This project has received funding from the European Union's Horizon 2020
research and innovation programme under grant agreement N° 732679

ACTIVAGE PROJECT

ACTivating InnoVative IoT smart living environments for AGEing well

D3.10 ACTIVAGE bridges for European Platforms

Deliverable No.	D3.10 (D3.4.2)	**Due Date**	31-Decembre-2018
Type	Report	**Dissemination Level**	*Public*
Version	1.0	**Status**	Release 1 (23-April-2019)
Description	Report describing the status update of the development of IoT bridges		
Work Package	WP3 ACTIVAGE Secure Interoperability Layer		

Authors

Author	Company	e-mail
Stéphane Bergeon	CEA	Stephane.BERGEON@cea.fr
Levent Gurgen	CEA	Levent.GURGEN@cea.fr
Byron Ortiz	TELEVES	byrort@televes.com
Andrea Carboni	CNR	andrea.carboni@isti.cnr.it
Michele Girolami	CNR	michele.girolami@isti.cnr.it
Nikolaos Kaklanis	CERTH	nkak@iti.gr
Stefanos Stavrotheodoros	CERTH	stavrotheodoros@iti.gr
Konstantinos Votis	CERTH	kvotis@iti.gr
Dimitrios Tzovaras	CERTH	dimitrios.tzovaras@iti.gr
Martin Serrano	NUIG	martin.serrano@insight-centre.org
Hugo Hromic	NUIG	hugo.hromic@insight-centre.org
Philippe Dallemagne	CSEM	philippe.dallemagne@csem.ch
Pierre Barralon	TECNALIA	pierre.barralon@tecnalia.com
Mary Panou	CERTH	mpanou@certh.gr
Saied Tazari	Fh-IGD	saied.tazari@igd.fraunhofer.de
Regel G. Usach	UPV	regonus@doctor.upv.es

Document History

Date	Version	Editor	Change	Status
November 2018	0.1 – 0.2	S. Bergeon	Initial version from D3.4 with update assignations to partners	draft
December 2018	0.3	S. Bergeon	Added updates from CERTH, CSEM, UPV and TELEVES	draft
February 2019	0.4	S. Bergeon	Added updates from CERTH, MYS, TEC, UPV and FhG	draft
March 2019	0.5	S. Bergeon	Added updates from UPM and CEA	draft
April 2019	1.0	S. Bergeon	Integrating contributions, internal review, building the final version	final

Document Key Data

Keywords	Internet of Things, UniversAAL, FIWARE, Sofia2, OpenIoT, sensiNact, IoTivity, SENIORSome, OneM2M, IoT platform, IoT protocol, middleware, interoperability, bridge
Editor Address data	Name: Stéphane Bergeon Partner: CEA
Internal Reviewers	Pierre Barralon, Tecnalia Mary Panou, CERTH

Abstract

This deliverable contains an update to the Deliverable 3.4 “ACTIVAGE Bridges for European Platforms”. For the sake of clarity and completeness, it begins with a summary of selected ACTIVAGE IoT platforms and edge protocols that are used in ACTIVAGE deployment sites. The second part presents the southbound bridges (connections to IoT devices) as they are implemented by each ACTIVAGE IoT platform. A final conclusion section provides a synthetic summary of the bridges per platforms, future work and recommendations for developping remainign support for IoT protocols in some of the deployment sites.

Scope

This document is about the IoT platforms used in the ACTIVAGE deployment sites and the communication bridges used to connect to the deployed IoT devices. The deliverable presents a review of the available bridges from/to sensor and actuator devices’ protocols included in the 7 selected IoT platforms considered within the ACTIVAGE project. It also presents the connections among the IoT platforms for the purpose of interoperability among them. Even if most of the developments have been finalised, some remaining implementations are also planned and presented in this deliverable. The final goal is to implement the ACTIVAGE interoperability layer integrating the bridge features from ACTIVAGE IoT platforms. This deliverable does not deal with the interoperability bridges between the SIL layer and the IoT platforms, which will be studied and reported in the Deliverable 3.11.

The following Figure illustrates the scope of the deliverable with respect to the overall architecture of the AIOTES.

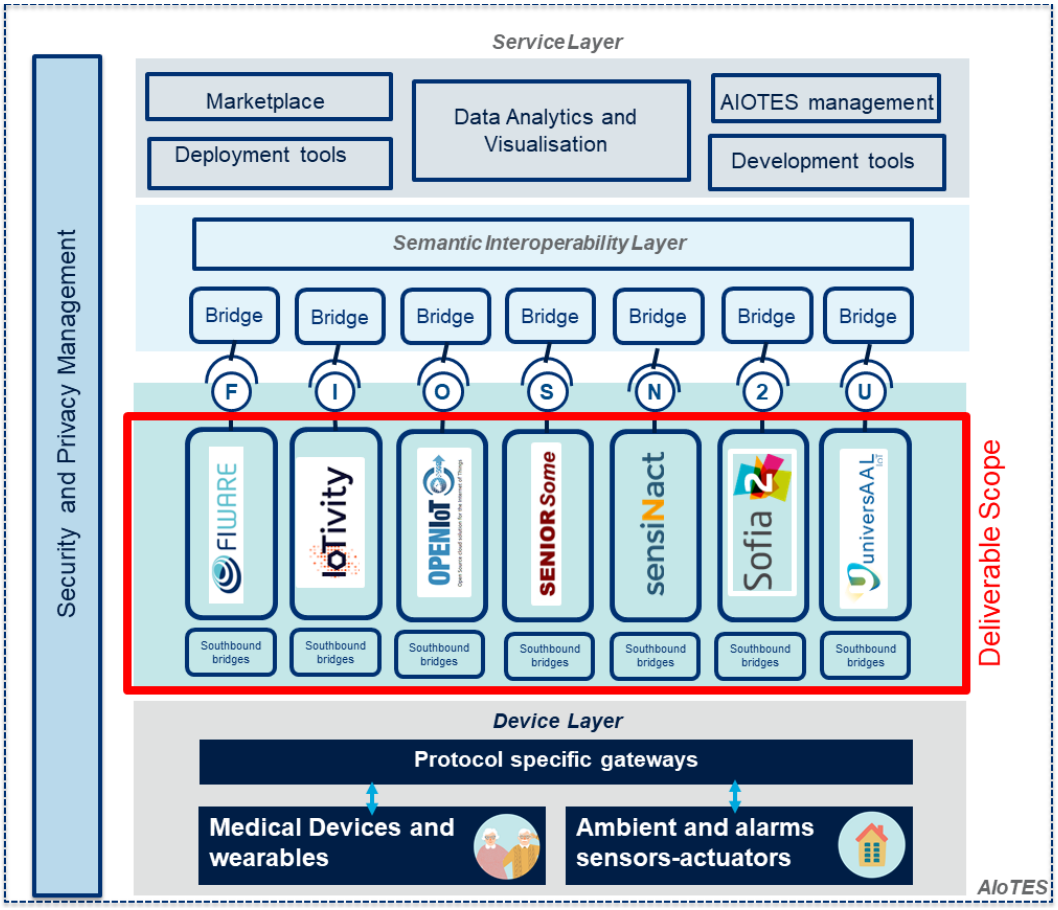

Figure 1 Deliverable scope within the ACTIVAGE overall architecture

Table of contents

List of tables

List of figures

About This Document

This deliverable reports the progress of the following tasks belonging to WP3, until M24: T3.3 “Building bridges to the IoT protocols and platforms”. It is the updated version of the previous D3.4 “ACTIVAGE bridges for European platforms”.

Deliverable context

The following table summarises the deliverable’s relation with respect to various project items.

Project item	Relationship
Objectives	O1 - Semantic interoperability layer to allow integration and interoperability of heterogeneous platforms; Framework and API to allow the connection of new services and interact transparently with IoT platforms.
Exploitable results	D3.4 and D3.10 contribute to the creation of the ACTIVAGE IoT Ecosystem Suite by describing the technologies provided at IoT platform level.
Work plan	Deliverable D3.10 is the main outcome of the task T3.3.
Milestones	MS1 - BUILD - All DS ready and solution integrated. D9.1 Completed. MS2 - DEMONSTRATE - Smart Living environments on each DS are deployed. D5.3 Completed. MS3 - EXPAND - DSs cooperate bi-laterally or tri-laterally. Use cases imported from other DS.
Deliverables	Deliverable D3.4 v1.0 was released on M10. The D3.4 v1.1 corrective version after the first project review was delivered in M14.
Risks	D3.10 contributes to gaining control of the following risks: 14 - Use of multiple technologies becomes too time consuming 15 - Difficult to achieve technical integration/interoperability between existing platforms. 19 - Harm to participants or violation of their fundamental rights due to data errors, datasets poor quality or reliability, or underestimated methodological limitations.

Version-specific notes

In this updated version of the document D3.10 “ACTIVAGE bridges for European platforms” the following changes have been applied with respect to the previous document:

- Specific updates per platform on implementing the new bridges and/or updating the existing ones
- A new section has been added in the current document, called “Summary of bridges updates”. This section summarizes updates that have been performed on bridges implementation among the ACTIVAGE IoT platforms, in a synthetized way so as to easily measure the progress of task T3.3 “Building bridges to the IoT protocols and platforms” during the period M9-M24.

1 Introduction

There is a variety of IoT platforms used today in the Ambient Assisted Living domain. Deliverable D3.1 proposed an overall view of the IoT platform market in Europe by presenting the current state-of-the-art of IoT platforms and a selection of those to be used in ACTIVAGE.

One of the main technical purposes of the ACTIVAGE project is to avoid creating silos of un-interoperable platforms which fragments the AHA application market, limits the reusability of hardware and software components in different contexts and creates compatibility problems among solutions.

In ACTIVAGE we deal with two layers of heterogeneity in response to the un-interoperability problem:

1) Heterogeneity of IoT device protocols, which is managed by individual IoT platforms deployed in ACTIVAGE deployments sites
2) Heterogeneity of IoT platforms, which is managed by ACTIVAGE's Semantic Interoperability Layer

This deliverable is specifically dealing with the first point, while the second point is dealt with the deliverable D3.11.

The following Figure succinctly illustrates the overall architecture of the AIOTES.

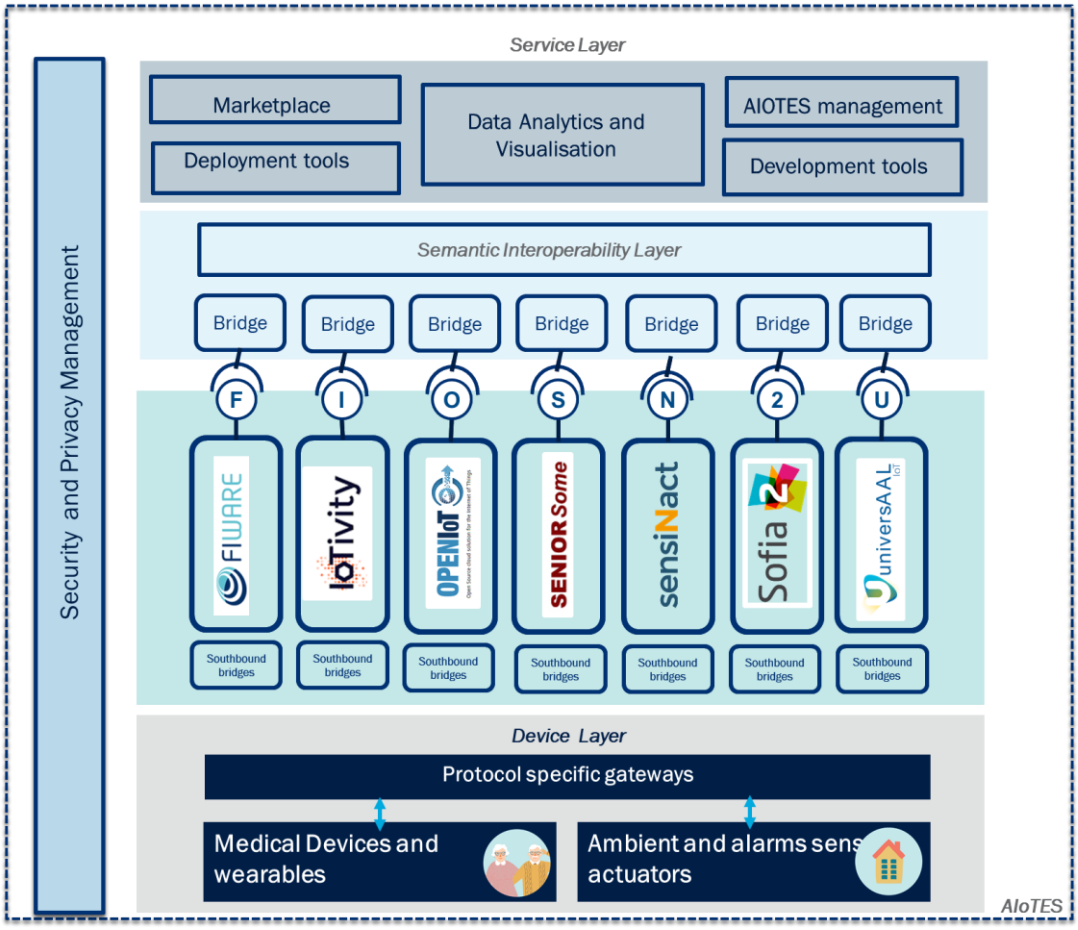

Figure 2 Overall architecture of AIOTES

The IoT platforms used in the project are in charge of managing the heterogeneity of the underlying IoT devices deployed in the environment (homes, care centers, outdoor in the cities, etc.). The following seven open-source application-oriented IoT platforms are being used in the project:

- universAAL IoT [1]
- Sofia2 [2]
- OpenIoT [3]
- sensiNact [4]
- FIWARE [5]
- IoTivity [6]
- SENIORSome [7]

The set of southbound bridges, which are detailed in this deliverable, are in charge of providingig the connection from and to the deployed IoT devices using various communication protocols. In this deliverable, for each of the aforementioned platforms, the main conceptual characteristics are presented, along with a description of the communication protocols and underlying data models. Such data models are used to transform raw data collected by the devices to meaningful (semantic) information that can be used in the application logic.

This report, aims to complement and precise the assets that will be used during the implementation of the ACTIVAGE components of WP3, WP4 and WP5, by making the state of the art on bridges implementation among the ACTIVAGE IoT platforms.

The ultimate goal of ACTIVAGE is to provide a secure and flexible interoperability layer capable of offering access to all the services (of the seven platforms) from a single-entry point.

2 Definition of IoT platform bridges

A **bridge**, in the IoT domain, is a software component that can read and write data necessary to communicate with a dedicated IoT platform. When reading, the IoT bridge unpacks the data from the received message, following a dedicated format as specified by the communication protocol. When writing, the bridge packs the data in the protocol telegram (also called payload).

We use the layer where the communication occurs to classify the IoT bridges:

- **Device layer** (or **Sensor layer**) **bridges** deal with communication towards and from the sensors and actuators.
- **Service layer** (or Middleware layer) **bridges** manage communication towards and from services (e.g. for applications, or for other IoT platforms). When a protocol is particularly devoted to communication between more than one IoT platform, we can precise **Interoperability layer**.

We also classify the bridges into five categories, depending on the source and destination edges of the communication:

- **Device to gateway** communication: device communicates directly to the gateway either through cable or radio with dedicated antenna (on-chip inside or with dongle plugged to the gateway). The bridge is hosted in the gateway among the IoT platform component, sending and receiving data to/from the sensors and actuators connected with the gateway.

  Examples of IoT protocol in this category: ZWave, EnOcean

- **Gateway to gateway** communication: the gateway communicates with another IoT platform (that could be hosted in another gateway, but could also be hosted in a cloud server). We use this dedicated naming for the protocol NGSI 9 & 10 v1 and v2.

- **Gateway to Wide Area Network (WAN)** communication: the gateway communicates with the rest of the world. The bridge is hosted in the gateway among the IoT platform components, and it is sending and receiving data to/ from the WAN, communicating with other IoT platforms, or providing top level services accessible from the application layer.

  Examples of IoT protocol in that category: RESTfull API, MQTT, CoAP

- **Device to Local Area Network (LAN)** communication: sensor and actuators are not directly connected to the gateway hosting the IoT platform. There is neither a direct cable connexion nor a direct Radio Frequency connexion with a dongle (or an antenna). The device is a node in the Local Area Network, as the gateway. Device and gateway communicate through Ethernet on this LAN. The IoT bridge hosted in the gateway manages the Ethernet communication with other nodes on the LAN.

  Examples of IoT protocol in that category: OpenHab, Philips hue

- **Device to WAN** communication: the sensors and actuators communicate to a third party IoT platform deployed in the cloud. The bridge (hosted either in the gateway or

in the cloud) manages the Ethernet communication with the WAN node of the third party platform.

Examples of IoT protocol in that category: LoRa, Sigfox

Sometimes the notion of southbound and northbound bridge is used. It has substance when a tree-like communication model is used with IoT platforms as nodes and sensors/actuators as leafs.

Southbound bridges are bridges dedicated to feed the IoT platform. The IoT platform gathers data from sensors and dispatches commands to actuators through southbound bridges. Mainly southbound bridges are at the device layer, but by gathering data from other IoT platforms and services, southbound bridges are also in the service and interoperability layers.

Northbound bridges are dedicated to communicate with a higher level node in the communication tree: an aggregator IoT platform, the interoperability layer or applications.

3 ACTIVAGE bridges for IoT edge protocols

This section presents the available bridges of the selected IoT platforms towards the IoT edge protocols used in the deployment sites.

3.1 FIWARE bridges status

FIWARE [5] is an open middleware platform for IoT, supported by the European Commission under the Future Internet Public Private Partnership Programme [9]. FIWARE provides public and royalty-free API specifications and interoperable protocols for the creation of new internet services and applications. Moreover, reference open-source implementations of its components are freely available.

FIWARE provides a series of Generic Enablers (GE) which implement interfaces to IoT devices and transforms the protocol artefacts into internal FIWARE calls. These GE, running on the cloud/edge domain, are built on top of the IDAS GE and each enabler implements an adapter to specific protocols.

The official agents provided by FIWARE are:

- JSON agent: a bridge between HTTP/MQTT messaging (with JSON payload) and NGSI
- LWM2M agent: a bridge between Lightweight M2M protocol and NGSI
- Ultralight agent: a bridge between HTTP/MQTT messaging (with UltraLight2.0 payload) and NGSI
- LoRaWAN agent: a bridge between LoRaWAN protocol and NGSI using CoAP transport
- OPC-UA agent: a bridge between the OPC Unified Architecture protocol and NGSI

There are other unofficial GEs for different protocols:

- OpenMTC agent: open source implementation of the OneM2M standard which includes a northbound interface with NGSI.
- RTPS agent: agent implementing the Real Time Publish Subscribe (RTPS) protocol [10], which is used to interface to robotic systems implementing the ROS2 operating system.

