This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n: 761999

EasyTV: Easing the access of Europeans with disabilities to converging media and content.

D3.9 Device Interoperability

EasyTV Project

H2020. ICT-19-2017 Media and content convergence. – IA Innovation action.

Grant Agreement n°: 761999

Start date of project: 1 Oct. 2017

Duration: 30 months

Document. ref.: D5.4

Disclaimer

The EasyTV Consortium consists of the following partners:

	Partner Name	Short name	Country
1	Universidad Politécnica de Madrid	UPM	ES
2	Engineering Ingegneria Informatica S.P.A.	ENG	IT
3	Centre for Research and Technology Hellas/Information Technologies Institute	CERTH	GR
4	Mediavoice SRL	MV	IT
5	Universitat Autònoma Barcelona	UAB	ES
6	Corporació Catalana de Mitjans Audiovisuals SA	CCMA	ES
7	ARX.NET SA	ARX	GR
8	Fundación Confederación Nacional Sordos España para la supresión de barreras de comunicación	FCNSE	ES
9	Unione Italiana dei ciechi e degli ipovedenti	UICI	IT

PROGRAMME NAME:	**H2020. ICT-19-2017 Media and Content Convergence – IA Innovation Action**
PROJECT NUMBER:	761999
PROJECT TITLE:	EASYTV
RESPONSIBLE UNIT:	ARX
INVOLVED UNITS:	ENG, MV
DOCUMENT NUMBER:	D3.9
DOCUMENT TITLE:	Device Interoperability
WORK-PACKAGE:	WP 3
DELIVERABLE TYPE:	R
CONTRACTUAL DATE OF DELIVERY:	31-10-2018
LAST UPDATE:	25-10-2018
DISTRIBUTION LEVEL:	PU

Distribution level:

PU = *Public,*

RE = *Restricted to a group of the specified Consortium,*

PP = *Restricted to other program participants (including Commission Services),*

CO = *Confidential, only for members of the LASIE Consortium (including the Commission Services)*

Document History

VERSION	DATE	STATUS	AUTHORS, REVIEWER	DESCRIPTION
v.0.1	05/11/2019	Draft	Stavros Skourtis ARX Chrysostomos Bourlis ARX Christos-Menelaos Vlemmas ARX Athanasios Chatziathanasiou ARX Serafeim Kosyvakis ARX	Table of Contents definition and document structure
V0.2	11/11/2019	Draft	Stavros Skourtis ARX Serafeim Kosyvakis ARX Athanasios Chatziathanasiou ARX	First draft
V0.3	04/12/2019	Draft	Giuseppe Vitolo ENG	Added 2.4, 2.5, 2.6
V0.4	07/12/2019	Draft	Christos-Menelaos Vlemmas ARX Serafeim Kosyvakis ARX Athanasios Chatziathanasiou ARX	Chapter 3
V0.5	18/12/2019	Draft	Stavros Skourtis ARX Chrysostomos Bourlis ARX Christos-Menelaos Vlemmas ARX Nicolamaria Manes MV	First full version ready for internal review
V0.6	20/12/2019	Peer-review	CERTH	General review
V0.7	20/12/2019	Peer-review	Stavros Skourtis ARX Chrysostomos Bourlis ARX	Made changes based on internal review

Definitions, Acronyms and Abbreviations

ACRONYMS / ABBREVIATIONS	DESCRIPTION
HbbTV	Hybrid broadcast broadband TV
IoT	Internet of Things
SDK	Service Development Kit
AVS	Alexa Voice Service
API	Application programming interface
LWA	Login with Amazon
BLE	Bluetooth Low Energy
GAP	Generic Access Profile
GATT	Generic Attribute Profile
UUID	Universally unique identifier
HA	Home Automation
MQTT	Message Queue Telemetry Transport
M2M	Machine to machine
AMQP	Advanced Message Queuing Protocol
CoAP	Constrained Application Protocol

Table of Contents

List of Figures

Executive Summary

The present document describes the research and development that is done for Task 3.5 “Device Interoperability”. This task focuses on technologies and services that will assist people with disabilities in order to control their home appliances.

The document will cover the following topics:

Chapter 1 will provide an introduction and an overview of this document.

