# INTER-Layer: A Layered Approach for IoT Platform Interoperability


**Andreu Belsa, Alejandro Fornes-Leal, Clara I. Valero, Eneko Olivares, Jara Suárez de Puga, Fernando Boronat, and Flavio Fuart**

**Abstract** A interoperability layer is fundamental to provide a global continuum interoperability among IoT platforms. To address this layer, the following activities have been carried out: (i) design of device-to-device interaction based on multiprotocol/access mechanisms; (ii) design of software defined interoperable modules for mobility and routing; (iii) development of an open management framework for smart objects; (iv) design and implementation of smart IoT application service gateway and virtualization; and (v) definition of a common ontology which will facilitate access to the heterogeneous data, data that will be collected and managed by integrated IoT platforms.

## 1 Introduction

The lack of interoperability in the IoT ecosystem causes many issues, from the impossibility of connecting non-interoperable devices into different IoT platforms, to difficulties in leveraging data of multiple platforms to conform applications and, to slowing the introduction of novel IoT technologies at large scale [1–3]. The INTER-IoT presents a layer-oriented solution to provide interoperability at any layer and across layers among different IoT systems and platforms. Although its design and development are more challenging in comparison to an application-level approach [4], the layered-oriented approach has a higher potential in order to provide interoperability. It facilitates a tight bidirectional integration, which in turn provides higher performance, complete modularity, high adaptability and flexibility, and presents increased reliability.

This layer-oriented solution is achieved through INTER-Layer. INTER-Layer is an instantiation of the INTER-IoT Reference Architecture (RA) presented in Chap. 3, which was designed specifically for the interoperability of IoT Plat-

A. Belsa (✉) · A. Fornes-Leal · C. I. Valero · E. Olivares · J. Suárez de Puga · F. Boronat
UPV, Universitat Politècnica de València, Camino de Vera, 46022 Valencia, Spain
e-mail: anbelpel@upv.es

F. Fuart
XLAB doo, Pot za Brdom 100, SI-1000 Ljubljana, Slovenia



C. E. Palau (eds.), *Interoperability of Heterogeneous IoT Platforms*, Internet of Things,
https://doi.org/10.1007/978-3-030-82446-4_4

forms. It includes several interoperability solutions (methods and tools) dedicated to specific layers: Device-to-Device (D2D), Networking-to-Networking (N2N), Middleware-to-Middleware (MW2MW), Application and Services-to-Application and Services (AS2AS), and Data and Semantics-to-Data and Semantics (DS2DS).

Each interoperability layer has a strong coupling with adjacent layers and provides an interface which can be used for interacting with the components. Interfaces are controlled by a meta-level framework to provide global interoperability. The different layers can communicate and interoperate with each other through these interfaces, therefore having cross-layering. Cross-layer components enable a deeper and more complete integration, while supporting security and privacy mechanisms for all the layers. In summary, INTER-Layer offers the following benefits at different layers or levels:

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

Except for the semantic interoperability layer, which has a dedicated chapter, the solutions developed for each layer are described and explained in the following subsections, considering all the relevant components, use cases the technologies applied [5, 6].

## 2 Device Interoperability

The Device Layer, in the context of an IoT ecosystem, comprises the lowest level layer in the IoT stack [7]. This layer comprises a range of interconnected small devices with limited CPU, memory, and power resources, the so-called “constrained devices”. It includes sensors/actuators, smart objects, and smart devices, and are used to conform a network which in turn may exhibit constraints as well (e.g., unreliable or lossy channels, limited and unpredictable bandwidth, and a highly dynamic topology) [8]. These constrained devices are in charge of gathering information from their respective ecosystems and send this information to one or more server stations. Additionally, they could act on the information, performing some physical action

(including displaying it). Other entities on an IoT deployment, like a base station or a controlling server, might have additional computational and communication resources to support the interaction between constrained devices and applications in a more traditional network approximation.

Interoperability at the device level implies that heterogeneous IoT devices are able to interact with each other, so that IoT devices can be both accessed and controlled through a unified interface and integrated into any IoT platform. At this level, interoperability is usually achieved through gateways deployed in dedicated nodes, although in can be implemented in other elements, such as smartphones [9]. In this subsection, the approach followed in INTER-IoT for achieving this type of interoperability as well as the architecture and components considered are presented. Besides, some used cases and results are depicted.

## *2.1 INTER-Layer Approach for Device Interoperability*

INTER-IoT, and more specifically INTER-Layer, aims to address the following device interoperability challenges:

- 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.

Historically, there have been several approaches and communication patterns to operate at device level in IoT systems, each one of them having different application areas and characteristics [10]: *Strict Device-to-Device Communication Pattern*, *Device-to-Cloud Communication Pattern*, and *Device-to-Gateway Communication Pattern*. The novelty introduced by INTER-IoT in this layer is a new communication pattern: ***Device-to-Edge Communication Pattern***. In this communication pattern, devices interact with a gateway, similar to the *Device-to-Gateway Communication Pattern*, but with some of the Device-to-Cloud capabilities shifted closer to the devices at the *Edge* or *Fog*. Fog and Edge are intermediate layers between the Cloud and IoT devices where smart agents provide processing and/or storage much closer to the device layer, typically those smart agents stay in the *MAN* or *WAN*.

In order to shift IoT cloud computing capabilities to the Edge of the network, closer to the devices, some of the functionalities that need more computing power must be virtualized in the Edge [11]. In INTER-IoT, apart from studying typical gateway approaches for providing interoperability [12], it also introduces a new paradigm for IoT Gateways that adjusts to this new communication pattern: the

**dual Physical-Virtual IoT Gateway**. This gateway is decoupled in two parts, (i) the **Physical Gateway**, which only performs lightweight network level operations and data aggregation, typically instantiated in a resource constrained device, and (ii) the **Virtual Gateway**, which represents the virtual counterpart of the physical gateway but in a less constrained device or virtualized service [13]. With this new communication pattern, three different connection levels are present:

- **Device to Physical Gateway Network Level**: Comprises all the different radio and access network protocols that devices will use to connect to the physical gateway, usually in the PAN or LAN range.
- **Physical to Virtual Gateway Network Level**: Is the connection between the physical and virtual gateway. This network level resides in the Fog or Edge, usually in the MAN or WAN range. This connection should be fast, secure and robust and should handle sessions to allow roaming.
- **Virtual Gateway to IoT Platform Network Level**: In this network level, the virtual gateway will connect and share its devices' state and information with an external IoT Platform. In a typical scenario, an IoT Platform resides in the Cloud, therefore this network level resides in the GAN.

## *2.2 Architecture of the Solution and Components*

The gateway architecture of INTER-Layer is shown in Fig. 1 [14]. It is designed considering always modularity in protocols and access networks, meaning that any access network (AN) can be inserted into the structure as long as it is interfacing accordingly with its corresponding controller. The same is true for the protocols and middleware (MW) modules. The gateway is build up so that once the system structure is functional, a split-up can be realized. Part of the gateway can be placed in the Cloud to allow functionalities that a physical gateway is not able to perform in an efficient way. The connector module is in charge of controlling the communication between the physical and the virtual part of the gateway. When connection is lost, the virtual part remains functional and will answer to requests of API and MW. There are three ways to connect to IoT sensors and actuators:

1. The lowest level where a connection can take place is at the Access Network controller (AN controller). This is for very simple sensors or actuators that either does not have or have very limited processing power and can be offline for longer time periods. Sensors of this kind are commonly battery powered, actuators may have a power grid connection but usually have very limited processing power. The AN controller will do all routing and will serve as a master or access point for the sensors, and afterwards the Protocol controller will manipulate the data and create the messages to be sent to the virtual counterpart of the gateway.
2. At the middle level, the dedicated sensors and actuators can be connected (COTS IoT devices). Usually these sensors and actuators have some dedicated communication protocol between the wireless sensor and some piece of electronics with

a small processing core. They are capable of handling their own access and protocol controllers, and can be connected through a dedicated extension module by implementing a Device controller.

3. At the highest level, the COTS IoT systems are found. They manage their own gateway, protocols and AN controllers. These systems can be connected via the connector directly to the Virtual Gateway of the Inter-IoT system. In any case, the related COTS system software would have to be modified to add specific connection capabilities in order to implement the reference Physical-Virtual Gateway communication protocol.

The architecture is composed of the following components:

- **Registry**: This component is responsible of registering all the devices with its multiple sensors and actuators in the gateway. It adds an entry in the Device Manager with the information about each sensor and actuator.
- **Device Manager**: The Device Manager is accessible to every other component that needs information of any sensor/actuator. The protocol and access network modules will call the Device Manager in order to resolve the metadata for each sensor/actuator.
- **Access Network Modules**: The Access Network modules provide the INTER-IoT gateway access to the following communication channels: WiFi, ZigBee, USB, LoRaWAN and other proprietary RF links accessible via SDR. They are in charge of establishing and terminating a connection with the sensor/actuator, requesting and sending data from/to them, and handling the data pushed by the sensor/actuator to the gateway, among other functionalities.
- **Protocol Module**: This components are located within the Protocol Controller and implement the specific features of any supported protocol (CoAP, MQTT, LWM2M, etc.) throughout standard interfaces towards the Protocol Controller and the Dispatcher.
- **Access Network Controller**: It allows access to the devices, providing the necessary interfaces between the devices and the protocol modules. The Device Manager configures the access network modules according to the registry.
- **Protocol Controller**: This component is located within the physical part of the gateway architecture and contains all the communication protocols supported by the gateway, implementing the common interfaces between those protocols and the other components such as the Gateway Configuration, the Access Network Controller, the Device Manager and the Dispatcher.
- **Gateway Configuration**: This component is duplicated in the virtual and physical part. Every other component can use this component to access the gateway configuration.
- **Connector**: It controls the communication between the physical and virtual part of the gateway.
- **Dispatcher**: The device sends a trigger to the Dispatcher whenever new data are available, being this component in charge of storing 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