An IoT Agent is a component that mediates between a set of Devices using their own native protocols and an NGSI compliant Context Provider (i.e. Fiware Context Broker).

The following picture (Figure 3) depicts the high-level architecture of the IoT Agents framework, implemented by the IDAS GE. An ‘IoT Agent Manager’.

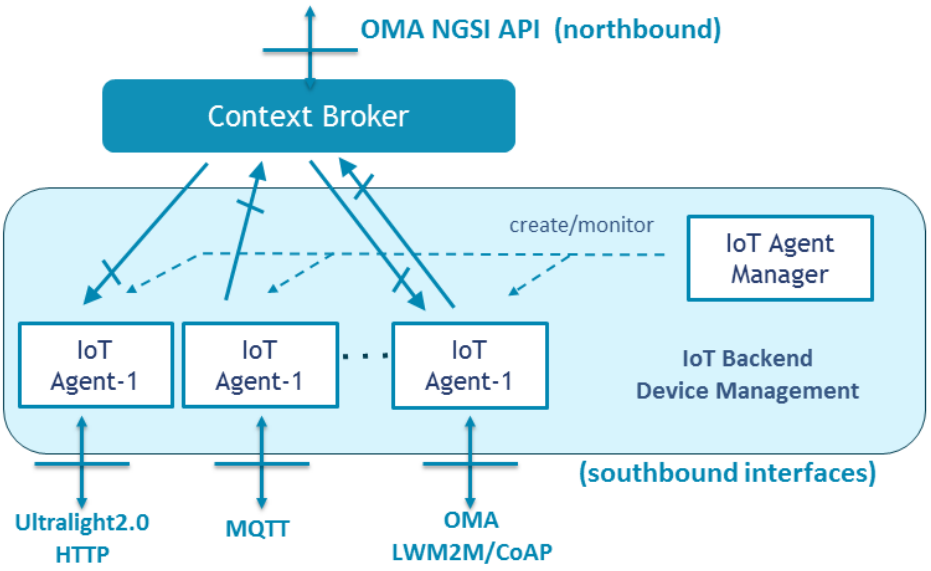

Figure 3: FIWARE IoT Agents to connect different protocols

Southbound Traffic (a.k.a. Commands) are HTTP requests generated by the Context Broker and passed downwards towards an IoT device via an IoT agent. Southbound traffic consists of commands made to actuator devices which alter the state of the real world by their actions. Northbound Traffic (aka Measurements) are requests generated from an IoT device and passed back upwards towards the Context Broker via an IoT agent. Northbound traffic consists of measurements made by sensor devices and relays the state of the real world into the context data of the system.

The IoT Agent Manager implements both the Configuration and the Provision API. The Provision API allows the manipulation on an IoT Agent in order to publish custom information in the IoT Platform. The provisioning API provides the functionality allowing devices to be preregistered, so all the information about service and subservice mapping, security information and attribute configuration can be specified in a per device way instead of relying on the type configuration.

The Configuration API allows the provision of different device groups, each of one mapped to a different type of entity in the context broker. This API is used when there is no need to provision individual devices.

3.1.1 CoAP Bridge

Constrained Application Protocol (CoAP) protocol [11] is standardized by the Constrained RESTful Environments (CoRE) workgroup, and is available as an active (IETF) Internet-draft. CoAP is based on a request/response communication model between entities (e.g. a sensing node and server), including key concepts found on the Web, thus enabling easy integration with the standard Web services.

FIWARE provides two GE which implements the use of the Ligthweight M2M protocol and LoRaWAN protocol over a CoAP transport. The CoAP capabilities are provided by the open source library coap [12], but the mapping to the LWM2M protocol only implements a subset of features provided by CoAP.

3.1.2 HTTP Rest bridge

REST services are the default communication pattern used by FIWARE components. All GEs expose a REST API which can be invoked to trigger the internal functionalities and communicate to the context broker.

As such, this is not really a bridge, but a core implementation pattern.

3.1.3 MQTT bridge

MQTT protocol follows a publish/subscribe paradigm, which allows multicast communications as opposed to the one-to-one communication offered by REST services through the request/response paradigm.

This bridge integrates the Mosquitto broker, an open source implementation of the MQTT infrastructure [13]. This bridge can be combined with protocol specific bridges to enable MQTT over different protocols. FIWARE already provides integration with Ultralight2.0 and JSON formats.

3.1.4 NGSI 9 & 10 v1 and v2

NGSI v1.0 is a technical specification created by the Open Mobile Alliance (OMA) with the focus of the standardization of functional interfaces for Data Configuration and Management, Call Control and Configuration, Multimedia List Handling, Context Management, Identity Control, Service Registration and Discovery functions.

NGSI 9/10 v1 [14] covers the interfaces regarding the context management component. In more detail:

- NGSI 9 defines the management of the Context Information about Context Entities and of its specific Context Information elements. The Context Information includes an identifier for a specific Context Entity, the type of Context Entity, attributes and their values, such as a person (=type) and its location(=attribute). It may also include meta data about the Context Information, for example, the lifetime and quality of information
- NGSI 10 defines the access (query, subscribe/notify) to the available Context Information about Context Entities, including Context access specific conditions such as operation scopes to reduce the query cost and restrictions based on the attribute values, etc.

FIWARE's Orion Context Broker implements both interfaces (NGSI 9 and NGSI 10) and it is the core of its architecture.

There are two versions of the NGSI API implemented, V1 and V2 (these versions are related to the API provided by the Orion Context Broker and have nothing to do with the v1.0 in the NGSI technical specification). The latest CB is based on the V2 API, but supports V1 for backwards compatibility, although V1 has been deprecated.

The complete documentation for both version of the API can be found at:

- V1: https://fiware-orion.readthedocs.io/en/1.7.0/user/walkthrough_apiv1/index.html
- V2: https://fiware-orion.readthedocs.io/en/1.7.0/user/walkthrough_apiv2/index.html

3.1.5 NGSI-LD bridge

NGSI-LD is a specification [15] produced by ETSI Industry Specification Group (ISG) cross-cutting Context Information Management (CIM), which aims at evolving the OMA NGSI-9/10 information model to support linked data, property graphs and semantics.

NGSI-LD is an information model prescribing the structure of context information and specifying the data representation mechanisms that the NGSI-LD API should use. The information model is defined at two levels:

1) Foundational classes which correspond to the core meta-model, which is a formal specification of the property graph model,
2) The cross-domain ontology, composed of transversal classes, aims at avoiding conflicts or redundant definitions of the same classes in each domain-specific ontology.

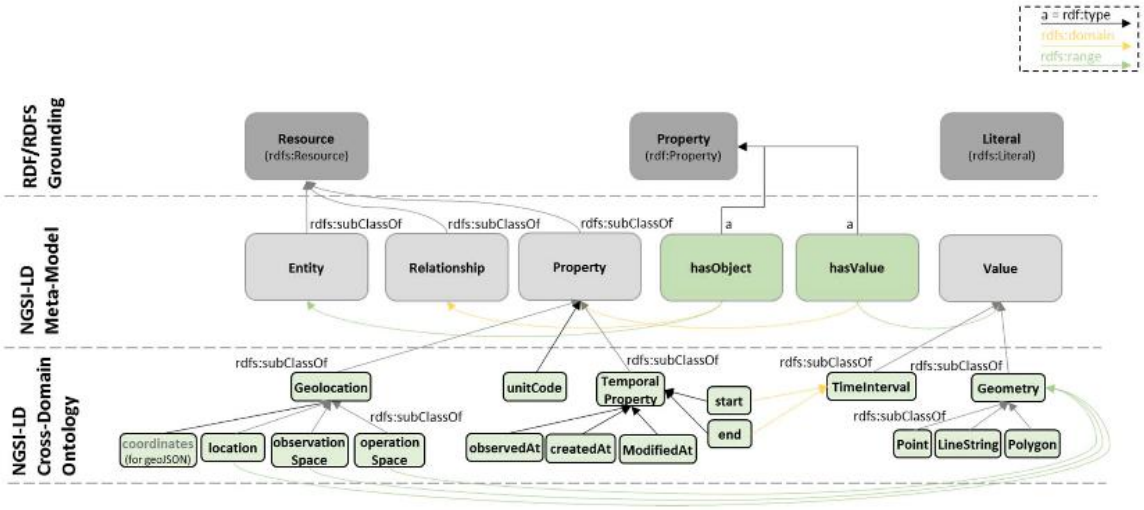

Figure 4: NGSI-LD Core meta model and cross-domain ontology

FIWARE provides a proof-of-concept bridge implementing the NGSI-LD standard on top of the Orion Context Broker. Since NGSI-LD is a generalization of NGSI-9/10, there is a great level of compatibility and a clear migration path between both models.

However, the current status of the NGSI-LD specification is still a draft and the roadmap does not foresee a public release before 2020, so the provided bridge implementation may change without providing backwards compatibility.

3.1.6 Sigfox bridge

FIWARE provides an official IoT Agent implementing a bridge between Sigfox call-backs and NGSI protocol.

For each device, the Sigfox backend provides a call-back mechanism that is used to send to types of information:

- Attributes defined by the Sigfox backend itself.
- Free data format, whose structure is defined in the device type.

The agent provides:

- Northbound functionalities

- A Sigfox endpoint listening call-backs from the Sigfox backend
- A Sigfox data parser that can be used to convert the data format as defined in the call-backs to a JavaScript array.
- A testing tool that simulates data coming from the device.

The Sigfox agent can be downloaded from the official repo: https://github.com/telefonicaid/sigfox-iotagent

3.1.7 Zigbee and Z-wave bridges

Both protocols require the use of dedicated hardware implementing the radio communications. As FIWARE is a cloud-based IoT platform, it does not natively implement the connectivity to those protocols.

However, FIWARE provides examples on how to implement a gateway application that can act as bridge between the radio interface and the IoT Agents framework provided by the IDAS GE. These examples are based on an open-source implementation of such gateway using a Raspberry Pi device, a device running macOS or any linux distribution, and it is publicly available at https://github.com/telefonicaid/fiware-figway

However, there is no implementation available mapping Zigbee or Z-Wave protocols.

3.1.8 BLE Bridge

The Bluetooth Low Energy bridge provides the mechanisms to translate northbound device messaging from BLE devices into NGSI context update requests.

The implemented bridge supports the Generic Access Profile (GAP) for manufactures specific services working over the broadcast channel with non-connectable devices (ADV_NONCONN_IND transmissions) advertising information to any listening device.

The current bridge version, implemented as an Android library, integrates the devices manufactured by MySphera (providing volumetric presence, magnetic switch events, temperature and humidity), and can be extended to support any other off-the-self device using the BLE standard.

3.1.9 Summary of FIWARE IoT bridges status

The Table below provides the IoT protocol bridges for FIWARE.

IoT Protocol bridges for FIWARE					
Protocol	**Domain**	**Layer**	**Connection edges**	**Status**	**Comment**
COAP	Generic	Sensor	Device to Gateway	Partial implementation	Abstract bridge tailored to LWM2M and LoRaWAN protocols.

HTTP REST	Generic	Service	Gateway to WAN	Implemented	Abstract bridge; useful for further HTTP bridges. Southbound and northbound
MQTT	Generic	Service	Gateway to WAN	Implemented	Northbound and southbound
NGSI 9 & 10 v1	Generic	Interoperability	Gateway to Gateway	Implemented	Core implementation in FIWARE. Deprecated in favour of V2
NGSI 9 & 10 v2	Generic	Interoperability	Gateway to Gateway	Implemented	Core implementation in FIWARE
NGSI-LD	Generic	Interoperability	Gateway to Gateway	Implemented	Provides support for the draft NGSI-LD specification
Sigfox	Generic	Sensor	Device to WAN	Implemented	RF
Zigbee	Generic	Service	Device to Gateway	To be implemented	RF
Z-Wave	Generic	Sensor	Device to Gateway	To be implemented	RF
BLE	Domotic	Sensor	Device to Gateway	Implemented	Northbound advertising communication

Table 1: FIWARE supported and planned IoT protocol bridges

FIWARE will be connected to other platforms via the SIL layer. In addition, FIWARE has direct bridges to some of the platforms, such as to sensiNact via NGSI and to IoTivity via MQTT.

3.2 SOFIA2 bridges status

Sofia2 [16] is a semantics-based IoT platform, utilizing ontological models for the connection between the low-level device information and the applications.

The basic concepts to understand how SOFIA2 works are:

- **Smart Space:** This is the virtual environment where different applications interoperate with each other to provide complex functionality. Commonly, there is a single SIB (that may be a cluster of SIBs) in each Smart Space but in specific cases a Smart Space may have a federation of SIBs to communicate different ontologies. A

Smart Space can communicate with other Smart Spaces by establishing trusted relationships with them. The Smart Space's core is the SIB.

- **Semantic Information Broker SIB**: This is the core element of the Platform and it receives, processes and stores all the information of applications connected to the SOFIA2 Platform, thus acting as the Interoperability Bus.

  All the existing concepts in the domain (reflected in the ontologies) and their current states (specific instants of the ontologies) are reflected on it.

  SOFIA2 proposes the use of JSON structures to exchange information (SSAP) and to define the ontologies. There are implementations in several languages and platforms. Indra provides a JEE SIB that runs over JEE Web Server (Tomcat, JBoss, etc.) and the SIB has connectors to communicate from different clients like: REST, MQTT, Web Services/JMS for Enterprise applications, etc.

- **Knowledge Processor KP:** The KP allows access to connect each application with the Smart Space through the SIB. Each application works with instances of the relevant concepts, inside the domain (ontology) for which it is designed. That access allows the execution of operations with the SIB (ontology).

Within a SOFIA2 SmartSpace, the SIB acts as a communication gateway with the KPs, receiving SSAP messages.

- The SIB supports several communication protocols /bridges out-of-the-box, allowing the KPs to connect with the SIB.
- Those protocols/bridges can be enabled or disabled using the settings.
- The SIB also offers the plug-in concept to develop new connectors and to enable new protocols/bridges that fit specific needs.

Below is showing the main protocol bridges provided by SOFIA2.

3.2.1 MQTT

MQTT (Message Queue Telemetry Transport) is a connectivity protocol focusing inM2M (machine-to-machine) and IoT (Internet of Things). It is a lightweight messaging protocol based on TCP and especially designed for remote devices with little memory and little processing power. It is based in a publish/subscribe messaging model that eases one-to-many distribution. Sofia2 has implemented MQTT bridge in order to communicate from its middleware to applications.

3.2.2 RESTful

Sofia2 provides a RESTful Gateway to invoke operations on that instance. This Gateway works around SSAPResource, which represents, along with the Gateway HTTP verbs, the different SSAP operations.

The RESTful Gateway accepts and returns information in JSON format. Therefore, both requests to and responses from the Gateway include in their headers the attribute Content-type: application/json, and the SSAPResources are sent in the request body coded as a JSON object.

3.2.3 Ajax Push (JavaScript) AJAX

AJAX (Asynchronous JavaScript and XML) is a web development technique to create interactive applications or (Rich Internet Applications). Those applications run in the client, meaning in the user's browser, while at the same time an asynchronous communication with the server is kept on the background. Ajax is an asynchronous technology, meaning that additional data are requested to the server and then are downloaded in the background, avoiding any interference with the page's display or behaviour. Ajax allows for the interaction with the server's Java code using the browser's JavaScript and vice versa: Making call to JavaScript functions in the user's browsers from the server's code (reverse Ajax) and editing the web pages with the results of the queries in a way that is transparent to the user.

3.2.4 Websocket

WebSocket is a technology that provides a bidirectional communication channel and full-duplex on a single TCP socket. It is designed to be implemented in browsers and web servers, but can be used by any client/server application. The use of this technology provides similar functionality to opening multiple connections in different ports, but multiplexing different WebSocket services over a single TCP port (at the cost of a small protocol overhead) .

3.2.5 Summary of SOFIA2 IoT bridges status

SOFIA2 provides several protocol bridges to communicate gateways and devices. The Deployment Site (DS1) has several needs witch force to have the control of the communications for devices, these needs are: security, communication in an easy way, medical standardization, management of the devices, etc. For these reasons the communications of all devices pass through the gateway and that device is responsible to provide: security, communication in an easy way, medical standardization, management of the devices, secure updates, etc.

Because of this, Televes has created several bridges inside the gateway (Device to Gateway) that allows the communication between all devices (medical devices, environmental devices, and alarm devices) with the gateway and through the gateway with SOFIA2 using the SOFIA API Rest.

Next table provides the IoT protocol bridges for SOFIA2.

IoT Protocol bridges for SOFIA2					
Protocol	**Domain**	**Layer**	**Connection edges**	**Status**	**Comment**
Z-Wave	Generic	device	Device to Gateway	Implemented	
RF	Generic	device	Device to Gateway	Implemented	
BLE	Generic	device	Device to Gateway	Implemented	
Zigbee	Generic	device	Device to Gateway	Implemented	

HTTP REST	Specific API Rest of SOFIA2	service	Gateway to Wide Area Network (WAN)	Implemented	
Websocket	Specific API for SOFIA2	service	Gateway to Applications	Implemented	
Ajax Push	Specific API for SOFIA2	service	Gateway to Applications	Implemented	
MQTT	Generic	service	Gateway to Applications	Implemented	

Table 2: SOFIA2 supported and planned IoT protocol bridges

Sofia2 will be connected to other platforms via the SIL layer.

3.3 universAAL IoT bridges status

universAAL [17] is an IoT open source platform that enables seamless interoperability of devices, services and applications in distributed systems. Its open source middleware can be integrated into many devices, products, and service solutions, regardless of branding. When embedded "universAALized", components will communicate automatically, exchanging data and functionality that can be processed and reacted to.

In universAAL, the conceptual approach for integrating not "native" components – e.g., sensors and actuators using different communication protocols – is following a pattern that is very common among IoT platforms, namely a general-purpose reference architecture consisting of three layers, as depicted in Figure 5.

