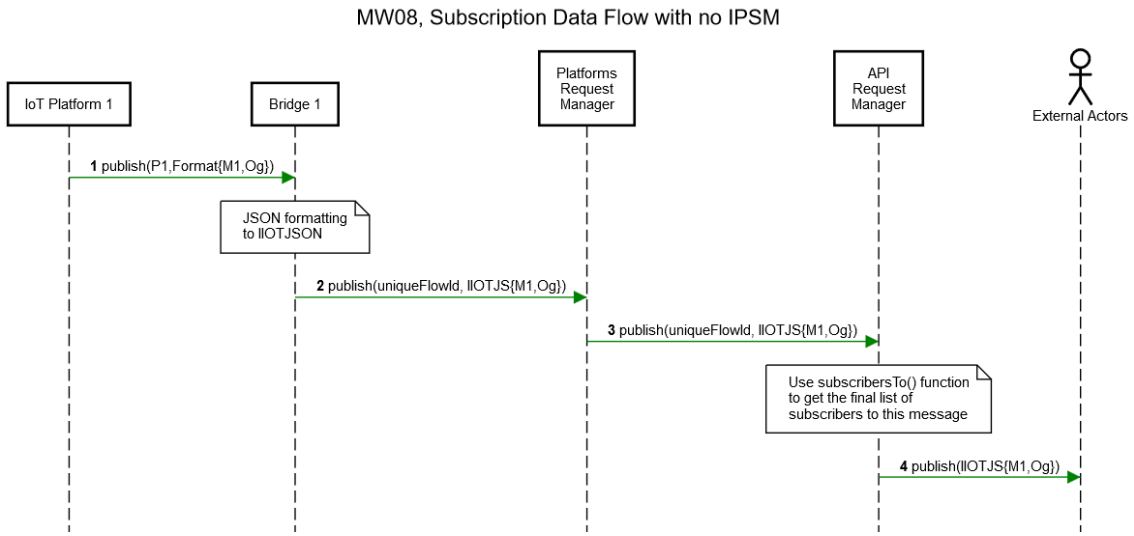

Figure 51: MW08, Subscription Data Flow with no IPSM

Step 1: P1 sends a notification to the callback mechanism provided by Bridge 1 (B1) with the data M1. Bridge 1 gets the the ID of the device which has generated the trigger. The data M1 observed by B1 is in Ontology 1 (O1).

Bridge 1 formats M1 in json as it came from the platform and puts it as a payload in a new json file. B1 adds identification and metadata to that file: platform id (*P1*), device id, timestamp of the current operation (which will be the reference timestamp for the event in INTER-IoT realm) and a unique id for the message (for storage and processing purposes).

Step 2: B1 sends the message to PRM through the topic for that purpose.

Step 3: PRM forwards the message to the API Request Manager (RM).

Step 4: API RM gets the list of callback for the topic and send them the notification.

MW05 Unsubscribe from topic

View online: http://tinyurl.com/mw05v02

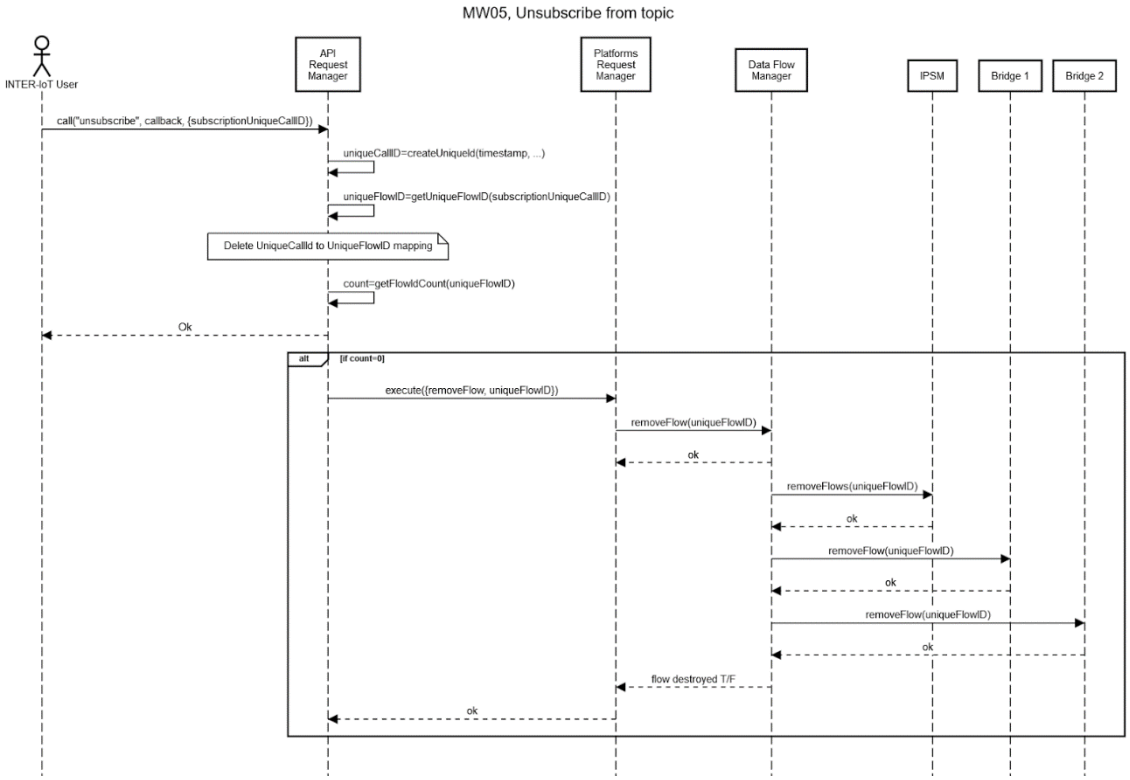

Figure 52: MW05, Unsubscribe from topic.

Step 1: Actor calls API with reference for call back and subscriptionUniqueCallID

Step 2: API forwards to RM

Step 3: RM creates uniqueCallID

Step 4: RM creates uniqueFlowID from previous subscriptionUniqueCallID

Step 5: RM deletes UniqueCallID to UniqueFlowID mapping

Step 6: count of uniqueFlowID made

Step 7: If count > 1, ok

Step 8: If count = 0,

- remove uniqueFlowID from PRM
- remove uniqueFlowID from DFM
- remove uniqueFlowID from IPSM
- remove uniqueFlowID from Bridges

MW06 Flow creation

View online: http://tinyurl.com/mw06v01

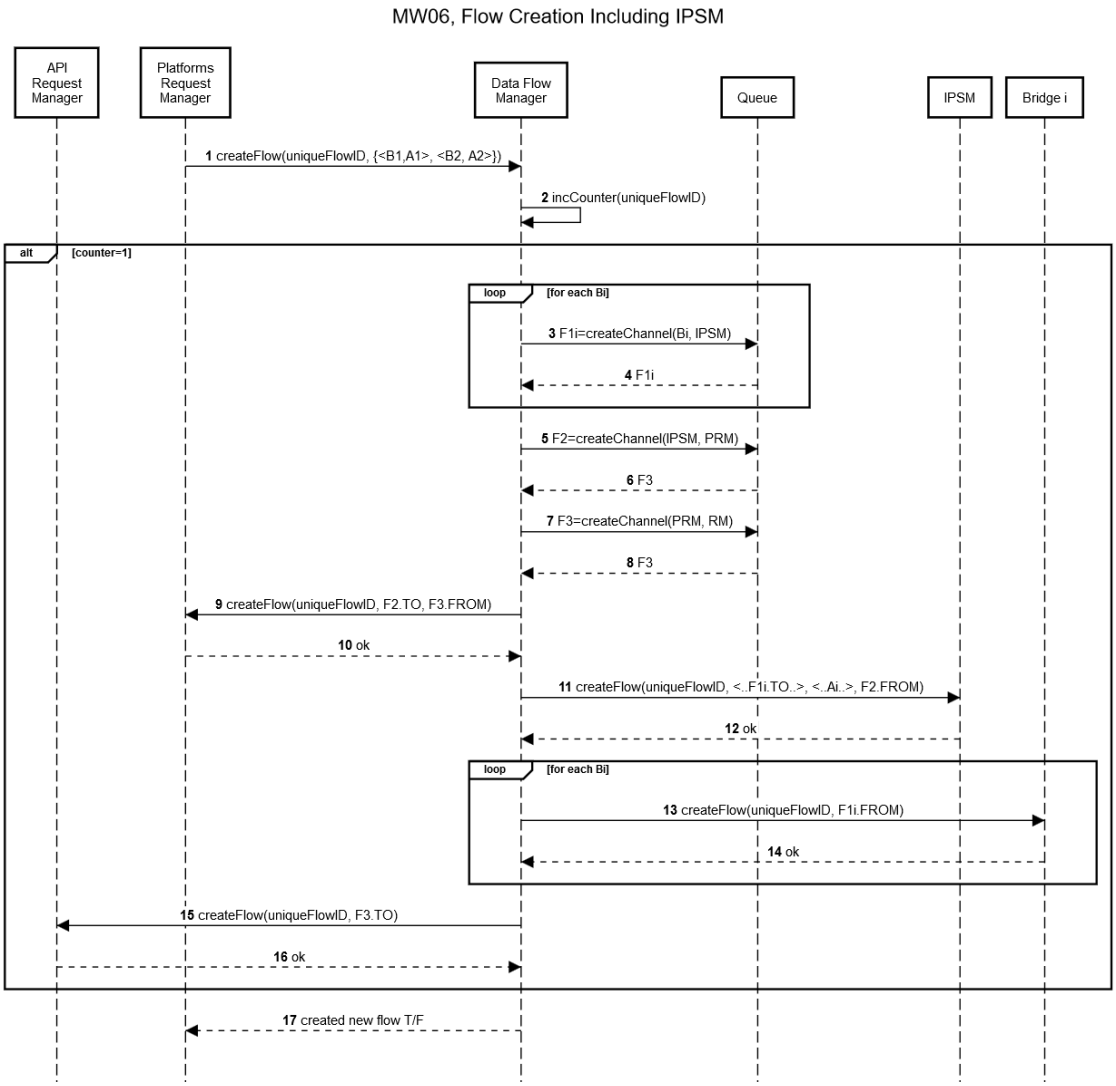

Figure 53: MW06, Flow Creation Including IPSM.

Step 1: API request data from bridge. PRM receives request

Step 2: PRM sets up communication with DFM

Step 3: DFM creates communication ID

Step 4: start loop

For each Bi create a channel from the queue

Queue sent ack, channel open

Step 5: Set up channel between IPSM and PRM -> ack

Step 6: Set up channel between PRM and RM -> ack

Step 7: PRM create flow: IPSM, PRM, RM

Step 8: DFM connect to IPSM -> ok

Step 9: start loop

For each Bi create a flow to the bridge

Bridge sent ack, channel open for each Bi

Step 10: DFM create flow to API RM -> ok

Step 11: DFM to PRM new flow created

Use case	MW2MW resource discovery
Use Case ID	#25
Description	An MW2MW user will be able to obtain the list of devices throughout the integrated platform it has access to, which comply with a search query or filter.
Objectives	To let application and services to discover, whenever possible, what devices, and with which properties, are available to the system.
Components Involved	API Request Manager, Platform Request Manager, Resource discovery, Resource Registry, Platform Registry & Capabilities, Data Flow Manager, IPSM, Bridges, Platforms
Requirements Involved	[2], [6], [13], [17], [43], [57], [72], [179], [234], [235], [236], [237], [238], [255]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-805

MW04 Resource discovery

View online: http://tinyurl.com/mw04v01

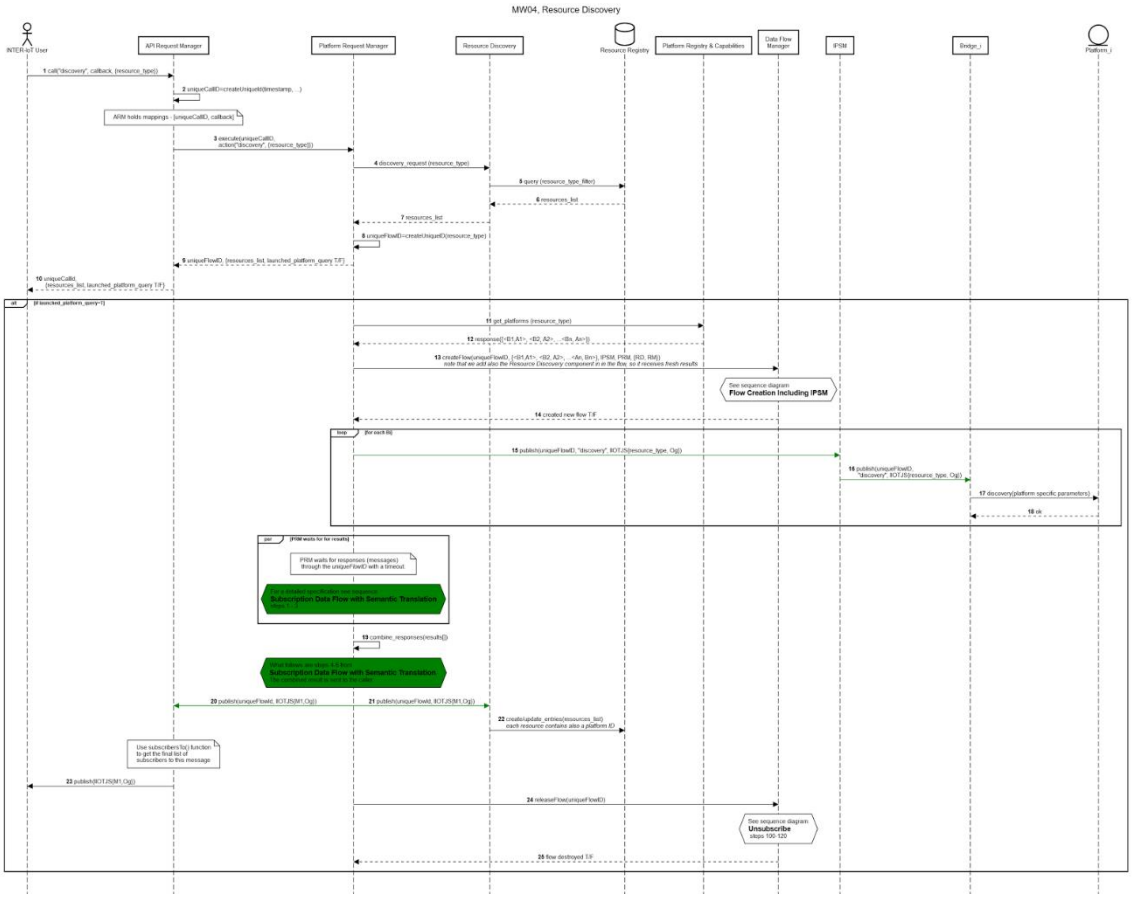

Figure 54: MW04, Resource discovery.

Step 1: Actor calls API ‘s method for resource discovery with references for callback.

Step 2: The RM creates a unique id for the call (we can think of it as “session”) and keeps the mapping between unique IDs and callbacks.

Step 3: The RM calls PRM with the instruction to execute the action for resource discovery.

Step 4-5: The PRM forwards the discovery request to RD.

Step 6-7: The list of authorized platforms is returned. The answer is composed by a list of references to bridges and corresponding semantic descriptions.

Step 8-10: A unique identifier for the data flow is created at this point. The identifier is a function of platforms and filters. This means, that for a repeated identical request from other clients we do not duplicate communications streams and requests to platforms. PRM creates a data flow using this unique identifier and returns resources list to actor.

If PRM decides to launch a remote resource discovery:

Step 11-12: PRM retrieves registered platforms from PRC

Step 13-14: PRM asks DFM to create a new Flow

Step 15-18: PRM sends discovery query message to relevant Bridges (through IPSM), with reference to the above created Flow. Each Bridge forwards (async) discovery request to Platform and each Platform responds to its Bridge.