Chapter 2 will cover all the technologies and services that were investigated

Chapter 3 will describe the integration that was done with the HbbTV Companion Screen application

1. Introduction

The Internet of Things (IoT) relies on products to exchange, share, and interpret data. Interoperability ensures all components work together according to requirements and expectations for performance, security, and data integrity. Interoperability verifies that products and systems will form an integrated solution, enabling seamless communication with one another. An IoT platform can be defined as an intelligent layer that connects the devices to the network and that abstracts applications from them with the goal to enable the development of services. IoT platforms achieve several main objectives, such as flexibility (being able to deploy things in different contexts), usability (being able to make the user experience easy) and productivity (enabling service creation in order to improve efficiency, but also enabling new service development). An IoT platform facilitates communication, data flow, device management, and the functionality of applications. The goal is to build IoT applications integrated with the EasyTV HbbTV Companion Screen application.

- Integration of the EasyTV Companion Screen with Philips hue

This deliverable contains many technical details in order to describe the usage and the features of the various components of the EasyTV SDK. Therefore, the reader of this document must have sufficient technical knowledge about software development in order to be able to properly understand it.

2. IOT and Smart Home Solutions

2.1. Amazon Echo & Alexa devices

2.1.1. Introduction

Alexa built-in is a category of devices created with the Alexa Voice Service (AVS) that have a microphone and speaker. Anyone can talk to these products directly with the wake word “Alexa,” and receive voice responses and content instantly. Alexa built-in products work with Alexa skills and Alexa-compatible smart home devices, bringing familiar capabilities from the Amazon Echo family of devices to a range of new form factors and use cases developed by leading brands.

2.1.2. Alexa Voice Service

The Alexa Voice Service enables access cloud-based Alexa capabilities with the support of AVS APIs, hardware kits, software tools, and documentation. We simplify building voice-forward devices with Alexa built-in by handling complex speech recognition and natural language understanding in the cloud, reducing development costs and accelerating time to market. Most of all, regular Alexa updates bring new features to the device and add support for a growing assortment of compatible smart home devices.

So the first idea was to start with AVS. To access the AVS API, any product needs to obtain a Login with Amazon (LWA) access token, which grants it access to call the API on a customer's behalf. There are multiple ways to authorize a product:

Remote Authorization is used to authorize devices with a companion website or mobile app, typically, remote authorization is used with headless devices, like a smart speaker.

- Authorize from companion App

To access the Alexa Voice Service (AVS), the Alexa Built-in product needs to obtain a Login with Amazon (LWA) access token, which is sent with each request to AVS. If the product lacks a graphical user interface (GUI), also known as a headless product, this is done by integrating a companion app with the LWA Mobile SDK for Android or iOS. The companion app is responsible for obtaining an authorization code and securely transferring it to the product. The product is responsible for using the authorization code to obtain access and refresh tokens from LWA, which are used to make calls to AVS.