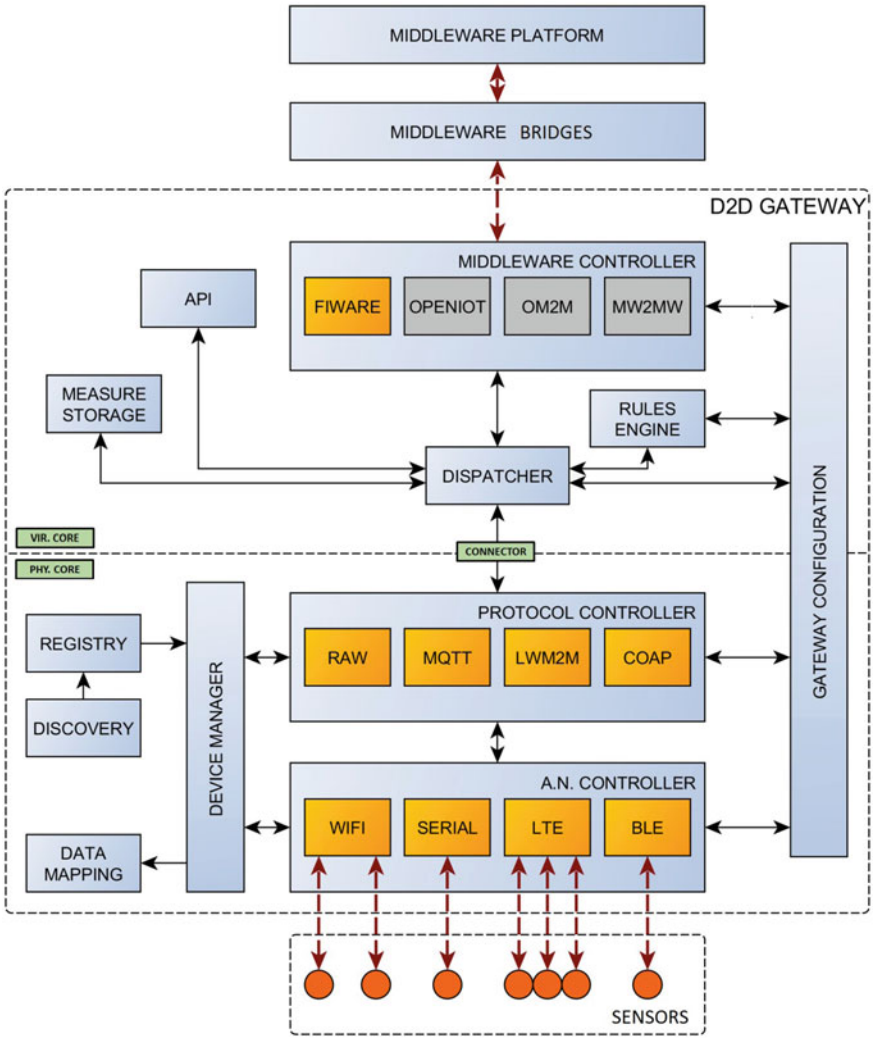


**Fig. 1** Gateway architecture

get the latest data sample from the Measurement Storage and will send it to the middleware.

- **Measurement Storage**: This component works as a cache in the gateway, storing 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.

- **Middleware Module**: This Module is specific to a IoT Middleware platform and handles the communication of the gateway with the platform. It is in charge of registering the sensors and actuators to the middleware platform as well as processing the requests and responses exchanged with it.
- **Middleware Controller**: It wraps the active Middleware Module in order to have a common interface for the gateway. This component creates the connection to the MW platform and handles the messages interchanged between the module and the platform, as well as the messages sent to the Dispatcher.
- **Commons**: Even if it does not appear in the architecture, it is a basic component that includes several classes, methods and tools to be leveraged by the rest of components.

## *2.3 Implementation and Use Cases*

In this section, the main technologies used for implementing the described architecture for interoperability at device layer are presented, and then some use cases in which the proposed solution has been utilized are briefly depicted. In particular, examples of integration at different levels of the device layer are showcased, including integration at device level, at physical gateway level and at virtual gateway level.

### 2.3.1 Implementation

The physical and virtual gateway implementation share a common base and runtime code. Both are based in an OSGi[1] framework wrapper (the OSGi framework has to be R4 compliant) with a customized bootstrap and initiation routines. This framework first load the third party libraries, then the core components and afterwards the extension modules. Finally, a routine for starting all the modules is launched, and the Physical and Virtual Core take the main thread to control the gateway. In Figs. 2 and 3 a schema and summary of the OSGi Framework, wrapper and components is shown.

This approach follows OSGI recommendations for a clear decoupled and modular system. As can be seen in the previous figures, these extensions can be developed to work in both parts of the gateway. Typically, physical extensions are centered in providing support for other device access network and protocols, creating new device controllers, whereas virtual extensions are centered in creating new middleware controllers to provide support for more IoT platforms. Common extensions can provide utilities to configurate or manage the gateway as a whole.

In the INTER-IoT device-to-device interoperability gateway, there are four different APIs:

[1] https://www.osgi.org/.

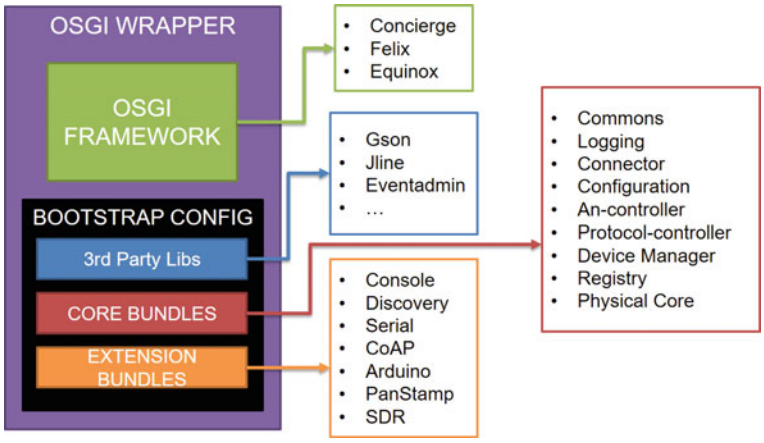


**Fig. 2** Physical Gateway components

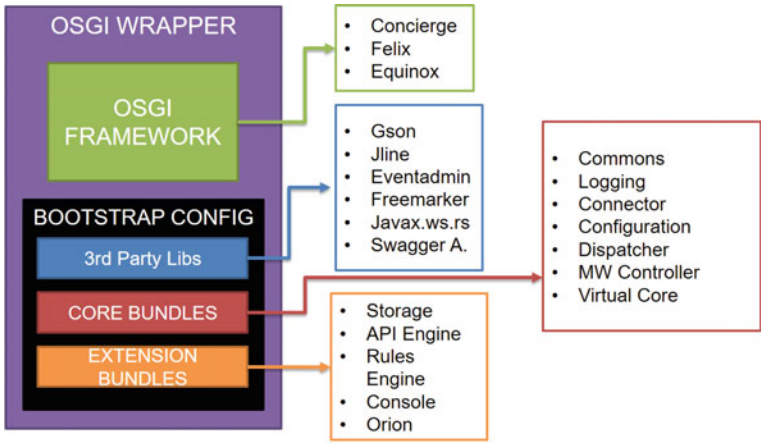


**Fig. 3** Virtual Gateway components

- **Gateway CLI**: The gateway console extension provides a Command-Line Interface (CLI) to control the physical or virtual gateway instance.
- **Gateway REST API module**: REST API exposed by the virtual gateway API Engine extension module to interact with the virtual and physical gateway.
- **Physical/Virtual Communication API**: Messages exchanged between the physical and virtual through the connector module.
- **Programmatic API**: Libraries and interfaces needed to develop new extension modules for the gateway.

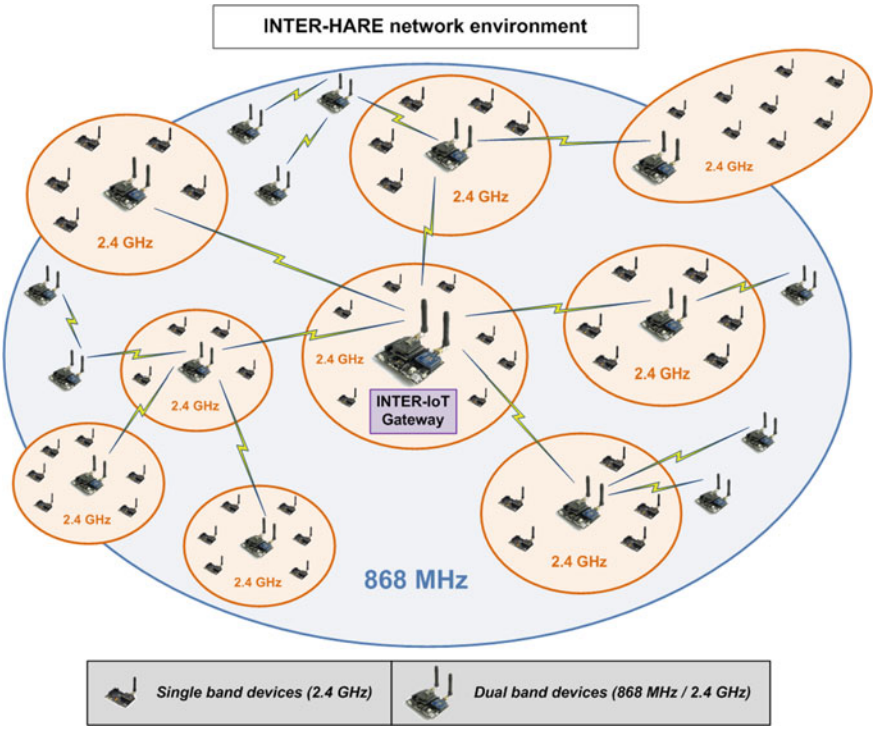


**Fig. 4** INTER-Hare network

### 2.3.2 Integration at Device Level: INTER-HARE

The INTER-HARE project is intended to design a new LPWAN technology flexible enough to transparently encompass both LPWAN devices and multiple so-called low-power local area networks (LPLANs) while ensuring overall system's reliability. A cluster-tree network is created [15], where the LPWAN acts not only as data collector, but also as backhaul network for several LPLANs, as shown in Fig. 4.

The communication within the LPWAN is based on the HARE protocol stack [16], which ensures transmission reliability, low energy consumption by adopting uplink multi-hop communication, self-organization, and resilience. The INTER-HARE platform is conceived as an innovative evolution of HARE protocol stack and can be considered as a dynamic multiprotocol by means of the integration with the INTER-IoT Gateway. The architecture of the INTER-HARE platform can be split into two networks with different purposes: the transport network and the integration network (as it can be seen in Fig. 5).

The transport network involves all internal infrastructures responsible for gathering and transporting information from the end-devices to the physical gateway. This internal infrastructure is formed by one single HARE protocol Gateway, several

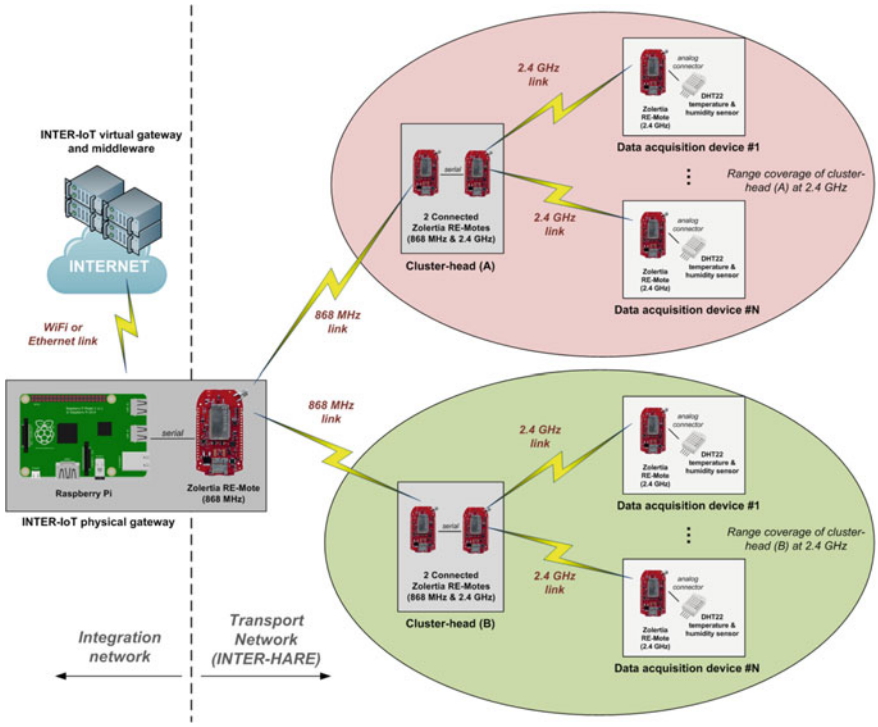


**Fig. 5** INTER-Hare architecture

Cluster Heads (CH) and Data Acquisition Devices (DAD). The integration network is formed by the INTER-IoT Gateway, which enables access to the whole IoT stack. Communication between the Physical Gateway and the Hare Protocol Gateway, is done with serial UART communication protocol. The INTER-IoT gateway is therefore considered as the brain of the INTER-HARE platform and the single point of contact between the physical network and the rest of the INTER-IoT system.