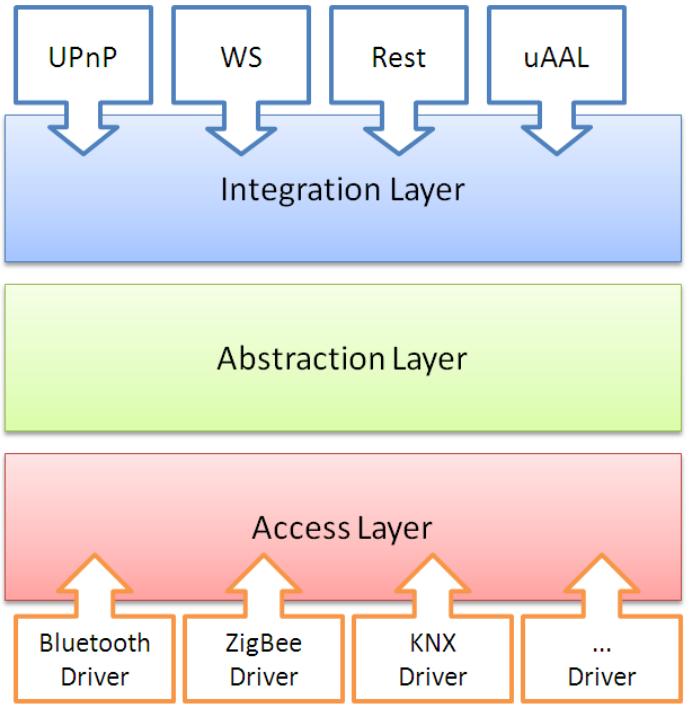

Figure 5: universAAL Access, Abstraction and Integration layers

Service or Access Layer to is usually composed of a number of technology drivers, sometime already integrated in the operating system with well-defined API (e.g. Bluetooth), sometime written from scratch either with an ad hoc solution or using established libraries if standardized API is not available at the OS level (e.g ZigBee, KNX). So, this is the level to deal with the communication protocols.

The **Abstraction layer** is about the abstract representation of devices independently from the communication protocols. At this level, proxies are created for each device that can be accessed via the Access Layer. The proxies hide the protocol-specific communication details and represent the actual devices in a technology-neutral way.

Finally, the **Integration Layer** publishes the proxies instantiated in the abstract layer by creating new endpoints in the universAAL network; however, Figure 5 reveals that this approach is general enough for using the device representations on the Abstraction Layer for integrating the devices not only into the universAAL network but also in other networks (e.g., a UPnP network) or make them available globally with a REST API or as a Web Service). This enables the universAAL community to share device bindings up to the Abstraction Layer also with other communities.

The whole of this three-layer architecture (including the integration into universAAL) serves as what has been called “the device layer bridge”. In centralized deployments of universAAL on one single device serving as “gateway”, the “device layer bridges” implementing this three-layer architecture will reside on the gateway. In decentralized deployments of universAAL, such device layer bridges can be distributed on any universAAL node in the uSpace (the virtual space covered by a certain universAAL installation – whether in centralized or in decentralized form – that is protected by a “space”-specific key).

3.3.1 KNX universAAL bridge

KNX is a standardized (EN 50090, ISO/IEC 14543), OSI-based network communications protocol for intelligent buildings. KNX defines several physical communication medias: Twisted pair wiring, Powerline networking, Radio, Infrared and Ethernet. It is designed to be independent of any particular hardware platform. A KNX Device Network can be controlled by anything from an 8-bit microcontroller to a PC, according to the needs of a particular implementation.

KNX features a twofold structure: on the one hand there are single devices (sensors/actuators) where one device can have multiple inputs or outputs, which are called *Communication Objects*. Those *Communication Objects* can be grouped in so called *Group Addresses*. *Group Addresses* can contain *Communication Objects* from sensors (e.g. switch) and actuators (e.g. light controller). Therefore, *Group Addresses* can be seen as Functions (e.g. switching/controlling a light, control the heating/cooling, control shutters/blinds, etc.). All devices which should be controlled or monitored must be included in a *Group Address*.

For the Home/Building Automation domain universAAL provides a solution for integrating KNX, with a bundle suite (github.com/universAAL/lddi/tree/master/lddi.knx) that provides interaction facilities with KNX sensor networks. The suite includes the KNX message protocol and provides monitoring and control features for KNX devices (sensors and actuators) and exports them to universAAL buses. UniversAAL LDDI focuses on *Group Addresses*, not on single inputs/outputs (sensors/actuators).

3.3.2 ZigBee universAAL bridge

ZigBee [18] is a specification for small, low-power digital radios based on an IEEE 802 standard for personal area networks. Applications include wireless light switches, electrical meters with in-home-displays, and other consumer and industrial equipment that requires short-range wireless transfer of data at relatively low rates. The technology defined by the ZigBee specification is intended to be simpler and less expensive than other WPANs (wireless personal area networks), such as Bluetooth. ZigBee is targeted at radio-frequency (RF) applications that require a low data rate, long battery life, and secure networking.

The core components (Access and Abstraction layers) are provided by the ZB4OSGi project [19] as an open contribution to the OSGi community. Therefore, the integration of ZigBee into universAAL (github.com/universAAL/lddi/tree/master/lddi.zigbee) builds on top of this general-purpose OSGi solution. It provides for the following features:

- ZigBee Commissioning: Simulated Device to bind on cluster and configure reporting
- ZigBee Exporter: Implements universAAL wrappers for ZB Home Automation devices
  - Sends context events for incoming sensor events to the universAAL context bus
  - Provides services on the universAAL service bus for querying devices.

3.3.3 Bluetooth universAAL bridge (Continua-certified products)

universAAL supportsthe Continua Health Alliance certified devices (also called agents or sources) from the healthcare domain that use the wireless and low-cost Bluetooth technology (github.com/universAAL/lddi/tree/master/lddi.bluetooth).

The integration of Continua certified devices [20] inside universAAL is based on three different components (some of them can be shared between all environments without exception):

A component in charge of managing Bluetooth HDP connections and disconnections. Interface with the lower layers and Bluetooth drivers.

A component in charge of understanding received HDP frames from Continua agents and accordingly move through the appropriate x73 state machine diagram.

A component in charge of integrating HDP messages/Health data inside universAAL middleware.

The first component had to be ad-hoc developed for Windows and GNU/Linux OS because the Bluetooth stack is not the same in both architectures. The second and the third components can be re-used on different platforms without adding external libraries neither dependencies.

The provided universAAL bridge for Bluetooth has been tested with at least two Continua-certified devices namely a Weighing Scale and a Blood Pressure Monitor. They both transmit over **Bluetooth Health Device Profile (HDP)**. Furthermore, the Continua-proposed standard for in-home sensors measuring user activity, so called *Independent Living Activity Hub* (ISO 11073-10471) is adopted for sensor integration in universAAL.

3.3.4 FS20 universAAL bridge

FS20 is a simple wireless protocol developed by the German company ELV for the low-cost market segment of home appliances [21]. Its main advantage is the cheap price of hardware devices. Though the protocol is not that robust than comparable protocols in this domain (e.g. KNX or ZigBee).

The protocol is based on the exchange of simple events/commands between event-receiver and event-sender. In the universAAL project the integration of FS20 is based on existing software/driver components from the PERSONA project.

The FS20 integration (github.com/universAAL/lddi/tree/master/lddi.fs20) uses a two layer design comparable to the KNX Integration. Contrary to the OSGi DAS all FS20 devices are registered in the access layer via a service registration to OSGi. The exporter bundle in the integration layer notices the OSGi ServiceEvents and handles them.

3.3.5 ZWave universAAL bridge

ZWave is a home automation communication specification, similar to ZigBee although it was developed as a proprietary solution. The integration into universAAL is dependent on specific vendor devices, unlike the other, standard-based technology integrations. The existing ZWave Exporter (github.com/universAAL/lddi/tree/master/lddi.zwave) provides for the following features:

- Connects to ZWave gateway hardware
- Implements universAAL wrappers for ZWave Home Automation devices
- Sends context events for incoming sensor events to the universAAL context bus.
- Provides services on the universAAL service bus for querying devices.

3.3.6 Eclipse Smarthome (openHAB) universAAL bridge

Eclipse Smarthome (ESH) is the 2.0 version of OpenHAB [22]. It is a framework that integrates with several home automation standards, protocols and technologies to represent all their devices in a unified way. By integrating it, universAAL can get access to a lot of devices (github.com/universAAL/lddi/tree/master/lddi.smarthome), leaving the hardware integration to be performed by ESH.

Although ESH and universAAL share the same runtime environment, but there are some incompatibilities due to system bundles and configurations. For this reason it is not possible to simply install standard universAAL Karaf features, instead there is a dedicated Karaf feature that has everything sorted out to install universAAL Middleware and the ESH Exporter in the ESH runtime. Then it is safe to rely on universAAL connectivity to link to other normal universAAL instances.

3.3.7 Modbus universAAL bridge

The MODBUS protocol defines a simple protocol data unit (PDU) independent of the underlying communication layers. The mapping of MODBUS protocol on specific buses or network can introduce some additional fields on the application data unit (ADU). The MODBUS application data unit is built by the client that initiates a MODBUS transaction. The function indicates to the server what kind of action to perform. The MODBUS application protocol establishes the format of a request initiated by a client. The function code field of a MODBUS

data unit is coded in one byte. Valid codes are in the range of 1 ... 255 decimal (the range 128 – 255 is reserved and used for exception responses). When a message is sent from a Client to a Server device the function code field tells the server what kind of action to perform. Function code "0" is not valid. Sub-function codes are added to some function codes to define multiple actions. The data field of messages sent from a client to server devices contains additional information that the server uses to take the specific action defined by the function code. This can include items like discrete and register addresses, the quantity of items to be handled, and the count of actual data bytes in the field.

The data field may be non-existent (of zero length) in certain kinds of requests, in this case the server does not require any additional information. The function code alone specifies the action. If no error occurs related to the MODBUS function requested in a properly received MODBUS ADU the data field of a response from a server to a client contains the data requested. If an error related to the MODBUS function requested occurs, the field contains an exception code that the server application can use to determine the next action to be taken.

MODBUS bases its data model on a series of tables that have distinguishing characteristics. The four primary tables are:

Primary tables	Object type	Type of	Comments
Discretes Input	Single bit	Read-Only	This type of data can be provided by an I/O system.
Coils	Single bit	Read-Write	This type of data can be alterable by an application program.
Input Registers	16-bit word	Read-Only	This type of data can be provided by an I/O system
Holding Registers	16-bit word	Read-Write	This type of data can be alterable by an application program.

Table 3 MODBUS primary tables

The MODBUS Addressing model is as following:

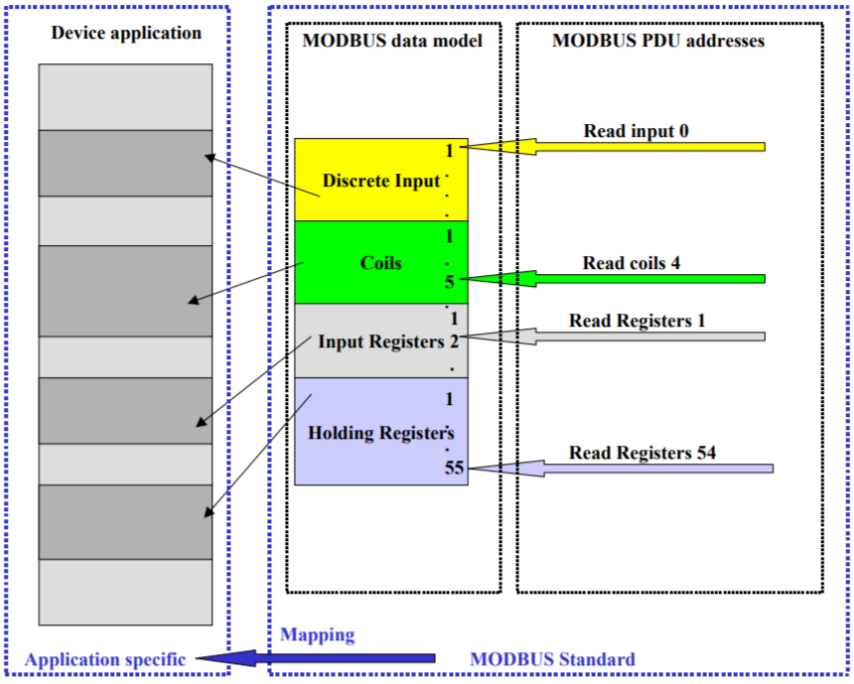

Figure 6 MODBUS Adressing model

The Modbus protocol is currently used as an exporter of the universal IoT open platform especially to control and communicate with the Programmable Logic Controllers created by WAGO Company and installed in the German DS.

3.3.8 MQTT universAAL bridge

MQTT: Message Queuing Telemetry Transport is an ISO standard (ISO/IEC PRF 20922) publish-subscribe-based messaging protocol. It works on top of the TCP/IP protocol. It is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited. The publish-subscribe messaging pattern requires a message broker. The MQTT is considered as a machine-to-machine /Internet of Things connectivity protocol and was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. It is also ideal for mobile applications because of its small size, low power usage, minimised data packets, and efficient distribution of information to one or many receivers.

Technically speaking, MQTT Information is organized in a hierarchy of topics. When a publisher has a new item of data to distribute, it sends a control message with the data to the connected broker. The broker then distributes the information to any clients that have subscribed to that topic. The publisher does not need to have any data on the number or locations of subscribers, and subscribers in turn do not have to be configured with any data about the publishers. If a broker receives a topic for which there are no current subscribers, it will discard the topic unless the publisher indicates that the topic is to be retained. This allows new subscribers to a topic to receive the most current value rather than waiting for the next update from a publisher. When a publishing client first connects to the broker, it can set up a default message to be sent to subscribers if the broker detects that the publishing client has unexpectedly disconnected from the broker. Clients only interact with a broker, but a system may contain several broker servers that exchange data based on their current subscribers' topics. A minimal MQTT control message can be as little as two bytes of data. A control message can carry nearly 256 megabytes of data if needed. There are fourteen defined message types used to connect and disconnect a client from a broker, to publish data, to acknowledge receipt of data, and to supervise the connection between client and server. MQTT relies on the TCP protocol for data transmission. A variant, MQTT-SN, is used over other transports such as Bluetooth. MQTT sends connection credentials in plain text format and does not include any measures for security or authentication. This can be provided by the underlying TCP transport using measures to protect the integrity of transferred information from interception or duplication. Currently, the integrated MQTT exporter within the universal IoT open platform enable the integration and the smooth communication with the smart bed sensors

3.3.9 “C++ DataProvider” universAAL bridge

“C++ Data Provider” is a C++ base class aiming at facilitating the communication of IoT-agnostic devices. This component intents to ease **the inclusion of not-yet connected devices**, running only C++ embedded code, **into AIOTES**. IoT communication is “hidden” into the source code of the proposed component (data model, communication protocol). By using that component, C++ developers can focus on their core business (low level sensor connection and data generation) and only need to adjust the connection configuration and to produce the data to be sent to UniversAAL.

The Data Provider class already provides appropriate threading mechanisms for (Figure 7):

- Extracting data from connected sensor(s) at a given framerate
- Providing permanent and asynchronous access to the latest data generated by the computation loop
- Providing means to log the generated data within files (local storage).
- Sending data to an IoT platform (universAAL in this case) through
    - Registration of a context publisher in UniversAAL and user authentication.
    - Data serialization using JSON.
    - Event pusblisher to the bus context using web requests and the UniversAAL REST-API.
    - Detection of communication errors.

All the previous operations are provided by the base class and are therefore accessible by any class derived from it. This way C++ developers can focus on their core business since they only have to provide the IoT-agnostic content of the following methods (in red on Figure 7):

- Configure: to connect and configure to the sensor
- GenerateRowData: to extract from sensor signal the appropriate digital information.
- GenerateIoTData: to combine latest data generated into the appropriate information to be exchanged with the IoT platform (presumably at a lower frequency than GenerateRowData).

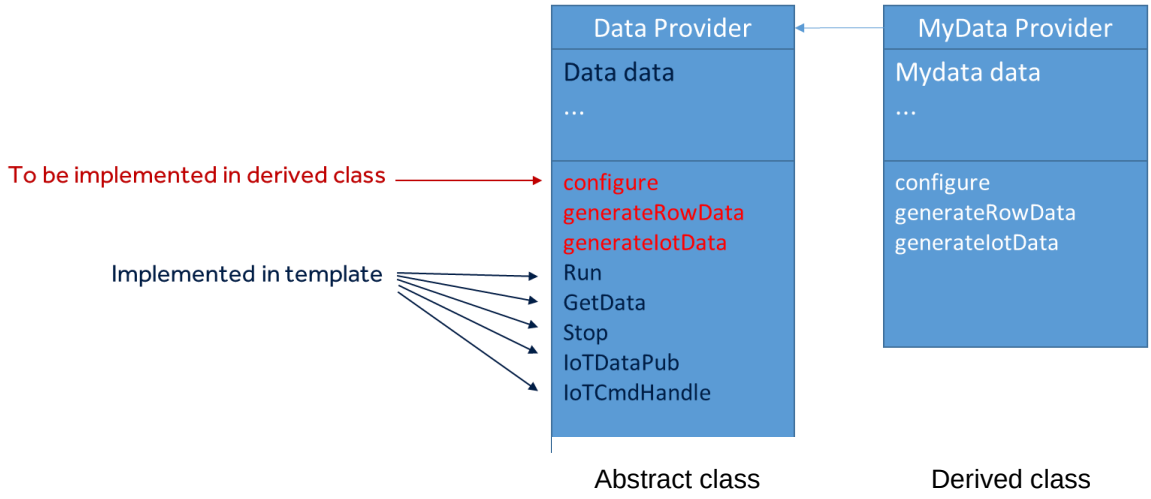

Figure 7. C++ DataProvider architecture

The C++ DataProvider aims at running inside as sensor (potentially a gateway) and is used as presented in the following pseudo-code:

```
New MyData data_prov

// Low level configuration of sensor(s)/driver(s)
data_prov->configure(parameters)

// Launch the data generation & transmission
data_prov->run

// running until stoped by data_prov->stop
while "running"
    // Row data are generated by the threading processing loop (at a specified framerate: e.g. 30Hz)
    (generateRowData)
    // Meta data are computed
    (generateIotData)
    // Data sent to IoT (universAAL) when needed (e.g. every minute, after an event)
    IoTDataPub
```

3.3.10 Summary of universaal IoT bridges status

IoT Protocol bridges for universAAL					
Protocol	**Domain**	**Layer**	**Connection edges**	**Status**	**Comment**
OpenHab	Domotic	Device	Device to LAN	Implemented	
Z-Wave	Domotic	Device	Device to Gateway	Implemented	
KNX	Domotic	Device	Device to Gateway Device to LAN	Implemented	
BLE-continua-x70	Health	Device	Device to Gateway	Implemented	Integration with the Continua certified devices
Zigbee	Generic	Device	Device to Gateway	Implemented	Home Automation Profile
Fs20	Domotic	Device	Device to Gateway	Implemented	German Domotic proprietary protocol
Philips Hue	Domotic lighting	Device	Device to LAN	To be implemented	
http/WSDL-SOAP	Generic	Service	Gateway to WAN (2way)	Implemented	
HTTP/REST	Generic	Service	Gateway to WAN (2way)	Implemented	
Modbus	Automatio n \Demotic	Service	Device to Gateway	Implemented	
MQTT	Generic	Service	Device to LAN	Implemented	
C++ DataProvi der	Generic	Device	Device to LAN	Implemented	

Table 4: universAAL supported and planned IoT protocol bridges

UniversAAL will be connected to other platforms via the SIL layer.