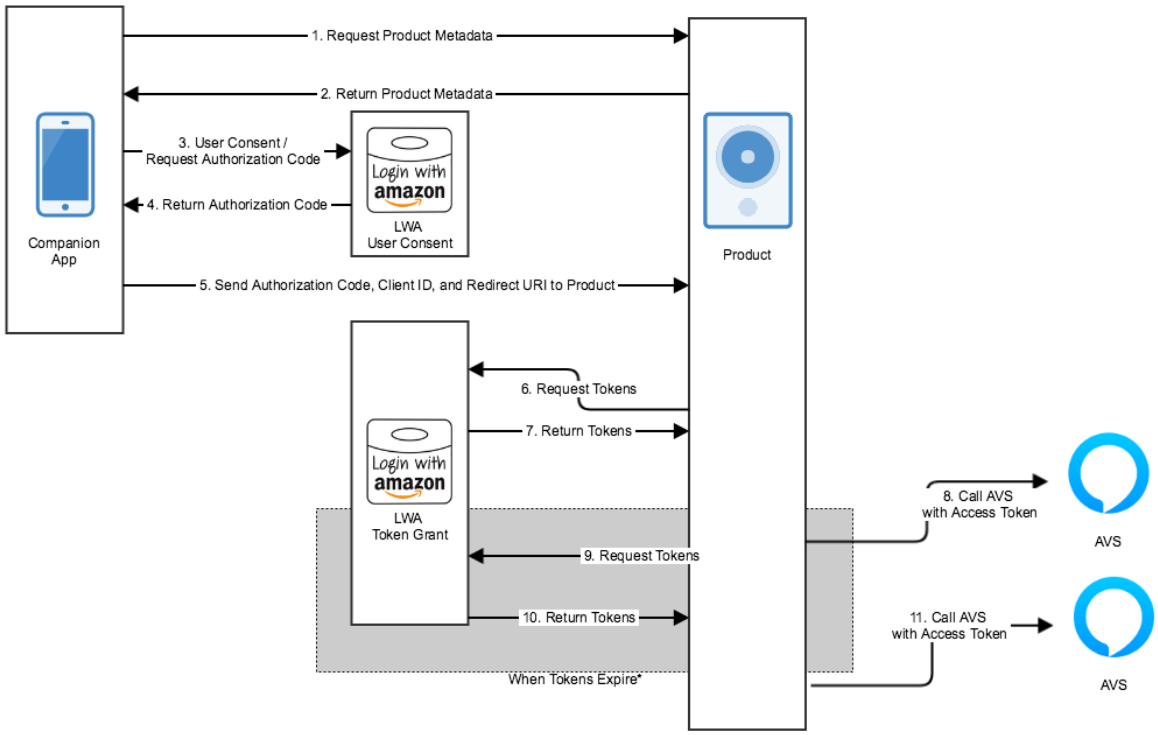

Figure 1 Alexa architecture

2.2. Google Smart Home

Google home work with actions. App Actions let users launch specific features in the app using the Google Assistant. By enabling App Actions to extend the app, users can easily deep link into the apps via the Assistant by simply speaking a request to the Assistant. If the user has the app already installed, the Assistant triggers deep linking when the user says an invocation phrase that includes the app name, such as *"OK Google, order a vanilla marshmallow smore from smores app"*.

Smart home Actions are structured differently than traditional Actions. The process of how users trigger actions and the actions' conversations are handled for them; all they need to do is handle smart home intents on the service.

To enable App Actions, an actions.xml[1] file must be added to the Android app project that tells Google what built-in intents[2] the app supports. For some use cases, it is strongly encouraged to build Android Slices[3] and associate them in the actions.xml file.

[1] https://developers.google.com/assistant/app/action-schema

[2] https://developers.google.com/assistant/app/intents

[3] https://developers.google.com/assistant/app/slices

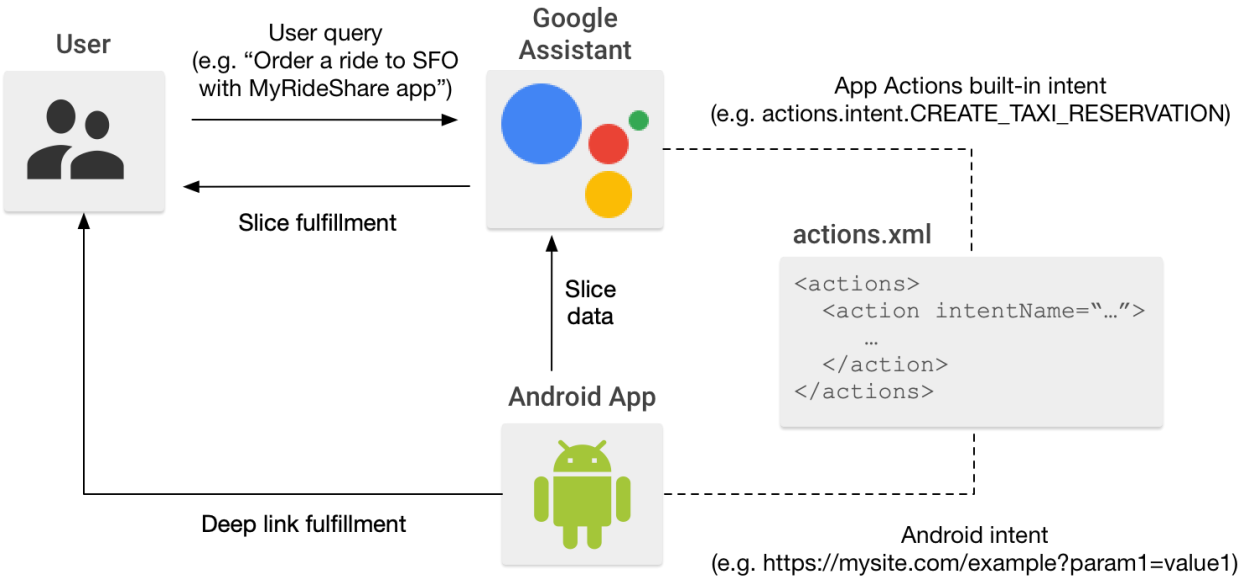

Figure 2 Google Home architecture

2.3. Infrared

In the user’s daily life, there are many appliances that are operated by a remote control, such as TV, air condition units, set-top-boxes, speakers. The most common technology used for these remote controls is infrared. The way this works is that the remote control has an infrared transmitter that converts the actions, digitally coded, into a message that is transmitted in infrared radiation. On the other side, the device has an infrared receiver, which receives the message, decodes it and finally performs the desired action.