Step 19: PRM receives responses as they arrive (through the Flow), and combines them into one

Step 20-21: PRM publish combined results to RM and RD

Step 22: RD creates (or updates) discovered resources into RR

Step 23: RM publish combined resources list to Actor

Step 24-25: RPM calls DFM to close (release) the Flow

Use case	Request query to MW2MW
Use Case ID	#23
Description	An MW2MW user will be able to request a list of values from [a set of] devices of the platforms it has access to with conditions regarding the geographical, temporal and other conditions as defined by the given set of filters.
Objectives	To let the inquirer to obtain a list of values of interest from a subset of devices.
Components Involved	API Request Manager, Platform Request Manager, DS2DS IPSM, Bridge
Requirements Involved	[2], [6], [13], [72], [179], [234], [235], [236], [237], [255]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-806

MW03 Query

View online: http://tinyurl.com/mw03v01

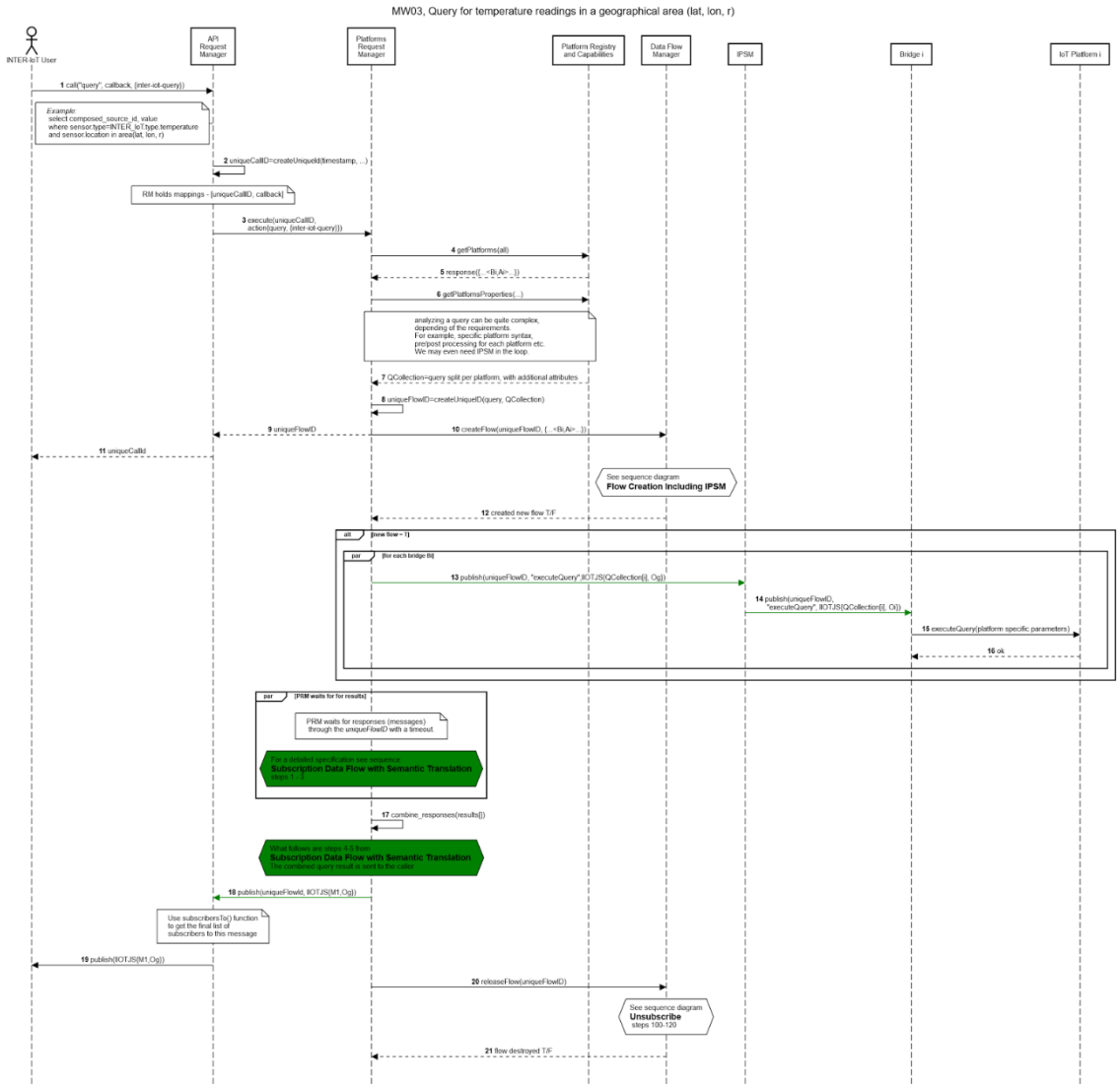

Figure 55: MW03, Query for temperature readings in a geographical area (lat, lon, r)

Step 1: The API Request Manager (RM) receives a request for a query from an external actor.

Steps 2-3: The RM creates a unique id for the call (we can think of it as “session”) and keeps the mapping between unique IDs and callbacks. It then sends a query request to the Platforms Request Manager (PRM). The request contains the query to be executed. PRM uses MW service components in further steps in order to interpret and execute the query.

Step 4: The PRM asks for the list of platforms according to the request by contacting Platform Registry and Capabilities (PRC) component.

Step 5: The list of authorized platforms is returned. The answer is composed by a list of references to bridges and corresponding semantic descriptions.

Steps 6-7: Furthermore, certain platform properties are requested in order to correctly parse and execute the query.

Step 8: A unique identifier for the data flow is created at this point. The identifier is a function of the query content. This means, that for a repeated identical query from other clients we do not duplicate

communications streams and requests to platforms. Although, for query execution it is highly unlikely that the same query would be repeated twice in a short timespan.

Steps 9-12: The data flow is created, if needed (see sequence diagram “Flow Creation Including IPSM”).

Steps 13-16: For each bridge Bi, if the Data Flow Manager (DFM) created a new flow Fi, then PRM sends a “executeQuery” request to Bi with the flow ID and the query adapted to a specific platform. Bi creates a publisher for Fi. Bi translates from the common query to the specific syntax of the platform and subscribes with the indicated conditions to the platform Pi.

Step 27: PRM waits for results of queries b listening to the created topic (see **Subscription data flow** sequence diagram). It waits until it gets all results or reaches a specific timeout.

Steps 18-19: PRM Publishes back the combined result (see **Subscription data flow** sequence diagram).

Steps 20-21: PRM shuts down the data flow (see **Unsubscribe** sequence diagram).

Use case	MW2MW sends information to a device (sensor or actuator)
Use Case ID	#65
Description	The Midleware can access to a device (sensor or actuator) and send it orders or actions (e.g. change the configuration, activate/deactivate).
Objectives	In order to manage a device, it is necessary to send orders to it, besides receiving data.
Components Involved	API Request Manager, Platform Request Manager, DS2DS IPSM, Bridge
Requirements Involved	[2], [6], [13], [25], [89], [179], [234], [235], [236], [237], [255], [283]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-837

MW07 MW2MW sends information to device(s)

View online: http://tinyurl.com/mw07v01

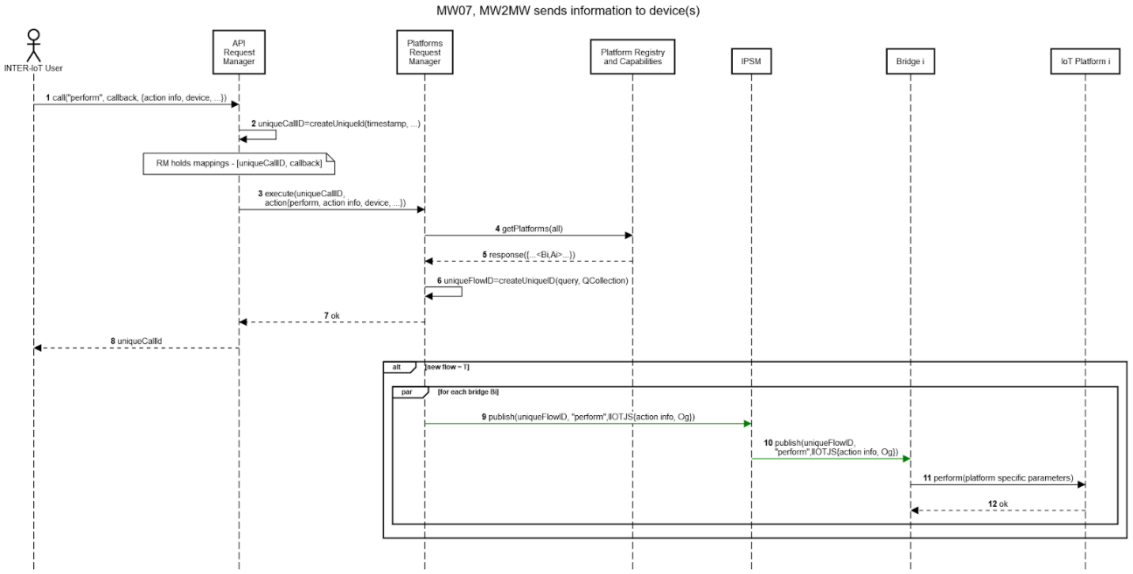

Figure 56: MW07, MW2MW sends information to device(s)

Step 1: The API Request Manager (RM) receives a request to perform an action upon the device from an external actor.

Steps 2-3: The RM creates a unique id for the call (we can think of it as “session”) and keeps the mapping between unique IDs and callbacks. It then sends an action request to the Platforms Request Manager (PRM). The request contains the information, which is to be sent to the device. PRM uses MW service components in further steps in order to interpret and execute the query.

Step 4: The PRM asks for the list of platforms according to the request by contacting Platform Registry and Capabilities (PRC) component.

Step 5: The list of authorized platforms is returned. The answer is composed by a list of references to bridges and corresponding semantic descriptions.

Step 6-8: A unique identifier for the data flow is created at this point. The identifier is a function of the action content. This means, that for repeated identical actions from other clients we do not duplicate communications streams and requests to platforms.

Steps 9-12: For each bridge Bi, if the Data Flow Manager (DFM) created a new flow Fi, then PRM sends a “publish” request to Bi with the flow ID and the information to be pushed onto the platform, adapted to that specific platform.

3.4 AS2AS proposed solution

The goal of this task is to achieve through INTER-IoT the identification and access to native services and applications from different IoT platforms that are under the INTER-IoT research. Then, get the specifications about how to access to it. As well, to offer a description or detailed information about this services and applications. Offering the possibility to obtain a list of services throughout the integrated platform it has access to, which comply with a search query or filter.

Finally, the objective of this task is to make interoperable application services furnished by heterogeneous IoT platforms. Making it possible to reuse and exchange heterogeneous services

from the different IoT platforms and allow application developers to produce new added value services from existing IoT services.

For that reason, the approach proposed for the AS2AS level is based on:

- Access APIs provided by IoT platforms: Almost all IoT platforms provide a public API to access their services. The APIs are usually based on RESTful principles. But in a case where the IoT platforms do not provide this RESTful API, it will be necessary to create a solution to access their services.
- Service Catalogue: Register services/applications with their description or detailed information to make them discoverable. With our solution based on Service Catalogue, AS2AS will be able to use the same metadata annotations (then creating a point of interoperability) and have a uniform data catalogue.
- Service Discovery: Creates requests to the Service Catalogue to obtain the necessary services from the IoT Platforms.
- Service Composition: Encompasses all those processes that create added-value services, called composite services, from existing services.

Using the following solutions to achieve the main objective:

- Orchestrator: Service composition is done from a centralized perspective, expressing how the composition has to act in order to integrate components.
- Data flows: In a composition, the input of one component is typically produced by another component output.
- Flow Based Programming (FBP): FBP describes a graph of nodes, which exchange messages containing data via the edges. The edges are defined outside the nodes, in others words nodes have no control on where the data comes from and where it goes to.
- Modeller: Allows to drag IoT services and internal services from the catalogue and connect them in a graphical environment and then create flows.

Finally, AS2AS has to provide an interoperability API (INTER-API) and integration toolbox. With this tool AS2AS users and organizations can develop new applications over the integration of their existing heterogeneous IoT infrastructures.

3.4.1 Architecture

The architecture for the Application and Services integration, as shown in Figure 57, shows the components that will perform the functionalities and goals that have been listed in the previous section. It also presents the communication that exists between these components.

Service Catalogue and Service Discovery are in charge of storing and managing the information and description of the services available on IoT Platforms. To interact with these components, we have the modules that are part of the graphical environment (GUI): Modeller and Register Client.

The Register Client provides users a tool to register new native IoT platform services and new composite services (also known as subflows). During the registration of one of these elements, it is possible to add a description about its features. Once the registration of the service occurs, it will be stored in the Service Catalogue.

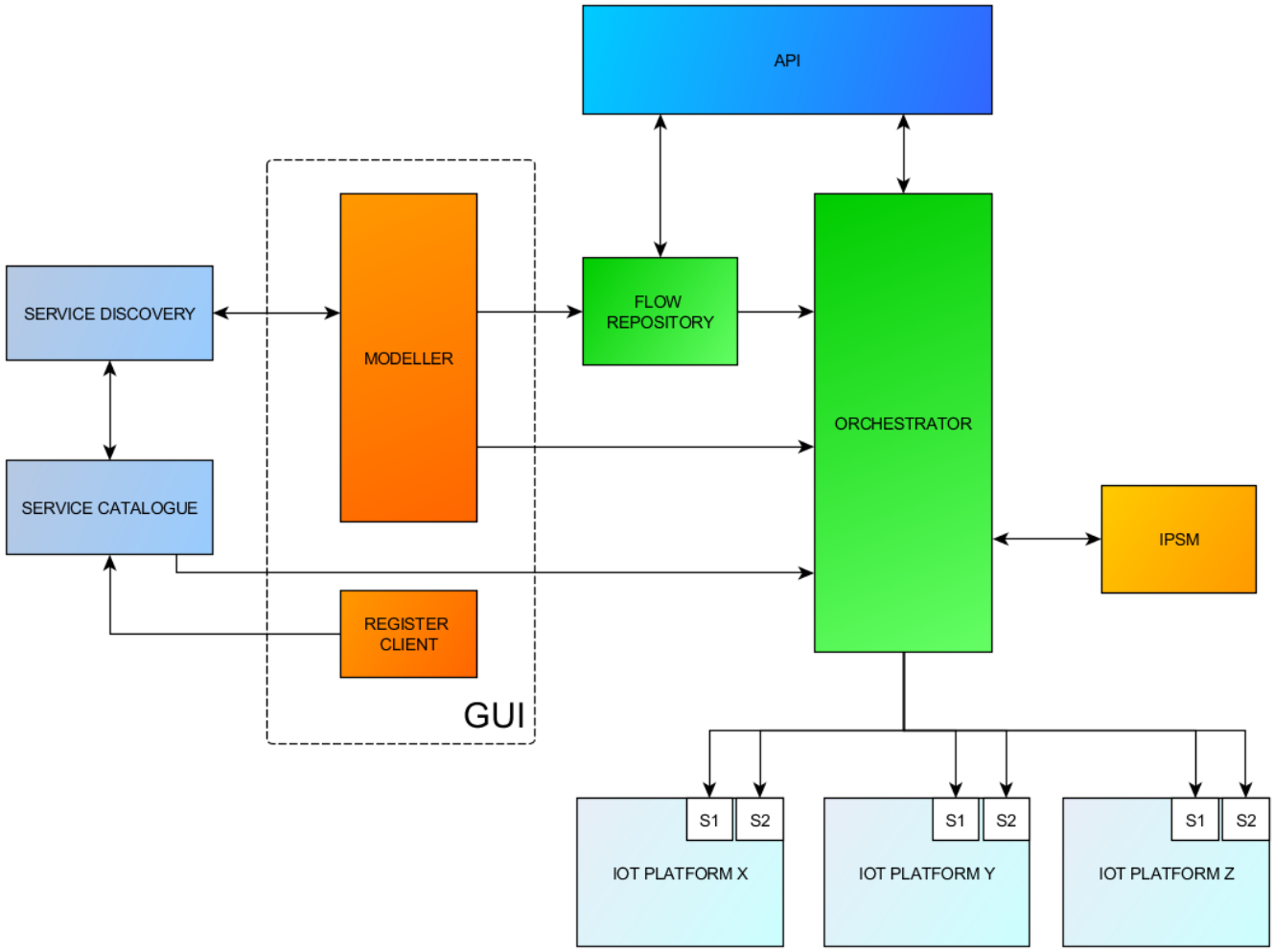

Figure 57: AS2AS architecture overview.

The Modeller is a graphical environment that has access to the services which have been registered (using the Service Discovery module, through which it calls the Service Catalogue) and internal functions to execute a particular process (for example, functions to perform transformations in the data resulting from the execution of a service, to display information, to determine a timeout between calls, to repeat a call to a service 'x' number of times…). With this tool the AS2AS users can design a solution based on the composition of services. The visual editor lets user drag and drop the services (visually represented as nodes) onto the design surface and then join them together by dragging lines between them. A solution based on flow based programing will be designed.

Once the design made by the Modeller is validated, the generated flow is stored in the Flow Repository. This component manages the information of all flows created.

The Orchestrator is the engine of the solution. It is responsible for loading the flows created with the Modeller and stored in the Flow Repository. Once the design is loaded, it makes the necessary calls to the service APIs of the IoT Platforms Services and executes the internal functions, in the order indicated in the model to run the service composition. It collaborates with the IPSM, which is responsible for performing semantic translation of data exchanged between artifacts...

Finally, the API is responsible for making the interaction tools to manage the Orchestrator and the flows stored in the Flow Repository available to the AS2AS user. For example: it would be like a

process manager; where a user can start/stop a flow of execution, view its status, load a flow in the orchestrator and so forth.