3.4 OpenIoT bridges status

The OpenIoT (Open source blueprint for large scale self-organizing cloud environments for IoT applications) is an open source Internet of Things platform [23], which allows the collection, storage and processing of data stemming from heterogeneous set of IoT entities and resources. The platform introduces Sensing as a Service paradigm by interweaving the cloud computing with Internet of Things technologies. In order to collect data from multiple sensing devices, OpenIoT uses the Global Sensor Networks (GSN) [24] middleware. OpenIoT extends GSN (that has been called X-GSN) with RDF features and Linked Data, and provides a GSN-CoAP wrapper to access sensor data using a RESTful approach. GSN can retrieve data from various data sources. In the context of the OpenIoT Project, OpenIoT leverages M2M/IoT techniques (such as CoAP), but also adds richer semantics to the inter-object communication. Furthermore, it provides user-friendly service interfaces for people to object interactions.

The OpenIoT internal communication between sensors within a deployed network is considered as a “black box”, therefore the input for OpenIoT sensor data delivery chain solely results from the data as provided by the gateway node. The OpenIoT as a Service also includes a sensor data middleware that eases the collection of data from virtually any sensor, located anywhere, while at the same time ensuring the proper data annotation following the W3C standards by means of semantic annotation. OpenIoT as a Service current online version offers a wide range of visual tools that enable the development and deployment of IoT applications with almost zero programming as result of the easy to implement data API’s.

Another key feature of OpenIoT is its ability to handle mobile sensors, thereby enabling the emerging wave of mobile crowd sensing applications. OpenIoT is currently supported by an active community of IoT researchers and developpers, while being extensively used for the development of IoT applications in areas where semantic interoperability is a major concern.

The work in ACTIVAGE has motivated the evolution of the OpenIoT platform, mainly from the proliferation for building and deploying IoT applications in multiple geographical locations that must be interconnected and federated in some way for facilitating the provisioning of common services (data exchange, data discovery, visualisation, analytics, etc.). Despite the emergence of numerous IoT platforms, the deployment in the cloud is yet the most common approach and at the same time this condition defines a big challenge, which is that there is still no simple way to integrate geographically and administratively heterogeneous platforms for use in the ACTIVAGE bridge implementations. In addition, considering also that the IoT platforms inherently include a considerable large number of dispersed sensors and IoT services, the interoperable activity and data exchange is more complicated as well. The Figure 8 illustrates the full life cycle and value chain of OpenIoT Bridges and their use with OpenIoT Platform towards offering the Data via APIs for Data Applications, This is known as OpenIoT Stack and up to date is used in several development and integration as reference implementation of multiple IoT Architecture standards.

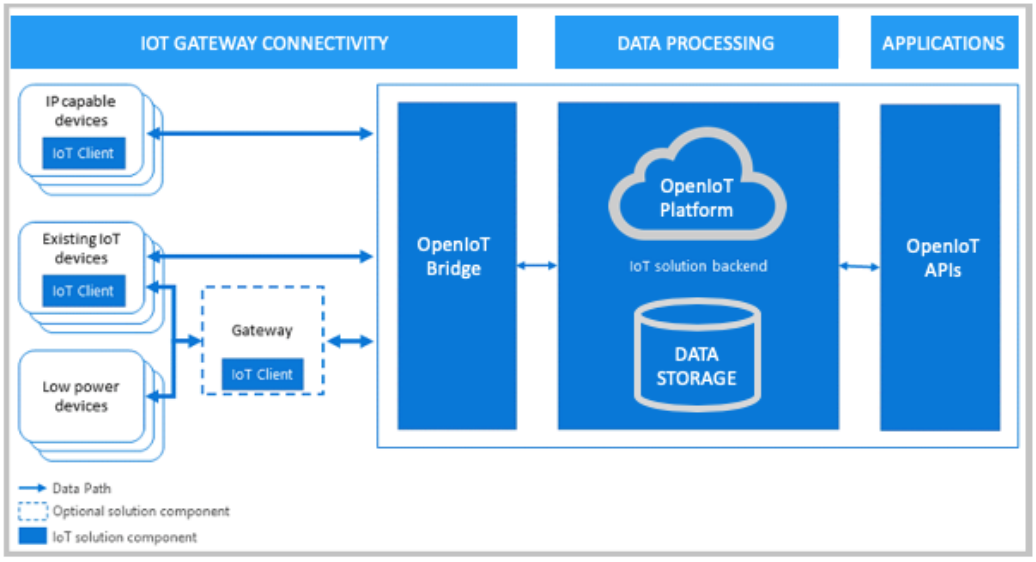

Figure 8: OpenIoT Bridge Access, Abstraction and Processing and Applications Stack

In the context of the ACTIVAGE project implementation, the following are the most widely adopted protocols that OpenIoT has selected to be included in the OpenIoT bridge, the development shall be used as a reference implementation and demonstration about the capabilities of using standard data format and interoperability principles. Below sections detail the status of each bridge. The Appendix chapter at the end of the document provides source code examples of using those bridges.

3.4.1 ZigBee Bridge

ZigBee is an IEEE 802.15.4-based specification for a suite of high-level communication protocols used to create personal area networks with small, low-power digital radios, such as for home automation, medical device data collection, and other low-power and low-bandwidth needs, designed for small scale projects which need wireless communication.

Typically, Zigbee-based IoT systems feed all the data to online cloud platforms on the Internet. In OpenIoT, the IoT devices typically connected to a Gateway can transfer data using the ZigBee protocol, the device samples some value locally, for example temperature, then sends the readings to any one of a number of online applications for logging, processing and data visualization. In OpenIoT, sensors measurements like temperature for example (using simple sensors) send the values to a remote IoT Gateway in the form of a “data stream” that can be visualized in different ways, accessed via open APIs or stored on the local repository or data base named LSM-Light (data base) for later use by data applications. Figure 9 provides a schematic representation on the basic architectural arrangements.

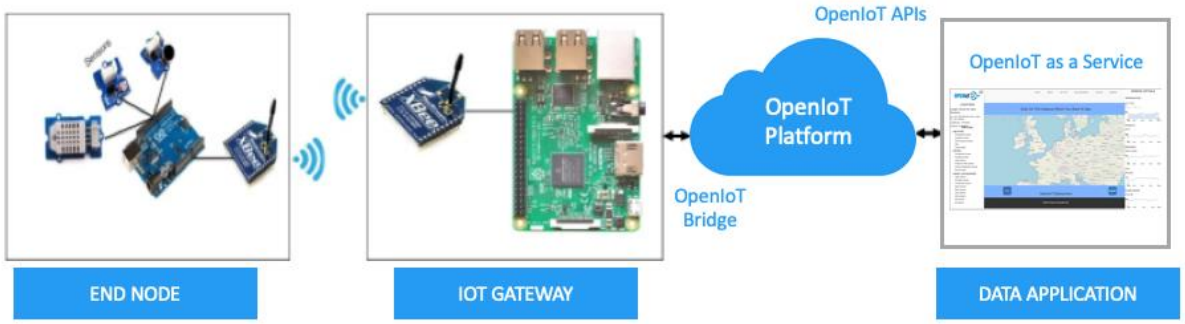

Figure 9: Zigbee Connectivity, Access, Abstraction and Processing Stack

3.4.2 BLE Bridge

The implementation of a Bluetooth Low Energy (BLE) bridge for OpenIoT adopts the reference implementation from IoT AWS, and currently there is only one example of a sample data collection using Amazon Cloud with FreeRTOS BLE boards. The support of the BLE protocol is a feature that makes possible for embedded developers to securely connect devices that use Bluetooth Low Energy (BLE) through Android or iOS devices. Developers can implement applications for devices that need lower power than any other forms of connectivity. In particular, with the use of BLE support (FreeRTOS BLE Board), developers can use the standard Generic Access Profile (GAP) and Generic Attributes (GATT) profiles through a universal API layer to create BLE applications that are portable across any FreeRTOS-qualified device and use companion Android and iOS SDKs to integrate with AWS IoT functionality. According to the BLE specifications, GAP defines how BLE devices broadcast availability and communicate with each other and GATT describes how data is transferred once a connection is established. In this example implementation, a BLE device connecting to AWS IoT through an Android proxy allows the BLE device to be agnostic to the underlying communication carrier, guaranteeing that BLE data communication protocol can be used. Because BLE offers lower power compared to Wi-Fi, devices can use the MQTT protocol to connect to AWS IoT services over BLE. This brings the best of low power usage and rich AWS IoT services, such as Amazon FreeRTOS over-the-air updates, to the devices in the field. Figure 10 shows the extension of BLE protocol using OpenIoT bridge following the AWS IoT architecture possible extension(s).

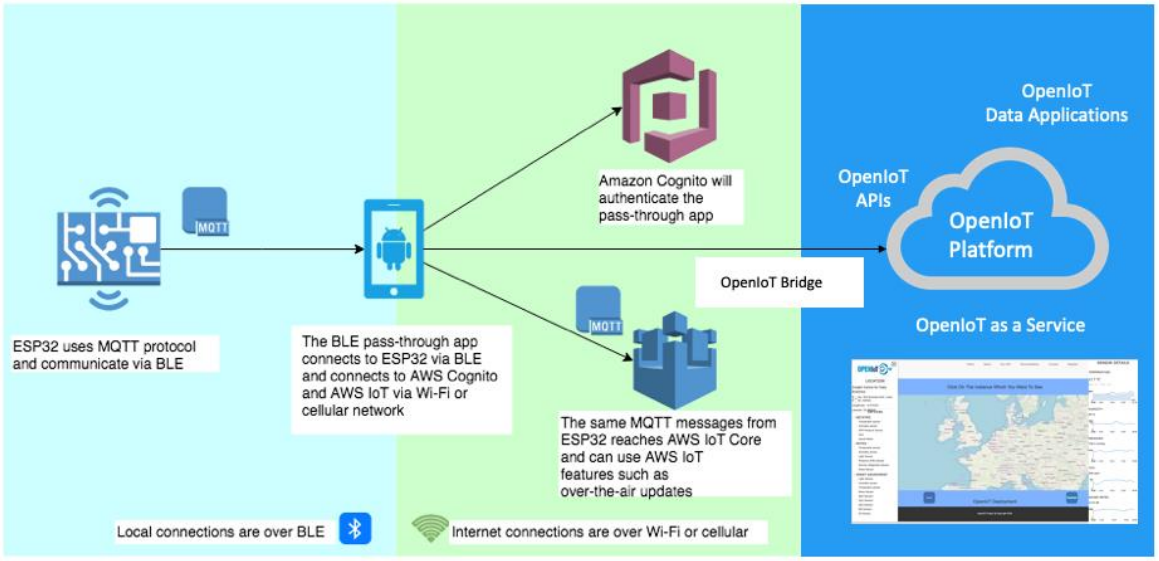

Figure 10: BLE Connectivity, Access, Abstraction and Processing Stack

3.4.3 MQTT Bridge

The Message Queuing Telemetry Transport (MQTT) protocol is a lightweight messaging protocol designed for constrained devices and low-bandwidth, high-latency and unreliable networks. It uses the publish/subscribe model of communication to minimize network bandwidth and device resource consumption while keeping the reliability and assurance of delivery. OpenIoT has implemented an MQTT bridge in order to interconnect the middleware with the applications.

An MQTT system consists of clients communicating with a server, often called a "broker". A client may be either a publisher of information or a subscriber after connecting to the broker. Data transmission in MQTT is organized in a hierarchy of topics. When a publisher has a new item of data to distribute, it sends a message with the data to the connected broker. The broker then distributes the information to any clients that have subscribed to that topic. The publisher does not need to have any data on the number or locations of subscribers, and subscribers in turn do not have to be configured with any data about the publishers.

If a broker receives data for a topic for which there are no current subscribers, it will discard the topic unless the publisher indicates that the data is to be retained. This allows for new subscribers to an existing topic to receive the most current value rather than waiting for the next data update from a publisher of the topic. When a publisher client first connects to the broker, it can set up a default message to be sent to subscribers if the broker detects that the publishing client has unexpectedly disconnected from the broker. Furthermore, clients can only interact with a broker, but a more complex system may contain several interconnected broker servers that exchange data based on their current subscribers' topics.

A minimal MQTT control message can be as little as two bytes of data, and a control message can carry nearly 256 megabytes of data if needed. There are fourteen defined message types used to connect and disconnect a client from a broker, to publish data, to acknowledge receipt of data, and to supervise the connection between client and server. MQTT relies on the TCP protocol for data transmission. A variant, MQTT-SN, is used over other transports such as UDP or Bluetooth. MQTT sends connection credentials in plain text format and does not include any measures for security or authentication. This can be alleviated by the underlying TCP transport using measures to protect the integrity of transferred information from interception or duplication such as secure sockets layer (SSL).

Sensor devices can connect to an MQTT broker as a publisher client and stream sensor measurements into dedicated topics for subscriber clients to receive data in real-time. Figure 11 shows the MQTT protocol following the MQTT Broker standard architecture, it also depicts the HTTP Protocol and the APIs for providing access to the data as part of the OpenIoT Stack.

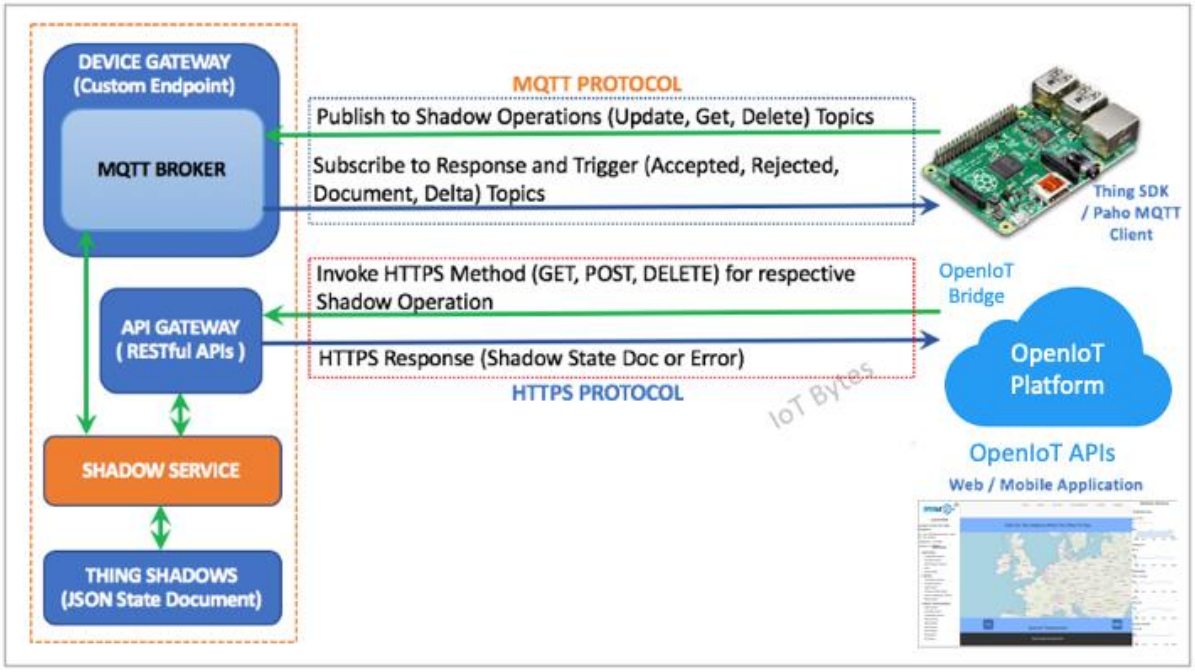

Figure 11: MQTT Protocol Process including OpenIoT Bridge and Platform as the OpenIoT Stack

3.4.4 CoAP Bridge

The Constrained Application Protocol (CoAP) is a specialized Internet Application Protocol for constrained devices. It enables those constrained devices called "nodes" to communicate with the wider Internet using similar protocols. CoAP is designed for use between devices on the same constrained network (e.g., low-power, lossy networks), between devices and general nodes on the Internet, and between devices on different constrained networks both joined by an internet. CoAP is also being used via other mechanisms, such as SMS on mobile communication networks. The Constrained Application (CoAP) protocol is standardized by the Constrained RESTful Environments (CoRE) workgroup and is available as an active (IETF) Internet-draft. CoAP is based on a request/response communication model between entities (e.g. a sensing node and server), including key concepts found on the Web, thus enabling easy integration with the standard Web services.

The nodes often have simple 8-bit microcontrollers with small amounts of ROM and RAM, while constrained networks such as IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs) often have high packet error rates and a typical throughput of only 10s of kbit/s. The CoAP protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.

The smallest CoAP message is 4 bytes in length, if omitting Token, Options and Payload. CoAP makes use of two message types, requests and responses, using a simple, binary, base header format. The base header may be followed by options in an optimized Type-Length-Value format. CoAP is by default bound to UDP and optionally to DTLS, providing a high level of communications security. Any bytes after the headers in the packet are considered the message body. OpenIoT implements a CoAP bridge for wireless sensor network data collection and distribution and the representation is shown in Figure 12.

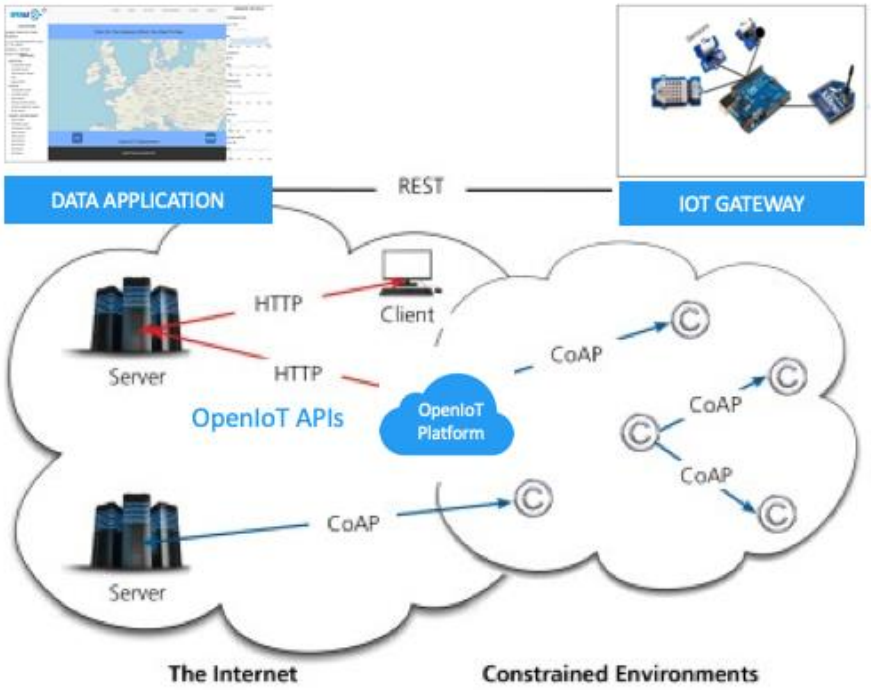

Figure 12: Architecture of CoAP between the Internet and devices in constrained environments.