Figure 3 IR remote control

In this task, the possibility to use infrared technology with Android smartphones was explored in order to provide an accessible way for the user to control their home devices. In this scenario, the user would be able to control multiple device in the home by using the EasyTV HbbTV Companion Screen application. Moreover, by using the Speech SDK it would become accessible for blind or visually impaired users.

The initial solution was implemented by creating a plugin for the apache Cordova framework that would expose a Javascript API to send IR commands using the phone’s IR transmitter. The Companion Screen application would use this API, after accepting a voice command, to send the appropriate infrared code and perform the desired action.

There are some issues though that make infrared technology less ideal for the needs of people with disabilities. The first limitation is that very few android devices come with an IR transmitter in their hardware. The second and most important limitation is that the IR transmitter (Android

smartphone) must be pointed in the direction of the device to be controlled, this is a very important problem for blind or visually impaired people who cannot know which way to point their device. For these reasons we continued our investigation for a better solution.

2.4. Bluetooth Low Energy (BLE)

2.4.1. Introduction

Bluetooth Low Energy (BLE) started as part of the Bluetooth 4.0 Core Specification. It was previously known as Wibree and it was, initially, planned to design a radio standard with the lowest possible power consumption, specifically optimized for low cost, low bandwidth and low power.

The BLE standard was introduced in 2010 and has seen a rather rapid adoption rate: early adoption of BLE by well-known mobile producers like Apple and Samsung paved the way for wider implementation of BLE.

As previously said, the **Bluetooth SIG** (Special Interest Group) introduced BLE with version 4.0 of the Bluetooth Core Specification. At the moment of writing, the current specification is the 5.1.

It is worth noting that the **classic Bluetooth (BR/EDR)** standard and the **Bluetooth Low Energy (BLE)** standard are not directly compatible. The on-air protocol, the upper layers and the applications are different and incompatible between the two technologies. However, a device manufacturer can choose to develop a dual-mode device, which implements both BR/EDR and BLE: in this way communication is possible with any other Bluetooth device.

2.4.2. Band, modulation schemes and operating range

As stated in [**1**], Bluetooth Low Energy (LE) devices operate in the unlicensed **2.4 GHz ISM** (Industrial Scientific Medical) band and two modulation schemes are defined. The mandatory modulation scheme (“1 Msym(megasymbol)/s modulation”) uses a shaped, binary FM to minimize transceiver complexity. The symbol rate is 1 Msym/s. An optional modulation scheme (“2 Msym/s modulation”) is similar but uses a symbol rate of 2 Msym/s. The LE system uses 40 RF channels in total.

BLE is focused on very **short-range communication** and a typical operating range is closer to 2 to 5 meters. Transmission power is configurable over a certain range (usually between -30 and 0 dBm), but the higher the transmission power, the higher the energy (battery) usage.

2.4.3. Communication mechanisms

- **Broadcasting/observing**: in broadcasting, a broadcaster sends data to multiple observers, and it is the only way for a device to transmit data to more than one peer at time. Broadcasting is fast and easy to use and it is a good choice if used for a small amount of data on a fixed interval. One of the main drawbacks is that it has no security or privacy provisions at all.
- **Connections**: A connection is a permanent and periodical data exchange between two devices. Connections involves two separate roles: a central (master) and a peripheral (slave). The master scans the preset frequencies for advertising packets sent by a slave and, when suitable, initiates a connection. Once the connection is established, the peripheral stops advertising and the two devices begin to exchange data in both directions.

2.4.4. GAP, GATT and data organization

The **GAP (Generic Access Profile)** is responsible for connection functionalities, it handles the access modes and procedures of the device including device discovery, link establishment and

termination, device configuration. On the other hand, the **GATT (Generic Attribute Profile)** defines the way two BLE devices transfer data between them using concepts like *services* and *characteristics*. From a GATT standpoint, when two devices are connected, each of them has one of two roles: GATT server or GATT client.

Data is organized around units called *services* and *characteristics*:

- *services* are used to separate data into logic entities, and contain specific chunks of data called *characteristics*. A service can have one or more characteristics, and each service has a unique 16-bit UUID.

Characteristics are defined attribute types that contain a single logical value. More specifically, they are items of data which relate to a particular internal state of the device or some state of the environment which the device can measure using a sensor.