3.4.2 Components

Component	Register Client
Description	It is responsible for allowing the registration of new services through the graphical environment. In addition, it is possible to include a description of these services.
Functionalities	Register new services: • register an IoT Platform service • register flows as a new service (subflows) • register functions created by the user.
Relation with other component	Service Catalogue
Use Cases Involved	[28]
Requirements Involved	[43], [73]

Component	Modeller
Description	It is used to perform a modeled solution with available services.
Functionalities	It allows to make a composition of services with its graphical tool. It calls the Service Catalogue through the Service Discovery module, to have a list of available services. Modeller indicates which services are compatible with other service and it's in charge for the validation of the design.
Relation with other component	Service Discovery Flow Repository Orchestrator
Use Cases Involved	[24], [29], [30]
Requirements Involved	[76], [240]

Component	Service Catalogue
Description	Provides storage and access to a uniform catalogue of existing and new services. Each service is described by listing its features. It uses the same annotations to facilitate a point of interoperability
Functionalities	This component provides the following functions: • Register services/applications to make them discoverable. • Offer a description or detailed information about the services/applications.
Relation with other component	Service Discovery Register Client Orchestrator
Use Cases Involved	[24, [28], [29]
Requirements Involved	[43], [73]

Component	Service Discovery
Description	Manages the detection of services provided by each IoT platform attending certain features. Also, it indicates the possibilities of composition between services.
Functionalities	It is responsible for converting the demands that come from the Modeller in queries to the Service Catalogue. In addition, it also converts the answers of the Service Catalogue into responses in the format of the graphic environment of the Modeller.
Relation with other component	Service Catalogue Modeller
Use Cases Involved	[24], [28], [29]
Requirements Involved	[43]

Component	Flow Repository
Description	Stores the composite services designed with the graphical tool.
Functionalities	The created flows will be stored with an identifier. The orchestrator will call this repository when it needs to load information from a flow.
Relation with other component	Modeller Orchestrator API
Use Cases Involved	[24], [30]
Requirements Involved	[57], [239]

Component	Orchestrator
Description	Using the designed flow made with the Modeller, the Orchestrator calls services of platforms to run the composite service. This action is done from a centralized perspective, expressing how the composition has to act in order to integrate components.
Functionalities	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.
Relation with other component	Flow Repository IPSM API IoT Platform Services
Use Cases Involved	[24], [28], [30]
Requirements Involved	[76], [239], [240], [241], [255]

Component	AS2AS API
Description	It is responsible of managing the orchestrator and flows.
Functionalities	It would be like a process manager, where the user can: start a flow, stop a flow, view its status...
Relation with other component	Flow Repository Orchestrator
Use Cases Involved	[24], [30]
Requirements Involved	[57], [221]

External components to AS2AS:

Component	IPSM
Description	The Inter Platform Semantic Mediator provides semantic translation of messages that is explained in section 3.5.1.
Functionalities	It performs semantic translation of input messages expressed in source semantics to output messages expressed in target semantics. It provides API for configuration before use.
Relation with other component	Orchestrator
Use Cases Involved	[24], [28], [30]
Requirements Involved	[179], [221], [255]

Component	IoT Platforms Services
Description	The IoT Platforms offers native services and applications.
Functionalities	Almost all IoT platforms provide a public API to access their services. The APIs are usually based on RESTful principles, and allow common operations such as PUT, GET, PUSH or DELETE. However, there are IoT

	platforms that did not include a REST API for easing the development of Web services, but use different interaction means.
Relation with other component	Orchestrator
Use Cases Involved	[24], [30]
Requirements Involved	[57], [205], [241]

3.4.3 Use cases

Use case	AS2AS Service Cataloguing
Use Case ID	#28
Description	An AS2AS user will be able to register services/applications with their description or detailed information to make them discoverable.
Objectives	The following objectives are mainly pursued: • Offer a description or detailed information about the services/applications. • The registered service becomes available in the Modeller. So it can be used by the Orchestrator. • To obtain an uniform data catalog.
Components Involved	Register Client Service Catalogue Service Discovery Orchestrator
Requirements Involved	[180], [236], [238]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-776

AS01 AS2AS Service Cataloguing

View online: http://tinyurl.com/as01v01

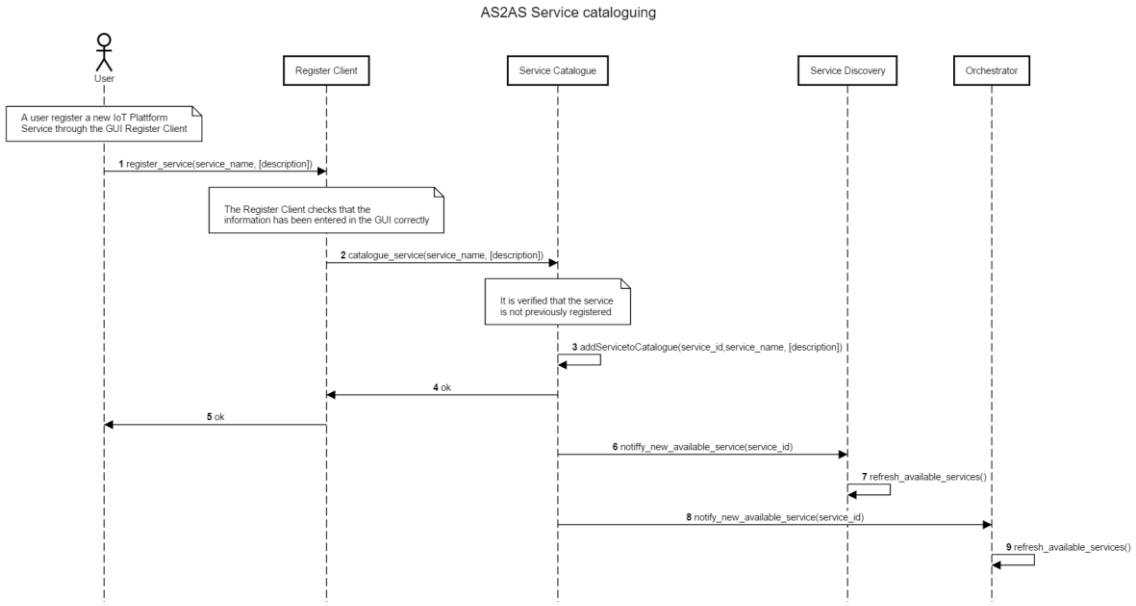

Figure 58: AS2AS Service Cataloguing.

Step 1: AS2AS user uses the GUI to register a service through Register Client. The user fills the information and description of the service.

Step 2: The Register Client checks that the information has been entered in the GUI correctly and sends it to the Service Catalogue. If there are validation failures, the user will be informed of the errors and must complete the fields again correctly.

Steps 3-5: The Service Catalogue confirms that the service has not been previously registered.

If all validations are correct, the new service is added to the catalogue. After that Service Catalogue confirms that the service has been successfully registered.

Else, an error indicating that this service has been previously registered is sent to the user.

Steps 6-7: The Service Catalogue notifies the availability of the service to the Service Discovery component. This component is refreshed to show this new service.

Steps 8-9: The Service Catalogue notifies the availability of the service to the Orchestrator component. This component is refreshed to show this new service.

Use case	AS2AS Service Discovery
Use Case ID	#29
Description	An AS2AS user will be able to obtain the list of available services in the catalogue, which comply with a search query or filter.
Objectives	Obtain a list of matching services and applications from the IoT platforms considered.

Components Involved	Modeller Service Discovery Service Catalogue
Requirements Involved	[180], [236], [238]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-777

AS02 AS2AS Service Discovery

View online: http://tinyurl.com/as02v01

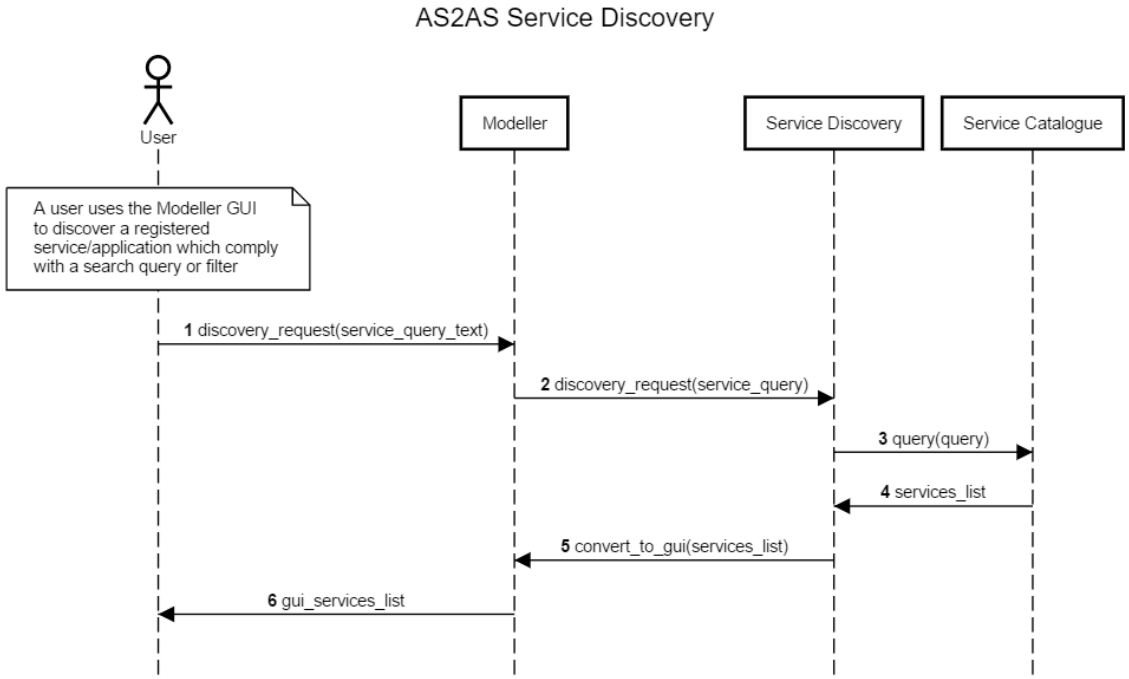

Figure 59: AS2AS Service Discovery.

Step 1: A user introduces a query text, to search or filter, in a search box in the Modeller GUI.

Step 2: The Modeller module forwards request to Service Discovery.

Step 3: The Service Discovery processes the request and sends a query to the catalogue.

Step 4: The Service Catalogue responds to Service Discovery with a list of services.

Step 5: The Service Discovery processes the result of the query and sends it back to the graphical environment.

Step 6: A list of matching services and applications from the IoT platforms considered is available for the user.

Use case	AS2AS Service Composition
Use Case ID	#30
Description	An AS2AS user will be able to create added-value services, called composite services, from existing services.
Objectives	The main objective is wiring together APIs and Services from the IoT Platforms creating new composite services.
Components Involved	Modeller Flow Repository AS2AS API Orchestrator IoT Platforms Native Services
Requirements Involved	[236], [239], [240], [241]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-778

AS03 AS2AS service composition

View online: http://tinyurl.com/as03v01

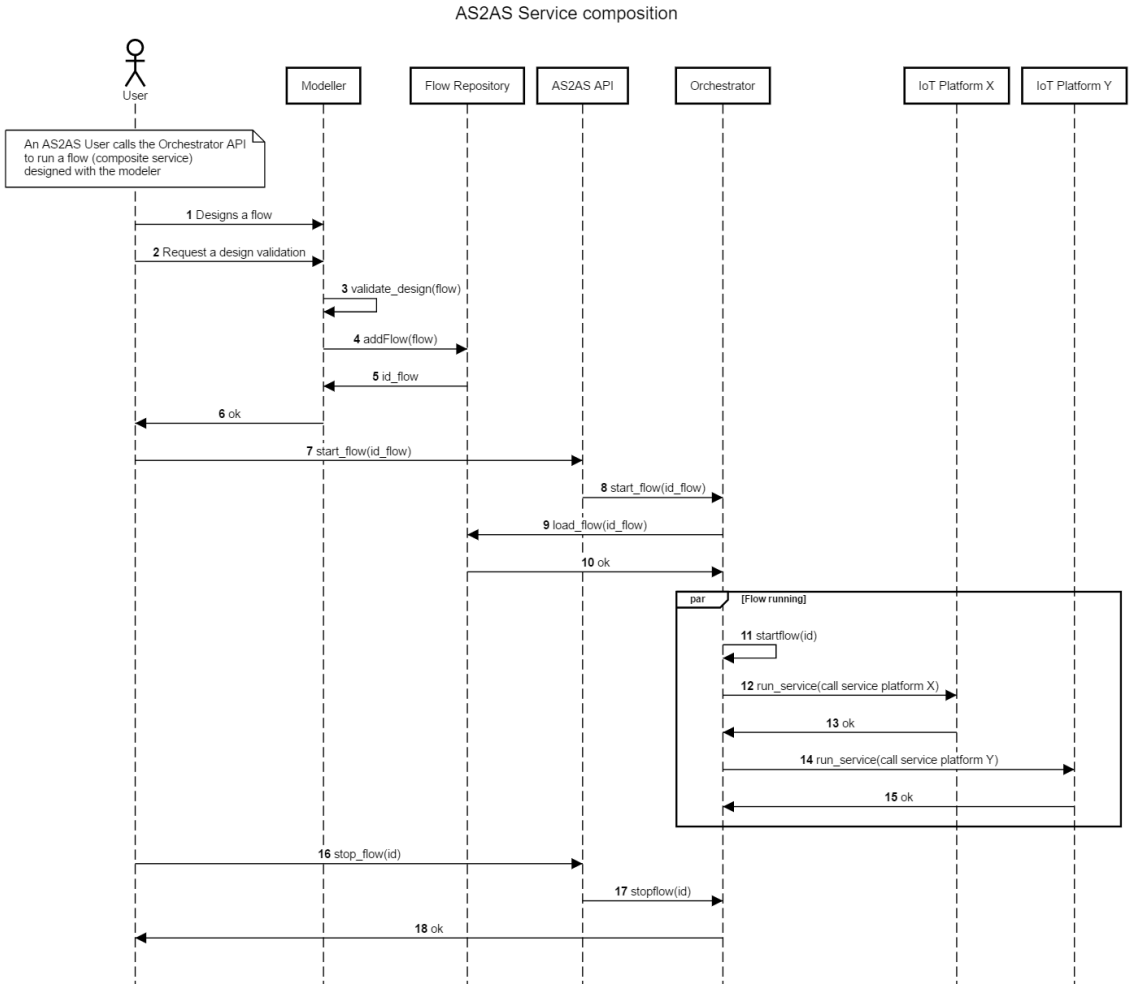

Figure 60: AS2AS Service Composition.

Steps for the creation, validation and storage of a flow:

Step 1: A user designs with the Modeller GUI a modeled solution with the available services.

Step 2: The user requests a validation of this design.

Step 3: The Modeller performs a validation of the created design. If there is any error in the validation, the user will receive a message with a list of errors.

Steps 4-6: The validated design (flow) is stored in the Flow Repository. An identifier is assigned to the flow. The user is notified of the correct validation.

Steps to run and stop a flow using the API:

Step 7: The user uses the API to start the desired flow.

Step 8: The API calls the Orchestrator to start the desired flow.

Steps 9-10: The Orchestrator loads the indicated flow from the Flow Repository.

Steps 11-15: The Orchestrator module starts the flow and calls IoT platform services in the order indicated in the design to run the composite service.

Steps 16-17: If the flow does not have a stop condition, the user can call the API to stop the flow.

Step 18: The Orchestrator informs the user of the correct execution of the flow. On the other hand, if there is any problem the user will be informed of any error in the execution of a service.

Use case	Request Query to AS2AS
Use Case ID	#24
Description	An AS2AS user (IoT platform, application, person, etc.) will be able to exchange information between IoT platform services/applications through the AS2AS system.
Objectives	To let the inquirer exchange information of interest with services and applications allocated in another platform.
Components Involved	Modeller Flow Repository AS2AS API Orchestrator IoT Platforms Native Services
Requirements Involved	[236], [239], [240], [241]
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-775

AS04 Request query to AS2AS

View online: http://tinyurl.com/as04v01

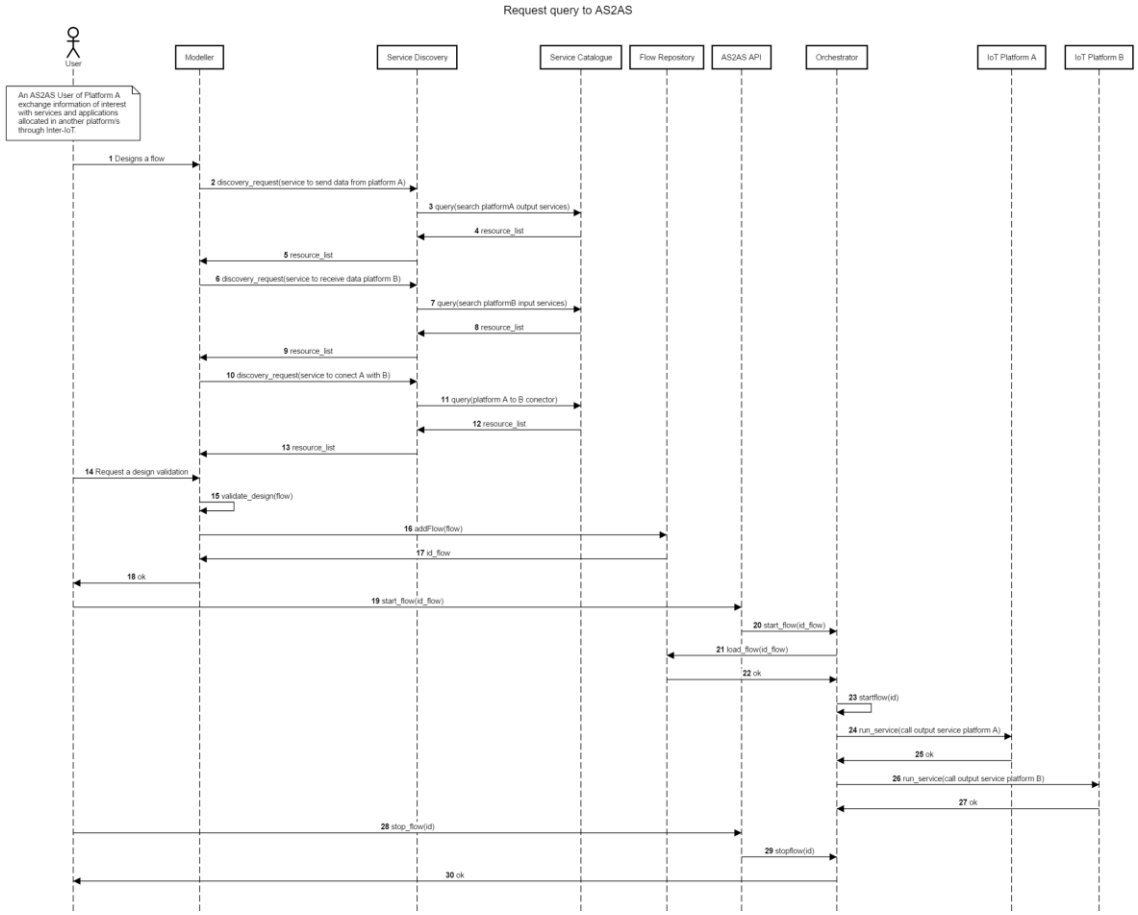

Figure 61: Request query to AS2AS.

An AS2AS User of Platform A exchange information of interest with services and applications allocated in another platform/s through INTER-IoT.

Steps for the creation, validation and storage of a flow request query:

Step 1-13: A user uses the Modeller GUI to create a query within available services of Platforms A and B. In this step the desired information exchange is defined.

Step 14: When the user has finished the design, request a validation of this.

Step 15: The Modeller performs a validation of the created design.

Steps 16-18: The validated design (flow) is stored in the Flow Repository. An identifier is assigned to the flow. The user is notified of the correct validation.

Steps to run and stop a flow using the API:

Step 19: The user uses the API to start the desired flow.

Step 20: The API calls the orchestrator to start the desired flow.

Steps 21-22: The Orchestrator loads the indicated flow from the Flow Repository.

Steps 24-27: The Orchestrator module starts the flow and calls IoT platform services in the order indicated in the design to run the composite service. First a call is made to platform A service to

receive information, then a call to platform B service is made to send information. During this process, the Orchestrator could execute some internal services to facilitate this exchange of information.

Steps 28-29: If the flow does not have a stop condition, the user can call the API to stop the flow.

Step 30: The orchestrator informs the user of the correct execution of the flow. On the other hand, if there is any problem the user will be informed of any error in the execution of a service.