3.4.5 HTTP-GET Bridge

In the OpenIoT architecture, the Global Sensor Network (GSN) serves as sensor middleware to publish sensor data form all kinds of physical devices via a common Web service interface. The Linked Sensor Middleware (LSM) sits on top of this network, fetching the data from GSN via HTTP, transforming it into Linked Data, enriching it with semantic information, and storing the data into an RDF storage system.

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser. HTTP was developed to facilitate hypertext and the World Wide Web and is designed to permit intermediate network elements to improve or enable communications between clients and servers. HTTP defines methods (sometimes referred to as verbs) to indicate the desired action to be performed on the identified resource. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. One such methods is the GET method that requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect, however they can also contain parameters for the server.

3.4.6 HTTP REST Bridge

Representational State Transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, termed RESTful Web services (RWS), provide interoperability between computer systems on the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations. By using a stateless protocol and standard operations, RESTful systems aim for fast performance, reliability, and the ability to grow, by re-using components that can be managed and updated without affecting the system as a whole, even while it is running. RESTful services must implement a number of HTTP verbs such as GET, POST, PUT, PATCH and DELETE. The OpenIoT HTTP REST bridge is then an extension of the HTTP GET bridge, that operates only using the HTTP GET verb.

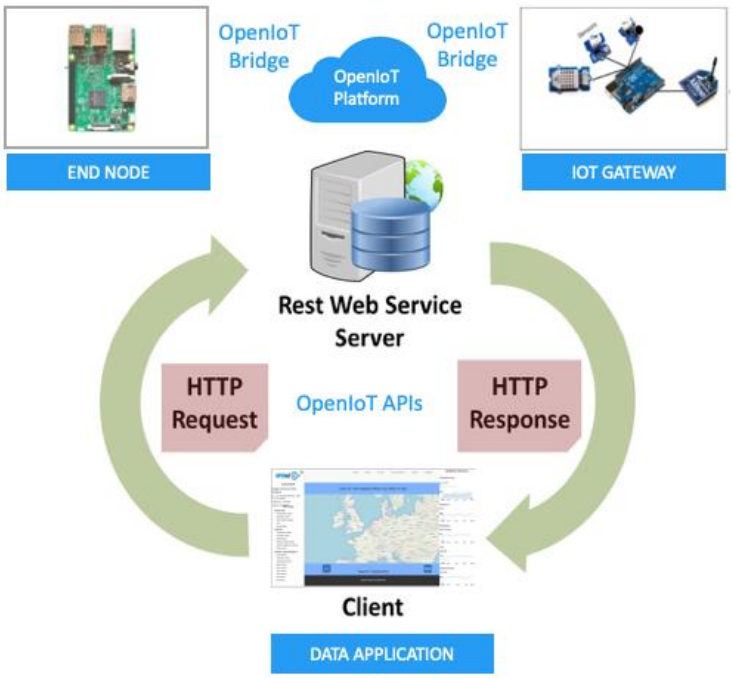

Figure 13: Architecture Style for REST Web Services

LD4Sensors is OpenIoT's tool for data annotation and is a RESTful Web server implemented using Java with the Jena library and the Apache Jena Triple DB, leveraging on the OpenIoT GUI. The current API allows to access, store, update, and delete specific resources that are typically involved in sensor measurements and sensor networks, after having semantically annotated them. The data can be accessed by querying a SPARQL endpoint, in addition to the REST API. The implemented bridge also connects to a remote GSN-service instance and uses the REST API to poll data from the server. Moreover, the authentication and authorization process is performed using the OAuth2 protocol. All components within OpenIoT provide RESTful web services which are accessible via defined URL as shown in Figure 13.

3.4.7 HTTP SOAP Bridge

SOAP (Simple Object Access Protocol) is a protocol specification for the exchange of structured information and the implementation of web services. It uses Extensible Markup Language (XML) as message format and typically uses HTTP for message negotiation and transmission. SOAP allows processes running on heterogeneous operating systems (such as Windows and Linux) to communicate using XML. Since Web protocols like HTTP are installed and running on nearly all operating systems, SOAP allows clients to invoke web services and receive responses independent of language and platforms.

SOAP has three major characteristics: (a) extensibility, for example there are security extensions, (2) neutrality, as SOAP can operate over any transport protocol, and (3) independence, as SOAP allows for any programming model. The neutrality characteristic explicitly makes it suitable for use with any transport protocol. Implementations often use HTTP as a transport protocol, but other popular transport protocols can be used. For example, SOAP can also be used over SMTP, JMS and message queues or brokers such as MQTT. When combined with HTTP post/response exchanges, SOAP can be tunnelled easily through existing firewalls and proxies, therefore not requiring modifying the widespread computing and communication infrastructures that exist for processing HTTP communication.

However, SOAP is also notorious for its larger and more complex design in comparison with other alternative protocols, therefore it is not well suited for smaller low-powered sensor

devices. Furthermore, the verbosity of the protocol, slow parsing speed of XML and lack of a standardized interaction model has placed SOAP behind other alternative services using the HTTP protocol more directly such as REST architectures (often compared to SOAP).

Using OpenIoT, the raw sensor data collected by GSN is stored in a database system and can be accessed through a defined HTTP SOAP API. Figure 14 depicts the use of SOAP and the integration with the OpenIoT bridge main functionalities, it is a protocol that enhance security and the trade-off is the use of SOAP messages.

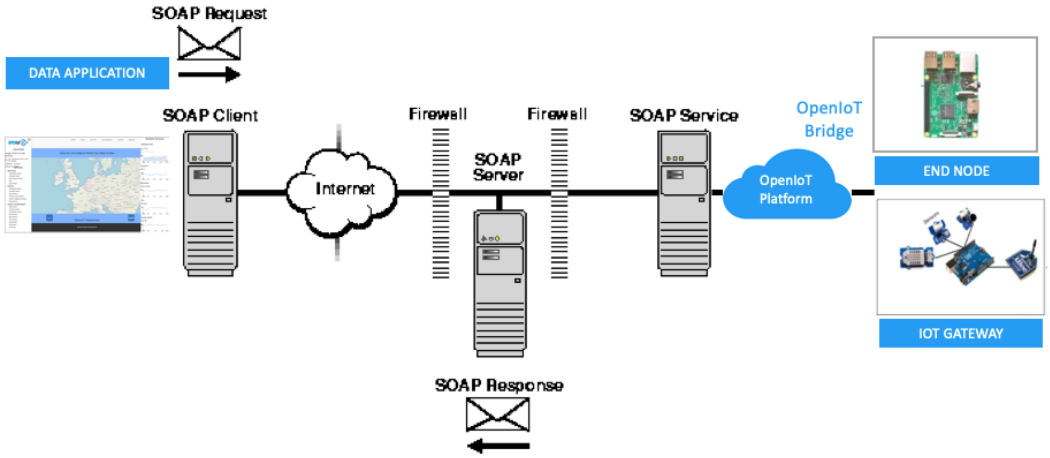

Figure 14: Architecture for SOAP-based Services

3.4.8 RSS Bridge

RSS (originally RDF Site Summary and later, Rich Site Summary or Really Simple Syndication) is a type of web feed which allows users and applications to access updates to online content in a standardized, computer-readable format. These feeds can, for example, allow a user to keep track of many different websites in a single news aggregator. The news aggregator will automatically check the RSS feed for new content, allowing the content to be automatically passed from website to website or from website to user. This passing of content is called web syndication. RSS feeds are typically used by Websites to publish frequently updated information, such as blog entries, news headlines, or episodes of audio and video series. RSS is also used to distribute podcasts. An RSS document (called "feed", "web feed" or "channel" includes full or summarized text, and metadata, like publishing date and author's name. A standard XML file format is adopted to ensure compatibility with many different machines/programs. RSS feeds also benefit users who want to receive timely updates from favourite websites or to aggregate data from many sites.

Despite RSS being originally designed for news and websites, it can be also used in other contexts such as global sensor networks due to its publisher/subscriber paradigm.

The RSS protocol is currently on decline due to several major sites such as Facebook and Twitter previously offering RSS feeds but have reduced or removed support. Additionally, widely used readers such as Google Reader have been discontinued having cited declining popularity in RSS. Nonetheless, RSS is a mature protocol for data aggregation.

The OpenIoT RSS bridge allows extracting an RSS feed from any given URL to be forwarded to the middleware and later to sensor applications.

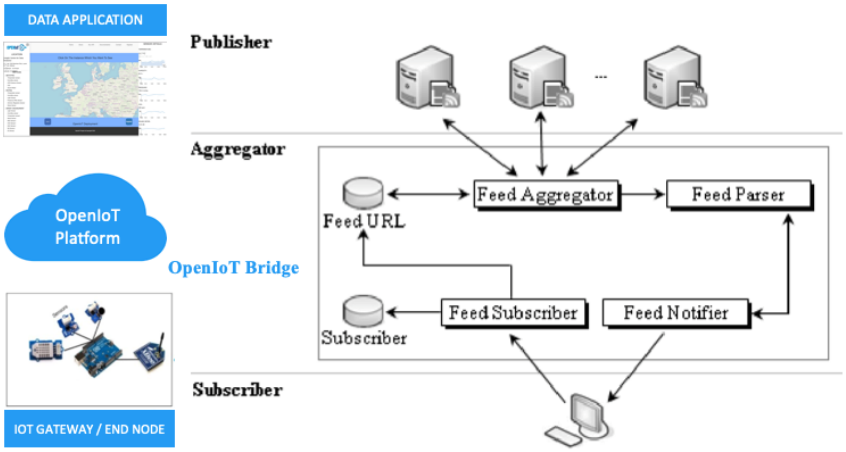

Figure 15: Architecture for RSS-based Services

3.4.9 UDP Bridge

The User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network. Prior communications are not required in order to set up communication channels or data paths. UDP uses a simple connectionless communication model with a minimum of protocol mechanism. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network; there is no guarantee of delivery, ordering, or duplicate protection.

UDP is suitable for purposes where error checking and correction are either not necessary or are performed in the application; UDP avoids the overhead of such processing in the protocol stack. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for packets delayed due to retransmission, which may not be an option in a real-time system. Due to its simplicity, the UDP protocol is a suitable choice for low-powered and resource constrained sensor devices. Figure 16 depicts the implemented OpenIoT UDP bridge allowing the reception of arbitrary data from any device on a defined UDP port of the machine on which OpenIoT is running.

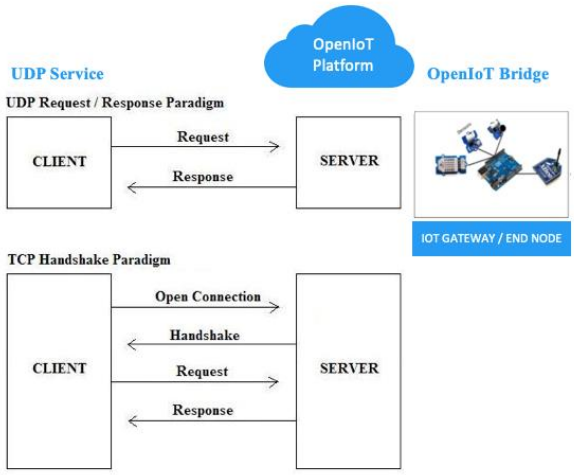

Figure 16: Comparison between the UDP and TCP communication paradigms.

3.4.10 XMPP Bridge

The eXtensible Messaging and Presence Protocol is an open standard based on XML for the near-instant exchange of messages and presence notifications. The main units of transferred information are called *stanzas*, which, in the context of XMPP, are self-contained XML snippets, for example, including the source and target of a stanza. In contrast to one-shot queries that are executed once and return a result, a continuous query over data streams generates new results each time new input data matches the query. To deliver results to any requesting application, OpenIoT supports various standard protocols including XMPP. Other protocols in this category include PubSubHubbub and WebSockets.

Designed to be extensible, the protocol has been used also for publish-subscribe systems, signalling for VoIP, video, file transfer, gaming, the Internet of Things (IoT) applications such as the smart grid, and social networking services. Unlike most instant messaging protocols, XMPP is defined in an open standard and uses an open systems approach of development and application, by which anyone may implement an XMPP service and interoperate with other organizations' implementations.

The XMPP network uses a client–server architecture; clients do not talk directly to one another. The model is decentralized - anyone can run a server. By design, there is no central authoritative server as there is with services such as AOL Instant Messenger or Windows Live Messenger. XMPP provides a general framework for messaging across a network, which offers a multitude of applications beyond traditional Instant Messaging (IM) and the distribution of Presence data.

As depicted in Figure 17, while several service discovery protocols exist today (such as zeroconf or the Service Location Protocol), XMPP provides a solid base for the discovery of services residing locally or across a network, and the availability of these services (via presence information).

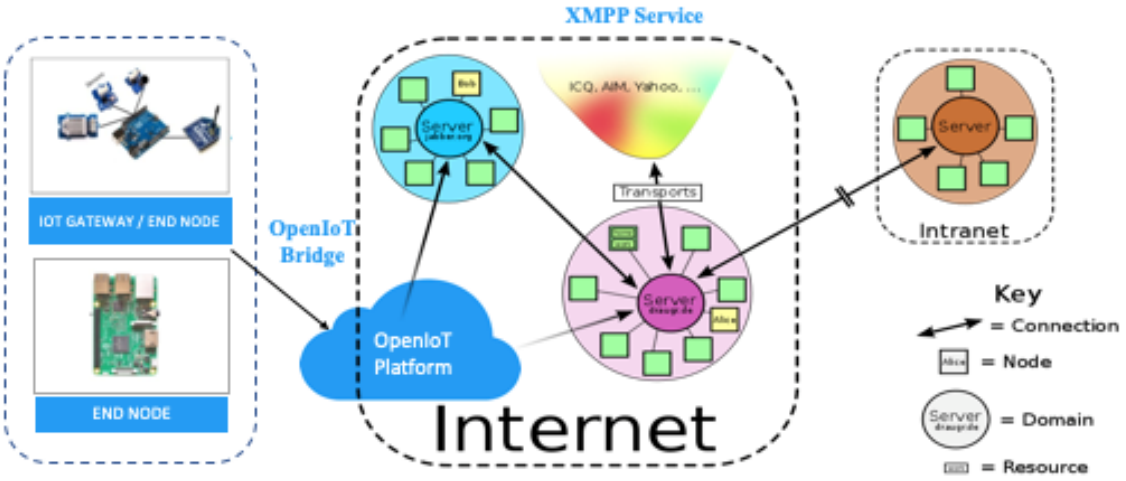

Figure 17: Architecture of the decentralised XMPP network.

3.4.10.1 Summary of OpenIoT IoT bridges

The Table 5 below summarizes the IoT protocol bridges (already supported and planned) for the OpenIoT platform.

IoT Protocol bridges for OpenIoT

Protocol	Domain	Layer	Connection Edges	Status
ZigBee	Generic	Device	Device to Gateway	Implemented
BLE	Generic	Device	Device to Gateway	In progress
MQTT	Generic	Middleware	Gateway to WAN	Implemented
CoAP	Generic	Device	Device to Gateway	Implemented
HTTP-GET	Generic	Device	Gateway to OpenIoT	Implemented
HTTP REST	Generic	Middleware	OpenIoT to WAN	Implemented
HTTP SOAP	Generic	Device	Gateway to OpenIoT	Implemented
RSS	Generic	Device	Device to Gateway	Implemented
UDP	Generic	Device	Gateway to OpenIoT	Implemented
XMPP	Generic	Middleware	OpenIoT to WAN	Implemented

Table 5: OpenIoT supported IoT protocol bridges

OpenIoT will be connected to other platforms via the SIL layer. In addition, OpenIoT has a direct bridge to the IoTivity via CoAP/MQTT.

3.5 sensiNact bridges status

The sensiNact platform is dedicated to IoT and particularly used in various smart city and smart home applications. sensiNact aims at managing IoT protocols and devices heterogeneity and provides synchronous (on demand) and asynchronous (periodic or event based) access to data/actions of IoT devices, as well as access to historic data with generic and easy-to-use API.

The sensiNact platform interconnects IoT devices using different southbound IoT protocols such as Zigbee, EnOcean, LoRa, XBee, MQTT, XMPP, as well as platforms such as FIWARE and allows access to them with various northbound protocols such as HTTP REST, MQTT, XMPP, JSON RPC and CDMI. The Figure 18 below shows an overview of the bridges (southbound and northbound) around the sensiNact platform.

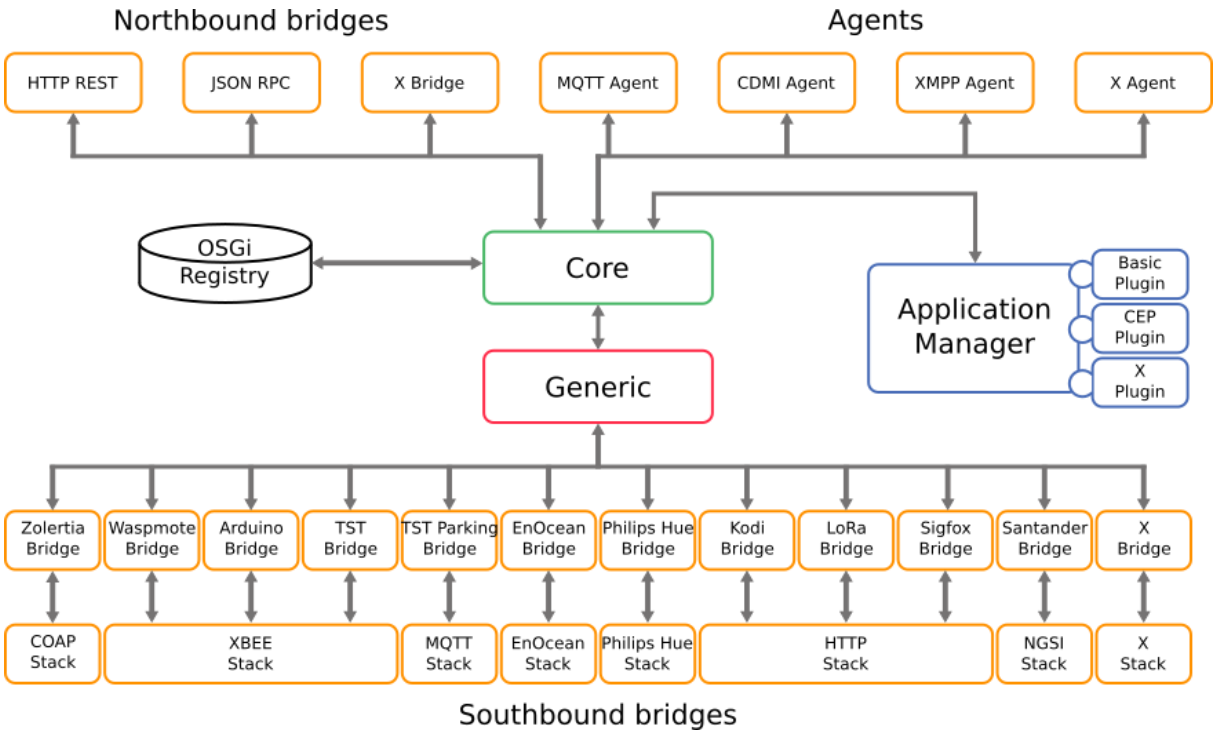

Figure 18: Overview of the sensiNact southbound and northbound bridges

The interactions between the gateway and other entities are performed through an extensible set of northbound and southbound bridges.