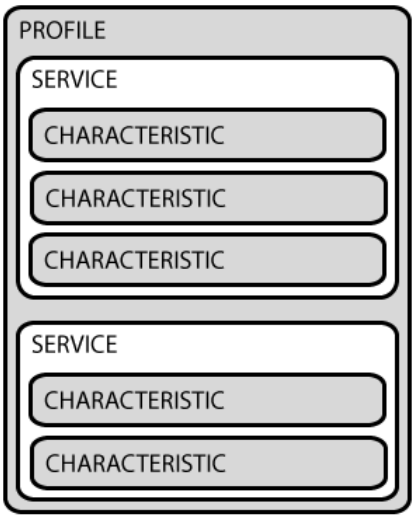

Figure 4 Profile, services and characteristics

2.4.5. **Topology**

As far as topology is concerned, the original design stated that BLE peripheral can only be connected to one central device at a time, while a central device can be connected to multiple peripherals.

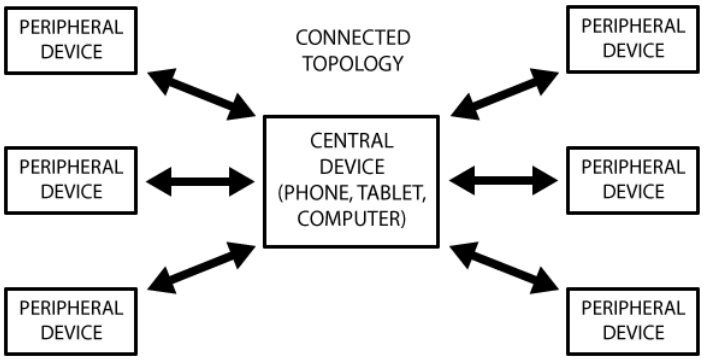

Figure 5 Star topology

Starting with Bluetooth 4.1, a device can act as a "**central device**" and as a "**peripheral device**" at the same time. This particular configuration opens the way to more interesting scenarios, allowing to build large-scale device networks by enabling **many-to-many** device communications. Technically it is referred as **Bluetooth Mesh** [2] and it is ideally suited for building automation,

sensor network, asset tracking and other IoT solutions that require many devices to communicate with one another.

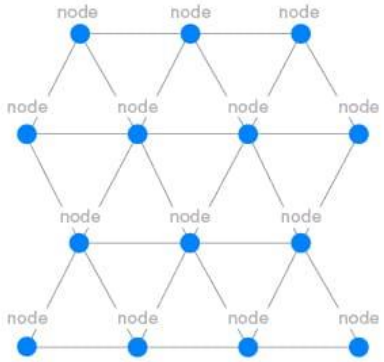

Figure 6 Mesh network topology

2.5. ZigBee

ZigBee is an enhancement to the IEEE 802.15.4 standard [3]. The **ZigBee Alliance** [4] supports and publishes the ZigBee standard and it offers full testing and certification of ZigBee compliant product to ensure interoperability.

The ZigBee specification 1.0 was approved in 2004 and publicly released in 2005. Over the years, the standard has been continuously evolving and, at the moment of writing, the current specification is **ZigBee 3.0**. ZigBee 3.0 is built on **ZigBee PRO** and it is designed to facilitate general wireless networks that are not market-specific.

The IEEE 802.12.4 protocol, on which ZigBee is built, provides radio-based network connectivity operating in one of three possible RF (radiofrequency) bands: 868, 915 or 2400 MHz In the 2.4 GHz ISM band, which is the most commonly used frequency, a data rate up to 250 kbps can be achieved with a total number of 16 channels (2MHz bandwidth each).

ZigBee PRO devices using the Green Power feature [5] are able to complete communication with an average of 100-500 micro Joule of energy. Due to its low-power consumption, communication ranges achievable are typically between 10-100 meters (line-of-sight).

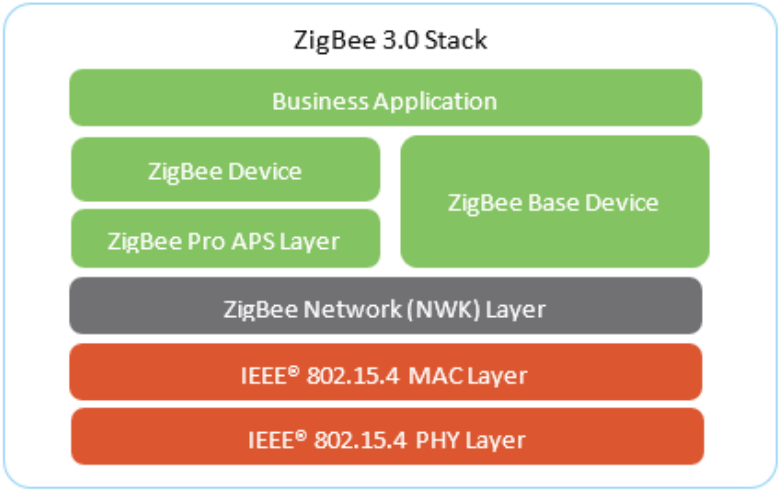

Figure 7 ZigBee 3.0 stack

2.5.1. Network topology and network nodes

The ZigBee PRO standard was designed to facilitate wireless networks with the **mesh topology**.