3.4.4 Technologies

We have decided to use Node-RED as the central technology of the AS2AS solution. In the SOTA section 2.4.2.5 we can find the main characteristics of Node-RED and the other tools we have considered. Explaining the advantages and disadvantages of these tools in an AS2AS interoperability solution.

In this section we will include a map of AS2AS and Node-RED features establishing a connection between its elements and our architecture.

AS2AS ARCHITECTURE	NODE-RED	Observations
IoT Native Services	A visual tool for wiring together hardware devices, APIs and online services.	INTER-IoT focuses on the creation of new nodes for IoT services, as well as on the study of existing nodes.
Service Catalogue	Node-RED comes with a core set of useful nodes and mechanisms to implement new custom nodes. The semantic annotations are not used.	INTER-IoT is working on the semantic description of the nodes. In order to extend the advantages of Node-RED.
Service Discovery	Users can search for available nodes in the Node-RED library or in the npm repository. New nodes can be installed, and existing nodes can be enabled or disabled .	INTER-IoT exploits the advantages that the use of semantic annotations in the nodes provide to the discovery of services.
Register Client	Node-RED lets implement nodes directly using the editor.	INTER-IoT is able to introduce semantic annotations when registering a new node.
Modeller	Node-RED provides a browser-based flow editor that makes it easy to wire together flows using a wide range of nodes in the palette.	The Node-RED graphical tool is a solution that provides the desired functionalities to the INTER-IoT modeller.
Flow Repository	A built-in library allows users to save useful functions, templates or flows for re-use.	Node-RED offers a solution to work with the flows.

Orchestrator	Flows can be deployed by the runtime with a single-click.	Using this element, users can make calls to the services in the desired order.
Data semantic translation support	Does not offer anything related	It is one of the main improvements that IINTER-IoT implements.
API	AdminHTTP-based API can be used to remotely administer the runtime. Runtime API can be used when embedding Node-RED into another application. Runtime API provides a pluggable way to configure where Node-RED runtime stores data	Some of the INTER-IoT API needs are already provided by the Node-RED API.

Table 7: Summary of Node-RED technologies for INTER-IoT.

Our work is focused on expanding the possibilities offered by Node-RED in these directions:

- Development of nodes for access to services of IoT platforms that have not yet been implemented, thus actively collaborating with the Node-RED community, offering new nodes that will fit the needs of the users.
- Development of a solution over the Node-RED platform or making changes in their design, to achieve the needs of the proposed architecture. For example, creating new solutions in the cataloguing and discovery of services or by bringing out performance to all the possibilities that the API can offer.
- Being able to take advantage of the benefits provided by containers engines (docker), combining it with the use of Node-RED.
- Development of new solutions related to semantics that can work with Node-RED.
- Use the solutions designed in this layer in the infrastructures provided by the project partners.

Next we will focus on explaining how the use of Node-RED and other technologies/platforms would be combined to offer a new interoperability solution in this layer.

Current catalogue of Platform Services Nodes

We are going to list the contributions that are currently offered in the Node-RED library to work with the main IoT platforms.

Platform	Node-RED nodes	License
OM2M[125]	• oM2M-Application - a node to define a new OM2M application • oM2M-Data-Container - A node to register a Data Container with OM2M • oM2M-Data-ContentInstance - a node to push application's data (a.k.a. Content Instance) @ Data Container • oM2M-Descriptor-Container - a node to register a Descriptor Container with OM2M • oM2M-Descriptor-ContentInstance - a node to set application's data metadata @ Descriptor Container • oM2M-Subscription - a node to define a new OM2M Subscription	Copyright 2015 the Apache 2.0 license.
FIWARE[126]	A set of nodes that allow one to interact with a FiWare instance: • fiware-device-out, fiware-device-in - a node that sends the message payload to FiWare's Orion Context Broker • Fiware-instance - configuration node	MIT
Azure IoT Hub[127]	A set of nodes that allow one to send messages and register devices with an Azure IoT Hub. This is a fork from the original Node-RED example by the Azure IoT team. Nodes: • azureiothub - this node allows one to send messages to one's Azure IoT Hub • azureiothubregistry - this node allows one to registers devices with one's Azure IoT Hub	
Azure IoT[128]	This Node-RED node adds Azure IoT connectivity to the Node-RED flow.	MIT
Azure IoT Hub Examples[129]	This flow has several examples to help to get started sending data from Node-RED to the Microsoft Azure IoT Hubs.	
Other (related to IoT platforms, but not necessarily INTER-IoT)		

[125] https://github.com/themaco/node-red-contrib-om2m
[126] http://flows.nodered.org/node/node-red-contrib-fiware
[127] http://flows.nodered.org/node/node-red-contrib-azure-iot-hub
[128] http://flows.nodered.org/node/node-red-contrib-azureiothubnode
[129] http://flows.nodered.org/flow/1270f25a183d67c5b50e6b4eb78cabea

SiteWhere[130]	Allows Node-RED to interact with the SiteWhere Open IoT Platform. This module allows a device running Node-RED to interact with SiteWhere via JSON over the MQTT protocol. It supports the following concepts: • registration of a new device with SiteWhere • sending measurements, alerts, and location data to SiteWhere • receiving system commands and custom commands from SiteWhere	Apache 2.0
IBM Watson[131]	A pair of Node-RED nodes for connecting to the IBM Watson Internet of Things Platform as a Device or Gateway.	Apache 2.0
IBM Watson Examples[132]	IoT Application Node-RED node for the Registered and Quickstart flows in the IBM Watson IoT Platform. Nodes: • ibmiot in - to receive device events, status, application status and device commands in case of Registered flow • ibmiot out - to send device events and device commands in case of Registered flow • ibmiot - configuration node	Apache 2.0
SOFIA2[133]	Experimental: A Node-RED node to interact with Indra's SOFIA2 IoT platform. Nodes: • sofia2 • sofia2-server - configurationnode	Apache 2.0
AWS[134]	A Node-Red node to read and write to the Amazon Web Services AWS IoT. Nodes: • aws-mqtt in - MQTT input node. Connects to a broker and subscribes to the specified topic • aws-mqtt out - Connects to a MQTT broker and publishes **msg.payload** either to the **msg.topic** or to the topic specified in the edit window • aws -thing - a thing shadow is a JSON document that is used to store and retrieve current state information for a thing. Use thing shadows to get and set the state of a thing over MQTT or HTTP	Apache 2.0

[130] https://libraries.io/npm/node-red-contrib-sitewhere
[131] https://www.npmjs.com/package/node-red-contrib-ibm-watson-iot
[132] http://flows.nodered.org/node/node-red-contrib-scx-ibmiotapp
[133] http://flows.nodered.org/node/node-red-contrib-sofia2
[134] http://flows.nodered.org/node/node-red-contrib-aws-iot-hub

	• Aws-iot-device - device configuration node	
ThingSpeak[135]	A simple node that sends the payload to thinkerspeak.com with a http get.	MIT
ThingSpeak Examples[136]	A node (thingspeak42) that posts data to a ThingSpeak channel. Capable of aggregating multiple messages into a single multi-field post.	MIT
AllJoyn[137]	A collection of Node-RED nodes to access Alljoyn services by node-alljoyn.	Apache 2.0

Table 8: Summary of existing IoT platform nodes and example flow for Node-RED.

IPSM related nodes

The Inter Platform Semantic Mediator (IPSM) is a solution, within the INTER-IoT, that provides semantic translation of messages (for details, see D3.1 section DS2DS). The IPSM receives messages for translation within a communication translation channel dedicated to an "instance" of communication with translation. Each channel is created dynamically, when needed, and translates between two chosen semantics. The IPSM communication channels have inputs (through which the IPSM receives messages) and corresponding outputs (to which translated messages are written). Before using the IPSM, a channel and an alignment configuration is required.

In the context of Node-RED, the semantic interoperability tasks are assumed to be realized with dedicated nodes, required to configure, send data to and receive data from the IPSM. The input and output messages (data) are in the RDF data format, and have, respectively, source and target artifacts semantics.

In the Node-RED, the following custom nodes need to be created:

- **IPSM Configuration** - the configuration node(s) that is(are) used by the IPSM in and out nodes. The initial configuration consists of:
    - alignment - at least one alignment must be added to the Alignments Repository IPSM component to become available to be used in the translation process,
    - channel - instantiation of semantic translation channel**,** which in turn creates a corresponding Alignment Applicator IPSM component and configures the IPSM communication channel.

The IPSM provides REST API to manipulate alignments and channels. This API shall be called from the Node-RED node(s). The IPSM in and out nodes use the configuration to interact with the IPSM.

- **IPSM in** - node that accepts data in the RDF format, expressed in semantic of the source artifact, and writes the data to the communication channel (set during configuration). Communication channel is available in the IPSM Communication Infrastructure.

[135] http://flows.nodered.org/node/node-red-contrib-thingspeak

[136] http://flows.nodered.org/node/node-red-contrib-thingspeak42

[137] https://www.npmjs.com/package/node-red-contrib-alljoyn

- **IPSM out** - node that consumes data in the RDF format, expressed in semantic of the target platform, from the communication channel (set during the configuration) available in the IPSM Communication Infrastructure.

Nodes that already exist in Node-RED and can be useful for implementing semantic translation tasks are:

- Kafka Node - Kafka node can produce/consume the messages to/from kafka cluster along with topic(s),
- HTTP Request - core Node-RED node (with available extension) for performing HTTP/HTTPS requests,
- Swagger API - a Node-RED node able to invoke Web APIs generically based on a Swagger description.

Semantic catalogue annotations

Standards that offer solutions to the functionalities requested by the Service Catalogue and Service Discovery are described in the section 2.4.2.2. One of these standards is Hypercat, which is the tool that most closely matches the needs of AS2AS. In this section you can find a description of Hypercat features.

Some of the factors that have led to the choice of this standard are listed briefly:

- HyperCat standard is a hypermedia catalogue format designed for exposing information about the Internet of Things. The Hypercat specification allows Internet of Things clients to discover information about IoT resources and services over the Web.
- HyperCat provides a standard means for resource discovery, which enables an interoperable ecosystem. The Hypercat standard makes easier to discover and combine data from connected things to create valuable new applications and services.
- HyperCat developer community is emerging, with open source tools available. More than 40 organisations (including Intel, ARM and IBM) have been working on this project.

In order to combine semantically annotated Service Catalogue with Node-RED, a set of dedicated nodes need to be implemented, to perform basic operations on the catalogue. HyperCat is an open, lightweight JSON-based hypermedia catalogue format for exposing collections of URIs (with any number of RDF-like triple statements about them). It is designed for exposing information about IoT assets as linked-data descriptions.

The HyperCat specifications have been implemented in Node.js, Java, Python, PHP. For example, Node.js node-hypercat is a small library for fetching, pushing and manipulating Hypercats. The following operations are supported:

- Opening/creating HyperCats
- Inspecting a HyperCat
    - Getting items
    - Getting sub-catalogues

To use HyperCat from Node-RED, we have to create nodes that cover specific API functions.

Service semantic annotations

A number of technologies can be used to describe semantic Web services. SADI (Semantic Automated Discovery and Integration) is a project that defines a set of design patterns for services. The project is rooted in presenting services as instances of OWL classes, expressed in RDF. A catalogue of SADI services is an RDF graph of instances.

The services themselves can be described in an OWL ontology. WSMO (Web Service Modelling Ontology) is one such ontology. It provides a basic model in which any Web service can be described. Because of inherent extensibility of OWL, this ontology may be combined with other (e.g. domain specific) ontologies to meet the needs of describing very specific services. The WSMO is built “on top” of the WSMF (Web Services Modelling Framework). Lightweight Semantic Descriptions for Services on the Web is provided by the WSMO-Lite. The SWSO (Semantic Web Services Ontology) is a part of the SWSF (Semantic Web Service Framework) - a framework that aims to enhance existing Web service technologies with semantics. The ontology itself is defined in a SWSF language, based on first-order logic, and is an extension of OWL-S.

OWL-S is a first major OWL ontology for describing semantic Web services, that was designed to enable automatic discovering, invoking, composing, and monitoring of Web resources.

More information on service semantics is in section 2.5.

3.5 DS2DS proposed solution

The approach proposed for the DS2DS level is based on:

- Defining explicit, OWL-demarcated, semantics for each IoT artifact that is to interoperate, communicate and collaborate
- Creating infrastructure that will translate messages / data / communication from its native format to the common format used across the INTER-IoT infrastructure
- Create an IoT Platform Semantic Mediator (IPSM) component that will be responsible for translating incoming information, representing semantics of artifact X to semantics of artifact Y. The IPSM will use ontological alignments to perform ontology-to-ontology translations.
- Alignments in IPSM are uni-directional.
- IPSM is configured with a core ontology and alignments to and from it.
- Each complete process of translation consists of a translation from input semantics to core semantics and from core semantics to output semantics.
- The IPSM will work concurrently servicing multiple communication channels, each representing a single “communication”.
- The IPSM will be designed in such way to maximize speed of translation (and, thus, to allow use on data streams).
- While each channel will have one input and output, multiple artifacts will be allowed to subscribe to 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.

- Channels are dynamically created and destroyed.

In what follows we describe in more details the architecture of the core of the IPSM, as well as auxiliary components that are needed for its functioning.

3.5.1 Architecture

The architectural model can be seen below in Figure 62.

The Inter Platform Semantic Mediator (IPSM) is a software component that performs semantic translation of data. In the context of the INTER-IoT, it is used to translate semantics of messages exchanged by IoT artifacts (platforms, gateways, applications, etc.) within the INTER-IoT software. It is composed of the IPSM Core and auxiliary components, i.e. Semantic Annotators, and exposes a REST API (for configuration). An additional Communication Infrastructure is required to enable communication between the IPSM and all other “artifacts” that are to use its semantic translation services.

The Semantic Annotators are “located” between the “outside world” and the IPSM. Their role is to produce RDF triples from data that they receive, e.g. from Bridges (component described in the MW2MW layer of the INTER-IoT architecture), and forward them to the IPSM Core through the Communication Channels, instantiated within the Communication Infrastructure. The IPSM Core performs the semantic translation of the RDF data, by applying pre-stored alignments (representing relationships between input and output ontologies). An instance of the IPSM can concurrently “service” multiple “conversations” taking place in separate Communication Channels. To achieve this goal, it can communicate with multiple instances of Semantic Annotators at the same time. Furthermore, each Alignment Applicator services a single Communication Channel and applies a separate alignment within the context of such channel.

Communication Channels work in publish-subscribe mode, which allows a single channel to serve both, one-to-one and one-to-many communication. It is explicitly assumed that:

- regardless of underlying semantics, each message flow through a channel is served by a pair of alignments,
    - One alignment is appointed to be the “input” alignment, and another one the “output”. One or both alignments can be empty (in the latter case no actual translation occurs),
- each alignment has a unique ID, and
- alignments are persisted / managed (added, removed), in an Alignment Repository, during the lifetime of the IPSM.

Main operations that are going to be served by the IPSM and that are described through the sequence diagrams (in what follows) are:

- management of alignments in the Alignment Repository
- instantiation of a Communication Channel, and
- sending (and translating on the way) a message from one IoT artifact to another (where another can be one, or more, artifact(s) subscribed to the given communication channel).

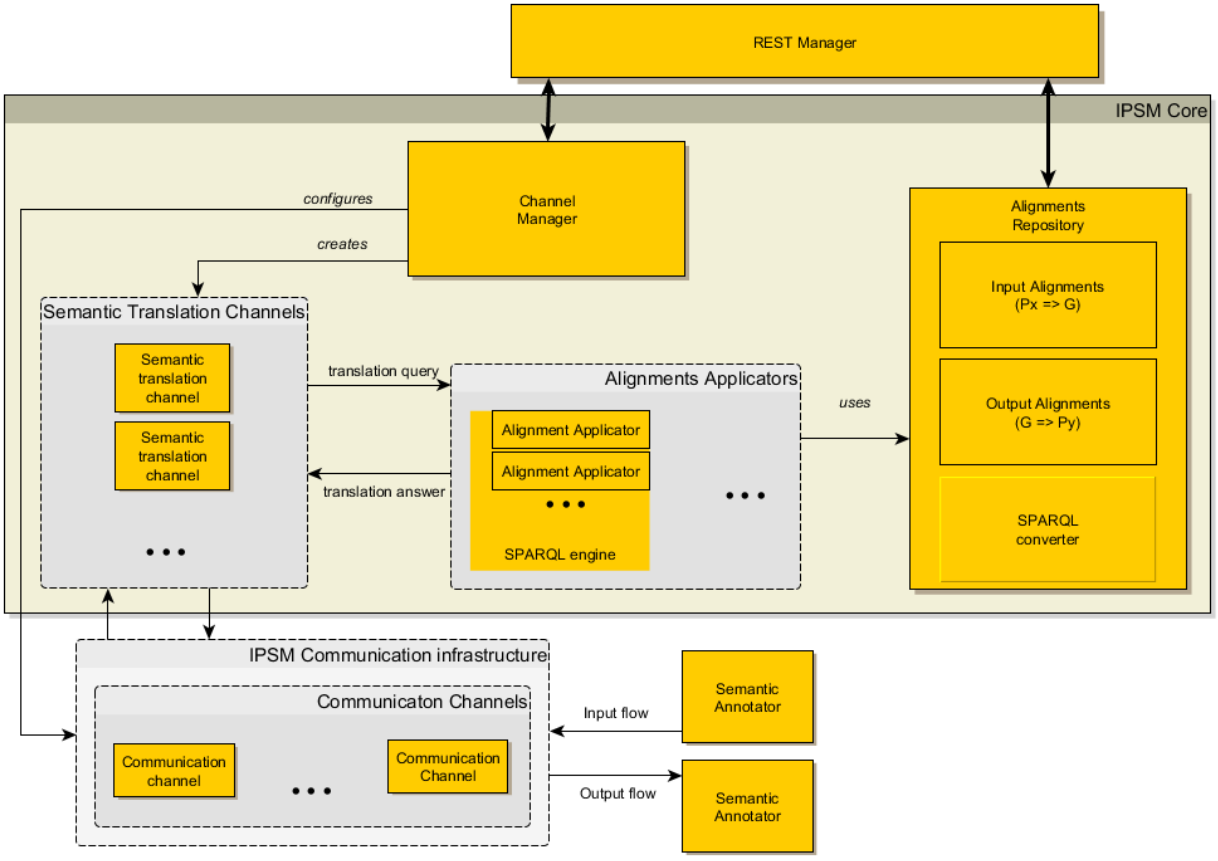

Figure 62: DS2DS architecture overview

3.5.2 Technologies

At the moment of writing of the deliverable, for the DS2DS solution we have decided to use the following technologies.