### 2.3.3 Integration at Physical Gateway Level: SensHook

SensHook is a IoT node focused on the prevention and detection of disease-vector mosquitoes. The node is composed by a Smart Mosquito Trap capable of mimicking the human body (scent and respiration) and of automatically counting captured mosquitoes, identify the gender and the species. The information collected by each node is then sent to a server. In this manner, SensHook aims at reducing inspection costs while improving surveillance programs, being the first solution in the world to combine human mimicking with automatic pest information in their value

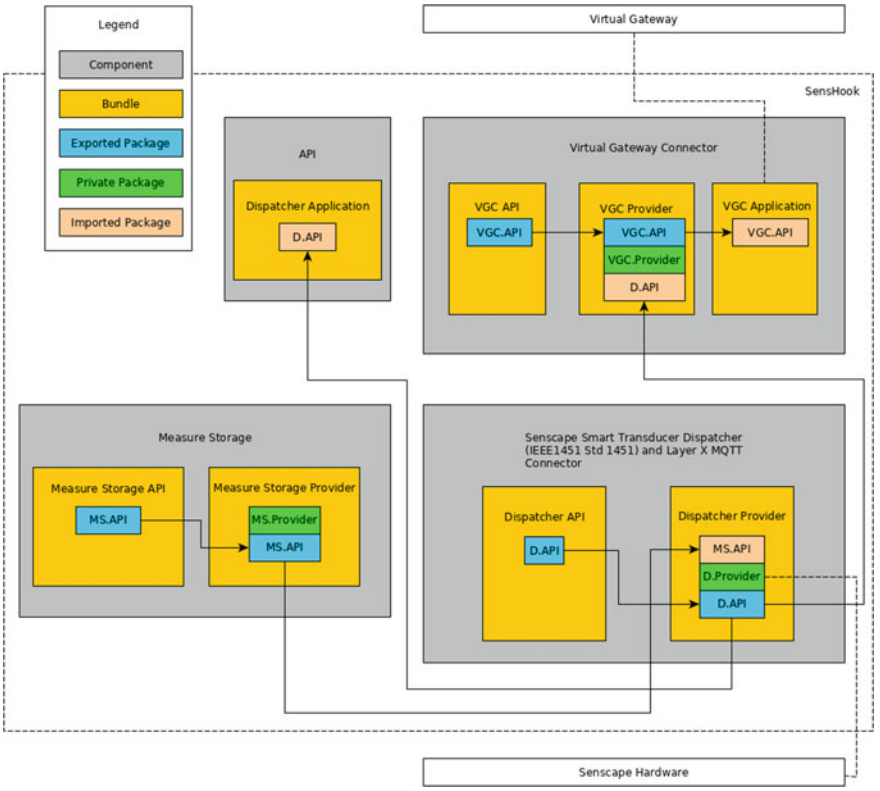


**Fig. 6** SensHook architecture

proposition. This will allow a whole new population of consumers to establish surveillance programs that were only accessible to those with significant resources.

In this use case, the integration in this case is performed at physical gateway level. Despite the fact that SensHook provides their own platform for performing low level communication and computing, the capability of sharing the information of its devices with IoT platforms is not available. Hence, aiming at enabling it, a connection is made to the Virtual Gateway, by developing a specific connector integrated in the SensHook platform that understands the Physical-Virtual communication protocol as can be seen in Fig. 6.

### 2.3.4 Integration at Virtual Gateway Application Level: ACHILLES

ACHILLES is a project that provides an advanced access control and endpoint authentication to devices attached to the INTER-IoT gateway. In general, these devices are usually limited in storage capacity, power, energy and processing capabil-

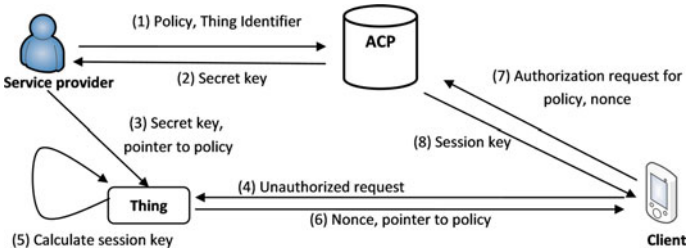


**Fig. 7** Achilles architecture

ities, presenting security risks in IoT deployments. Since these devices are not usually able to perform complex cryptographic operations, security management becomes an impossible task from the device perspective. ACHILLES project overcomes these limitations by allowing the delegation of security operations to a third party (ACP, Access Control Provider) which can be implemented by a trusted separate entity as depicted in Fig. 7.

ACHILLES is integrated in the gateway as an extension of the Virtual part. This extension implements the core client functionality and configuration, being able to perform read/write calls to the supported physical devices. The main idea of the ACHILLES concept is that IoT service providers store access control policies in ACPs and in return ACPs generate secret keys which are stored in the device (steps 1–2). These keys are generated, during a setup phase, using a secure hash with the device identifier as input. Additionally, devices are configured with pointers (e.g., a URL that points to an ACP and a particular file) to the access control policies that protect sensitive resources (step 3). Every time a client requests access to a protected resource (step 4), the device uses a secure hash function to generate a session key (step 5). The secret key used by that function is the key generated by the ACP, and the hash inputs are the pointer to the policy that protects the resource and a random nonce. The device transmits the nonce and the pointer to the client (step 6), which in return requests authorization from the appropriate ACP (over a secure channel, step 7). The ACP has all the necessary information required to calculate the session key: if the client is authorized, the ACP calculates the session key and transmits it back to the client (step 8). Providing that the device has not lied about its identity and the messages exchanged between the client and the device have not been modified, the device and the client end up sharing a secret key. This key can be used for securing subsequent communications (e.g., by using DTLS).

## 3 Network Interoperability

In the traditional OSI reference model for computer networking, the network layer is conventionally located within the third one, directly standing over data link layer (layer 2) and responding to the transport layer (layer 4). However, with the birth

of new radio access technologies and IoT protocols, this model had to adapt to be compliant with the IoT reference layer architecture, hence enabling the embracement of a large heterogeneous range of devices.

INTER-IoT understands the network layer of an IoT deployment as the protocols, systems and devices that work on layers 2, 3, and even 4 in some cases, of the traditional OSI model. IoT products encompass many different data communication scenarios: (i) some of them may involve sensors that send small data packets at low frequency without prioritizing timely delivery; (ii) others may involve storage capabilities to sustain periods when the communication link is down (e.g., Delay Tolerant Networks); (iii) some scenarios may need high bandwidth without having strict latency requirements; (iv) while others may need high quality, high bandwidth, and low latency. Besides, particular characteristics have to be taken in into account, such as the mobility of objects through different access networks, secure seamless mobility and backing of real time data among the network. The operation in highly constrained environments is also an important issue to analyze. Finally, the use of many heterogeneous protocols (6LowPAN, RPL, LoRa, SIGFox, etc.) and mechanisms (tunneling mechanisms over IP, GRE and 6LoWPAN, etc.) on IoT network level are problems that need of a network interoperability solution.

## *3.1 INTER-Layer Approach to Network Interoperability*

The particularity of an IoT deployment network is the treatment of different types of data flows as well as protocols to support communication. The great challenge that interoperability in the network layer must face is caused by the following problems:

- Difficulty to manage large amount of traffic flows generated by smart devices.
- Poor system scalability, which difficulties the integration of new devices.
- Hard interconnection of gateways and platforms via networks used by different providers.
- Several devices with totally different radio network access have to be accessed from a single gateway as an access point.
- Management of device's mobility through different access points.
- Great number of heterogeneous protocols (6LowPAN, RPL, LoRa, SIGFox, etc.) and mechanisms (tunnelling mechanisms over IP, GRE and 6LoWPAN, etc.) at IoT network level.

One of the main approaches to face these problems is the virtualization of the network layer, providing an extra tier of abstraction that facilitates management, scalability, seamless support for smart objects mobility (roaming) and packet routing, hence allowing the design and implementation of fully connected IoT ecosystems. To achieve network-to-network interoperability, the solution proposed by INTER-Layer is based on virtualization and software-defined paradigms, specifically in two approaches: Network Function Virtualization (NFV) and Software Defined Networks (SDN). The following characteristics have been considered:

- Decoupling of data plane from control 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 based on their priority.

### 3.1.1 SDN and NFV

Before getting into the details of the proposed interoperability solution, in this section it is described how these technologies operate and how they have been included in the INTER-IoT ecosystem. The term *virtualization* refers to the technologies that allow the decoupling or abstracting logical resources from the real physical infrastructure. Logical resources are named after the abstract vision that the software has of the physical resources of the system. The creation of these logical resources aims at offering a simpler high-level interface to isolate users and programmers from the details and characteristics of the internal hardware devices as storage, processor, memory or communication elements.

Virtualization can be applied in several domains. For instance, if the storage is virtualized, the real size and distribution of machines' storage is hidden and a logical division of this one is created, which can be leveraged by other elements. Virtualization of resources as processing capacity, as another example, can be useful for aggregating several CPUs to create virtual machines with higher capacity over a combined physical infrastructure. Besides, applying this concept to the components of the network receives the name of Network Virtualization, in which its physical resources (firewalls, routers, switches, load balancers, etc.) are virtualized and assigned to different virtual instances [17]. Network virtualization can stand for either aggregating physical networks into a single logical one, thus resulting in a Virtual LAN (known as external network virtualization), or providing network-like functionality within an operating system (internal network virtualization). In general, hardware and operating system virtualization are applied in the latter, obtaining a virtual network interface to communicate with. In this case, the Internal approach is exploited. The complexity and scale of today's data centres that are based in virtualization of machines makes network virtualization even more complex that traditional ones. Moreover, the hosting of new types of virtual machines as IoT platforms, virtual devices, or containers makes this approach a mandatory need. Hence, thanks to the implementation of virtualization, new architecture approaches can be deployed, as in the case of SDN/NFV.