In a mesh topology, there is a **co-ordinator**, a certain number of **routers** and a certain number of **end devices**. The co-ordinator is associated with a set of routers and end devices (its children), and a router can be associated with more routers and end devices (its children). This kind of association can continue on multiple levels, but note that on ZigBee PRO the maximum number of levels below the co-ordinator is 15.

Three types of nodes can exist in a ZigBee network (**mesh topology**):

- **Co-ordinator**: in a ZigBee mesh network this node must always be present and it should be the only co-ordinator in the network. It is the first node to be started and it provides initial frequency channel selection, network bootstrapping and it also allows child nodes to join the network through it.
- **Router**: the main task of a router node is the relaying of messages from one node to the other.
- **End device**: this kind of node sends and receives messages and can only communicate directly with its parent, so all messages to/from and end device pass through its parent.

2.5.2. ZigBee HA (Home Automation) profile

A **profile** is a domain space of related applications and devices. A profile can be either **public** (specified by the ZigBee Alliance) or **private** (specified by individual OEMs).

Home Automation (HA) is a public application profile which defines a wide range of ZigBee networked devices intended for use in the home, including lights and switches, wall outlets, remotes, thermostats, air-conditioners, and heaters. Another public profile, Commercial Building Automation, defines ZigBee devices such as advanced lights, switches, keyless entry and security systems.

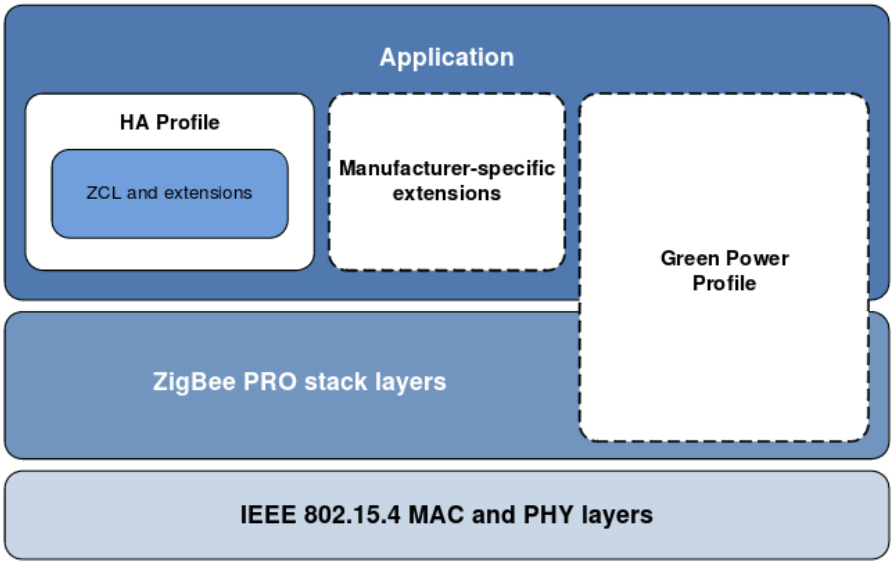

Figure 8 ZigBee HA software stack

The ZigBee HA profile defines the following specification for home automation [**6**]:

- *HA clusters*: specifically, clusters are group of commands and attributes that define what a device can do, like a group of actions by function. A device can support multiple clusters to do a variety of tasks. The majority of clusters are defined by the ZigBee Alliance and listed in the **ZigBee Cluster Library** [**7**]. There are also profile specific clusters that are defined by their own ZigBee profile like **Home Automation** or **ZigBee Smart Energy**, and

Manufacture Specific clusters that are defined by the manufacture of the device. These are typically used when no existing cluster can be used for a device. Furthermore, HA clusters can be classified based on their function: general clusters, measuring and sensing clusters, HVAC, security and safety.