Technology	Justification
Scala	A general purpose programming language with executable code running on a JVM, designed to overcome Java shortcomings (but easily integrable with Java).
Akka	Toolkit for building highly concurrent, distributed, message-driven applications, running on a JVM. Language bindings exist for both Java and Scala. Akka provides libraries that enable the parallel nature and independence of IPSM Semantic Translation Channels.
Banana-RDF	A library for RDF, SPARQL and Linked Data technologies in Scala. It provides an interface to, and a layer of separation from, SPARQL engines used by Alignment Applicators, and SPARQL converter in the Alignments Repository.
Apache Kafka	A distributed streaming platform that runs in production in thousands of companies. It is high-throughput, low-latency platform for handling real-time data feeds with architecture based on a scalable pub/sub message queue. It is the basis of IPSM Communication Infrastructure.

Table 9: Summary of technologies used for DS2DS.

3.5.3 Acronyms

For the sake of components and sequence diagrams descriptions, the following acronyms are used:

- **IPSM** – IoT Platform Semantic Mediator
- **G** – core IPSM ontology. While the ontology will be modular - to facilitate efficiency and flexibility of the INTER-IoT approach - the architecture of IPSM does not place such requirement on it. For sake of simplicity, in the description that follows, it will be treated as a single entity.
- **Px** – ontology in which input data is described (possibly, to which non-semantic input data was lifted)
- **Py** – ontology in which output data is described (possibly, to which non-semantic input data was lifted)
- **AA** – Alignments Applicator
- **AR** – Alignment Repository
- **CC** - Communication Channel
- **CI** - Communication Infrastructure
- **CM** - Channel Manager
- **RM** – REST Manager
- **SA** – Semantic Annotator
- **STC** – Semantic Translation Channel

3.5.4 Components

Component	Communication Infrastructure
Description	The role of the **CI** is to facilitate communication between IoT artifacts and the IPSM.
Functionalities	Acts as a message broker that is used for every communication act. It exposes API for management of **CC**s and performing common operations.
Relation with other component	It manages instantiated **CC***s.*
Use Cases Involved	[38], [67]
Requirements Involved	[2], [20]

Component	Communication Channel
Description	A channel through which **SA**s and the IPSM communicate. Communication channels consist of a Source, Sink, and a series of Flows between sources and sinks. Specifically: • Source - the data originates at an entity (e.g. Bridge - a MW2MW component) that writes to a Semantic Annotator, • Flow - process within which data goes through a **SA** into a **STC**, with an Alignment Applicator (where it is translated from **Px** to **Py**), back to the **STC**, to a potentially different (output) **SA**, • Sink - the output through which the **SA** sends the data to a receiving entity (e.g. a platform). The proposed architecture allows for dynamic creation of channels between any number of components, e.g. one **SA** may write data through multiple **STC**s (and thus multiple alignments) to multiple receiving **SA**s. A communication channel can be configured to allow semantic translation that uses a (possibly different) set of **STC**s for any combination of sending and receiving **SA**s.
Functionalities	Each **CC** provides a one-directional communication-translation. It receives data sent by the “in-artifact” and allows reading “out-data” by (one or more) “out-artifacts”.
Relation with other component	CC allows communication between other components: **SA**s, **IPSM**. A **STC** is a part of **CC**.
Use Cases Involved	[38], [67]
Requirements Involved	[20]

Component	Channel Manager
Description	**CM** is a component used for IPSM configuration.
Functionalities	**CM** manages (creates, destroys, lists) Communication Channels i.e. flows in message broker and Semantic Translation Channels.
Relation with other component	**CM** receives requests from **RM**. CM configures **CC**s and creates **STC**s.

Use Cases Involved	[67]
Requirements Involved	[20]

Component	**REST Manager**
Description	Interfacing component used for management of alignments in the Alignments Repository and for configuration of Communication Channels within the IPSM.
Functionalities	**RM** provides an API to facilitate alignments and channel management functionalities within the IPSM.
Relation with other component	Provides interface to **CM** and **AR** (dedicated API for handling requests).
Use Cases Involved	#66 IPSM Alignment Configuration, #67 IPSM Channel Configuration
Requirements Involved	N/A

Component	**Semantic Translation Channel**
Description	A lightweight component that stores information about: • Where to receive data from (i.e. an in-flow in the **CC**) • Which alignment to use (from the **AR**) • Where to send data to (i.e. an out-flow in the **CC**) Within the IPSM, there may be multiple **STC**s instances and they may use the same or a different pair of alignments.
Functionalities	Each **STC** instance receives data from an (input) flow in the **CC** and forwards it to an **AA**, along with information, which alignment to use. **AA** sends back translated data, which the **STC** forwards to an (output) flow. Each **STC** operates on a designated pair of (input/output) alignments.
Relation with other component	**STC** is part of **CC**. Communication with **AA**.

Use Cases Involved	[38], [67]
Requirements Involved	[180], [20]

Component	Alignments Repository
Description	Component for persistence and management of alignments used in the translation process. Each alignment is stored in an original SRIPAS alignment format, based on the Alignment Format from Alignment API, and as SPARQL statements. Alignments are divided into cells, each cell representing a one-way semantic transformation between two *entities* described in the RDF format. • Input alignments are unidirectional alignments between **Px** and **G** • Output alignments are unidirectional alignments between **G** and **Py** • SPARQL converter performs one-time conversion for each new alignment written into the repository. Alignment Applicators use SPARQL queries to apply each cell of the alignment.
Functionalities	Stores and manages alignments (read/write alignments).
Relation with other component	Used by **AA** and managed by **RM**.
Use Cases Involved	[38], [66]
Requirements Involved	[180]

Component	Alignments Applicators
Description	A component, instances of which are performing semantic translation. **AA**s are created and destroyed dynamically, as needed, to facilitate scalability.
Functionalities	Each **AA** has a unit that takes an RDF graph and an alignment (in SPARQL) as input, and returns RDF (translated through the alignment).
Relation with other component	Each applicator has access to a SPARQL engine, on which the alignments in the SPARQL format are applied to RDF data. **AA**s may share SPARQL engines. **STC**s communicate with **AA**s. **AA**s use **AR**.

Use Cases Involved	[38], [67]
Requirements Involved	[180]

Component	SPARQL Converter
Description	SPARQL converter is called when new alignment is placed in the **AR** to convert alignments' cells from SRIPAS format to internal SPARQL format. This is done to speed up future usage of the alignment.
Functionalities	Conversion of SRIPAS Alignment format into SPARQL queries.
Relation with other component	Used within **AR**.
Use Cases Involved	[66]
Requirements Involved	N/A

Component	SPARQL engine
Description	SPARQL engine is a service used to execute SPARQL queries (i.e. cells of an alignment stored in SPARQL format), as instructed by an instance of **AA**. It is possible that, based on experimental data collected while testing the initial implementation, multiple instances of the SPARQL engine will be created, to achieve higher throughput within the IPSM.
Functionalities	Library for management of RDF data and SPARQL execution.
Relation with other component	Used by **AA**.
Use Cases Involved	[38]
Requirements Involved	[180]

Component	Semantic Annotator
Description	This component instances perform two-way syntactic translation between RDF (which is always the input and output of an **STC**) and another format. This translation preserves the semantics of the data.
Functionalities	Produce RDF triples from received data.
Relation with other component	Sources and/or sinks for **CC**.
Use Cases Involved	[38], [67]
Requirements Involved	[180]

3.5.1 Use cases

Use case	IPSM Alignment Configuration
Use Case ID	66
Description	Before utilizing IPSM translation services an appropriate alignment has to be added to the repository to be used in the translation process.
Objectives	IPSM needs configuration before being utilized
Components Involved	REST Manager, Alignment Repository, SPARQL Converter
Requirements Involved	N/A
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-846

DS01 DS2DS IPSM Alignment Configuration

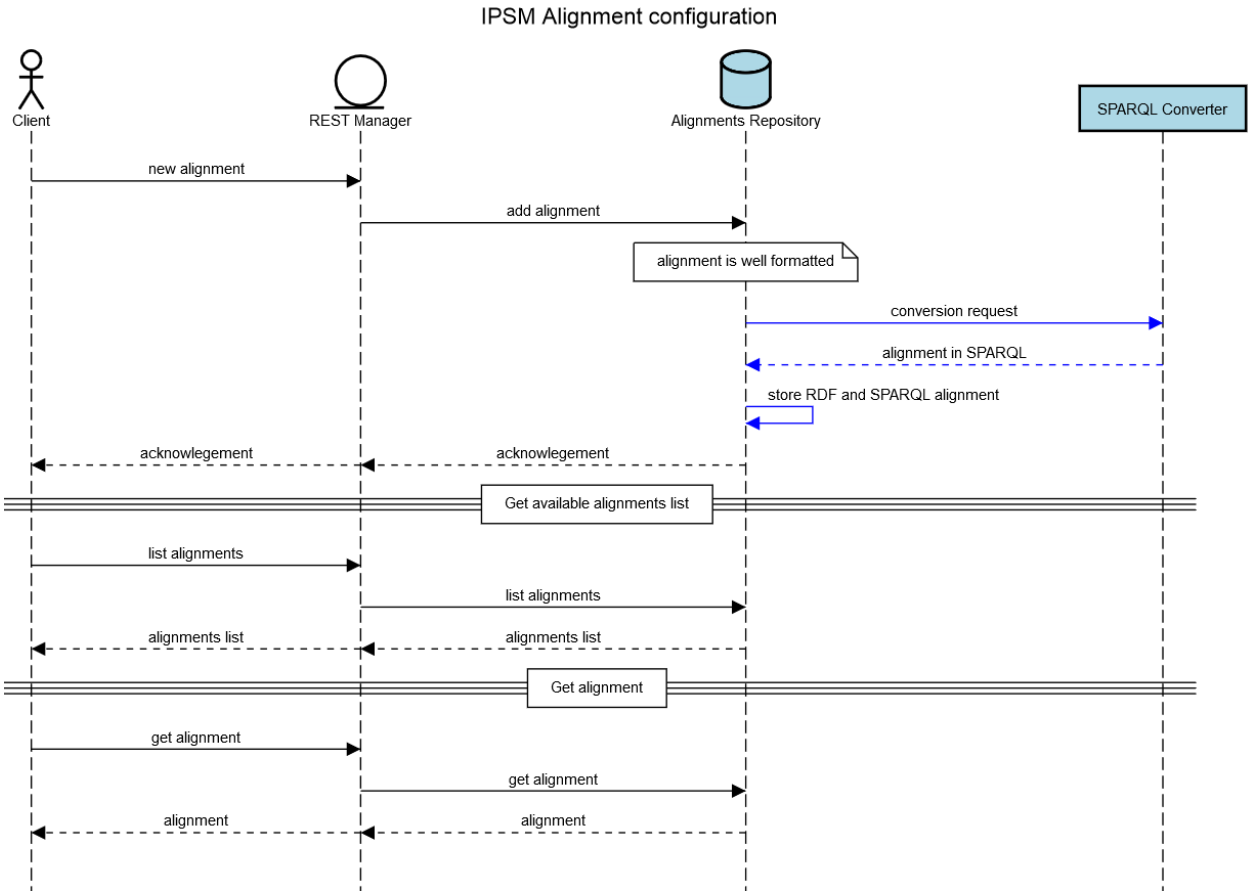

Figure 63: IPSM Alignment configuration.

Alignment configuration enables operations, such as add, list and get, to be performed.

Step 1: A Client sends a new alignment and a request to add it to the repository. Before utilizing IPSM translation services at least one alignment must be added to the Alignments Repository to become available to be used in the translation process.

Step 2: REST Manager (that exposes a dedicated API) receives the request (with the alignment attached) and forwards it to the Repository.

Step 3:The repository calls the SPARQL converter to produce required internal SPARQL representation of the alignment.

Step 4: Assuming that the alignment is well-formatted and no error has occurred, the repository stores the alignment received from client and its SPARQL representation.

Step 5: A message acknowledging the fact of adding a new alignment is returned from Repository, through REST Manager to the client.

The remaining two operations (get an alignment and list alignments) are straightforward in their sequence diagram description.

Use case	IPSM Channel Configuration
Use Case ID	67
Description	IPSM Channel configuration is required before establishing communication, between designated IoT artifacts, which involves semantic translation.
Objectives	IPSM needs configuration before being utilized
Components Involved	REST Manager, Channel Manager, Communication
Requirements Involved	N/A
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-847

DS02 DS2DS IPSM Channel Configuration

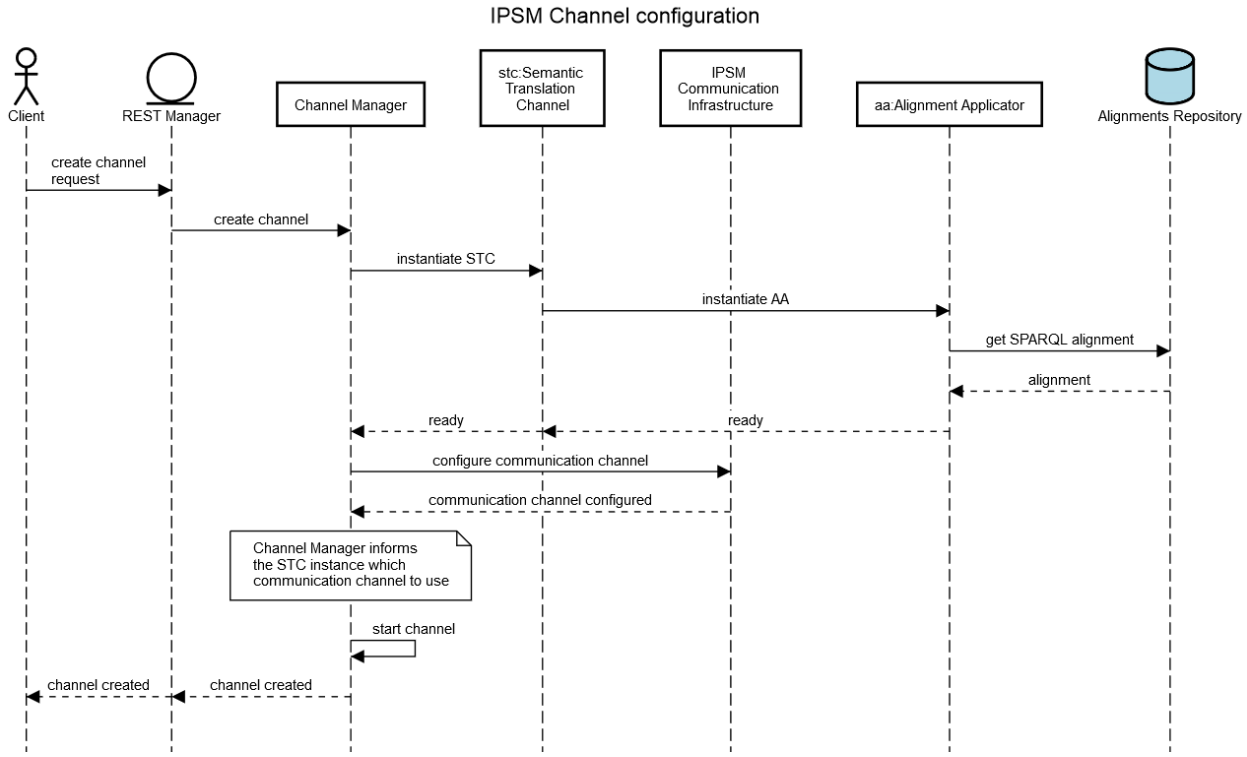

Figure 64: IPSM communication channel configuration.

IPSM Channel configuration is required before establishing communication, between designated IoT artifacts, which involves semantic translation.

Step 1:The client sends a request to the REST Manager to create a new communication channel. It passes configuration information that includes identifiers of used alignments and input/output topics.

Step 2:REST Manager communicates with the Channel Manager, in order to instantiate an **STC**

Step 3: The new **STC**creates a corresponding **AA**

Step 4: AA retrieves alignments form the repository**.**

Step 5: Once AA and STC are ready, the Channel Manager sends a configuration message to the Communication Infrastructure, in order to configure the **CC**. The **STC** instance needs to know which **CC** to use and which alignments to apply. It informs **AA** about the alignments (as part of **AA** instantiation process), so that it can retrieve the right alignments from the repository. Upon successful completion of the process, a **CC** is ready to send and receive messages.

Step 6: Channel Manager returns the information about channel creation to the Client. The CC is now ready to send and receive messages.