Software Defined Networking consists in the separation of the network functions in two planes: the control plane and the data (or forwarding) plane. On the one hand, the intelligence and network state, as well as the management and routing algorithms, are centralized, and the underlying network infrastructure is abstracted from applications and services. On the other hand, the elements that compose the network, as switches, routers, etc., become mere forwarders which route the information in an efficient manner, according to flow tables which are filled according to

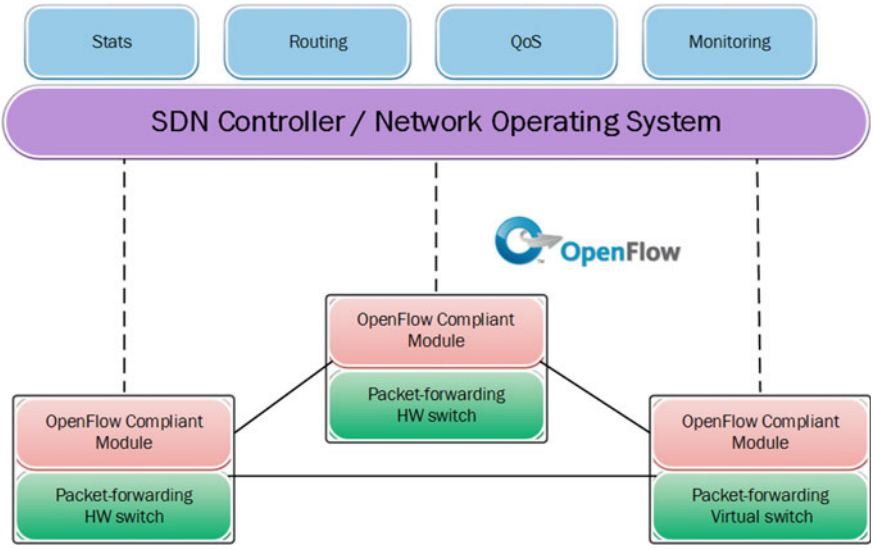


**Fig. 8** SDN basic architecture and components

the decisions of the control plane. Thanks to the NFV approach, the aforementioned network elements are virtualized within generic servers instead of making use of dedicated single-purpose equipment, being thus NFV and SDN highly compatible and complementary [18]. The SDN basic architecture and components is shown in Fig. 8.

## *3.2 Architecture of the Solution and Components*

The immense amount of traffic flows generated by smart devices is extremely hard to handle, and thus so is the scalability of IoT systems. Besides, creating the interconnections between gateways and platforms is not a trivial task. Thus, the network-to-network solution aims at providing seamless support for smart objects mobility and information routing. It will also allow offloading and roaming, which implies the interconnection of gateways and platforms through the network. The approximation that INTER-IoT proposes uses the SDN/NFV paradigm, achieving interoperability through the creation of a virtual network, with the support of the N2N API. The implementation of the N2N solution in INTER-Layer is depicted in Fig. 9.

The data plane (lower components of Fig. 9) is composed of virtual switches. They are connected to each other in a determined topology and all of them securely connected to the controller. The upper part is the control 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 connection with the forwarding

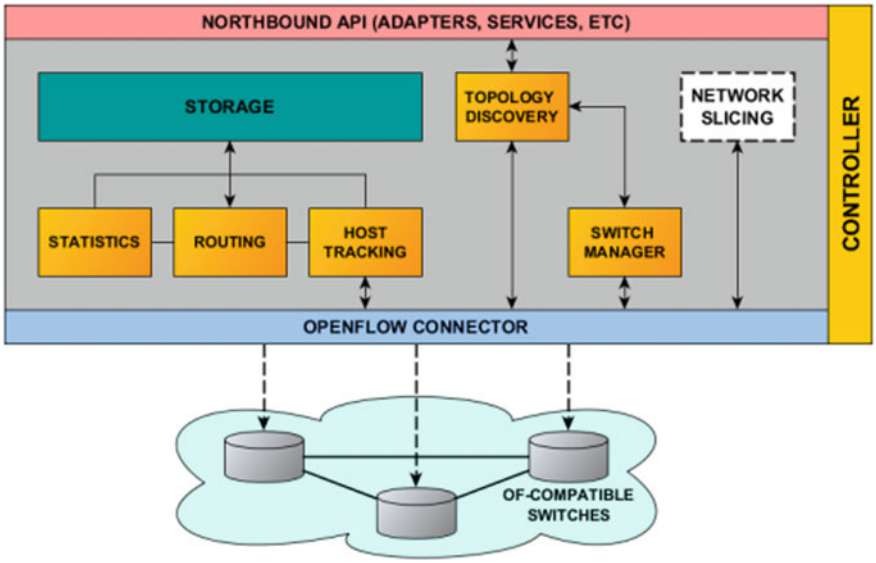


**Fig. 9** Network-to-network architecture high level building blocks

devices is implemented by a southbound API, which allows the entrance of OpenFlow packets into the controller and formalizes the way the control and data planes interact. The core of the controller consists of different modules containing all the logic that will dictate how to route the packets as well as to obtain statistics. Specifically, the modules that compose the control plane jointly with the controller are:

- **OpenFlow connector**: It is an OpenFlow understanding plugin that communicates, by OpenFlow protocol, with all the switches that conforms the virtual network. Is the Bridge between the Controller and the nodes of the network.
- **Switch Manager**: The Switch Manager API holds the details of the network elements. When a network element is discovered, its attributes (e.g. what switch/router is, version, capabilities, etc.) are stored in the database by the Switch Manager. Hence, it has all the information about the nodes of the network, the number of switches, their configuration and state, etc.
- **IoT Routing**: In this module, some headers of the packets are introduced to perform a routing algorithm previously configured and resolve the next hop in the network.
- **IoT Host Tracking**: Module in charge of handling the information from a host, including the address, the position in the network, etc. It tracks the location of the host relatively to the SDN network topology.
- **Statistics**: This module storage and provides information about the number of packets analyzed through the Controller. It can return the number of packets attending to some filters.

- **Storage**: In this module, the information about statistics, topologies, direction, and other data related with the network is stored and updated for other modules to access them.
- **Topology Discovery**: It contains a set of services that allow conveying topology information. It keeps track of the nodes in the network along with their links, and creates a graph representing the state of the network with additional information about the state of the links.
- **Northbound API**: It has been created and exposed so upper applications and services can configure the controller or gather data from it.

## *3.3 Implementation and Use Cases*

In this section, the main technologies used for implementing the described architecture for network interoperability are presented, and then some use cases in which the proposed solution has been utilized are briefly depicted.

### 3.3.1 Implementation

In order to implement the aforementioned modules, the most suitable available technologies have been implemented for creating a fully virtualized network with QoS capabilities that enables connectivity between the components that traditionally conform an IoT deployment. On the one hand, the virtual switches of the data plane have been implemented through Open VSwitch.[2] They are connected to each other in a determined topology, being all of them securely connected to the controller through TCP/SSL, leveraging the OpenFlow northbound protocol to update the flow tables and the OVSDB management protocol to retrieve information about their status and other statistics. On the other hand, the control plane contains the controller (RYU[3]), which provides OpenFlow and OVSDB connectors to parse all data packets coming from the network to the different services running on top it.

**OpenVSwitch as a Virtual Switch**

Open vSwitch (OpenVSwitch) is a production quality, multilayer virtual switch designed to enable massive network automation through programmatic extension, supporting standard management interfaces and protocols. Among its features one can find:

- VLAN 8021.Q support with trunk and access ports.
- Traffic flow-based statistics (NetFlow and sFlow).
- Traffic mirroring for monitoring (SPAN and RSPAN, among others).
- Link aggregation and bonding with LACP.

[2] https://www.openvswitch.org/.

[3] https://ryu-sdn.org/.

- Routing with Spanning Tree (STP).
- Quality of service management.
- Traffic queuing and shaping.
- Tunnelling (GRE, VXLAN, etc.).
- Security: VLAN isolation and traffic filtering.
- Automated Control: OpenFlow/OVSDB management protocol among others.

OpenVSwitch provides a more complex design than simple "bridges", being these the basic components to be used but, whereas bridges are only executed in host kernel space, the virtual switch makes use of both kernel and user spaces, which allows creating more complex rules of packet processing. The main components that compose a virtual switch are: (i) **ovs-vswitchd**, which is a daemon that implements the switch, jointly with a compilation of the Linux kernel module for flow-based switching; (ii) **ovsdb-server**, a lightweight database server to store and obtain switch configuration; (iii) **ovs-dpctl**, which is a tool for configuring the switch kernel module; (iv) **ovs-brcompatd**, which is a daemon that allows ovs-vswitchd acting as a substitute of Linux bridge; (v) **ovs-vsctl**, a command for queuing and updating the configuration of daemons; (vi) **ovs-appctl**: which is a utility that sends commands to the switch daemons that are running; and (vii) **ovs-ofctl**, a utility that implements the OpenFlow protocol to communicate with the controller.

The programmability and virtualization capabilities of OpenVSwitch have motivated its selection to deploy and manage the INTER-Layer virtual network solution. Besides, this switch supports different versions of the OpenFlow protocol, so it can be programmed to make specific actions with specific data flows. Hence, after the processing and decision-making that takes place in the specific modules of the controller, the adequate flow entry is inserted in the tables of the switches so that when data packet arrive it is already prepared to execute the necessary actions (forwarding, dropping, etc.).

**OpenFlow as Southbound Communication Protocol**

OpenFlow was the first SDN standard defined and vital element of an open SDN architecture. It is a communication protocol that gives access to the data plane as well as to the remote programming of network switches and routers over the network. This protocol decouples the intelligence required to route a packet from the act of forwarding it through the correct interface of the router, switch or network component, thus enabling the remote programming of the forwarding plane. This is achieved by inserting flow tables, designed by the protocol, within the switches managed by the controller.

The flow entries that compose the flow table are inserted and managed in the virtual switches by this protocol. They require three fields: (i) Match Fields, which defines a set of ingress ports, packet header fields and other metadata; (ii) Instructions, which is the action that the virtual switch has to make when a match is found (ports and fields of the data matches with the ones defined on the first field), and (iii) Counters, in charge of updating the number of packets matched against the Match Field. The OpenFlow protocol also allows representing 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 aid of this protocol and the programmability of the switches, different policies can be applied to manage the flows coming from the devices through the gateway. Additionally, with the information provided by the headers of the protocol, informative statistics can be obtained so the controller has a better overview of the state of the network and the flows being carried out [19]. Besides, the Quality of service possibilities implemented by OpenFlow includes:

- **Queues**: Associated to a port, define a priority treatment depending on the configuration, and could define the rate of the packets.
- **Rules**: Implemented in the queues, define the aforementioned treatment.
- **Meters**: Switch element which measures and controls the ingress rate of packets, i.e., the rate of packets prior to the output.

This protocol has several stable releases, starting from 0.8 and being versions 1.1 and 1.3 the most used ones. There are not many differences among version except for some QoS aspects like:

- *OF1.0*: In this version, an OpenFlow switch can have one or more queues for its ports. It is also possible to read/write headers for VLAN priority and IP type of service.
- *OF1.1*: This version improves the matching and tagging of VLAN and MPLS labels and traffic classes.
- *OF1.2*: Supports querying all queues of a switch, and introduces the OF-CONFIG protocol 5 to reconfigure queues within the switch. Max-rate property can be set to the queue. Flows can also be mapped to queues attached to ports.
- *OF1.3*: This version introduces meters.