- *HA networks requirements*: requirements and recommendations for an HA network such as device polling rate, channels and so on.
- *Device descriptions (function, supported cluster)*: devices used in HA are categorized as generic, lighting, closures, HVAC, or intruder alarm systems.
- *Commissioning*: commissioning is a method to configure devices in a ZigBee network. Commissioning is a tool that enables the installer to install devices, check network operation and troubleshoot (this tool can be a laptop, a PDA, or a simple push-button switch on a ZigBee device).

2.6. IoT Messaging protocols: MQTT, AMQP, CoAP

There are different protocols employed in the IoT ecosystem, which operate at different layers of the ISO/OSI model. In this document, the three most common **application layer** protocols suitable for IoT use will be described: **MQTT**, **AMQP** and **CoAP**.

2.6.1. MQTT

MQTT stands for "**MQ (Message Queue) Telemetry Transport"** [8] and it is a **M2M (machine-to-machine)** connectivity protocol, originally developed by IBM in the late 1990's. It is a lightweight publish/subscribe messaging protocol, designed for constrained devices and low-bandwidth networks. The protocol design aims to minimize network bandwidth and device resources requirements while at the same time attempting to ensure reliability.

MQTT version 5.0 [9] and version 3.1.1 [10] are both **OASIS** standards and v3.1.1 has also been ratified by ISO [11].

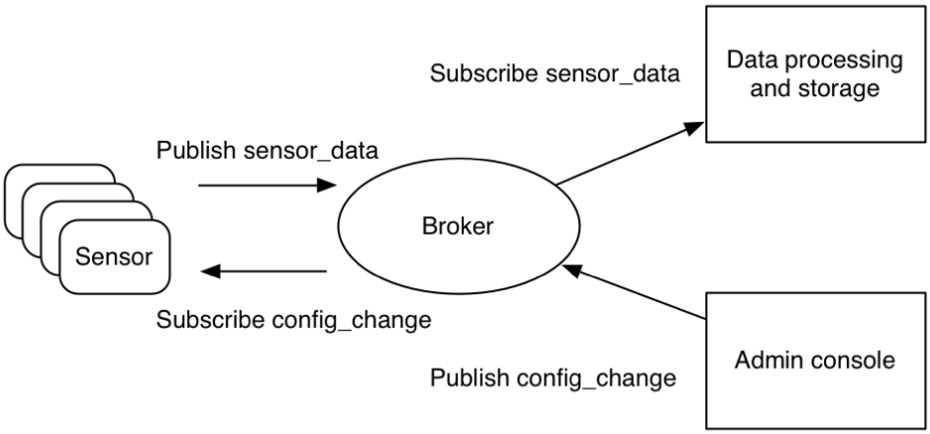

Figure 9 MQTT pub/sub model for IoT sensors

MQTT specifies how data bytes are organized and transmitted over a **TCP/IP** network (assuming TCP/IP is used in the lower layers), but developers do not need to understand the wire protocol. A developer should only know that each message carries a **command** and a **data payload** (optional). Some common message types are:

- CONNECT – used to **establish** a connection to the broker
- CONNACK – connection **acknowledgement** message from the broker
- SUBSCRIBE – sent to the broker and used to receive messages related to a specific **topic**
- SUBACK – subscribe **acknowledgment** message from the broker
- UNSUBSCRIBE – unsubscribe from a specific **topic**
- PUBLISH – used to **publish** a message to the broker

Many MQTT broker implementations exist. Some of the most popular are **Mosquitto**[4], **HiveMQ**[5], **Paho MQTT**[6] and **Mosca**[7], just to name a few. As far as client libraries are concerned, every major programming language / platform is covered since at least one library implementation exists for each language (i.e. **Eclipse Paho Java Client**[8] for Java, **MQTTnet**[9] for Microsoft .NET).

2.6.2. AMQP

AMQP (Advanced Message Queuing Protocol) is defined as an open Internet protocol for business messaging and it has been approved (version 1.0) as an International Standard [**12**]. It has also been approved as an OASIS standard [**13**]. Previous versions of AMQP were 0-8, 0-9 and 0-9-1 (November 2008): these earlier releases differ significantly from the current 1.0 specification, and can be considered as completely independent versions.

AMQP is comprised of several layers. The lowest level defines an efficient, binary, peer-to-peer protocol for transporting messages between two processes over a network. Above this, the messaging layer defines an abstract message format, with concrete standard encoding. Every compliant AMQP process must be able to send and receive messages in this standard encoding [**14**].

Currently, AMQP version **0-9-1** is often used over the 1.0 version because it is better supported through existing software applications (e.g. **RabbitMQ** message broker, **Apache QPid** message broker).