Use case	IPSM Translation
Use Case ID	38
Description	Data is exchanged between artifacts that use different semantics. The message should be translated from the semantics of the source platform to the semantics of the target platform.
Objectives	To establish communication between platform / systems that use different semantics.
Components Involved	Semantic Annotator, Communication Channel, Semantic Translation Channel, Alignment Applicator
Requirements Involved	N/A
Use case link	http://jira.inter-iot.eu/browse/INTERIOT-826

DS03 DS2DS IPSM Translation

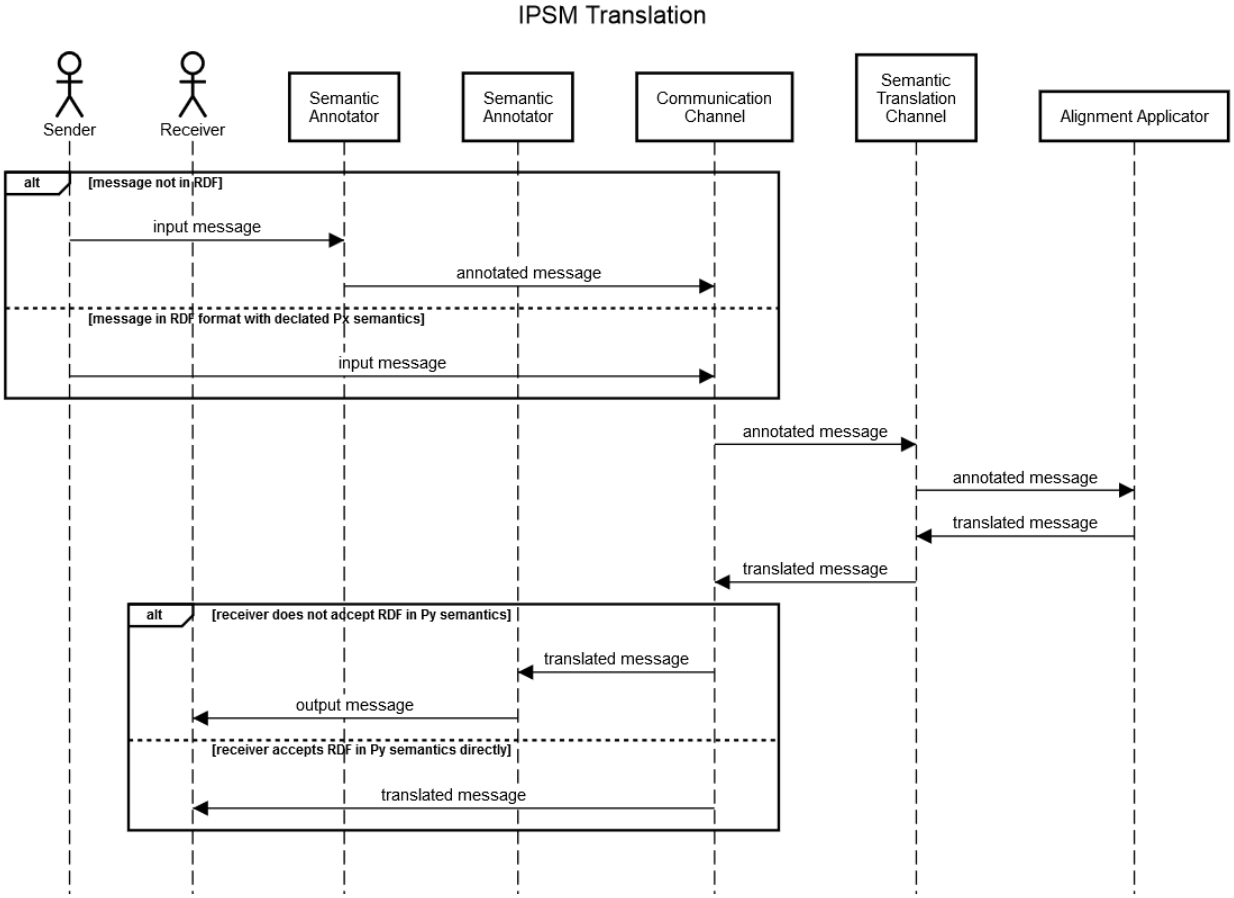

Figure 65: Semantic translation through IPSM communication channels.

IPSM translation is the basic process performed within the DS2DS layer of the INTER-IoT infrastructure. The sequence diagram describes actions to be performed to translate semantics of the input message to the semantics of the output message. Before the communication takes place, a communication channel and alignments need to be configured (see previous sequence diagrams).

Step 1: **SA** receives message from sender entity in a given format and transforms it into RDF triples. A Sender may send messages directly to a **CC**, if they are already annotated.

Step 2:The annotated message is sent to the **CC**

Step 3: CC forwards the message to the corresponding **STC**

Step 4: STC sends the message to the corresponding **AA,** where the semantic translation is performed.

Step 5: The translated message is then sent back, through the **STC** to the **CC**

Step 6: The message travels through another instance of **SA** to the Receiver entity. It should be stressed, that the role of SAs is to convert formats used by sender/ receiver to/from RDF with explicit Px/Py semantics, that are consumed, or received by/from IPSM core. Consequently, the **SA**s are an optional intermediary that is not required if sender or receiver can use RDF with declared semantics directly. In such case, they may directly read and write to communication channels.

3.6 INTER-Layer relation with INTER-Framework

The IoT Interoperability Framework (INTER-FW) aims at providing mechanisms, tools and helper contents to make proper use of the Layer Interoperability Infrastructures (LIIs) and Interoperability Layer Interfaces (ILI).

As described in previous sections, each LII provides generic interoperability means for different identified technology layers in IoT platforms. The INTER-FW provides a way to 1) select the appropriate LIIs for the particular scenario (e.g. middleware and services); 2) abstract generic behaviours present in the most common scenarios (e.g. security administration, device discovery) even though these behaviours apply to different LIIs (e.g. device discovery); 3) extend the generic functionalities provided to the specific cases of the scenario (e.g. extend the service interoperability for a particular natively supported service); 4) provide a single entry point (the framework) to start using the different capabilities of INTER-IoT, with a common, harmonized API with all the documentation in one place; and 5) unify when possible the development process with a homogenized approach to access libraries, develop, configure, deploy and test the software developed with the framework.

Thus, INTER-FW will provide access to INTER-LAYER structures and mechanisms through the APIs provided by the five layers' components, as described in the previous sections. In INTER-IoT, these APIs are exposed only internally (i.e., the ILIs are not accessible directly by an INTER-IoT user or third party), however, a good part of these APIs will be exposed almost identically through the INTER-FW API. For this reason, INTER-FW design and ILIs are processes that depend between them. INTER-FW specifications (especially those coming from the Reference Architecture and the Meta-Data Model tasks have an influence in the ILIs design, and, at the same time, the LIIs capabilities affect the framework definition and the global relation among ILIs.

INTER-FW will rely on different specifications and developments from WP4 that are directly linked to WP3 and INTER-LAYER definitions:

- An Architecture Reference Model for IoT interoperable platforms, whose initial version is described in D4.1. This model is the highest abstraction of features and components that enable the IoT Platform interoperability. A first version is delivered D4.1, based on the theoretical background, the platforms capabilities study and the first hands-on experience with widespread IoT platforms. The results of this specification define a complete set of reference model diagrams, based on existing efforts, for interoperable and open IoT platforms and provide architectural blueprints necessary to build concrete architectures for IoT platforms including INTER-LAYER components. The architectural meta-model follows a top-down approach (model-driven engineering), the definition of such diagrams will allow the creation of specific architectures with well-defined technologies. During next period it is expected a refinement of the current document, including (or merging) new interoperability mechanisms if detected, simplifying when possible the definition and interfaces identified and, finally, providing support for those aspect that still are under study, such as cross-layer interoperability or IoT-specific security mechanisms.
- A metadata-model for IoT interoperable semantics, developed in T4.2 and whose preliminary results are included in D4.1. It provides definition high-level data structures of smart objects & services for interoperable IoT platforms and relationships between data structures. The meta-data model developed in WP4 as part of INTER-FW has a direct link with the IPSM and the GOIoTP (Global Ontology for IoT Platforms) specified and developed in T3.5.

Additionally, the meta-data model of INTER-FW is associated with the semantic needs of the mechanisms included in the five layers of INTER-LAYER.

- A programming API and tools allowing global-level management of the integrated IoT platforms. As already mentioned, the API of INTER-FW will provide also access to the APIs of the different INTER-LAYER components, but it will be complemented by additional functions and tools to manage interoperability between platforms. This API will be defined in D4.3 in collaboration with the LIIs leaders (WP3 task leaders). The use of the INTER-API will allow third parties to develop new application and services compatible with the INTER-IoT paradigm and contribute to the creation of the development ecosystem.

In the following diagram are depicted the modules participating in the INTER-FW definition. Initially, the framework is envisaged as an upper layer consuming the ILIs and providing ways to specialize the generic features provided by LIIs. Thus, the INTER-FW acts sometimes as a mere proxy of the ILIs, while in other cases, it combines, complements and/or particularizes through configuration the behavior of the lower modules. In the framework are located (or abstracted) some essential features that, because of the heterogeneity of the implementation in the LIIs (security, authentication) or simply due to the incapability to implement in such as lower levels (meta-data repositories), are provided at framework level. It also includes framework specific tasks such as configuration, deployment or management.

Finally, all those features are exposed in a common API which tries to offer a homogeneous access to the INTER-IoT features regardless the configuration or particularities of the scenario. APIs management is supported by an adjacent layer and cares about versioning, security, features supported, documentation, testing and accessibility to documentation, among others. API management is a relatively new concept that is growing with new Internet services wave, whose APIs tend to be publicly available, supporting the access to rich and complex contents as those present in the INTER-FW.

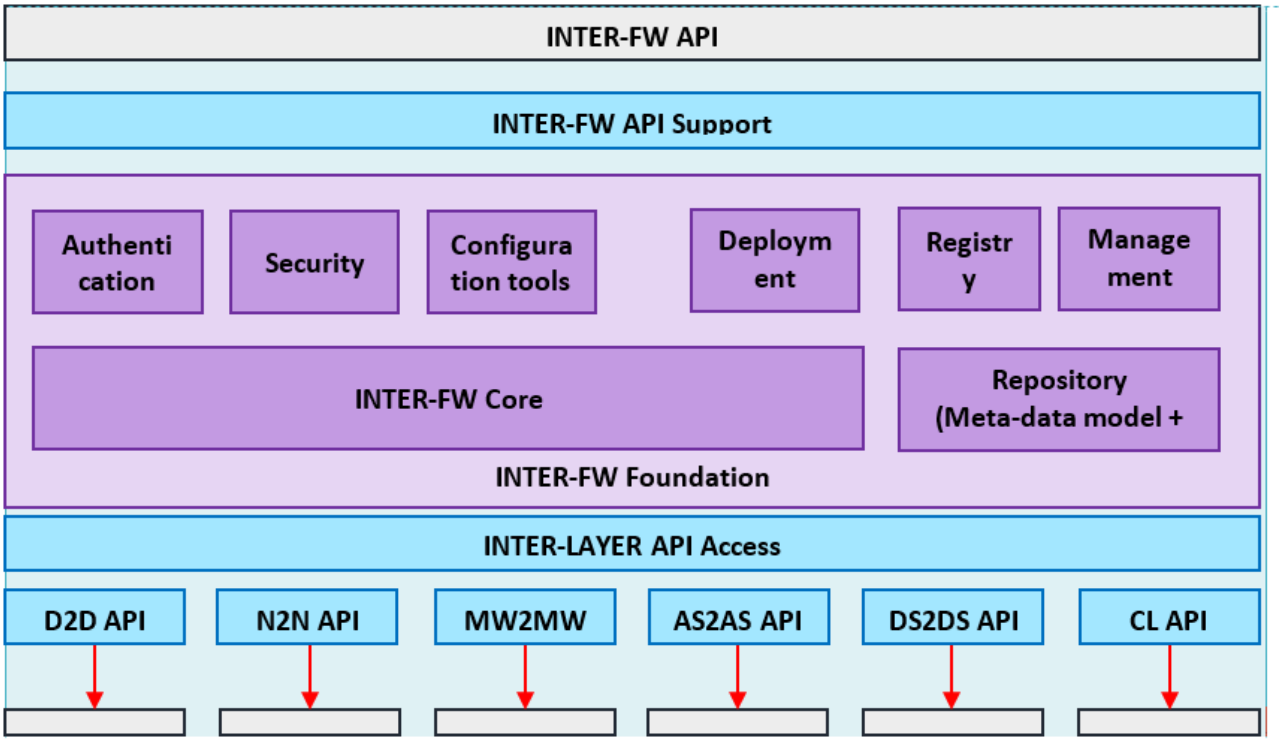

Figure 66: INTER-FW diagram.

As it can be seen in this very first draft design of INTER-FW, there will be an INTER-LAYER API Access layer responsible for the interaction with the bottom layers. Cross-Layer API (CL-API) will be defined as soon as T3.6 starts in M13.

The design of the relationship between INTER-LAYER and INTER-FW will have a double round trip approach. First, each layer will design its own API attending to their specific features with a bottom-up approach, in the sense that INTER-FW will be designed to interact with each interface coming from the bottom layers. Then, from the upper point of view, that is, from INTER-FW point of view, the interfaces with all the layers will be homogenized and reviewed. This redesign will be applied downwards towards the different layers.

4 Conclusions

This document presents the initial collection of INTER-IoT interoperability mechanisms and reports on the first version of the different components, entities and interfaces of the layered approach proposed by the project, and grouped in the product named INTER-LAYER. INTER-IoT requirements that have been derived during an iterative process and based on use cases, scenarios and feedback from stakeholders; have driven the design of INTER-LAYER components. They are related to a wide range of features across the IoT stack, from smart devices and gateways to cloud-based platform components and applications, mobility, and data semantics.

The document sets the foundations of the INTER-IoT functional components of INTER-LAYER as described in the Description of Activity in the context of various interoperability aspects which are being supported by the INTER-IoT interoperability proposal (syntactic, semantic and layered interoperability).

INTER-IoT defines an interoperability framework for IoT platforms and thus does not strive to become another standard IoT platform. INTER-IoT does not store any sensor-generated data outside of IoT platform boundaries, but rather acts as a mediator between different IoT platforms ensuring secure and uniform access to platform resources through well-defined interfaces. The definition of APIs in the different layers of INTER-LAYER allows for the restriction of interoperability to a certain layer of the IoT platforms involved. It makes the interoperability flexible and allows it to reflect the interests/needs of the stakeholders, integrators or application developers.

The functional structure of INTER-Layer is built using a layered stack and defines five domains: Device, Network, Middleware, Applications and Services and Data and Semantics. It is motivated by different analyzed interoperability proposals as indicated in the state of the art. Device to Device layer is based on the use of an IoT gateway, which will bring connectivity to devices using different access networks and legacy gateways and that virtualizes the network intelligence and APIs to access the upper layers. Network to network layer is based on the use of SDN, SDR and NFV and will be linked to the virtualized component of the gateway allowing to manage both interoperability layer in the cloud. The network layer will provide support for mobility and offloading. Middleware to middleware interoperability is implemented in the way of a super-middleware that allows transparent access to different IoT platforms, both existing and newly developed ones. It will flexibly be integrated in INTER-LAYER. Application and services to application and services interoperability allows the orchestration and connectivity between services from different IoT platforms, and will be based on a customized version of Node-RED, that is a common approach used by the IoT-EPI projects. Finally, Data and semantics to Data and semantics layer brings the GoIoTP Ontology that will extend different state of the art IoT ontologies as indicated and has developed the IPSM, that will be used by the different layers.

Our approach incorporated different novel contributions for example: the virtual gateway that may allow integration of IoT interoperability with operators' data centers; link between SDN/NFV with IoT and support for interoperability at network layer, which has been also proposed as link between IoT and 5G and the introduction of the semantic mediator. Additionally, INTER-LAYER includes some state-of-the-art components that will be used for example for application and service interoperability (e.g. customized version of Node-RED for IoT platforms).

In this document we have focused on defining the components of the prioritized areas from the requirements. Thus, the focus has been placed in device to device, middleware to middleware and

data and semantics interoperability, however network to network and application and services to application and services have also been developed, but considering that both may need inputs respectively from the device to device and middleware to middleware layers.

Developments have already started and there are first releases of the virtual gateway, MW2MW and IPSM. In the next phase that will be described in D3.2, a second version of every layer and the integration of the gateway and the network component, the middleware component, and the applications and services component will be provided. Adjustments in the different layers will be done in order to achieve performance and scalability. Additionally, as the cross-layer interoperability task will start in M13 all the functions assigned to this task like QoS, security, privacy and trust will be included in the following release of INTER-LAYER. And finally as WP4 and WP5 activity progresses the next release of the deliverable will provide further integration with INTER-FW and INTER-METH. Both products are under design in the moment of releasing this deliverable and INTER-LAYER has served as input to them.

5 References

[1] K. Waters, Prioritization using moscow, Agile Planning 12, 2009.

[2] N. Omnes, M. Bouillon, G. Fromentoux and O. Le Grand, “A programmable and Virtualized Network & IT Infrastructure for the Internet of Things,” *2015 18th International Conference on Intelligence in Next Generation Networks,* 2015.

[3] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, “Software-Defined Networking: A Comprehensive Survey,” *Proceedings of the IEEE,* vol. 103(1), pp. 14-76, 2015.

[4] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka and T. Turletti, “A survey of software-defined networking: Past, present, and future of programmable networks,” *IEEE Communications Surveys & Tutorials,* vol. 16(3), pp. 1617-1634, 2014.

[5] N. McKeown and G. Parulkar et al., OpenFlow: Enabling Innovation in Campus Networks, Stanford University, March 14, 2008.

[6] *IEEE Standard for Local and metropolitan area networks--Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer,* IEEE Std. 802.15.4e-2012, 2012.

[7] *IEEE Standard for Local and metropolitan area networks--Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs),* IEEE Std. 802-15.4-2011, 2011.

[8] A. Tinka, T. Watteyne and K. Pister, “A decentralized scheduling algorithm for time synchronized channel hopping,” in *International Conference on Ad Hoc Networks*, Springer, 2010.

[9] N. Accettura, E. Vogli, M. R. Palattella, L. A. Grieco, G. Boggia and M. Dohler, “Decentralized traffic aware scheduling in 6TiSCH networks: Design and experimental evaluation,” *IEEE Internet of Things Journal,* vol. 2, no. 6, pp. 455-470, 2015.

[10] *RFC 6550 - RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks,* IETF, 2012.

[11] R. Soua, P. Minet and E. Livolant, “Wave: a distributed scheduling algorithm for convergecast in IEEE 802.15.4e TSCH networks,” *Transactions on Emerging Telecommunications Technologies,* 2015.

[12] S. Duquennoy, B. Al Nahas, O. Landsiedel and T. Watteyne, “Orchestra: Robust mesh networks through autonomously scheduled TSCH,” *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems,* 2015.

[13] Q. Liu, X. Wang and G. Giannakis, “A cross-layer scheduling algorithm with QoS support in wireless networks,” *IEEE Transactions on vehicular Technology,* vol. 55(3), pp. 839-847, 2006.

[14] B. Ji, C. Joo and N. B. Shroff, “Delay-based back-pressure scheduling in multihop wireless networks,” *IEEE/ACM Transactions on Networking,* vol. 21(5), pp. 1539-1552, 2013.

[15] B. Li, A. Eryilmaz and R. Srikant, “On the universality of age-based scheduling in wireless networks,” *2015 IEEE Conference on Computer Communications (INFOCOM),* 2015.