In INTER-Layer, the controller chosen to communicate through this protocol will support all versions in order to connect with legacy switches that have implemented one of them. The network layer provides a QoS API in order to satisfy the potential QoS requirements of the deployment. When using the QoS API of Inter-IoT, the developer can add/delete/monitor rules, queues and meters. Rules determine whether the specified traffic is assigned to a certain queue or meter. Queues are designed to provide a guarantee on the rate of flow of packets placed in the queue. Different queues at different rates can be used to prioritize specific traffic. And meters complement the queue framework already in place by allowing for the rate-monitoring of traffic prior to output.

**OVSDB as a Southbound Protocol to Manage OVS Database**

The state of OpenVSwitch is stored in a database server. The Open vSwitch Database management protocol (OVSDB) is used to manage this database, thus leveraged for controlling the cluster database and determine the configuration of the virtual switch including its ports, bridges, interfaces and other important switch information. The OVSDB Protocol uses the JavaScript Object Notation (RFC 4627 [20]) for its schema and wire protocol format, and JSON-RPC 1.0 for its wire protocol.

The differences between OF-CONFIG and OVSDB protocols are several. The most important one is related to the fact that OVSDB is focused on the configuration of virtual switches implemented with OpenVSwitch while OF-CONFIG is focused on the configuration of physical switches. Still, vendors are also tending to implement OVSDB within their physical switches. These protocols are quite different regarding encoding, features, commands, etc. depending on the characteristic to be configured.

**Ryu as a Base Controller**

The component-based SDN framework Ryu has been chosen to handle the control plane and manage the virtual switches that compose the network-to-network solution. Ryu is simple, modular and highly designed to increase the agility of the network through its management and versatility. It is composed by a main component, Ryu-manager, which is in charge of (i) providing the environment where the different modules and applications will run, and (ii) the communication between the different modules. Besides, some modules have been implemented to extend the capabilities of the controller and adapt it to the particular needs of the network interoperability solution. Some of these modules are: (i) the Topology Discovery, which is in charge of obtaining the network information to create a graph that represents the current state of both the network and its components; (ii) a statistics module for accounting the number of packets processed, dropped or queued; (iii) the IoT Routing and Host Tracking modules, which goals are to create and manage the routes that each packet has to take to reach their destination; and (iv) a set of modules related with QoS and Security to prioritize some traffic flows and isolate them from others in order to create network slicing over the virtual infrastructure. Since the controller is entirely developed in Python 2.7, the modules created at the top of it to conform the network solution have been also developed in this programming language. Finally, a Northbound APIs REST-based have been developed for providing access to the software components that compose the controller and facilitate the deployment of new applications by future developers. This API is aggregated to other layers' APIs and published through INTER-API, which is presented in the following chapter.

### 3.3.2 Use Case: Traffic Priority in E-Health Environment

This use case implements the network-to-network solution with virtualized functions and central management of the cloud in an e-health environment. The virtual network is managed from a central monitoring application, using the API to request topologies, statistics, historical, etc. Additionally, the implementation of the SDN paradigm allows prioritizing data flows using traffic engineering and QoS. A real proof of concept was designed in order to provide an example of the usability of the solution. In this use case, the scenario presented a Central Hospital with different care houses in charge, located around the city. In these care houses, there were nurses that take care of the patients, measuring different health values (heart rate, temperature, sugar in blood, etc.) with the devices located at those houses. However, this information

never leaved the care houses, and thus the doctors had to visit continuously each one of them to gather the information, with the consequent cost that it entails.

The proposed architecture was based on the implementation of the network-to-network solution together with the device-to-device solution. Each care house was provisioned with a physical gateway that could obtain the information measured by the health tools (pulse meter, thermometer, glucose monitor and others). The physical gateway is directly connected and synchronized with its virtual counterpart located in a private cloud at the Central Hospital. This cloud was designed by means of the virtualization and SDN solution proposed by the INTER-Layer network approach. This way, the different virtualized gateways, together with other resources as IoT platforms or applications, were able to communicate through the virtual infrastructure that can be automatically re-configured.

Once the information arrives at the Central Hospital border router, which is connected to the SDN network, this information is automatically routed to the proper virtual gateway instantiated in the private cloud. Once the information arrives at this point, it can be filtered, aggregated or dropped off, and afterwards, all data could be forwarded to another IoT/Big Data Cloud Platform for future processing. One of the advantages of the implementation of SDN techniques, in this case, is the scalability that brings to the network. For instance, if a new virtual resource has to be deployed in the cloud infrastructure, it can be done in a seamless manner and its connection and configuration is made quite straightforward. Moreover, a different doctor could have access to a subset of care houses, hence to a subset of virtual resources. The possibility to define network slices within the SDN networks allows the separation of sets of virtual resources in order to define roles of data access, thus bringing privacy to patients and doctors. Finally, as the controller provided QoS capabilities, the prioritization of specific types of traffic brings a myriad of advantages to this scenario. For instance, data from a care house with severely ill patients or from a specific type of health devices can be prioritized in order to not get lost, for lowering the latency to the destination, or to trigger an alarm to notify the expert in charge.

## 4 Middleware Interoperability

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

Middleware thus abstracts several aspects of an end-to-end communication including the service name, address and location, the message transport protocol, service instance, interoperability features, etc. For example, a client can issue a request to a service without knowing which instance of that service will communicate with, thus hiding some of the complexities of service scalability. Middleware is also a convenient layer for placement of additional system-wide meta-services such as security, anonymization, auditing and monitoring.

### 4.1 INTER-Layer Approach to Middleware Interoperability

In the IoT domain, there is a need for dedicated and powerful middleware technologies to take the critical role of interconnecting the heterogeneous ecosystem of applications communicating over several interfaces using and operating on diversified technologies. Interoperability, context awareness, device discovery and management, data collection/storage/processing/visualization, scalability, privacy, and security are among the most significant aspects that have to be addressed by such solutions. Development of middlewares in the IoT domain is an active area of scientific and industrial research, and a number of interesting solutions have been developed so far [21, 22] (Fig. 10).

Due to the intrinsic difficulty to define and enforce a common standard among all these complex scenarios, IoT platforms 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. However, there is no clear division line between

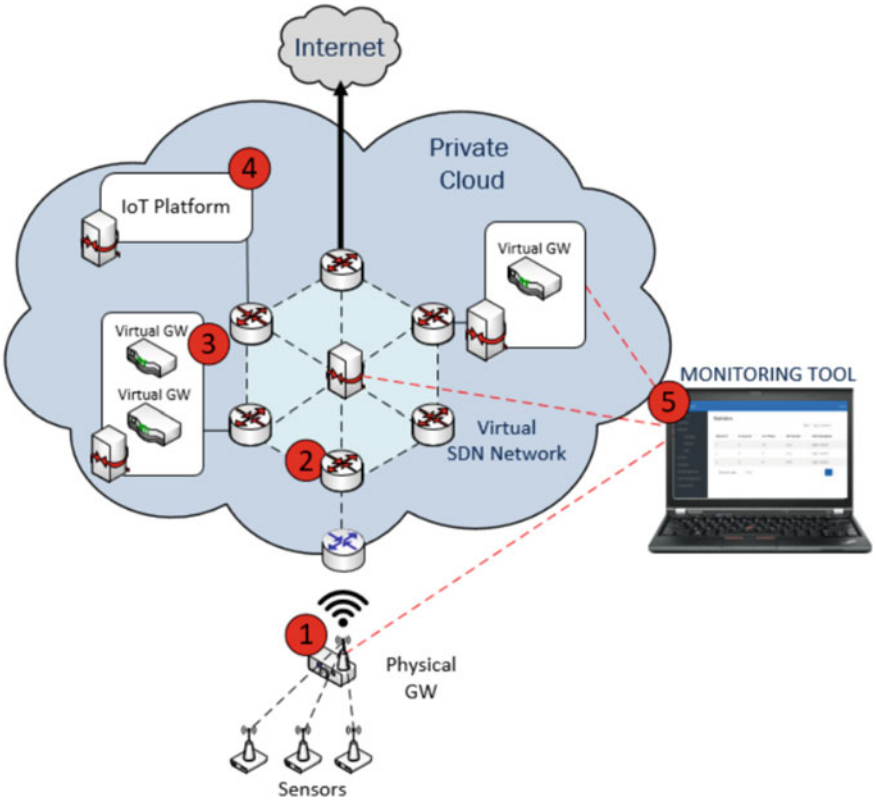


**Fig. 10** Use case on Health Vertical

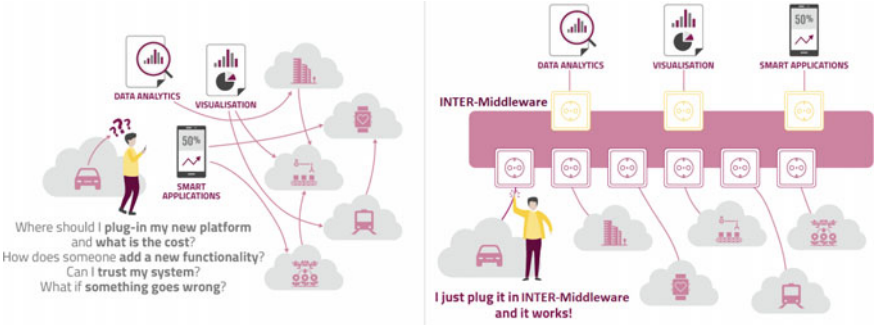


**Fig. 11** INTER-middleware complex usage scenarios made simple

a middleware solution and IoT platforms. Most IoT platforms provide some middleware functionalities, although their focus is on providing efficient IoT platform services and not on solving interoperability issues [23].

Figure 11 shows a scenario where a system composed of ad hoc solutions has been created. This approach exponentially increases the complexity (and thus development and operating costs) of the system with the addition of each new application or platform. As the system grows, it creates a complex network of different communication channels between applications and platforms. The main challenge is to interconnect systems that were never meant to work together, in a user-friendly, cost-effective, transparent, and secure way. This raises many issues, introduces technical, legal, privacy and security risks, and often places an insurmountable hurdle in front of delivery of such innovative systems. As a result, we can claim that developing value-added services on top of IoT platforms is expensive and time-consuming.

To address this issue, a middleware-to-middleware interoperability solution called INTER-Middleware has been created. At the time of writing, INTER-Middleware is in the incubation phase, where a robust product is being created from project results that have been validated in e-health and port logistics. We present in this chapter an IoT-focused middleware solution dedicated to the provision of the most critical interoperability services in the IoT domain.