Southbound bridges are specialized in the interaction with devices, which can be sensors or actuators. Each bridge is in charge of communicating with a specific kind of device, using a given protocol. Out of the box, sensiNact ships with southbound bridges such as Zigbee (motion sensors, force sensor, etc.), EnOcean (remote controls, windows opener detectors, etc.), CoAP (sliders, buttons, etc.). It also provides a bridge for retrieving context information using NGSi 9/10 protocol. Thanks to an OSGi based architecture, it is possible to add bridges “on the fly”, while the gateway is running, to allow the communication with new devices. Of course, the creation of bridges relies on an API which delegates most of the integration work to the gateway, letting the programmer focus on the communication protocol and the data model of the device to be integrated.

Symmetrically to the southbound bridges, northbound bridges are in charge of publishing information to remote systems. It can be using common protocols, for example MQTT, XMPP, NGSi 9/10. The set of northbound bridges is also extensible, for tailoring special needs of singular systems. The REST API, which is a northbound bridge, is a key part of our architecture. It is designed for the administration of the gateway, thanks to a well-documented API.

A tutorial for southbound bridge implementation inside the sensiNact platform is provided in the public wiki, see [36].

The next sections give details about some of the currently available or planned sensiNact bridges.

3.5.1 Android IMU sensiNact bridge

Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful if you want to monitor three-dimensional device

movement or positioning, or if someone wants to monitor changes in the ambient environment near a device.

The Android platform supports three broad categories of sensors:

- Motion sensors:
  These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.
- Environmental sensors:
  These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.
- Position sensors:
  These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

Android sensor framework provides the following capacities:

- Determine which sensors are available on a device.
- Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.
- Acquire raw sensor data and define the minimum rate at which you acquire sensor data.
- Register and unregister sensor event listeners that monitor sensor changes.

Personal Connected Health alliance [26] provides an overview of the sensors that are available on the Android platform. It also provides an introduction to the sensor framework.

This Android IMU sensiNact bridge allows sensiNact to receive Inertial Measurements from an Android device.

- Operating Requirements
  - The android-imu bridge must be activated at sensiNact configuration (via sensinact -c)
  - The Android device must run the Chrome browser
- How to use it
  - Point the Chrome (from your device) to the URL http://sensinact.address.com/imu/.
  - From that point on you can check the position of your device checking the service called android that is available in sensiNact
- Limitations
  - Multiple android devices are not supported yet.

3.5.2 CoAP Generic sensiNact bridge

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks in the Internet of Things.

The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.

Like HTTP, CoAP is based on the wildly successful REST model: Servers make resources available under a URL, and clients access these resources using methods such as GET, PUT, POST, and DELETE.

More details about the CoAP protocol, specifications, implementation and tools, can be found in [32].

The future CoAP sensiNact bridge will handle this IoT protocol. The implementation of this new generic bridge could reuse previous implementation done in a previous sensiNact version.

3.5.3 EchoNet sensiNact bridge

ECHONET [30] is a Japanese communication protocol designed to create the “smart houses” of the future. Today, with Wi-Fi and other wireless networks readily available in ordinary homes, there is a growing demand for air-conditioning, lighting and other equipment inside the home to be controlled using smartphones or controllers, or for electricity usage to be monitored in order to avoid wasting energy.

To achieve this kind of low-energy, comfortable, safe and reassuring lifestyle, we first need to create a system of rules or a “communication protocol” that can be read by any manufacturer’s equipment. This is why ECHONET was chosen and integrated.

The ECHONET Lite specification [31], in particular, is a communication protocol compatible with the now ubiquitous Internet. It is designed for ease of use and is simpler than the ECHONET specification.

The ECHONET Lite specification is already compatible with more than 100 types of device, and is also being adopted by the smart electric energy meters that will be installed in all households in future.

3.5.4 e-Lio app sensiNact bridge

e-lio is a user friendly platform dedicated to residential care markets (retirement homes, assisted-living residences, hospitals, etc.) that enhances the relationships between the caregivers, the residents and their families.

- For professionals, from any computer or telephone:
    - Animation of the establishment (activities, menus, pictures)
    - Communication with the families (news, pictures, messages)
    - Video Calls
    - Nurse call coupling
    - Adaptation to the resident's autonomy
- For families, from any computer, tablet or connected smartphone:
    - Audio and video Calls
    - On-line information news
    - Communication with the resident (pictures, messages, diary)

- Communication with the establishment (messages, pictures)
- Advises and information

More details can be found in [27]

The future e-lio sensiNact bridge (planned) will make possible to get data coming from the e-lio application (information about the occurrences of audio and video calls, tv interactions through the e-lio remote control, etc.) into the sensiNact service layer.

3.5.5 EnOcean sensiNact bridge

The EnOcean technology is an energy harvesting wireless technology used primarily in building automation systems, and is also applied to other applications in industry, transportation, logistics and smart homes. Modules based on EnOcean technology combine micro energy converters with ultra-low power electronics, and enable wireless communications between batteryless wireless sensors, switches, controllers and gateways.

The sensiNact platform includes an EnOcean bridges that manages 17 profiles for switches (F6-02-01, F6-02-02, F6-02-03, F6-02-04, D5-00-01), occupancy sensors (A5-07-01), temperature and humidity sensors (A5-02-01, A5-02-02, A5-02-03, A5-02-04, A5-02-05, A5-04-01, A5-04-02, A5-04-03), and energy measurement sensors and actuators (D2-01-06, D2-01-08, D2-01-0A). This sensiNact EnOcean bridges is constantly upgraded with frequent additions of new profiles.

3.5.6 Free Mobile sensiNact bridge

The French telephone company Free mobile provides a "Notifications by SMS" feature, making it possible to send SMS on his mobile phone from any device with an Internet connection through a simple RESTfull API [46]. [47] gives documentation and proposes a tutorial using this API.

The sensiNact Free Mobile bridge implementation is made on the top of this RESTFull API.

3.5.7 LoRaWAN sensiNact bridge

The LoRaWAN sensiNact bridge makes possible to connect with the Orange LoRa platform (France). Some technical details can be found in [35].

3.5.8 MQTT sensiNact bridge

This bridge allows sensiNact to subscribe to a MQTT topic, and materializes a sensiNact device based on the messages. Details about configuration and activation of this MQTT bridge are available in the sensiNact wiki [25].

During the last period, some updates have been made on the implementation of the MQTT sensiNact bridge, and pushed onto the eclipse code source repository, in order to improve the connectivity persistence.

3.5.9 Netatmo sensiNact bridge

The existing sensiNact netatmo bridge has been deprecated at the beginning of the ACTIVAGE project, and, not maintained, is no more part of the official sensiNact Eclipse release [4]. This bridge was dedicated to the netatmo face recognition camera, that does not fit with the used devices in the DS6 deployment use cases.

At the time of this report, the work remains to update the direct sensiNact netatmo bridge, but the priority has been set on an indirect netatmo bridge, through MQTT (see next section 3.5.10) in order to support the netatmo weather station deployed in DS6 homes.

3.5.10 openHAB sensiNact bridge

The open Home Automation Bus (openHAB) [33] is an open source, technology agnostic home automation platform which runs as the center of your smart home.

The openHAB sensiNact bridge is able to find the openHAB instances running on the local network and instantiate its switch devices into sensinact without further configuration.

The switch type devices on openHAB are devices that respond to ON/OFF command.

No configuration is necessary, as long as the UDP packages between openHAB and the sensiNact machines are not been blocked, sensiNact will be able to find the openHAB instance without issues. Be aware that some routers are configured to block this kind of package in multi-hop, and VPN tunneling as well. In this situation you can manually specify the openHAB address. Details about the configuration of the openHAB sensiNact bridge can be found in [34].

Now, the hardware installation is completely defined in the homes of deployment site 6 (for all panels).Since openHAB handles stable and reliable zwave connection and peering mechanism, this openHAB sensiNact bridge has been used to implement the first version of the zwave sensiNact bridge.

Since the beginning of the ACTIVAGE project, some improvements have been done on this openHAB sensiNact bridge. Among all implemented updates, the ones that have been required by the ACTIVAGE project make the new openHAB sensiNact bridge:

- better organize sensiNact providers: one per openHAB device node. openHAB node sub-features are gathered in services/resources of the node providers.
- handle useful features:
    - “pretty name” is accessible through admin/friendlyname sensiNact resource
    - “battery level” is accessible through the battery/level sensiNact resource
    - The connectivity status is accessible through the status/connected resource, and the status/detail give the error message in case of disconnection issue
- support set actions, useful for some actions on actuators

3.5.11 Philips hue sensiNact bridge

Philips hue is a line of color changing LED lamps and white bulbs created by Philips. It was introduced in October 2012. The hue system uses the Zigbee lighting protocol to communicate, and can be controlled over Wi-Fi via a Zigbee–WiFi bridge. The sensiNact hue bridge is implemented on the top of the provided Philips hue RESTFull API.

More details about the hue developer program and documentation on the API can be found in [37].

3.5.12 SIP sensiNact bridge

The Session Initiation Protocol (SIP) allows to synthesize text messages into voice records and phone them to a given phone number. The sensiNact SIP bridge uses the REST API provided by the SIP service to send both text message and phone number.

In DS6 ACTIVAGE deployment site, the SIP service is used to synthesize voice messages when notifications or alerts are triggered through the sensiNact platform, by some of the provided AHA functions. Six different kinds of voice messages have been specified, and implemented in French language (since addressed to the French medical staffs in the institutions). Below are listed the voice synthesized messages in French, with a wild card {0} that is automatically set to the relevant room numbers where the alert is triggered from:

- Alert message in case of suspicious inactivity in the shower (***Situation anormale dans la douche de la chambre {0}***)
- Alert message in case of suspicious inactivity in the bedroom (***Situation anormale dans la chambre {0}***)
- Alert message in case of suspicious inactivity in the bathroom (***Situation anormale dans la salle de bain {0})***
- Notification message to ask for help for the washing (***le patient dans la chambre {0} demande de l'aide pour sa toilette***)
- Alert message for a high pain level declared by the patient (***le patient dans la chambre {0} souffre***)
- Alert message when temperature is uncomfortable in the patient room (***la température dans la chambre {0} est inconfortable)***

3.5.13 Summary of sensiNact IoT bridges

The Figure 19 below summarizes the installation topology and the required communication protocols around the home gateway (raspberry pi).

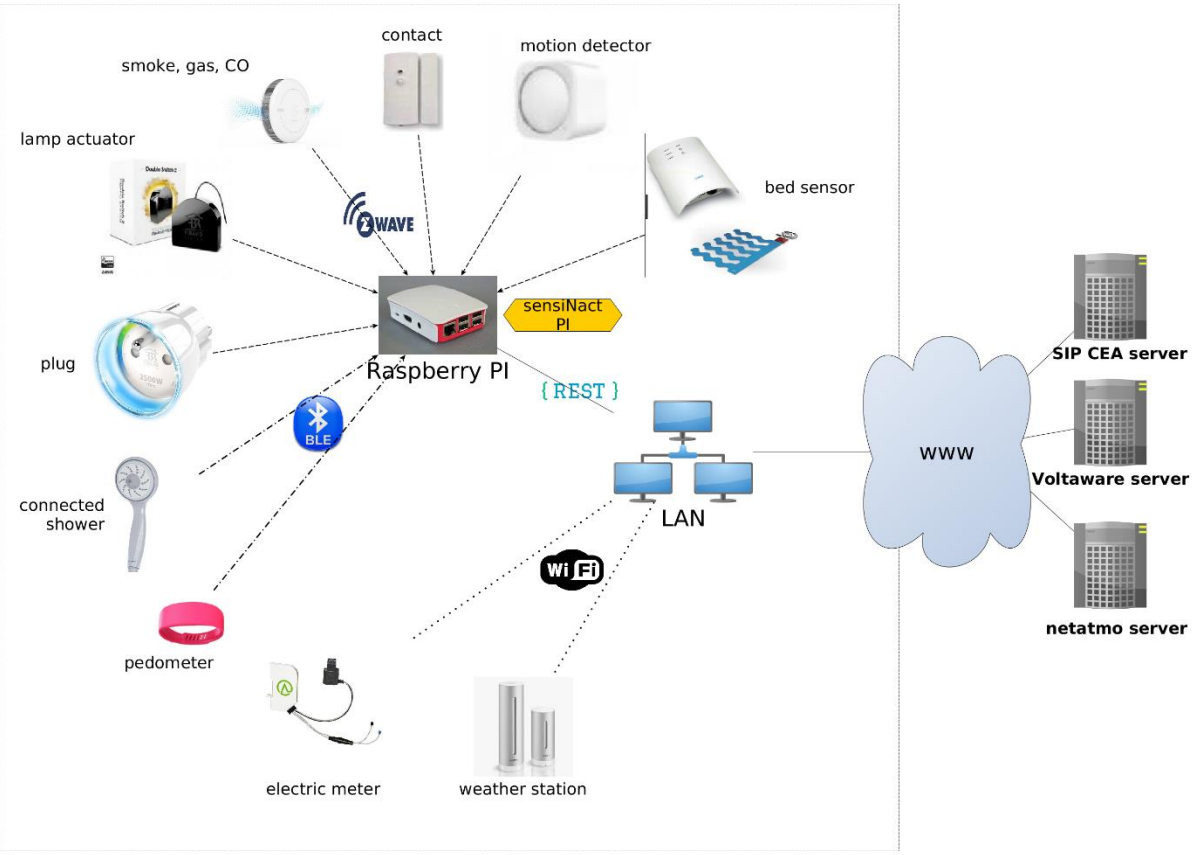

Figure 19: required protocol support in DS6 deployments

Figure 19 shows that the required protocols around the gateway are

- Zwave, for plugs, lamp actuators, air quality sensors, contact sensors, motion detectors and bed sensors
- BLE, for the connected shower and the pedometer
- REST for SIP (voice message synthesizer), electric meter and weather station that data are provided through web services.

Priority has been pushed on these three kinds of bridges to be handled by the sensiNact IoT platform deployed in DS6.

For the SIP REST support, a dedicated bridge has been implemented. See 3.5.12 for more details.

For the other two REST protocols, a first implementation of the three kinds of bridges is made involving openhab and mqtt existing bridges. The sensiNact edge protocol bridge architecture is described in Figure 20:

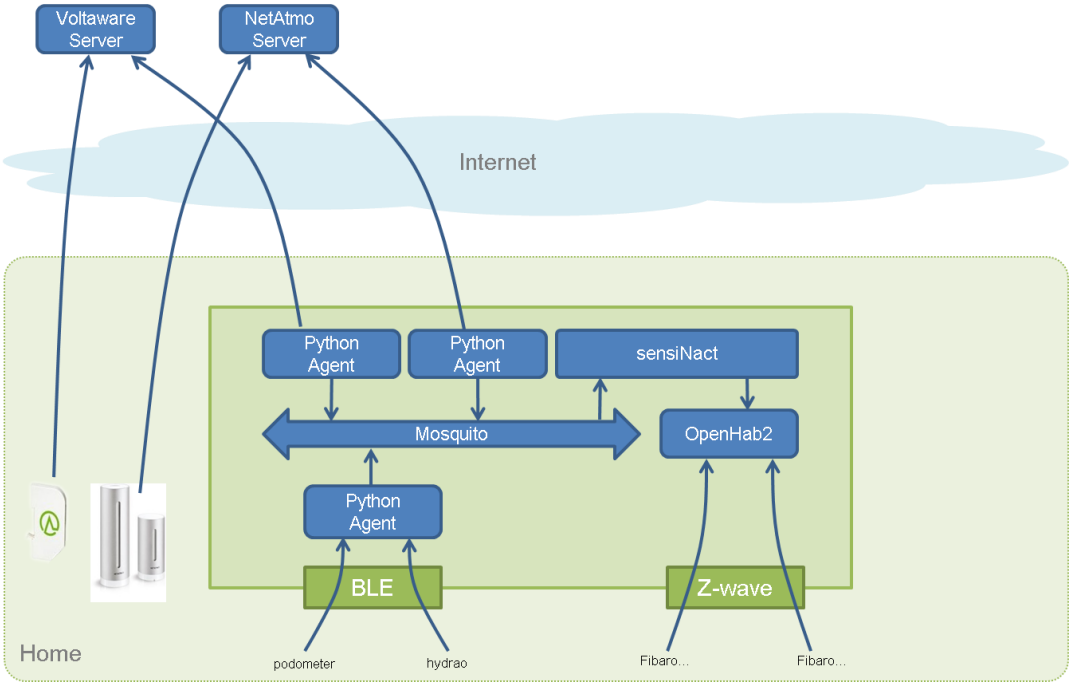

Figure 20: Architecture of the sensiNact edge protocol bridges for the ACTIVAGE project

The Table 6 below summarizes the IoT protocol bridges (already supported and planned) for the sensiNact platform.