[16] A. Saifullah, J. Li, K. Agrawal, C. Lu and C. Gill, “Multi-core real-time scheduling for generalized parallel task models,” *Real-Time Systems,* vol. 49(4), pp. 404-435, 2013.

[17] M. J. Neely, “Delay-based network utility maximization,” *IEEE/ACM Transactions on Networking (TON),* vol. 21(1), pp. 41-54, 2013.

[18] M. R. Palattella, N. Accettura, L. Grieco, G. Boggia, M. Dohler and T. Engel, “On Optimal Scheduling in Duty-Cycled Industrial IoT Applications Using IEEE802.15.4e TSCH,” *IEEE Sensors Journal,* vol. 13, no. 10, 2013.

[19] O. D. Incel, A. Ghosh, B. Krishnamachari and K. Chintalapudi, “Fast data collection in tree-based wireless sensor networks,” *IEEE Transactions on Mobile computing,* vol. 11, no. 1, pp. 86-99, 2012.

[20] Y. Wu, J. A. Stankovic, T. He and S. Lin, “Realistic and efficient multi-channel communications in wireless sensor networks,” *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE,* 2008.

[21] R. Soua, P. Minet and E. Livolant, “MODESA: An optimized multichannel slot assignment for raw data convergecast in wireless sensor networks,” *2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC). IEEE,* 2012.

[22] S. Kuhlins and R. TredWell, “Toolkit for Generating Wrappers – A seurvey of Software Toolkit for Automated Data Extraction from Web Sites,” *Net.ObjectsDays,* 2002.

[23] A. H. Laender et al., “A brief Survey of Web Data Extraction Tools,” *ACM SIGMOD Record,* pp. 84-93, 2002.

[24] A. Firat, “Information Integration Using Contextual Knowledge and Ontology Merging,” *MIT Sloan PhD thesis,* 2003.

[25] G. Huck, P. Fankhauser, K. Aberer and E. Neuhold, “Jedi:Extracting and Synthesizing Information from the Web,” *German National Research Center for Information Technology Integrated Publication and Information Systems Institute IPSI, Dolivostr. 15, 64293 Darmstadt, Germany.*

[26] M. E. Califf and R. J. Mooney, “Relational Learning of Pattern Match Rules for Information Extraction,” in *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, Orlando, FL, 1999, pp. 328-334.

[27] A. Soderland and F. Azavant, “Learning information extraction rules for semi-structured and free text,” *Machine learning,* Vols. 34 (1-3), pp. 233-272, 1999.

[28] N. Kushmerick and B. Grace, “The Wrapper Induction Environment,” *AAAI Technical Report WS-98-10.*

[29] I. Muslea, S. Minton and C. Knoblock, “Hierarchical wrapper induction for semistructured information sources,” *Autonomous Agents and Multi-Agents Systems,* vol. 4, pp. 93-114, 2001.

[30] B. Adelberg, “A Tool for Semi-Automatically Extracting Structured and Semi-Structured data from text documents,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1998, pp. 283-294.

[31] D. Pallmann, Network Query Language, Wiley, 2002.

[32] J. Fujima and K. P. Jankte, “Web Service Wrapping Technologies for Customizable Consumer Electronics,” *IEEE 3rd Global Conference on Consumer Electronics (GCCE),* October 2014.

[33] M. N. Kuwahara and Y. Tanaka, “Webbles:Programmable and customizable meme media objects in a knwoledge federation framework environment on the web,” *Second Intl. Workshop on Konowledge Federation,* Oct. 2010.

[34] “FI-WARE Applications/Services Ecosystem and Delivery Framework,” [Online]. Available: https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Applications/Services_Ecosystem_and_Delivery_Framework. [Accessed 15 December 2016].

[35] K. Tollmar, F. Bentley and C. Viedma, “Mobile Health Mashups: Making sense of multiple streams of wellbeing and contextual data for presentation on a mobile device,” *6th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth) and Workshops,* 2012.

[36] F. Montesi, “Choreographic Programming,” Ph.D. thesis, IT University of Copenhagen, 2013. [Online]. Available: http://www.fabriziomontesi.com/files/choreographic_programming.pdf.

[37] “Service Orchestration,” [Online]. Available: https://github.com/universAAL/service/wiki/Service-Orchestration. [Accessed 15 Dec. 2016].

[38] M. E. Cambronero, G. Díaz, E. Martínez and V. Valero, “A Comparative Study between WSCI, WS-CDL, and OWL-S,” *2009 IEEE International Conference on e-Business Engineering (ICEBE 2009) Macau, China,* pp. 377-382, 2009.

[39] J. Mendling and M. Hafner, “From WS-CDL choreography to BPEL process orchestration,” *Journal of Enterprise Information Management,* vol. 21, no. 5, pp. 525 - 542, 2008.

[40] “Internet of the Things Architecture, Orchestration of distributed IoT service interactions,” [Online]. Available: http://www.meet-iot.eu/deliverables-IOTA/D2_3.pdf.

[41] T. R. Gruber, «Toward principles for the design of ontologies used for knowledge sharing.,» *International Journal of Human-Computer Studies,* vol. 43, nº 4-5, pp. 907-928, 1995.

[42] “Resource description framework (RDF),” [Online]. Available: https://www.w3.org/RDF.

[43] “RDF schema 1.1,” 2014. [Online]. Available: https://www.w3.org/TR/rdf-schema.

[44] “Web Ontology Language,” [Online]. Available: https://www.w3.org/TR/owl2-overview.

[45] “SPARQL 1.1 overview,” [Online]. Available: https://www.w3.org/TR/sparql11-overview.

[46] “oneM2M standards for M2M and the Internet of Things,” [Online]. Available: http://www.onem2m.org.

[47] “Fiware openstack-based cloud environment and open APIs for Internet of Things,” [Online]. Available: https://www.fiware.org.

[48] “Gambas generic adaptive middleware for behavior-driven autonomous services,” [Online]. Available: http://www.gambas-ict.eu.

[49] “IoT.est Internet of Things environment for service creation and testing,” [Online]. Available: http://ict-iotest.eu/iotest.

[50] S. De, T. Elsaleh, P. M. Barnaghi and S. Meissner, “An Internet of Things platform for real-world and digital objects,” *Scalable Computing: Practice and Experience,* vol. 13 (1).

[51] Internet of Things Success Stories, Series 1-3, Internet of Things European Research Cluster (IERC) and Smart Action, 2014-2015. URL http://www.smart-action.eu/publications, P.Cousin.

[52] P. M. Barnaghi, W. Wang, C. A. Henson and K. Taylor, “Semantics for the Internet of Things: Early progress and back to the future,” *International Journal on Semantic Web and Information Systems,* vol. 8 (1), pp. 1-21, 2012.

[53] C. Perera, C. H. Liu, S. Jayawardena and M. Chen, “Context-aware computing in the Internet of Things: A survey on Internet of Things from industrial market perspective,” *URL http://arxiv.org/abs/1502.00164.*

[54] “Csiro sensor ontology,” [Online]. Available: http://www.w3.org/2005/Incubator/ssn/wiki/SensorOntology2009.

[55] “Swamo ontology,” [Online]. Available: http://www.w3.org/2005/Incubator/ssn/wiki/Review_of_Sensor_and_Observations_Ontologies#SWAMO.

[56] A. Underbrink, K. Witt, J. Stanley and D. Mandl, “Autonomous mission operations for sensor webs,” *AGU Fall Meeting Abstracts C5,* 2008.

[57] “Mmi device ontology,” [Online]. Available: https://marinemetadata.org/community/teams/ontdevices.

[58] “Seek extensible observation ontology,” [Online]. Available: https://semtools.ecoinformatics.org/oboe.

[59] C. A. Henson, J. K. Pschorr, A. P. Sheth and K. Thirunarayan, “SemSOS: Semantic sensor observation service,” *Proc. of the 2009 International Symposium on Collaborative Technologies and Systems (CTS 2009),* pp. 44-53. URL http://knoesis.wright.edu/library/publications/C%2B09-Sem SOS_CTS.pdf, 2009.

[60] K. Janowicz and M. Compton, “The stimulus-sensor-observation ontology design pattern and its integration into the semantic sensor network ontology,” *Proc. 3rd International*

Workshop on Semantic Sensor Networks 2010 (SSN10) in conjunction with the 9th International Semantic Web Conference (ISWC 2010), Shanghai, China, pp. 64-78. URL http://ceur-ws.org/Vol-668/paper12.pdf, 2010.

[61] "Sensor Model Language (SensorML)," [Online]. Available: http://www.opengeospatial.org/standards/sensorml.

[62] "SSN Ontology," [Online]. Available: http://www.w3.org/2005/Incubator/ssn/ssnx/ssn.

[63] "Semantic Sensor Network XG Annual report (2011)," [Online]. Available: http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628.

[64] M. Compton, P. Barnaghi, L. Bermudez, R. Garcia-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W. D. Kelsey, D. L. Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth and K. Taylor, "The SSN ontology of the W3C semantic sensor network incubator group," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 17, pp. 25-32. URL http://www.websemanticsjournal.org/index.php/ps/article/view/312, 2012.

[65] "DOLCE+DnS Ultralite," [Online]. Available: www.ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite.

[66] R. Bendadouche, C. Roussey, G. D. Sousa, J. Chanet and K. M. Hou, "Extension of the semantic sensor network ontology for wireless sensor networks: The stimulus-wsnnode-communication pattern," *Proc. 5th International Workshop on Semantic Sensor Networks, SSN12, Boston, MA, USA,* vol. 904, pp. 49-64. URL http://ceur-ws.org/Vol-904/paper5.pdf, 2012.

[67] H. Müller, L. Cabral, A. Morshed and Y. Shu, "From RESTful to SPARQL: A case study on generating semantic sensor data," *Proc. 6th International Conference on Semantic Sensor Networks Volume 1063, SSN'13, CEUR WS.org, Aachen, Germany,* pp. 51-66. URL http://dl.acm.org/citation.cfm?id=2874543.2874547, 2013.

[68] G. Atemezing, O. Corcho, D. Garijo, J. Mora, M. Poveda-Villalón, P. Rozas, D. Vila-Suero and B. Villazón-Terrazas, "Transforming meteorological data into Linked Data," *Semantic Web,* vol. 4, no. 3, pp. 285-290, 2013.

[69] C. Wang, N. Chen, C. Hu, S. Yan and W. Wang, "A general sensor web resource ontology for atmospheric observation," *Proc. IEEE International Geoscience and Remote Sensing Symposium, IGARSS 2011, Vancouver, BC, Canada, IEEE,* pp. 3436-3439, 2011.

[70] A. J. G. Gray, R. García-Castro, K. Kyzirakos, M. Karpathiotakis, J. -P. Calbimonte, K. Page, J. Sadler, A. Frazer, I. Galpin, A. A. A. Fernandes, N. W. Paton, O. Corcho, M. Koubarakis, D. Roure, K. Martinez and A. Gómez Pérez, "A semantically enabled service architecture for mashups over streaming and stored data," *The Semantic Web: Research and Applications: 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, Proceedings, Part II, Springer,* pp. 300-314. doi:10.1007/978-3-642-21064-8_21. URL http://dx.doi.org/10.1007/978-3-642-21064-8_21, 2011.

[71] "Stream Annotation Ontology," [Online]. Available: http://iot.ee.surrey.ac.uk/citypulse/ontologies/sao/sao.

[72] S. Kolozali, M. Bermudez-Edo, D. Puschmann, F. Ganz and P. Barnaghi, “A knowledge-based approach for real-time iot data stream annotation and processing,” *Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing (CPSCom), IEEE,* pp. 215-222, 2014.

[73] “IoT lite Ontology,” [Online]. Available: http://www.w3.org/Submission/iot-lite.

[74] “Smart appliances reference ontology (saref),” [Online]. Available: https://sites.google.com/site/smartappliancesproject/ontologies/reference-ontology.

[75] “oneM2M ontologies,” [Online]. Available: http://www.onem2m.org/technical/onem2m-ontologies.

[76] “Fiesta-IoT Project,” [Online]. Available: http://fiesta-iot.eu.

[77] “Fiesta-IoT achieves semantic interoperability between FIWARE and OneM2M,” [Online]. Available: http://fiesta-iot.eu/fiesta-achieves-semantic-interoperability-between-fiware-and-onem2m.

[78] J. Soldatos, N. Kefalakis, M. Hauswirth, M. Serrano, J. -P. Calbimonte, M. Riahi, K. A. Aberer, P. P. Jayaraman, A. Zaslavsky, I. Podnar Karko, L. Skorin-Kapov and R. Herzog, “OpenIoT: Open source Internet-of-Things in the cloud,” in *Interoperability and Open-Source Solutions for the Internet of Things*, Springer, 2015, pp. 13-25.

[79] J. Calbimonte, S. Sarni, J. Eberle and K. Aberer, “XGSN: an open-source semantic sensing middleware for the web of things,” in *Joint Proceedings of the 6th International Workshop on the Foundations, Technologies and Applications of the Geospatial Web, TC 2014, and 7th International Workshop on Semantic Sensor Networks, SSN 2014, co-located with 13th International Semantic Web Con*, Riva del Garda, Trentino, Italy, 2014, pp. 51-66.

[80] K. Aberer, M. Hauswirth and A. Salehi, “A middleware for fast and flexible sensor network deployment,” in *Proceedings of the 32nd international conference on Very large data bases*, VLDB Endowment, 2006, pp. 1199-1202.

[81] C. Stadler, J. Lehmann, K. Höhner and S. Auer, “LinkedGeoData: A core for a web of spatial open data,” *Semantic Web Journal,* vol. 3(4), pp. 333-354, 2012.

[82] “Basic Geo (WGS84 lat/long) vocabulary,” [Online]. Available: https://www.w3.org/2003/01/geo.

[83] D. Le-Phuoc, H. Quoc, J. Parreira and M. Hauswirth, “The linked sensor middleware-connecting the real world and the semantic web,” *Semantic Web Challenge (ISWC),* 2011.

[84] L. Otero-Cerdeira, F. J. Rodríguez-Martínez and A. Gómez-Rodríguez, “Ontology matching: A literature review,” *Expert Systems with Applications,* vol. 42(2), pp. 949-971, 2015.

[85] “50 Ontology Mapping and Alignment Tools,” [Online]. Available: http://www.mkbergman.com/1769/50-ontology-mapping-and-alignment-tools.

[86] “50 Ontology Matching,” [Online]. Available: http://ontologymatching.org/projects.html.

[87] N. Duyhoa and Z. Bellahsene, “Overview of YAM++ -(Not) Yet Another Matcher for ontology alignment task,” *Research report . URL http://hal-lirmm.ccsd.cnrs.fr/lirmm-01079124,* 2014.

[88] E. Jiménez-Ruiz and B. C. Grau, “LogMap: Logic-based and scalable ontology matching,” in *International Semantic Web Conference (ISWC)*, Springer, 2011, pp. 273-288.

[89] E. J. Ruiz, B. C. Grau, Y. Zhou and I. Horrocks, “Large-scale interactive ontology matching: Algorithms and implementation,” in *Proc. of the 20th European Conference on Artificial Intelligence (ECAI)*, 2012, pp. 444-449.

[90] W. F. Dowling and J. H. Gallier, “Linear-time algorithms for testing the satisfiability of propositional Horn formulae,” *The Journal of Logic Programming,* vol. 1(3), pp. 267-284, 1984.

[91] D. Aumueller, H. -H. Do, S. Massmann and E. Rahm, “Schema and ontology matching with COMA++,” in *Proc. of the 2005 ACM SIGMOD Int. Conf. on Management of Data*, 2005, pp. 906-908.

[92] D. Faria, C. Pesquita, E. Santos, M. Palmonari, I. F. Cruz and F. M. Couto, “The Agreement Maker Light ontology matching system,” in *OTM 2013 Conferences “On the Move to Meaningful Internet Systems”*, Springer, 2013, pp. 527-541.

[93] D. Faria, C. Martins and A. Nanavaty, “AgreementMakerLight results for OAEI 2014,” in *ISWC Int. Workshop on Ontology Matching (OM), CEUR Workshop Proceedings*, 2014.

[94] D. Faria, C. Martins, A. Nanavaty, D. Oliveira, B. S. Balasubramani, A. Taheri, C. Pesquita, F. M. Couto and I. F. Cruz, “AML results for OAEI 2015,” in *ISWC Int. Workshop on Ontology Matching (OM), CEUR Workshop Proceedings*, 2015.

[95] J. David, J. Euzenat, F. Scharfle and C. Trojahn dos Santos, “The alignment API 4.0,” *Semantic Web,* vol. 2 (1), pp. 3-10, 2011.

[96] J. Volz, C. Bizer, M. Gaedke and G. Kobilarov, “Silk-A Link Discovery Framework for the Web of Data,” *LDOW,* vol. 538, 2009.

[97] F. Giunchiglia, A. Autayeu and J. Pane, “S-Match: An open source framework for matching lightweight ontologies,” *Semantic Web,* vol. 3 (3), pp. 307-317, 2012.

[98] G. Fortino, R. Gravina, W. Li and C. Ma, “Using Cloud-assisted Body Area Networks to Track People Physical Activity in Mobility,” in *Proc. of the International Conference on Body Area Networks (BodyNets 2015)*, Australia, Sep 2015, pp. 85-91.

[99] R. Gravina, C. Ma, P. Pace, G. Aloi, W. Russo, W. Li and G. Fortino, “Cloud-based Activity-aaService cyberphysical framework for human activity monitoring in mobility,” *Future Generation Computer Systems,* Available online 22 September 2016, DOI: 10.1016/j.future.2016.09.006.

[100] G. Fortino and R. Gravina, “A Cloud-Assisted Wearable System for Physical Rehabilitation,” in *ICTs for Improving Patients Rehabilitation Research Techniques*, vol. 515, Nov 2015, pp. 168-182.

[101] G. Fortino, R. Gravina, A. Guerrieri and G. Di Fatta, “Engineering Large-Scale Body Area Networks Applications,” in *Proc. of the International Conference on Body Area Networks (BodyNets 2013)*, Boston, United States, ACM press, 2013, pp. 363-369.