## *4.2 Architecture of the Solution and Components*

INTER-Middleware architecture (Fig. 11) provides core functionalities related to facilitation of interoperability among IoT middleware platforms as well as provision of a common abstraction layer to provide access to IoT platform features and information. The architecture acts as an abstraction layer unifying the view on all interconnected platforms, devices and services. Moreover, its scalability permits the easy addition of new IoT platforms. INTER-Middleware architecture is composed by five main components (Fig. 12):

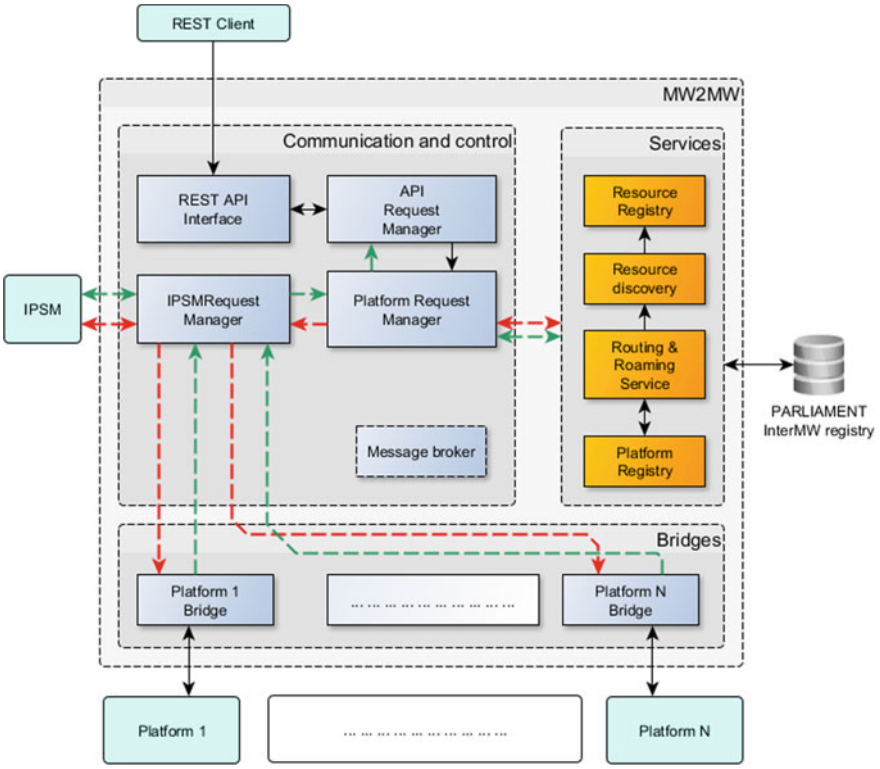


**Fig. 12** INTER-middleware architecture

- **Ontology**: Common ontology to represent all messages routed through the system.
- **Bridges**: Act as a middleman between INTER-Middleware and IoT platforms.
- **Communication and control**: Orchestrates all the communications that take place between the different components of the architecture
- **Services**: Common services offered by INTER-Middleware to facilitate interoperability between platforms.
- **REST Interface**: Extends the usability of the abstraction layer by exposing this functionality through a widely used technology.

## *4.3 Implementation and Use Cases*

### 4.3.1 Ontology

Data model, used in INTER-Middleware, 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 INTER-Middleware components improves the 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. INTER-Middleware uses the common INTER-IoT ontology (GOIoTP) to represent all messages routed through the system. It is implemented through JSON-LD messages and is in the core of INTER-Middleware, thus tightly coupled with the common abstraction layer that unifies the view on all interconnected platforms, devices and services. It does not matter what device belongs to what platform, or what service is in which platform. There is, however, no requirement of compliance with the INTER-Middleware data model placed upon IoT platforms that use it. Data models of platforms participating in communication through INTER-Middleware are mapped to ontologies and semantically translated in IPSM. As a result, the commonalities between data models of IoT platforms can be expressed through a common data model, despite the possibility of having different semantics.

### 4.3.2 Bridges

Interoperability at the middleware layer is achieved through the establishment of an abstraction layer and subsequent integration of all IoT platforms. INTER-Middleware open architecture is extended through the development of IoT platform bridges that provide specific functionalities to connect INTER-Middleware with an IoT platform. This way there is no need interconnect all platforms among themselves, but rather connect them to the abstraction layer and provide a mechanism for their communication within this layer. It supports actuation and subscription to observations as core IoT platform functionalities. Virtual devices management, which is basically mirroring devices across IoT platforms, is implemented for those platforms that support this functionality.

INTER-Middleware provides a common Java interface that defines bridge features that have to be implemented: subscriptions, actions, virtual devices management and discovery. One important step in bridges development is the implementation of a syntactic translator to/from platform-specific format and JSON-LD. Definition of rules for semantic alignments is still necessary, but not at the bridge level. That part of the process is fully implemented in IPSM. The integration with IPSM is achieved through the IPSM Request Manager component that orchestrates the communication between IPSM and INTER-Middleware components (Bridges, Platform Request Manager).

### 4.3.3 Communication and Control

Data flow is managed through a the introduction of *conversations*. A group of messages belongs to the same conversation if they share the same unique conversation identifier. For example, in a single conversation we would typically have first a message which subscribes to a particular group of sensors, and then messages with sensor readings, going upstream from the sensors to the application. Subscriptions in INTER-Middleware are also tracked by the unique conversation identifier. Technically, data flows are implemented through a message broker. This allows complete decoupling between components as well as isolation of the communication responsibility in a single element, which in turn makes profiling, scaling and adaptation to enterprise infrastructures easier. An abstraction mechanism enables interchangeability of the message broker implementation [24].

### 4.3.4 Services

The INTER-Middleware solution maintains a registry of all devices present in connected IoT platforms and provides meta-information about those devices. The Parliament™[4] triple store database provides persistence and advanced querying mechanisms for the Services subsystem. All registry-related requirements that need persistence or querying support, such as Platform Registry, Resource Registry and Subscriptions Registry, are implemented through this database. This allows the implementation of an efficient querying mechanism and seamless access to device information across IoT platforms. Maintenance of the device registry is not a trivial task, as there are several approaches utilized by IoT platforms to provide meta-data about attached devices. INTER-Middleware implements discovery strategies that can be used to populate the registry: full-query at regular time intervals, difference query at regular time intervals or, with more advanced IoT platforms, registry updates with callbacks.

### 4.3.5 REST Interface

Applications communicate with INTER-Middleware through a REST API. It further extends the usability of the abstraction layer by exposing most of the functionalities through a widely used technology and making them available to application layer components. Requests and results may be provided in either a simple JSON format, that fulfils most of the basic user requirements, or in the more complex JSON-LD format that also offers a richer set of functions and full semantic interoperability.

Security at application level is provided through the integration with the REST API Manager and Identity Manager. Platform security, on the other hand, is a respon-

[4] http://parliament.semwebcentral.org/.

sibility of bridge developers. In principle, authentication information can be passed through platform registration messages.

### 4.3.6 E-Health Use Case

In the area of health, INTER-Middleware has been used to develop e-Health application whose objective is the prevention of obesity-associated diseases through patient monitoring. The application integrates data collected from two different data sources, universAAL platform and Body Cloud platform. Both are health-focused IoT platforms that use Bluetooth technology to collect sensor measurements. However, they are not interoperable from a technological point of view. Instead of consuming data directly from the application through their APIs (leaving the resolution of interoperability problems as a task to be solved within the application), a new element is added to the architecture of the solution, INTER-Middleware, to be responsible for providing data interoperability between the platforms involved. To this end, it has been necessary to develop a platform bridges for each platform and connect them to INTER-Middleware.

Thus, the application consumes the data from the API provided by INTER-Middleware. In this way, if in the future it is desired to incorporate new platforms or devices associated to the platforms to the solution, these changes will be made in the interoperability layer, being transparent for the health application. An exhaustive description of the interoperability application in e-Health can be found in the Chap. 8.

### 4.3.7 Port Logistics Use Cases

INTER-Middleware has been validated in a port environment through three use cases whose main purpose is to improve the efficiency of resources in the transport chain of a port system through the monitoring and automation of processes involving different actors. The main actors are:

- Port authority: organism that manages the collection of different terminals, facilities and auxiliary systems that enable the activity of the port itself.
- Container terminal: installation or set of port installations constituting the interface between the mode of maritime transport and other modes.
- Haulier company: company that owns the fleet of trucks that access the port daily.

On the other hand, 3 use cases have been defined where each of them is focused on solving a different problematic:

- IoT access control, traffic and operational assistance.
- Dynamic lighting.
- Wind gusts detection.

The main challenge presented by this scenario is precisely the difficulty of interacting between systems that have not been designed to work together. The port authority

IoT platform is based on WSO2 whereas the container terminal uses its own IoT platform (SEAMS) and the haulier company has an Azure IoT platform in the cloud.

In order to establish a common base on which to build the new solutions associated with the different use cases in a user-fiendly, secure and scalable manner, it was decided to use the INTER-Middleware interoperability solution. For this purpose, the systems are integrated with INTER-IoT through the middleware layer and the API layer. To this end, each system implements a specific platform bridge. An exhaustive description of these cases can be found in the INTER-LogP chapter.

## 5 Application and Services Interoperability

There are multiple types of services in IoT ecosystems, such as Complex Event Processing, Historical Database, Big Data Processing, Visualization, Analytics, etc. IoT platforms do not have the capability to interact between each other at application and service level. This lack of interoperability is mainly produced by the heterogeneity of the services, the different domains involved, the lack of standardization of the technologies and the large amount of protocols involved, which in the end prevents prevents the emergence of vibrant IoT ecosystems [25]. There are main aspects to consider in order to achieve interoperability at this layer:

- The native access to the IoT Platform services. It should be considered as a method to access IoT platforms applications and services. Most IoT platforms provide a public API to access their services, APIs that are usually based on RESTful principles and allow common operations such as PUT, GET, PUSH or DELETE. However, there are other IoT Platforms that do not include a REST API or SOAP for easing the development of Web services, but provide different ways to interact with them.
- The use of wrappers. The term wrapper in this context refers to a specific program able to extract data from Internet sites or services and convert the information into a structured format.
- The creation of enablers to the applications and services. They guarantee, organize and simplify access to the IoT Platform services.
- The development of application, data and device catalogues dedicated to the IoT services. They are generally missing in the market. A solution based on a Service Catalogue will be able to register applications to make them discoverable. Furthermore, it will offer a description or detailed information about services/applications.
- The virtualization of services and applications. There are some benefits like simplifying the monitoring of the infrastructure, network issues and security incidents. Furthermore, it provides flexible mechanisms like the creation of additional instances of the services whenever needed. This allow to handle the additional load while maintaining the quality of the service.

Service composition solutions facilitate achieving interoperability between applications and services [26]. These solutions encompass all those processes that provide

added-value services, so-called composite services, from existing services in the IoT platforms. 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. The composition of such services can be as simple as one service making use of a second service, to very complex and flexible schemes of interconnection that need a coordinator to manage the mesh of requests and responses. There are several techniques of service composition like mash-up, orchestration, choreography of flow based programming and tools to facilitate its implementation. In the solution that will be described in the following subsections, the flow based programming approach is the selected technique.

## *5.1 INTER-Layer Approach at Service and Application Layer*

The main objective at this layer is to guarantee a solution that is capable to offer a layer of abstraction to achieve interoperability between the applications and services of IoT platforms. In order to provide benefits like access, use, import, export, catalog, discovery and combination of heterogeneous services between different IoT platforms (Fig. 13).