IoT Protocol bridges for sensiNact					
Protocol	**Domain**	**Layer**	**Connection edges**	**Status**	**Comment**
Android IMU	accelerometer gyroscope and magnetometer	Sensor	Device to WAN	Implemented	Mobile phone as an accelerometer gyroscope and magnetometer
BLE	Personal Area Network	Sensor	Device to Gateway	todo	RF; Profiles: TI_cc2650
COAP	Generic	Sensor	Device to Gateway	todo (done in a previous version)	RF
EchoNet	Smart Building	Sensor	Device to Gateway	Implemented	Wired
e-Lio app	TV activity, social networking, sound	Gateway	gateway	todo	API

EnOcean	Smart Building	Sensor	Device to Gateway	Implemented (17 profiles)	RF; Profiles: F60201, F60202, F60203, F60204, D50001, A50701, A50201, A50202, A50203, A50204, A50205, A50401, A50402, A50403, D20106, D20108, D2010A
Free Mobile	sms	Service	Gateway to WAN	Implemented	sms sender
HTTP REST	Generic			Implemented	Abstract bridge; useful for further HTTP bridges
KNX-RF	Generic	Sensor	Device to Gateway	To be implemented	RF
KODI	TV	Device	Device tyo Gateway	Implemented	Based on HTTP
LoRa	Outdoor	Sensor	Device to Gateway	Implemented	RF ; Development of ad-hoc profiles
LoRaWAN	Outdoor	Sensor	Device to WAN	Implemented	Connection to Orange platform
MQTT	Generic	Service	Gateway to WAN	Implemented	Northbound and southbound
Netatmo	Weather station	Sensor	Device to LAN	Implemented	wifi or wired ethernet
NFC ACR12	RFID card reader	Sensor	Device to Gateway	To be implemented	USB
NGSI 9 & 10 v1	Generic	Interoper ability	Gateway to Gateway	Implemented	Related to FiWare
NGSI 9 & 10 v2	Generic	Interoper ability	Gateway to Gateway	To be implemented	The v2 specification is not yet ready
OpenHab	Generic	Gateway	Gateway	Implemented	API
Philips Hue	Domotic lighting	Sensor	Device to LAN	Implemented	wired ethernet
Sigfox	Generic	Sensor	Device to WAN	Implemented	RF
Tikitag	RFID card reader	Sensor	Device to Gateway	Implemented	RFID ; deprecated
X3D	Generic	Sensor	Device to Gateway	To be implemented	RF, Delta Dore proprietary protocol
Xbee	Generic	Sensor	Device to Gateway	Implemented	RF

XMPP	Generic	Sensor	Device to Gateway	Implemented	
Z-Wave	Generic	Sensor	Device to Gateway	To be implemented	RF

Table 6: sensiNact supported and planned IoT protocol bridges

sensiNact will be connected to other platforms via the SIL layer. In addition sensiNact has direct bridge to FIWARE via NGSI.

3.6 SENIORSOME bridges status

The SeniorSome platform enables the use of different IoT protocols, sensors and means to connect and build a large scale IoT network. Interoperability is achieved in two main layers: 1) physical device layer whereas there exists an API for connecting devices through different bridges (with APIs). 2) Further a layer is provided at the backend where full restful API resources are available. SeniorSome provides (aims to provide) open interoperability for the other IoT platforms.

IoT Protocol bridges for SENIORSome					
Protocol	**Domain**	**Layer**	**Connection edges**	**Status**	**Comment**
Bluetooth	Generic	Sensor	Device to Gateway	Implemented	
Wifi-direct	Generic	Sensor	Device to Gateway	Implemented	
COAP	Generic	Sensor	Device to Gateway	To be implemented	
HTTP REST	Generic	Sensor	Device to Gateway	Implemented	
HTTP/REST	Generic	Service	Gateway to WAN (2way)	Implemented	
MQTT	Generic	Service	Gateway to WAN	Implemented	
NGSI 9 & 10 v1	Generic	Interoperability	Gateway to Gateway	To be implemented	
NGSI 9 & 10 v2	Generic	Interoperability	Gateway to Gateway	To be implemented	

Table 7: SENIORSome supported and planned IoT protocol bridges

SENIORSome will be connected to other platforms via the SIL layer.

3.7 IoTivity bridges status

IoTivity is an open source software framework and SDK for building IoT applications and is backed by the Open Connectivity Foundation (OCF). It implements the OIC standards in order to ensure device-to-device connectivity for addressing the emerging needs of Internet of Things. The OIC protocol is a REST-like interface similar to HTTP and CoAP. However, it is a higher level abstraction of those protocols to allow for additional transports including Bluetooth Classic, Bluetooth Smart (BLE), Zigbee or others in the future.

The basic layer concepts to understand how IoTivity works are the core and the service layers [38]. The **core layer** provides the required infrastructure for handling connectivity (connectivity abstraction) and resource representation. It uses the interfaces provided by the operating system to access and control standard radio technologies like Bluetooth, WiFi, Zigbee. It can provide the required connectivity layer abstraction for applications to talk to devices irrespective of the connectivity type.

The **service layer** provides higher level abstractions for the resources like, groups of "things", virtual and logical sensor management and protocol plugin manager for interacting with non-OIC devices.

In an IoTivity network, things are represented as resources along with a set of REST style getters and setters (based on CoAP PUT, POST, GET, etc.). Services in IoTivity are built using resource representations of the devices/things.

Applications can make use of the IoTivity API via the SDK to access the "things" (OCResource) in an IoTivity network and perform operations on them.

Moreover, the **Web Service Interface (WSI)** [39] offered also by of IoTivity aims to simplify representation of web based services in a network of IoTivity devices. It can act as a conceptual bridging layer between the IoTivity network and web based services. Figure 21 presents the functional blocks of the Web Services Interface.

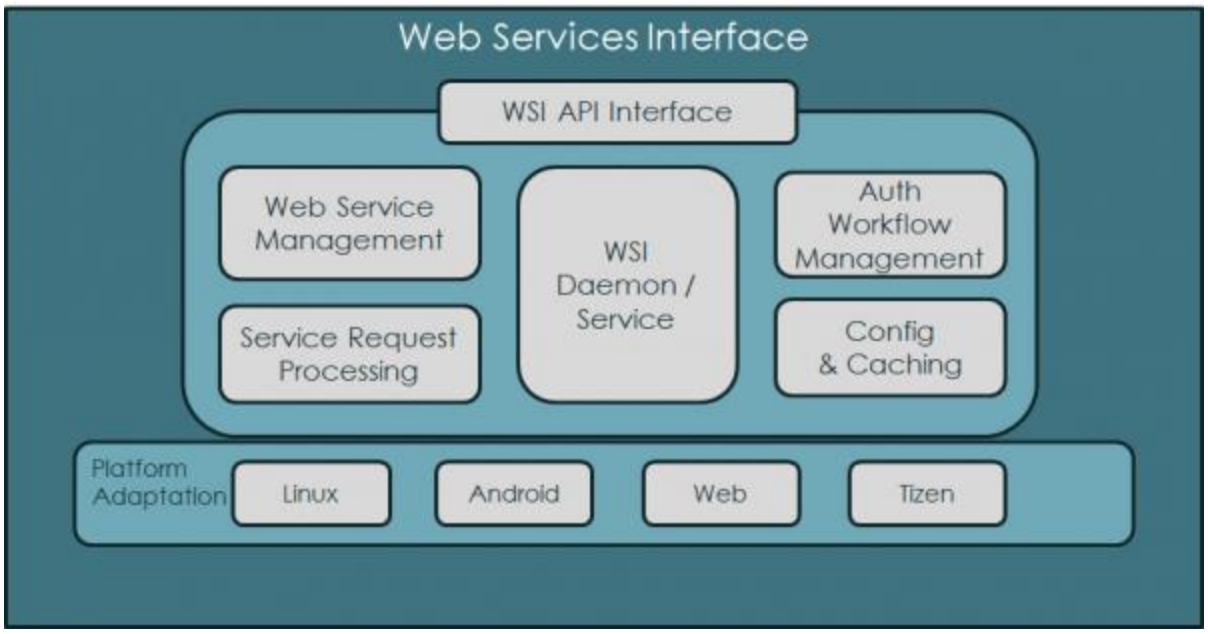

Figure 21: Functional blocks of the *Web Services Interface* of IoTivity

Except from the WSI, there are also 3 additional frameworks acting as bridges:

- The **Bridging Project** [40] – As IoTivity is compliant with only the OCF [41] ecosystem the community needs a way to bridge into other ecosystems to leverage ongoing support for the ever-growing OCF ecosystem. It is beneficial for developers and product manufacturers to support the OCF ecosystem as early as possible. The Bridging Project aims at providing the proper infrastructure and corresponding plugins for supporting developers and product manufacturers. Using protocol plugins, an application is able to communicate with various heterogeneous protocol devices using IoTivity API.
- **UPnP-Bridge** [42] – UpnP-Bridge is an open source software framework for discovering existing UPnP devices/services. Then, it can translate UPnP devices/services/actions into IoTivity devices/services and register them as IoTivity devices through the Resource Container [43]. This allows existing UPnP devices/services/action to be managed and used through IoTivity like any other IoTivity device.
- **AllJoyn-Bridge** [44] – AllJoyn Bridge is an open source software framework that implements generic "on-the-fly" translation between AllJoyn [45] and IoTivity devices (both AllJoyn consumers/producers and IoTivity clients/servers).

Below is showing the main protocol bridges provided by IoTivity:

3.7.1 CoAP Bridge

The Constrained Application Protocol (CoAP) is one of the IoTivity supported transports. It is designed to enable the communication of very simple devices with the wider Internet. For simplified integration with the web. it is modelled after HTTP and it is designed to easily translate to HTTP. It is based on UDP (instead of TCP) and provides support for multicast. CoAP is now a standard (RFC7252) as defined by the Internet Engineering Task Force (IETF) CoRE Working Group.

3.7.2 XMPP Bridge

The Extensible Messaging and Presence Protocol (XMPP) is a communication protocol for message-oriented middleware based on XML. It enables the exchange of data between any two or more network entities and supports multiple communication patterns, including Asynchronous Messaging, Publish/Subscribe and Request/Response. It is supported by IoTivity and can be used for remote communication scenarios (https://github.com/iotivity/iotivity-xmpp).

3.7.3 MQTT Bridge

MQTT (Message Queuing Telemetry Transport) is a lightweight publish and subscribe system where you can publish and receive messages as a client. It works on top of the TCP/IP protocol and it is designed for connections with remote locations and constrained devices with low-bandwidth. An implemented plugin enables the utilization of MQTT by IoTivity.

3.7.4 BLE Bridge

Bluetooth Low Energy is a variant of Bluetooth personal area network (PAN) technology, designed for use by Internet-connected machines and appliances. The IoTivity supports BLE Transport for GNU/Linux distributions through an implementation of the proposed Bluetooth GATT-based OIC Transport Profile. For bridging the IoTivity connectivity abstraction (CA) adapter interface and the underlying BLE stack it provides GAP central and peripheral roles, and GATT client and server roles. A build in adapter provides the aforementioned functionality (https://github.com/iotivity/iotivity/tree/master/resource/csdk/connectivity/src/bt_le_adapter).

3.7.5 Bluetooth Bridge

Bluetooth EDR (Enhanced Data Rate) or Bluetooth 2 is an upgrade of the original Bluetooth specification. It is based on the original Bluetooth standard which is well established as a wireless technology. IoTivity supports the Bluetooth EDR connectivity using RFCOMM for Android and Tizen platforms through a build in adapter (https://github.com/iotivity/iotivity/tree/master/resource/csdk/connectivity/src/bt_edr_adapter).

3.7.6 ZigBee Bridge

Zigbee is a suite of high level communication protocols based on IEEE 802.15.4 used to create personal area networks with small, low-power digital radios, such as for home automation, medical device data collection, and other low-power low-bandwidth needs. IoTivity is able to utilize ZigBee by enabling the corresponding implemented plug-in (https://github.com/iotivity/iotivity/tree/master/plugins/zigbee_wrapper).

3.7.7 UPnP Bridge

Universal Plug and Play (UPnP) is a standard that uses Internet and Web protocols to enable networked devices, such as personal computers, Internet gateways, Wi-Fi access points and mobile devices to discover each other in a network and establish functional services for data sharing, communications, and entertainment. The utilization of UPnP by IoTivity is made through the aforementioned UPnP bridge (https://github.com/iotivity/iotivity-upnp-bridge).

3.7.8 Summary of IoTivity IoT bridges status

Although IoTivity provides several protocol bridges to communicate gateways and devices, these are mostly related to OIC compliant devices. CERTH has implemented device specific bridges in order to support the communication between the gateway and some of the devices that are used by DS5, which are non-OIC compliant. These bridges were implemented by using the IoTivity SDK and include communication through Z-Wave and BLE.

Next table provides the IoT protocol bridges for IoTivity.

IoT Protocol bridges for IoTivity					
Protocol	**Domain**	**Layer**	**Connection edges**	**Status**	**Comment**
Constrained Application Protocol (CoAP)	Generic	Sensor, Service	Device to Gateway and Gateway to WAN	Implemented	
XMPP	Generic	Service	Gateway to WAN	Implemented	
MQTT	Generic	Service	Device to Gateway and Gateway to WAN	Implemented	MQTT is supported as a protocol plug-in
Wi-Fi Direct	Generic	Sensor	Device to Gateway	Implemented	
Bluetooth low energy	Generic	Sensor	Device to Gateway	Implemented	
Bluetooth	Generic	Sensor	Device to Gateway	Implemented	
Zigbee	Generic	Sensor	Device to Gateway	Implemented	ZigBee is supported as a protocol plug-in
UPnP Device Control Protocol (DCP)	Generic	Sensor	Device to Gateway	Implemented	https://openconnectivity.org/developer/specifications/upnp-resources/upnp

Table 8: IoTivity supported and planned IoT protocol bridges

IoTivity will be connected to other platforms via the SIL layer via MQTT

4 Summary and Conclusions

Six out of seven ACTIVAGE supported IoT platforms have been deployed into the 9 Deployment sites. The following table lists the IoT platforms deployed at each deployment site.

DS \ Platform		universAAL	SOFIA2	OPENIOT	FIWARE	sensiNact	SENIOR SOME	IoTivity
DS1	GAL		deployed					
DS2	VLC	deployed			deployed			
DS3	MAD	deployed						
DS4	RER				deployed			
DS5	GRC	deployed			deployed			
DS6	GNB					deployed		
DS7	WOQ	deployed						
DS8	UK							deployed
DS9	FIN						deployed	

Table 9: ACTIVAGE IoT platforms as they are deployed among deployment sites

The below table provides a synthetic list of the supported protocols by the ACTIVAGE IoT platforms, which can thus be considered as ACTIVAGE supported IoT protocols. This means that any application or platform compatible with ACTIVAGE (aka AIOTES) can be compatible with all of these protocols, thanks to the Semantic Interoperability Layer that has been developed by the project.

Protocol \ Platform	UNIVERS AAL	SOFIA2	OPENIOT	FIWARE	sensiNact	SENIOR SOME	IoTivity
Android IMU					done		
BLE	done for Continua certified devices		ongoing		ongoing		done
Bluetooth						done	done
COAP			Done	todo (done in a previous version)	todo (done in a previous version)	todo	done

EchoNet					done		
e-Lio app					todo		
EnOcean					done (17 profiles)		
Free Mobile					done		
Fs20	done						
HTTP REST	done	done	done	done	done	done	
HTTP SOAP	done	done	done	done			
KNX(-RF)	done				todo		
KODI					done		
LoRa					done		
LoRaWAN					done		
MQTT		done	done	done	done	done	done (MQTT as protocol plug-in)
Netatmo					done		
NFC ACR12					todo		
NGSI 9 & 10 v1				done	done	todo	
NGSI 9 & 10 v2				done	todo	todo	
OpenHab	done				done		
Philips Hue	todo				done		done

RSS			done				
Sigfox				done	done		
Tikitag					done		
UDP			done				
UPnP DCP							done
Wi-fi Direct						done	done
X3D					todo		
Xbee					done		
XMPP			done		done		done
Zigbee	done		done	todo			done (ZigBee as protocol plug-in)
Z-Wave	done			todo	todo		

Table 10: IoT Edge protocol bridges, including open call protocols for each one of the 7 IoT platforms in ACTIVAGE project

Some of the above mentioned protocols remain critical for some of the deployment sites. The table below lists those protocols (rows) and the support of the protocols at the relevant deployment site (columns) by the used IoT platform. It also shows new development bridges developped since the last version of the table given in D3.4

Latest IoT protocols in Deployment Sites									
IoT protocol	**DS1 GAL**	**DS2 VLC**	**DS3 MAD**	**DS4 RER**	**DS5 GRC**	**DS6 ISE**	**DS7 WOQ**	**DS8 UK**	**DS9 FIN**
BLE	**NEW bridge! SOFIA2**	UNIVERSAAL (continua certified devices)			UNIVERSAAL (continua certified devices)	sensiNact			

BlueTooth	not supported by SOFIA2 **PLANNED 2019**		not supported by UNIVERSAAL **PLANNED 2019**		Fiware (Implemented support)			IoTivity	SENIORsome
proprietary bed sensor RF protocol						not supported by sensiNact **Planned 2019**	UNIVERSAAL		**NEW bridge! SENIORsome**
SIP (phone ip)						**NEW bridge! sensiNact**	UNIVERSAAL		**NEW bridge! SENIORsome**
Zigbee	SOFIA2								SENIORsome
Zwave			UNIVERSAAL		UNIVERSAAL	sensiNact	UNIVERSAAL		

Table 11: Check table between request and availability of edge protocol for each deployment site

From Table 10 above, we can extract the required edge protocols that are not yet managed by the installed IoT platforms. For instance, in the DS1 GAL deployment site, the SOFIA2 deployed IoT platform did not support needed Bluetooth and BLE protocols at the device layer. A new bridge now has been developped to cover the BLE. The support for Bluetooth is planned to be developped in 2019. Similarly UniversAAL will be adding the Bluetoorh suport for the Madrid deployment. Isere site will be developping a bridge for a proprietary bed sensor.

As for the BLE support in Greece site, although the deployed universAAL platform for the smart home scenario supports BLE protocol, it is only for Continua certified devices. There may be a need to develop additional bridge for some other specific devices for measuring blood pressure/glucose levels, which is not Continua certified.

Below table summarises the ations taken to complete the development/integration of the missing bridges.