[102] G. Fortino, S. Galzarano, R. Gravina and W. Li, “A framework for collaborative computing and multi-sensor data fusion in body sensor networks,” *Information Fusion,* vol. 22, pp. 50-70, 2015.

[103] A. Ricauda, N. G. Isaia, V. Tibaldi, G. Bestente, A. Frisiello, A. Sciarappa, S. Cavallo, M. Ghezzi and G. Larini, “Telecare and Telemedicine in Home Care Practice: Field Trial Results,” in *Distributed Diagnosis and Home Healthcare (D2H2)*, vol. 2, American Scientific Publishers (ASP), 2011, pp. 281-303.

[104] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo and C. Savaglio, “Enabling IoT interoperability through opportunistic smartphone-based mobile gateways,” *Journal of Network and Computer Applications,* Available 20 October 2016, DOI: 10.1016/j.jnca.2016.10.013.

[105] World Health Organization, Obesity: Preventing and Managing the Global Epidemic. WHO Obesity Technical Report Series 894, Geneva, Switzerland, 2000.

[106] World Health Organization, Physical Status: The Use and Interpretation of Anthropometry. Technical Report Series 854, Geneva, Switzerland, 1995.

[107] World Health Organization, Global status report on non communicable diseases, 2010.

[108] F. Bellifemine, G. Fortino, A. Guerrieri and R. Giannantonio, “Platform-independent development of collaborative Wireless Body Sensor Network applications: SPINE2,” *IEEE International Conference on SMC,* pp. 3144-3150, 2009.

[109] H. Arslan, Cognitive Radio, Software Defined Radio, and Adaptive Wireless Systems, Springer, 2007.

6 Annex

In this section we present complementary data mentioned in the other sections of this document.

6.1 INTER-LogP pilot requirements table

ID	Name	Type	Category	MoScow priority
248	Create new services to access different platforms	Interoperability	Non-functional	Must
	Access to resources and services of a virtual entity from another IoT platform or application when certain rules are met. Modification of services or creation of new ones that take advantage of shared information.			
193	Allow communication between legacy systems	Operational	Functional	Must
	Connect legacy systems with new services through standard based protocol gateways to free data from proprietary constraints.			
247	Need of object virtualization	Operational	Non-functional	Must
	The IoT platforms use virtual objects of its physical entities for managing data from different sources. They can share these virtual objects or part of them with other IoT platforms when necessary. Capacity to define access rules to different attributes of the virtual entity among platforms is required.			
249	Semantic interoperability among platforms	Semantics, Interoperability	Non-functional	Must
	The data provided by an IoT platform to another IoT platform must be understandable for the receiver platform.			
54	High response time	Communications	Non-functional	Should
	High time responses for accessing data should be established.			
84	Priority in alarms	Functionality	Non-functional	Should
	Alarms should go in priority way, not more than a second, and launch triggers.			
168	Provide an alert system	Functionality	Functional	Should
	INTER-IoT needs to provide an alert system among heterogeneous IoT platforms associated with a subscription system (requirement 201) that will notify events when the attributes of a virtual entity change according to predefined values or ranges.			

252	IoT platforms are able to stop sharing an object at any moment	Interoperability	Functional	Should
	The road haulier company is able, at any moment, to finalize the connection with the port IoT platform if it decides to do that.			
166	Detection of passive physical entities to start communication with other platforms	Interoperability	Functional	Should
	When you have multiple passive physical entities, you need a mechanism for quick identification of objects that ensure you know where they are at all times. This allows to identify entities in any environment.			
194	Provide exchange of virtual objects between platforms	Interoperability	Functional	Should
	INTER-IoT need to provide that a company shares a virtual object with other company when the physical object is on its facilities. You can share the whole virtual object or a part of it.			
167	Provide services to associate and link two virtual entities	Interoperability	Functional	Should
	INTER-IoT needs to provide services to associate and link two virtual entities handled by different and heterogeneous IoT platforms when they are in proximity. So that they can exchange information immediately between them. You can also disassociate the virtual entities.			
195	Provide the creation and monitoring of geofences	Operational	Functional	Should
	There are actions that must be performed when an object enters or leaves an area. Therefore, there must be a mechanism to detect it, by using geofences.			
197	Beacons to request the communication from other platforms and devices	Communications	Functional	Could
	There are some objects that need to send data to nearby devices. This communication may be indoor or outdoor and low energy consumption, as it will not have access to a power supply.			
79	Service to manage energy consumption of devices	Feature	Functional	Could
	The framework provides methods for energy management (status, enable/disable, power saving mode, etc.) to end users, when native platforms allow it.			
198	Capacity to achieve a heterogeneous computing platform environment	Functionality	Functional	Could
	In INTER-IoT are several platforms, and each has multiple devices or sensors. This generates a lot of information that must be stored and processed. Therefore it is needed			

	tools that enable processing a large amount of data from several different platforms, such as Big Data tools.			
246	Identification of an object through multiple techniques	Operational	Functional	Could
	There should be the possibility of identify an object through different techniques, giving priority to one of them.			
81	Services should provide Quality of Service	QoS	Non-functional	Could
	In the INTER-IoT project there are two business use cases related to health and logistics and transport. In the case of health, if communication fails, a patient may suffer serious consequences. In the logistics case, you can assume considerable losses for a company. It is therefore essential to ensure the quality of services. In addition, safety and security could also be in risk when quality of services is not guaranteed (e.g. dangerous goods inside container boxes).			
139	Multiple interface options	Usability	Non-functional	Could
	The system shall provide the users with several user responsive interfaces, according to the individual needs and preferences of a user or the situation the user is in. Possible types of user interface are: - graphical user interfaces over all kind of devices (smart phones, tablet PCs, video screens, virtual reality glasses, touch screens etc.). - geographical representations of data like map representations. - audio interfaces, both for alarms/notifications and for interaction.			
196	Position detection of objects through WiFi	Operational	Functional	Won't
	The need of getting the position of the objects with accuracy and reliability becomes necessary to detect the position through different mechanisms. Therefore, the WiFi signal received will be use at different access points to calculate the position of the object, as a complement to other methods such as GPS.			

Table 10: INTER-LogP pilot minimum requirements.

6.2 INTER-Health pilot requirements table

ID	Name	Type	Category	MoScow priority
71	Application response time	Functionality	Non-functional	Must

	The "navigation" functionalities on different contents by using both Smartphone or Personal Computer to access to the platform, have to guarantee a response time of a few seconds.			
127	Availability of sensor data	Functionality	Non-functional	Must
	Health monitoring data must be viewable from a remote location to facilitate patient triage and inform decision making.			
176	User Access Gateway for Patients	Operational	Functional	Must
	Gateway main functionalities for patients are: • Access to services (providing username and password). • Setting Profile communication and devices pairing. • Managing measures on the device and releasing them to the gateway which stores them on a local database. • Possibilities of inserting measures manually. • Sending measures to the platform. • Reporting locally measures already stored. • In term of interoperability, the INTER-Health gateway uses the gateway architectural scenarios described in the requirement 175.			
146	Information sheet. Processing of personal data	Privacy	Non-functional	Must
	The person tasked with processing must provide the information sheet, to the involved person or the person from whom you collect the data, in the manner determined by the data processor of the structure and using the forms, where prepared by the Company. During the health use case, the information sheet shows in a simple but detailed way: • Purpose of the study; • Detailed study protocol; • Advantages and the risks involved; • Any costs or compensation for subjects who choose to participate; • Voluntary participation; • Right of withdrawal at any time; • References of the person responsible of the project.			
145	Informed consent. Processing of personal data	Privacy	Non-functional	Must
	The informed consent must be: • freely express, • specifically performance,			

	• must be known by the involved person, of legal age, not banned and able to understand or want. For different reasons of incapacity or inability, the privacy code has the legitimacy of one of the following subjects: 1. operator of the power, in the case of under-age or person prohibited or subject to support administration; 2. family, close relative or partner (all placed on the same level) for cases of physical impossibility or inability to understand or want the individual; 3. residually, the Data Processor of the institution where is the involved subject.			
218	Personal data collected on Computerized Nutritional Folder during Experimental and Traditional Nutritional Counselling	Privacy	Functional	Must
	The data of the recruited subjects, collected on electronic nutritional folder refer to: • Personal and identification data • Anthropometric data (weight, height, BMI, waist circumference) • Food research (eating habits and physical activity)			
103	User Authentication to access INTER-Health services	Security	Non-functional	Must
	Nowadays users shall authenticate to the services using their username and password (see Non Functional Requirements); if needed, a stronger way of authentication will be implemented. To guarantee the correct identification of the INTER-Health user (such as Patients and Sanitary staff) but also the technical addicts such as platforms administrators.			
106	Definition of reference meaning for health information	Semantics	Functional	Must
	Health information can be detected using different devices according to different way of measurement (unit of measure that could differ from country to country and also depending on devices manufacturers). To use same information coming from different systems and going to others, it is mandatory to establish specific criteria to: • Define a common meaning if it is possible. • Determine a correspondence between different data that have the same meaning and different values. • Set transcoding tables between different values of the same data.			
104	Personal data and user profile management	Data model	Functional	Should
	User data and provisioning will be based on: • A set of identification data (such as surname, name, tax code, country and so on). • Role and profiling.			

	• Contact data and addresses. • Anthropometric and health information. Users can be recorded both locally (on the owner platform) or in the Cloud (on one or more client platforms).			
164	Medical Device informatics	Functionality	Non-functional	Should
	ISO/TC 215 Health informatics sets international standards for medical data transfer.			
107	Exchanging synthetic or statistical health information between platforms	Interoperability	Functional	Should
	Events, Dashboards, Images, Reports, Graphs and Charts should be exchanged or executed independently of the owner platforms. Different information is produced at different levels all over different platforms interoperable in IoT Galaxy; the purpose of this requirements is to use synthetic or almost worked data where they are reusing the results without reworking them totally. Many configurations are possible. • To use a platform (master) as main point of access, linking the other functions accessed through APIs to the platforms owner where the functions are executed. • To use a distributed approach calling different APIs with many interfaces between different platforms at the same level. • To use a Business Intelligence platform for synthesis, elaboration, statistics and presentation, keeping operational analytical data in the owner platforms. In case 1 e 2 dashboards and reports, produced by the owner platforms are exchanged without adding elaboration; in case 3 some algorithms could be written and executed directly on the Business Intelligence platform.			
101	Exchanging discrete medical measures across platforms	Interoperability	Functional	Should
	A discrete health measure must be accessed and used in many platforms, also different from the one that first physically picked up the information. Semantic Analysis about the meaning of the data is needed.			
102	Exchanging complex medical measures across platforms	Interoperability	Functional	Should
	A complex health measure (i.e. a file made of different parameters covering a period of time) can be used in many platforms different from the one that first physically picked up the information. Data usage and elaboration must be done in accordance with the protocol used to store the information.			
62	Constraints on processing of personal and health data	Legality	Non-functional	Should
	The processing of Personal and Health data must conform to the rules and criteria laid down in “Personal Data Protection Code - Legislat. Decree no.196 of 30 June 2003” and in			

	"Measures and arrangements applying to the controllers of processing operations performed with the help of electronic tools -27 november 2008". For Italian privacy regulation some specific instances are required: • Informed Consent: the person in charge of data processing must collect the informed consent of the involved person to the processing of data disclosing health status (for more details see requirement 145). • Information sheet: Art. 13 of the Privacy Codex provides that the people in charge of data processing or collecting need to be informed orally or in writing about the processing of data (for more details see requirement 146). • Privacy Codex: The behaviour of healthcare operators must be respectful of the right of personal dignity and confidentiality of every citizen and it has to be appropriate to various situations in which benefits are provided according to the Legislative Decree 196/2003 (for more details see requirement 143). For United Kingdom regulation some specific instances are required: • Compliance with the Data protection act: Compliance with the data protection act is required in all cases of personal data collection, usage and storage. Sensitive data (such as health data) require a higher level of protection (for more details see requirement 151). • Information Security and Information Governance good practice guidelines: The health and social care information center implements good practice guidelines for use within the NHS. • Adoption of any system by NHS funded organizations requires compliance with best practice guidelines for information security and governance (for more details see requirement 152).			
158	National, regional and local Bioethic Committee	Operational	Functional	Should
	The clinical trial is an extraordinary means to evaluate the efficacy of a drug or a medical device, the risks involved and, ultimately, to decide whether it is appropriate to make it available for the population. However, the importance of research can never justify the violation of the rights of persons participating in the trial. For this reason, the European Union has adopted a set of rules, called Good Clinical Practice, which govern the research correct. To ensure the observance of "Good Clinical Practice", they have been set up specific bodies (Ethics Committees) who evaluate and closely monitor each trial overseeing the correctness and evolution over time. Ethics Committees are independent bodies formed by health operators and not, with the task of evaluating the protocols of each trial from the point view of the scientific, ethical and feasibility. In Italy it is not possible to conduct any human trials without first this has been assessed and approved by an ethics Committee. Actually there are more than 200 distributed among hospitals and local health authorities.			

	Other tasks of ethics Committees are: • monitor the progress of the studies; • promote information and training for doctors and patients; • provide opinion and directions in the case of specific requirements, both individually (in case of uncertainties concerning the treatment to be applied), both at a general level (e.g. in case you need to make decisions related to patient groups); • check the economic coverage for the costs of the trial; • check that the protocol of the trial must be guaranteed the right to dissemination and publication of the results by the investigators regardless of the opinion of the promoter and in compliance with applicable laws regarding the processing of sensitive data and intellectual protection confidentiality.			
157	Seamless patient monitoring	Operational	Functional	Should
	All physiological data (e.g. from ECG, Blood Pressure and SpO2 monitors) should be accessible by specific applications connected to the INTER-IoT.			
174	User Access Service for Administrators	Operational	Functional	Should
	Administrators main duties in INTER-Health are: • User provisioning. • Platform authorization and activation of APIs. For the first process, in accordance to the architectural choices, different options are possible. • Keeping the main directory to register users, on the “A” platform (that acts as master, the only which can update, through platform provisioning on “A” portal, the database) and then giving access to data, via API to the interested platforms (only query). • Keeping users on “A” platform (master Director) allowing other platforms to insert update and query users by APIs. • Keeping the “A” user database as master and synchronizing the slave directories in other platforms (redundant way). • Use third parties Directory as master and keep it synchronized with all platforms interested. For the second process the main functionalities for administrators will be: • Define platforms and systems that may interact (client platforms, suppliers platforms). • Set protocols and standards for exchanging data. • Set APIs authorization and so on. • Activate/deactivate the use of interface and so on.			
177	User Access Gateway for Caregivers	Operational	Functional	Should
	INTER-Health gateway for Caregivers is intended to integrate gateways functionalities for assisted measurements (i.e. measures that patients are not able to do or it’s better to take with the aid of an expert person). Architectural options are the same of gateways for patients.			

	Caregivers gateway main functionalities in addition to what already described for patients, are: • Authentication as assistant. • Authorization to manage data of different users. • Choice of patients between a set of cared users. • Possibilities of inserting measures manually. • Setting Profile communication and devices pairing. • Managing measures on the device and releasing them to the gateway that stores them on a local database. • Sending measures to the platform. • Reporting locally measures already stored. In term of interoperability, the INTER-Health gateway uses the gateway architectural scenarios described in the requirement 175. Specific Instances and implementation possible solutions: • In particular situation, for example assisting patients on an ambulance, all text boxes must be 'speech to text' able. • Besides, sensor data must be immediately available on gateway display unit to allow triage to continue in an uninterrupted path. Please see requirement 153 for how to address drops in server connectivity.			
172	User Access Service for Patients	Operational	Functional	Should
	User health main functionalities for patients are: • Access to services (providing username and password). • Personal settings (contacts, measurements reminders and so on), managed in registration and updated by patients, later. • Reporting: access to measures by chronological reports or using graphics and dashboards. Each group of user functionalities can be implemented in different ways in accordance to the choices done in the architectural scenario. For the patients functionalities and health services specific features is worth what said for user access services for doctors apart from' Personal data collected on Computerized Nutritional Folder'.			
173	User Access Service for Doctors	Operational	Functional	Should
	User health main functionalities for doctors are: • Access to services (providing username and password). • Personal information (contacts, receiving alerts) managed by administrators in registration and eventually updated by doctors.			

	• Assigned Patients medical parameters settings (measurements schedules, thresholds). • Medical report management (special reporting allowing specialists to follow report workflow). • General purpose (chronological) or ad-hoc (oximetry, images and so on) reports.			
154	Timestamped event recording	Operational	Functional	Should
	To highlight specific events within a patient history, a time stamped generic event could be generated. The time stamp associated with this event should be to the accuracy of seconds. The ability for INTER-IoT to handle time stamped events with associated meta data should be supported.			
217	Wearable devices support to detect physical activity level (e.g. number of steps taken, consumed calories and minutes of physical activity) took place during the Experimental Nutritional Counselling	Operational	Functional	Could
	Wearable Mobile Devices used are equipped with wireless interface and are CE labelled in accordance with Directive 93/42/EEC, which certifies that the device meets the minimum essential requirements of safety of operators and citizen. The detection of the physical activity practice will occur daily and in mobility. The surveys recorded and sent in INTER-Health platform, will be available from both health staff and the subject. The information gathered from wearable mobile devices will allow to capture real-time status of physical activity of the subject and check the achievement of objectives. If necessary health staff will require additional counselling for subjects who have particular risk situations. Mobile Devices wearable will detect Routes Steps Number, calories burned and Minutes of physical activity carried out. Information gathered from wearable mobile devices will be used to divide patients into different categories (e.g. Inactive or Sedentary Subjects, Moderately active Subjects, Sedentary Subject, etc...).			
70	User interface	Usability	Non-functional	Could
	All end user interfaces should be easy to use. For older people the confidence with the use of smart phones, Internet applications and information technology in general is very limited. For younger people the confidence on the use of smartphones Internet technologies and applications increases. As services are offered to both types of users, this requirement needs to be considered as a reference target for the design of web and mobile interfaces.			

	It is assumed that health care providers (doctors, nurses, technicians) have experience with Internet applications. Access to the management, monitoring and consultation could be done through a web interface.

Table 11: INTER-Health pilot minimum requirements.