INTER-Layer approach provides a detailed plan about how to perform access to IoT platform services and implements a complete architecture to interact with these services and to create and manage new composed applications. The technique selected to create interoperability between services is Flow Based Programming paradigm. This paradigm defines applications as black-box process, which exchange data through predefined connections with message passing. These black-box processes can be connected to create different solutions without the need of being modified internally [27].

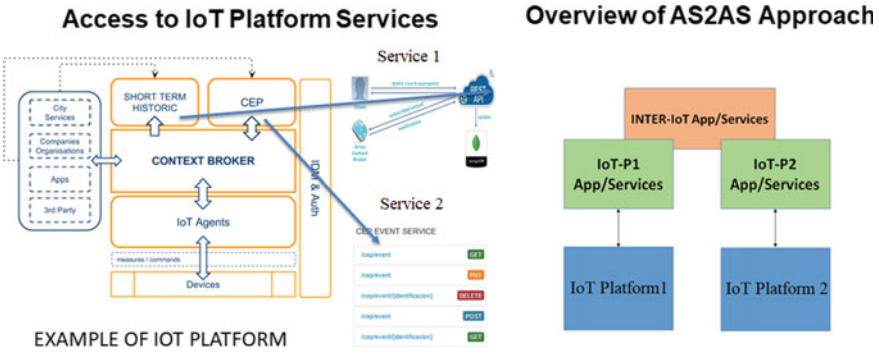


**Fig. 13** AS2AS basic overview

The main functionalities offered at AS2AS level are based on access to APIs or interfaces provided by IoT platforms, register those services/applications with their description or detailed information to make them accessible, offer requests to the catalogue to obtain the necessary services from the IoT Platforms, providing an abstraction layer of interoperability that facilitates the common access to these services and helping developers to make them interoperate with others of the catalog.

## 5.2 *Architecture of the Solution and Components*

The following architecture [28] is designed to perform the functionalities and goals that have been listed in the previous subsection (Fig. 14).

The components and the relation that exists between these components are the following:

- Service Catalogue and Service Discovery are in charge of storing and managing the information and description about the services available on IoT Platforms. To interact with these components, users can make use of the of the graphical environment (GUI): Modeller and Register Client.

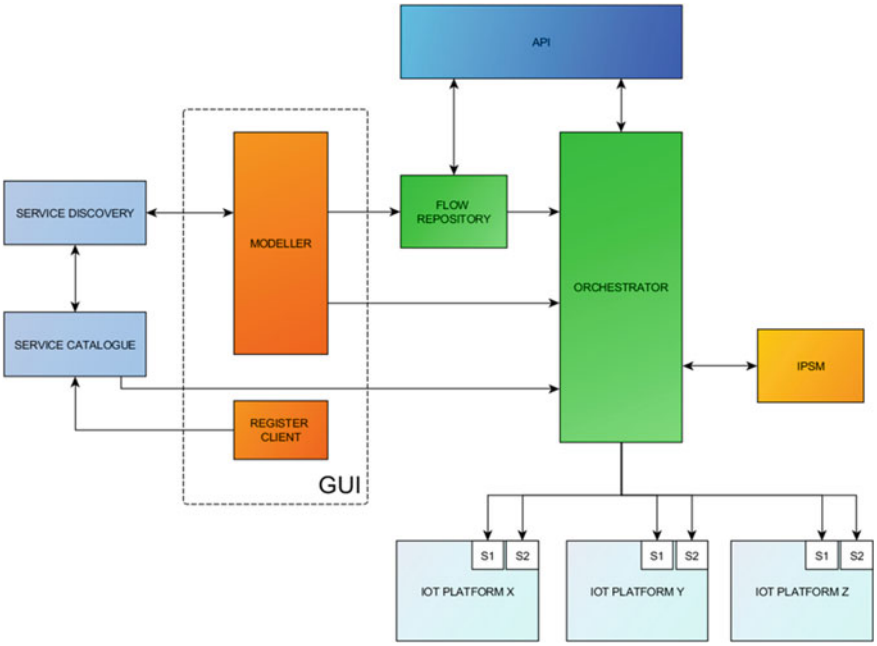


**Fig. 14** AS2AS basic architecture and components

- The Register Client provides a tool to register new native IoT platform services and new composite services (also known as flows). During the registration of a service, it is possible to add a description about its features. Once the registration of the service occurs, it is stored in the Service Catalogue.
- The Modeller is a graphical environment that has access to the services that have been registered. It uses the Service Discovery module, through which it calls the Service Catalogue. In addition, it can access to util 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 a number of times, etc.) that facilitate the interaction with the available services. Using 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. 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 central engine of the interoperability 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 its internal functions, in the order indicated in the model to run the service composition. It collaborates with the semantics module, which is responsible for performing semantic translation of data exchanged.

The orchestrator and modeller are based in Flow Based Programming paradigm. For that reason, it is necessary to take into account that in this technology the main element is the *node*. These nodes provide a mechanism to access and interact with the IoT services. A node needs input parameters and provides output information. It executes a series of internal processes in the application that is calling. The interaction between the different nodes will be defined by an execution flow, which defines and manages this interoperability process between services.

The Catalog, Register and Discovery components work with the nodes and its properties. The Modeller is responsible to create and modify the flows, which are composed by the interconnection of several nodes. The purpose of the Modeller is to define the service composition. These flows are stored and loaded in the Flows Repository. Finally, they are executed by the Orchestrator, which is the one who starts the service composition operation.

Finally, the API is responsible to manage the Orchestrator and the flows stored in the Flow Repository. It offer the functionalities of a process manager, allowing users to start/stop a flow of execution, view its status, load a flow of interoperability in the orchestrator or access to specific services.

## 5.3 Implementation and Use Cases

The core solution is based in Flow-Based Programming using as core a tool, Node-RED,[5] that implements this paradigm according to the AS2AS INTER-Layer interoperability requirements. This tool interacts with the components designed and developed to offer the interoperability solution. It transfers the advantages of this service composition paradigm to IoT. The solution enables a number of IoT services to be available in a development environment. Access to IoT services has been achieved by accessing its REST APIs and wrapping them through a node with a series of functionalities for the user. For those services that do not have REST API, other alternatives have been looked (e.g. SOAP web services).

### 5.3.1 Node-RED Integration in INTER-IoT

Node-RED offers a visual tool for wiring together hardware devices, APIs, IoT Native Sevices and online services [29]. From the point of view of INTER-Layer, Node-RED offers tools for creating new nodes for IoT services, as well as a legacy and huge core set of useful nodes. These nodes can be stored in a catalogue. Users can search available nodes in the catalogue or in the npm repository. New nodes can be registered and installed, and existing nodes can be enabled or disabled. In order to design and orchestrate interoperability, 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. This flows can be deployed in runtime with a single-click. In addition, it allows users to save useful functions, templates or flows for re-use [30, 31]. Finally, it offers an API to remotely administer the runtime.

### 5.3.2 Nodes

Paying attention to technical issues, a node consists in a JavaScript file that runs in the Node-RED service, and an HTML file consisting in a description of the node. The description appears in the node panel with a category, colour, name and icon, code to configure the node, and help text. Nodes can have at most one input, and zero or more outputs. During the initialization process, the node is loaded into the Node RED service. When the browser accesses the Node RED editor, the code for the installed nodes is loaded into the editor page. Node RED loads both HTML for the editor and JavaScript for the server from the node packages.

[5] https://nodered.org/.

### 5.3.3 Flows

The flows are a collection of nodes wired together to exchange messages, the data contained in the flow is stored in a file in JSON format. It consists of a list of JavaScript objects that describe the nodes and their configurations, as well as a list of downstream nodes they are connected to, the wires. Wires define the connections between node input and output endpoints in a flow. The messages passed between nodes in Node-RED are, by convention, JavaScript Objects called messages. Messages are the primary data structure used in Node-RED and are, in most cases, the only data that a node has to work with when it is activated. This ensures that a Node-RED flow is conceptually clean and stateless.

### 5.3.4 Implementation

Regarding a mapping between the architecture and the implementation, it is necessary to consider how to access to a complete instance of the interoperability solution. For that reason, during the implementation it has been taken into account that the flow designed by the modeller and executed by the orchestrator can be accessed by one or more users, at the same time and with different permissions, through the APIs. Still, only one flow can be executed in each instance of the solution. Therefore to have several flows of the interoperability solution running at the same time, it is necessary to work with different instances of the solution.

The core solution could be virtualized inside a docker container image allowing deploying the solution in the same way regardless of the environment. It allows to offer different instances of the interoperability solution located at the same host. Each instance of the server have different nodes, its own internal folders and files with its running configuration. It should be highlighted that instances access the same service catalogue and flow repository.

Different graphical interfaces and web services have been developed to interact with the components at various levels. The first level is from the point of view of the management of instances, allowing users to generate and manage different instances of the solution dynamically. The second level is to communicate with each specific instance, to design flows and load configurations or services. The third level is to interact with the catalogues. The framework designed in INTER-IoT starts integrating in its graphic environment the complete management of these components of the interoperability solution, including the management of the different users and security.

### 5.3.5 Node Design Methodology

Regarding the validation of the correct operation of the interoperability solution and the correct design of the nodes. The first steps after the creation of nodes to access to the desired IoT services involve the design and implementation of interoperability

use cases. These use cases consist in a flow of execution to develop a new composite application with a specific purpose. To homogenize the process, it has been defined a procedure to integrate services and applications in the interoperability solution. If a developer follows these steps, the result is the creation of an accessible and totally functional node. As a summary of these procedure:

- Firstly, it is necessary to perform a complete analysis of the service, to obtain access to an instance of the platform with the service running, to perform test with data, to analyze the functionalities offered by the service, to study the provided methods to access to the service, to document the functionalities and to analyze the messages or actions that return the execution of each functionality of the service.
- In second place, the node has to be implemented: to group the functionality of the service, to identify the parameters needed to access the service, to create configuration nodes, to create the interface that collects the parameters that will consume the service, to develop the code that will execute the functionalities, and to define the messages that the node sends and receives.
- Then, the correct actuation of the node has to be tested considering real data, fixing the bugs and catching errors.
- Finally, the deployment of the service with real data and the characteristics of the node have to be documented.

### 5.3.6 Interoperability Use Cases

These nodes are used in the interoperability flows. They implement uses cases consisting of a flow of execution to perform a new composite application with a specific purpose. The flow and nodes involved are inside a instance of the interoperability solution deployed as a docker container.

There are implemented several use cases, so in order to give a practical approach to the information of this chapter, some of the most outstanding ones are going to be briefly described. For instance, considering a use case of a Port Environment, a CEP can be connected to trigger actions when the trucks monitored by different and heterogeneous road haulier companies platforms are physically close to a specific location, to perform actions in a platform from the port authority domain, like queries or to store historical information about this truck and show it in a dashboard. This implementation facilitates the improvement of logistic processes, shows alerts and allow to consider several application domains (road haulier companies, port and terminal) working together in a single composite application.