Overview of actions at the DS for the missing protocol support					
DS		**Missing IoT protocol**	**Recommendation**	**Status**	**Action**
DS1	GAL	Bluetooth + BLE	interoperate SOFIA2 with other IoT platform supporting BLE	OK planned 2019	For their first deployments, DS1 uses a proprietary IoT platform developed by TELEVES, hosted in gateway. OK to switch to ACTIVAGE platform in 2019, following the progress of the AIOTES interoperability layer implementation

DS3	MAD	Bluetooth + BLE	Develop specific bridge or interoperate UNIVERSAAL with IoTivity	On-going developments	DS3 developed a module dedicated to their bluetooth lamp: specific bridge for the brand, modularized in such a way that it could become an official universAAL bridge. IoTivity bridges may not be appropriate for the DS3 bluetooth devices that most are not standard profiles (device manufacturers typically create a proprietary or even adhoc protocol over the serial profile: this is one of such cases for DS3 devices).
DS5	GR	BLE communication with a non Continua certified device	Develop specific bridge or interoperate UNIVERSAAL with IoTivity	Ongoing developments	Both solutions will be implemented by CERTH. They will be evaluated by comparing several technical parameters like performance, memory consumption, etc., and the solution with the best technical KPIs will be finally deployed in the Greek DS.
DS6	ISE	bed sensor proprietary protocol	interoperate sensiNact with UNIVERSAAL	OK planned 2019	Once SIL level interoperability level finalised, interoperability tests will be conducted
DS9	FIN	bed sensor and SIP proprietary protocol	interoperate SENIORSOME with UNIVERSAAL	Ongoing discussion	Once SIL level interoperability level finalised, interoperability tests will be conducted

Table 12: Updated overview and status of the recommendations to DS for ACTIVAGE interoperability

Any remaining missing support for any particular protocol for any deployment site will be covered by ACTIVAGE's platform level interoperablity (AIOTES). This is the power of ACTIVAGE which will provide a complete portfolio of IoT protocols used in the Ambeint Assisted Living domain.

References

[1] UNIVERSAAL IOT, SEMANTIC INTEROPERABILITY FOR RAPID INTEGRATION & DEPLOYMENT: HTTPS://GITHUB.COM/UNIVERSAAL

[2] SOFIA2 WEB SITE: HTTP://SOFIA2.COM/HOME_EN.HTML

[3] OPENIOT – OPEN SOURCE CLOUD SOLUTION FOR THE INTERNET OF THINGS: HTTP://WWW.OPENIOT.EU

[4] SENSINACT WEB SITE: HTTPS://PROJECTS.ECLIPSE.ORG/PROJECTS/TECHNOLOGY.SENSINACT

[5] FIWARE WEB SITE: HTTPS://WWW.FIWARE.ORG/

[6] IOTIVITY WEB SITE: HTTPS://WWW.IOTIVITY.ORG/

[7] SENIORSOME WEB SITE: HTTP://WWW.SENIORSOME.COM

[8] ONEM2M WEBSITE: http://www.onem2m.org

[9] THE FUTURE INTERNET PUBLIC-PRIVATE PARTNERSHIP (FI-PPP) HTTPS://WWW.FI-PPP.EU/

[10] FAST RTPS DOCUMENTATION: HTTPS://EPROSIMA-FAST-RTPS.READTHEDOCS.IO/EN/LATEST/

[11] THE CONSTRAINED APPLICATION PROTOCOL (COAP) HTTPS://TOOLS.IETF.ORG/HTML/RFC7252

[12] NODE-COAP LIBRARY: HTTPS://WWW.NPMJS.COM/PACKAGE/COAP

[13] ECLIPSE MOSQUITTO: HTTPS://MOSQUITTO.ORG/

[14] NGSI CONTEXT MANAGEMENT V1: HTTP://WWW.OPENMOBILEALLIANCE.ORG/RELEASE/NGSI/V1_0-20120529-A/OMA-TS-NGSI_CONTEXT_MANAGEMENT-V1_0-20120529-A.PDF

[15] ETSI GS CIM 004 V1.1.1: HTTPS://WWW.ETSI.ORG/DELIVER/ETSI_GS/CIM/001_099/004/01.01.01_60/GS_CIM004V010101P.PDF

[16] SOFIA2 WEB SITE: HTTP://SOFIA2.COM/HOME_EN.HTML

[17] UNIVERSAAL IOT, SEMANTIC INTEROPERABILITY FOR RAPID INTEGRATION & DEPLOYMENT: HTTPS://GITHUB.COM/UNIVERSAAL

[18] ZIGBEE ALLIANCE HTTP://WWW.ZIGBEE.ORG/

[19] ZB4O - ZIGBEE™ API FOR OSGI™ SERVICE PLATFORMS: HTTP://ZB4OSGI.AALOA.ORG/

[20] PERSONAL CONNECTED HEALTH ALLIANCE: HTTP://WWW.PCHALLIANCE.ORG/

[21] FS20 PROTOCOL: HTTP://FHZ4LINUX.INFO/TIKI-INDEX.PHP?PAGE=FS20%20PROTOCOL

[22] ECLIPSE SMARTHOME, A FLEXIBLE FRAMEWORK FOR THE SMART HOME: HTTPS://WWW.ECLIPSE.ORG/SMARTHOME/INDEX.HTML

[23] OPENIOT – OPEN SOURCE CLOUD SOLUTION FOR THE INTERNET OF THINGS: HTTP://WWW.OPENIOT.EU

[24] OPENIOT GITHUB: HTTPS://GITHUB.COM/OPENIOTORG/OPENIOT

[25] SENSINACT MQTT BRIDGE WIKI: HTTP://WIKI.ECLIPSE.ORG/SENSINACT/BRIDGE_MQTT

[26] DEVELOP > API GUIDES > LOCATION AND SENSORS > SENSORS OVERVIEW : HTTPS://GITHUB.COM/LSIR/GSNANDROID SENSOR OVERVIEW: HTTPS://DEVELOPER.ANDROID.COM/GUIDE/TOPICS/SENSORS/SENSORS_OVERVIEW.HTML

[27] TECHNOSENS WEB SITE: HTTP://WWW.TECHNOSENS.FR/

[28] NETATMOCONNECT WEB PAGE: HTTPS://DEV.NETATMO.COM/EN-US/DEV

[29] NETATMOCONNECT RATE LIMITS: HTTPS://DEV.NETATMO.COM/EN-US/RESOURCES/TECHNICAL/GUIDES/RATELIMITS

[30] ECHONET WEB PAGE: HTTPS://ECHONET.JP/ABOUT_EN/

[31] ECHONET OPEN SPECIFICATIONS: HTTPS://ECHONET.JP/SPEC-EN/#STANDARD-01

[32] COAP TECHNOLOGY: HTTP://COAP.TECHNOLOGY/

[33] OPENHAB WEB SITE: HTTPS://WWW.OPENHAB.ORG/

[34] SENSINACT OPENHAB BRIDGE WIKI: HTTP://WIKI.ECLIPSE.ORG/SENSINACT/BRIDGE_OPENHAB

[35] DISCOVER THE LORA® EXPLORER KIT: HTTPS://PARTNER.ORANGE.COM/DISCOVER-ORANGE-EXPLORER-LORA-KIT/

[36] A SENSINACT BRIDGE TUTORIAL - THE WEATHER STATION EXAMPLE: HTTP://WIKI.ECLIPSE.ORG/SENSINACT/TUTORIAL_BRIDGE

[37] PHILIPS HUE DEVELOPER PROGRAM: HTTPS://WWW.DEVELOPERS.MEETHUE.COM

[38] IOTIVITY WEB SERVICE INTERFACE (WSI): HTTPS://WIKI.IOTIVITY.ORG/WEB_SERVICE_INTERFACE

[39] IOTIVITY WEB SERVICE INTERFACE (WSI): HTTPS://WIKI.IOTIVITY.ORG/WSI

[40] IOTIVITY BRIDGING PROJECT: HTTPS://WIKI.IOTIVITY.ORG/BRIDGING_PROJECT

[41] OPEN CONNECTIVITY FUNDATION: HTTPS://OPENCONNECTIVITY.ORG/DEVELOPER/REFERENCE-IMPLEMENTATION/IOTIVITY

[42] IOTIVITY UPNP BRIDGE: HTTPS://WIKI.IOTIVITY.ORG/IOTIVITY_UPNP_BRIDGE

[43] IOTIVITY WIKI: HTTPS://WIKI.IOTIVITY.ORG/RESOURCE_CONTAINER

[44] IOTIVITY GITHUB: HTTPS://GITHUB.COM/IOTIVITY/IOTIVITY-ALLJOYN-BRIDGE

[45] ALLSEEN ALLIANCE: HTTPS://ALLSEENALLIANCE.ORG/FRAMEWORK/DOCUMENTATION/DEVELOP/BUILDING

[46] NOUVELLE OPTION « NOTIFICATIONS PAR SMS » CHEZ FREE MOBILE: HTTPS://WWW.FREENEWS.FR/FREENEWS-EDITION-NATIONALE-299/FREE-MOBILE-170/NOUVELLE-OPTION-NOTIFICATIONS-PAR-SMS-CHEZ-FREE-MOBILE-14817

[47] HTTP://WWW.DOMOTIQUE-INFO.FR/2014/06/NOUVELLE-API-SMS-CHEZ-FREE/

[48] INTER-IOT PROJECT: HTTP://WWW.INTER-IOT-PROJECT.EU/

Appendix. Open IoT code examples

This appendix gives some examples of code using the bridges for OpenIoT platform. Examples for other platforms can be found in the relevant documentation of each platform.

Zigbee bridge

The following is an example of source code for sending sensor data using Zigbee Xbee3 boards from Digi manufacturer[1]. This example particularly shows how to send data streams over a gateway after some sensor readings are locally collected.

```
# Digi XBee3 Cellular DRM Example
# uses a TMP36 to measure temperature and post it to Digi Remote Manager
# by default repeating once per minute, 1440 times total, stopping after a day
# ENTER YOUR DRM username and password, REPLACING "your_username_here" etc. BEFORE UPLOADING THIS CODE!
from remotemanager import RemoteManagerConnection
from machine import ADC
from time import sleep
from xbee import atcmd
cycles = 1440 # number of repeats
wait_time = 60 # seconds between measurements
username = 'your_username_here' #enter your username!
password = 'your_password_here' #enter your password!
# Device Cloud connection info
stream_id = 'temperature'
stream_type = 'FLOAT'
stream_units = 'degrees F'
description = "temperature example"
# prepare for connection
credentials = {'username': username, 'password': password}
stream_info = {"description": description,
               "id": stream_id,
               "type": stream_type,
               "units": stream_units}
```

[1] https://www.digi.com/blog/hands-on-micropython-programming-examples-for-edge-computing-part-2/

```
ai_desc = {
    0x00: 'CONNECTED',
    0x22: 'REGISTERING_TO_NETWORK',
    0x23: 'CONNECTING_TO_INTERNET',
    0x24: 'RECOVERY_NEEDED',
    0x25: 'NETWORK_REG_FAILURE',
    0x2A: 'AIRPLANE_MODE',
    0x2B: 'USB_DIRECT',
    0x2C: 'PSM_DORMANT',
    0x2F: 'BYPASS_MODE_ACTIVE',
    0xFF: 'MODEM_INITIALIZING',
}
def watch_ai():
    old_ai = -1
    while old_ai != 0x00:
        new_ai = atcmd('AI')
        if new_ai != old_ai:
            print("ATAI=0x%02X (%s)" % (new_ai, ai_desc.get(new_ai, 'UNKNOWN')))
            old_ai = new_ai
        else:
            sleep(0.01)
# Main Program
# create a connection
rm = RemoteManagerConnection(credentials=credentials)
# update data feed info
print("updating stream info...", end ="")
try:
    rm.update_datastream(stream_id, stream_info)
    print("done")
except Exception as e:
    status = type(e).__name__ + ': ' + str(e)
    print('\r\nexception:', e)

while True:
    print("checking connection...")
    watch_ai()
    print("connected")
    for x in range(cycles):
        # read temperature value & print to debug
        temp_pin = ADC("D0")
        temp_raw = temp_pin.read()
        print("raw pin reading: %d" % temp_raw)
```

```
    # convert temperature to proper units
    temperatureC = (int((temp_raw * (2500/4096)) - 500) / 10)
    print("temperature: %d Celsius" % temperatureC)
    temperatureF = (int(temperatureC * 9.0 / 5.0) + 32.0);
    print("temperature: %d Fahrenheit" % temperatureF)

    # send data points to DRM
    print("posting data...", end ="")
    try:
        status = rm.add_datapoint(stream_id, temperatureF) # post data to Device Cloud
        print("done")
        print('posted to stream:', stream_id, '| data:', round(temperatureF), '| status:', status.status_code)
    except Exception as e:
        print('\r\nexception:', e)
    # wait between cycles
    sleep(wait_time)
```

BLE Bridge

The following is a data format example for AWS IoT services for sending sensor data using a FreeRTOS BLE Board. This example particularly shows how to send data streams to Amazon Cloud after some sensor readings are locally collected.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:<aws-region>:<aws-account-id>:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": "arn:aws:iot:<aws-region>:<aws-account-id>:*"
    },
    {
      "Effect": "Allow",
```

```
      "Action": "iot:Subscribe",
      "Resource": "arn:aws:iot:<aws-region>:<aws-account-id>:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": "arn:aws:iot:<aws-region>:<aws-account-id>:*"
    }
  ]
}
```

MQTT Bridge

The following is an example of source code for sending sensor data with MQTT using a standard Arduino board and the “PubSubClient” library. This example particularly shows how to send data streams over a gateway after some sensor readings are locally collected.

```
#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h>

// Update these with values suitable for your network.

byte mac[]    = {  0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };
IPAddress ip(172, 16, 0, 100);
IPAddress server(172, 16, 0, 2);

EthernetClient ethClient;
PubSubClient client(ethClient);

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    if (client.connect("arduinoClient")) {
      client.publish("tempC", "85");
    } else {
      Serial.print("failed, rc=");
```

```
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);  // Wait 5 seconds before retrying
    }
  }
}

void setup() {
  Serial.begin(57600);
  client.setServer(server, 1883);
  Ethernet.begin(mac, ip);
  delay(1500);  // Allow the hardware to sort itself out
}

void loop() {
  if (!client.connected())
    reconnect();
  client.loop();
}
```

CoAP Bridge

The following is an example of source code for sending sensor data with CoAP using a standard Arduino board and the “CoAP-simple” library. This example particularly shows how to send data streams over a gateway after some sensor readings are locally collected.

```
#include <SPI.h>
#include <Dhcp.h>
#include <Dns.h>
#include <Ethernet.h>
#include <EthernetUdp.h>
#include <coap-simple.h>

byte mac[] = { 0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };

// UDP and CoAP class
EthernetUDP Udp;
```

```
Coap coap(Udp);

void setup() {
  Serial.begin(9600);
  Ethernet.begin(mac);
  coap.start();  // start coap server/client
}

void loop() {
  Serial.println("Send Request");
  int msgid = coap.get(IPAddress(172, 16, 0, 2), 5683, "tempC", "85");
  delay(1000);
  coap.loop();
}
```

HTTP-GET Bridge

The following is an example of source code for sending sensor data with HTTP GET using a standard Arduino board and the "HttpClient" library. This example particularly shows how to send data streams over a gateway after some sensor readings are locally collected.

```
#include <HttpClient.h>
void loop() {
  HttpClient client;
  client.get("http://openiotserver.org/report?sensor=temp&value=85");
  while (client.available()) {
    char c = client.read();  // discard the response for now
  }
  delay(5000);
}
```

HTTP REST Bridge

The following is an example of source code for sending sensor data with HTTP REST using the PUT method on a standard Arduino board and the "HttpClient" library. This example particularly shows how to send data streams over a gateway after some sensor readings are locally collected.

```
#include <HttpClient.h>
```

```
void loop() {
  HttpClient client;
  const char *data = "id=1234,tempC=85";
  client.put("http://iotserver.org/sensors", data);
  while (client.available()) {
    char c = client.read();  // discard the response for now
  }
  delay(5000);
}
```

HTTP SOAP Bridge

The following is an example of source code for sending sensor data with SOAP using the Python programming language and the “Zeep” library. The code can run on mid-range embedded devices such as the Raspberry Pi. This example particularly shows how to send data streams over a gateway after some sensor readings are locally collected.

```
from zeep import Client
client = Client('http://iotserver.org/Sensor?WSDL')
if not client.service.ReportTemperature(62, 'Celsius'):
    raise Error("could not report temperature to OpenIoT")
if not client.service.ReportHeartRate(85):
    raise Error("could not report heart rate to OpenIoT")
```

RSS Bridge

The following is an example RSS message in XML for publishing sensor data to an RSS aggregator. The data can be sent to the RSS server using any HTTP client library available to the sensor device. This example particularly shows how to send data streams over a gateway after some sensor readings are locally collected.

```
<?xml version="1.0" encoding="UTF-8" ?>

<rss version=”2.0”>
<channel>
 <title>Temperature Sensor</title>
 <description>This is the RSS feed for the temperature sensor</description>
```

```
 <link>http://172.16.0.1/sensorManagement</link>
 <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000 </lastBuildDate>
 <pubDate>Sun, 06 Sep 2009 16:20:00 +0000</pubDate>
 <ttl>1800</ttl>
 <item>
  <title>Temperature Reading</title>
  <description>Here is the temperature reading from the sensor.</description>
  <link>http://127.16.0.1/sensorReading/7bd204c6-1655-4c27-aeee-53f933c5395f</link>
  <guid isPermaLink="false">7bd204c6-1655-4c27-aeee-53f933c5395f</guid>
  <pubDate>Sun, 06 Sep 2009 16:20:00 +0000</pubDate>
 </item>
</channel>
</rss>
```

UDP Bridge

The following is an example of source code for sending sensor data with UDP on a standard Arduino board and the “Ethernet” library. This example particularly shows how to send data streams over a gateway after some sensor readings are locally collected.

```
#include <Ethernet.h>
#include <EthernetUdp.h>

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192, 168, 1, 177);

// The IP address of the remote system
IPAddress remoteIp(172, 16, 0, 1);
int remotePort = 9000;
// An EthernetUDP instance to let us send and receive packets over UDP
EthernetUDP Udp;

void setup() {
  Ethernet.begin(mac, ip);
}
```

```
void loop() {
  Udp.beginPacket(remoteIp, remotePort);
  Udp.write("temperature=85");
  Udp.endPacket();
}
```

XMPP Bridge

The following is an example of source code for sending sensor data with XMPP on a standard Arduino board and the “CiaoXMPP” library. This example particularly shows how to send data streams over a gateway after some sensor readings are locally collected.

```
#include <Ciao.h>
// requires setting up and enabling the XMPP connector for the sketch
// https://www.arduino.cc/en/Reference/CiaoXMPP
String USER = "sensor@openiot.org";
void setup() {
  Ciao.begin();
}
void loop() {
  CiaoData data = Ciao.read("xmpp");
  if (!data.isEmpty() && !data.isError()) {
    String id = data.get(0);
    String sender = data.get(1);
    String message = data.get(2);
    message.toLowerCase();
    if (message == "read temp") {
      Ciao.writeResponse("xmpp", id, "value=62, unit=celsius");
    }
    else if (message == "read heartbeat") {
      Ciao.writeResponse("xmpp", id, "value=85, unit=bpm");
    }
  }
}
```