Another use case is related to Active Assisted Living Environment for accessing to historical information from different IoT platforms with heterogeneous formats, to perform syntactic and semantic translation in a common data model and to store data in a centralized database. All this heterogeneous information is stored in a common format and there exists the possibility of using different kinds of databases. Hence, external applications can directly access and use all these data from various sources

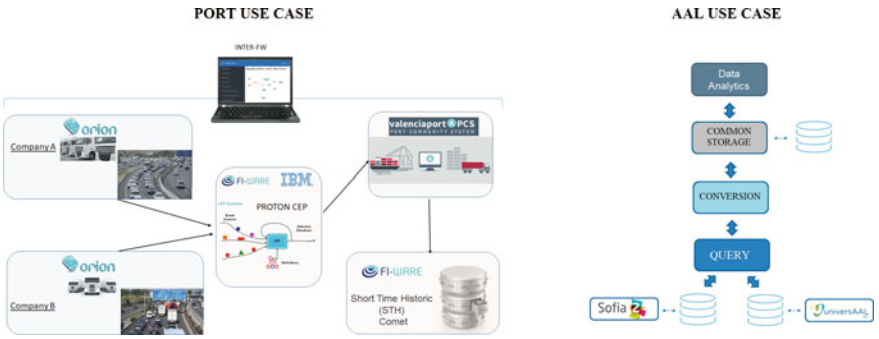


**Fig. 15** AS2AS example of use cases

in their applications without making different connectors, providing extra value to this data (Fig. 15).

Finally, the following are some examples of generic use cases of this level: accessing to different services from different IoT platforms and use a dashboard to show the information in a mashup composition way, to use external services to store information in an application that provides an extra security layer, to connect and interoperate local IoT platforms services with services provided by main providers of web applications (mail, social networks, cloud services, etc.) or performing conversion and translation between services that use different formats or protocols.

### 5.3.7 Results and Discussion

Different solutions and tools have been developed that are available in the public GIT repository of the INTER-IoT project, together with the a guide that explains the technical information, how to deploy the components and how to develop new elements to extend the solution. In addition, the different components of Application and Services Interoperability solution are integrated inside the INTER-IoT Framework, which offers a visual interface to access and interact easily with the components in a way that integrates the common security and privacy functionalities of INTER-IoT. The main accessible components are:

- Instance Manager. It performs the deployment and management of the solution instances. It facilitates both extensibility and usability of the solution in a local or cloud deployment.
- Service registry. It offers the registration of new platforms and services in the interoperability solution.
- Flow registry. It provides access to the new services created and its information, the easy execution and the possibility of reuse the flows.

An API is provided to manage, deploy, access and modify different instances of the interoperability core solution without the use of v Framework. In addition, an

automatic Docker instantiation of the solution has been developed, since the use of containers allows a centralized management of different instances of the solution that can work concurrently in a scalable way and can be located in different servers.

As a result of this work there are available around 20 nodes and 10 interoperability flows available in the INTER-IoT git repository. They are related directly with elements of the INTER-IoT project. These nodes cover different aspects, like access to IoT Platform services or translation of formats of messages exchanged. All the nodes and flows available are compatible with the Node-RED solution and can be reused without effort.

## 6 Conclusions

This chapter describes the approach followed by INTER-IoT to solve the interoperability problem at each one of the IoT layers. These solutions have been proposed as an attempt to resolve one of the main remaining challenges that blocks the growth of IoT systems in real environments.

The solutions proposed by INTER-Layer are based in the creation of adaptors, gateways and higher level components, like the middleware, which provides a new layer of abstraction that allows the communication of the different components through it. Each one of the solutions can be implemented independently, following its own architecture aiming at solving a concrete interoperability problem in a specific layer. However, some of them are usually implemented altogether. For instance, the Device-to-Device solution and Network-to-Network solution can conform an IoT deployment solution for lower levels, although they could perfectly work separately. The only exception is the Middleware-to-Middleware solution, as it requires the IPSM component of the Semantics module to allow interoperability at Platform and Semantic levels as a whole.

A summary of the main and derived products extracted from each layer can be observed in the following table:

| Layer | Core solution | Derived products |
|---|---|---|
| D2D | Physical and virtual gateway | Installer tool, automatic deployment tool, gateway extensions and SDK |
| N2N | SDN IoT controller | CLI, QoS application, dashboard and utilities |
| MW2MW | INTER-MW | IoT bridges, Docker-compose, examples and demo dashboard |
| AS2AS | INTER-AS | Instance manager, INTER-IoT services, flows and nodes |

Moreover, each solution exposes a standardized API that facilitates the extensibility of the system and the creation of new application tools at the top of INTER-Layer.

An example of this is INTER-FW, a management utility used to monitor the status of the layered solutions and to create new instances of each one. This utility makes use of the API and is installed combined with one or more INTER-Layer solutions. More information about this tool is described in this chapter.

## References


1. Gravina, R., Palau, C.E., Manso, M., Liotta, A., Fortino, G. (eds.): Integration, Interconnection, and Interoperability of IoT Systems. Internet of Things. Springer International Publishing (2018)
2. Fortino, G., Savaglio, C., Palau, C.E., de Puga, J.S., Ghanza, M., Paprzycki, M., Montesinos, M., Liotta, A., Llop, M.: Towards multi-layer interoperability of heterogeneous IoT platforms: the inter-IoT approach. Internet of Things 199–232 (2018)
3. Fortino, G., Palau, C.E., Guerrieri, A., Cuppens, N., Cuppens, F., Chaouchi, H., Gabillon, A. (eds.): Interoperability, Safety and Security in IoT—Third International Conference, InterIoT 2017, and Fourth International Conference, SaSeIot 2017, Valencia, Spain, November 6–7, 2017, Proceedings. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 242. Springer (2018)
4. Fortino, G., Garro, A., Russo, W.: Achieving mobile agent systems interoperability through software layering. Inf. Softw. Technol. **50**(4), 322–341 (2008)
5. Vermesan, O., Friess, P. (eds.): Digitising the Industry Internet of Things Connecting the Physical, Digital and Virtual Worlds. Riverpublishers (2016)
6. Broring, A., Zappa, A., Vermesan, O., Främling, K., Zaslavsky, A., Gonzalez-Usach, R., Szmeja, P., Palau, C., Jacoby, M., Zarko, I.P., Sour-sos, S., Schmitt, C., Plociennik, M., Krco, S., Georgoulas, S., Larizgoitia, I., Gligoric, N., Garcìa-Castro, R., Serena, F., Orav, V.: Advancing IoT Platform Interoperability. River Publishers, The Nederlands (2018)
7. ITU-T. Y.2060: Overview of the Internet of things. Technical report, The International Telecommunication Union SG13 (2012)
8. Internet Engineering Task Force (IETF). RFC 7228: Terminology for Constrained-Node Networks (2014)
9. Aloi, G., Caliciuri, G., Fortino, G., Gravina, R., Pace, P., Russo, W., Savaglio, C.: Enabling IoT interoperability through opportunistic smartphone-based mobile gateways. J. Netw. Comput. Appl. **81**, 74–84 (2017)
10. Tschofenig, H., Arkko, J., Thaler, D., McPherson, D.: Architectural considerations in smart object networking. Technical report, Internet Architecture Board (2015)
11. Ramalho, F., Neto, A.: Virtualization at the network edge: a performance comparison. In: WoWMoM 2016—17th International Symposium on a World of Wireless, Mobile and Multimedia Networks. Institute of Electrical and Electronics Engineers Inc., July 2016
12. Yacchirema, D.C., Esteve, M., Palau, C.E.: Design and implementation of a gateway for pervasive smart environments. In: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 4454–4459, October 2016
13. Zambrano, A., Perez, I., Palau, C., Esteve, M.: Quake detection system using smartphone-based wireless sensor network for early warning. In: 2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS), pp. 297–302 (2014)
14. Zambrano, A., Perez, I., Palau, C., Esteve, M.: D3.2. Methods for Interoperability and Integration v.2. INTER-IoT H2020 project, October 2017. https://inter-iot.eu/deliverables
15. Varga, L.-O.: Multi-hop energy harvesting wireless sensor networks: routing and low duty-cycle link layer. Ph.D. thesis, Grenoble, December 2015
16. Vázquez, T.A., Barrachina-Muñoz, S., Bellalta, B., Bel, A.: HARE: supporting efficient uplink multi-hop communications in self-organizing LPWANs. Sensors **18**(2), 115 (2018)

17. Omnes, N., Bouillon, M., Fromentoux, G., Grand, O.L.: A programmable and virtualized network it infrastructure for the Internet of Things: how can NFV SDN help for facing the upcoming challenges. In: 2015 18th International Conference on Intelligence in Next Generation Networks, pp. 64–69 (2015)
18. Kreutz, D., Ramos, F.M.V., Veríssimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-defined networking: a comprehensive survey. Proc. IEEE **103**(1), 14–76 (2015)
19. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008)
20. Internet Engineering Task Force (IETF). RFC 4627: The application/JSON Media Type for JavaScript Object Notation (JSON). Technical report, IETF Network Working Group (2006)
21. Zdravković, M., Trajanovic, M., Sarraipa, J., Jardim-Gonçalves, R., Lezoche, M., Aubry, A., Panetto, H.: Survey of Internet of Things platforms, February 2016
22. Partha Pratim Ray: A survey of IoT cloud platforms. Future Comput. Inf. J. **1**(1), 35–46 (2016)
23. Pileggi, S.F., Palau, C.E., Esteve, M.: Building semantic sensor web: knowledge and interoperability. In: Proceedings of the International Workshop on Semantic Sensor Web: SSW, (IC3K 2010), vol. 1, pp. 15–22. INSTICC, SciTePress (2010)
24. Giménez, P., Molìna, B., Palau, C.E., Esteve, M.: SWE simulation and testing for the IoT. In: IEEE International Conference on Systems, Man, and Cybernetics, pp. 356–361 (2013)
25. Bröring, A., Schmid, S., Schindhelm, C.K., Khelil, A., Käbisch, S., Kramer, D., Le Phuoc, D., Mitic, J., Anicic, D., Teniente, E.: Enabling IoT ecosystems through platform interoperability. IEEE Softw. **34**(1), 54–61 (2017)
26. Lemos, A.L., Daniel, F., Benatallah, B.: Web service composition: a survey of techniques and tools. ACM Comput. Surv. **48**(3) (2015)
27. Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., Reinfurt, L.: Comparison of IoT platform architectures: a field study based on a reference architecture. In: 2016 Cloudification of the Internet of Things (CIoT), pp. 1–6 (2016)
28. Belsa, A., Sarabia-Jacome, D., Palau, C.E., Esteve, M.: Flow-based programming interoperability solution for IoT platform applications. In: 2018 IEEE International Conference on Cloud Engineering (IC2E), pp. 304–309 (2018)
29. Blackstock, M., Lea, R.: Toward a distributed data flow platform for the web of things (distributed node-red), pp. 34–39, October 2014
30. Blackstock, M., Lea, R.: Fred: a hosted data flow platform for the IoT, pp. 1–5, December 2016
31. Kleinfeld, R., Steglich, S., Radziwonowicz, L., Doukas, C.: glue.things: a mashup platform for wiring the Internet of Things with the Internet of Services, October 2014