AMQP 0-9-1 defines both the network protocol and the server-side services through [**15**]:

- A defined set of messaging capabilities called the "*Advanced Message Queuing Protocol Model*" (AMQ model).
- A network wire-level protocol

The **AMQ model** specifies a set of components and standard rules for connecting these components together. There are three main component types connected into processing chains in the server to create the desired functionality:

- The **exchange** receives messages from publisher applications and routes these to "message queues" based on a specific criteria.
- The **message** queue stores messages until they can be consumed by a client application or multiple applications.
- The **binding** defines the relationship between a message queue and an exchange and provides the message routing criteria.

[4] https://mosquitto.org/

[5] https://www.hivemq.com/

[6] https://www.eclipse.org/paho/

[7] https://github.com/mcollina/mosca

[8] https://www.eclipse.org/paho/clients/java/

[9] https://github.com/chkr1011/MQTTnet

By focusing on **AMQP 1.0** instead, the main difference from the previous versions is that version 1.0 focuses on the messaging layer only, thus all attempts to define a broker model have been removed. All notions related to broker management (e.g. route configuration) will be specific to the particular AMQP broker that will be used.

For instance, the "**exchange**" concept has been removed: producers and consumers (called "**links**") interact respectively with targets and sources. Routing key concept too has been removed because of the "address" concept introduction.

Unlike MQTT, AMQP can guarantee complete transactions, which can be useful for some particular IoT applications. However, due to its heaviness, AMQP is not always suitable for sensor devices with limited memory, power and network bandwidth.

One of the most known AMQP Pre-1.0 broker is **RabbitMQ**, which could also support AMQP 1.0 through an experimental plugin. Native AMQP 1.0 support is offered instead by **Apache Qpid**[10] and **Apache ActiveMQ**[11]. A certain number of client-side libraries also exist: RabbitMQ offers a range of libraries for all the major programming languages. Microsoft in particular offers a wide support for AMQP through AmqpNetLite[12], AzureAMQP[13] and uAMQP[14].

2.6.3. CoAP

CoAP stands for "*Constrained Application Protocol*" and as the name suggests it is a specialized protocol for use with **constrained nodes** and **constrained networks** in an IoT environment. The standard has been defined by the IETF and the core of the protocol in specified in RFC 7252 [**16**]. As stated in the RFC, one of the main goals of CoAP is to design a **generic web protocol** for the special requirements of this constrained environment, especially considering energy, building automation, and other **machine-to-machine (M2M)** applications.

The standard realizes **a subset of REST** common with HTTP but optimized for M2M applications (as such, M2M features like multicast support, discovery and asynchronous message exchanges are offered). A client/server interaction model similar to the HTTP model is employed and request/response semantics are carried in CoAP messages, which include either a method code or a response code.

[10] https://qpid.apache.org/

[11] https://activemq.apache.org/

[12] https://github.com/Azure/amqpnetlite

[13] https://github.com/Azure/azure-amqp

[14] https://github.com/azure/azure-uamqp-c

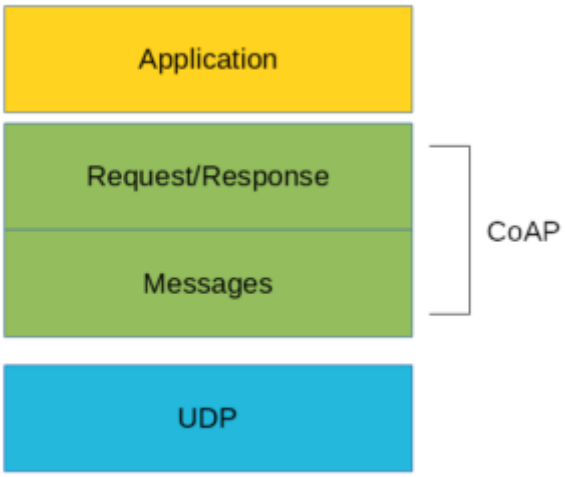

Figure 10 - Abstract layering of CoAP[15]

As seen in the figure, it is possible to consider CoAP as using a **two-layer approach**, one to deal with UDP and the asynchronous nature of the interactions, and the other one to deal with the request/response interactions. Note however that CoAP is a **single protocol**, with messaging and request/response as just features of the CoAP header.

Since CoAP is bound to unreliable transport protocols like UDP, messages may arrive out of order, duplicated or missing without notice: for this reason, a lightweight reliability mechanism is implemented, without re-creating the full feature set of a transport protocol like TCP.

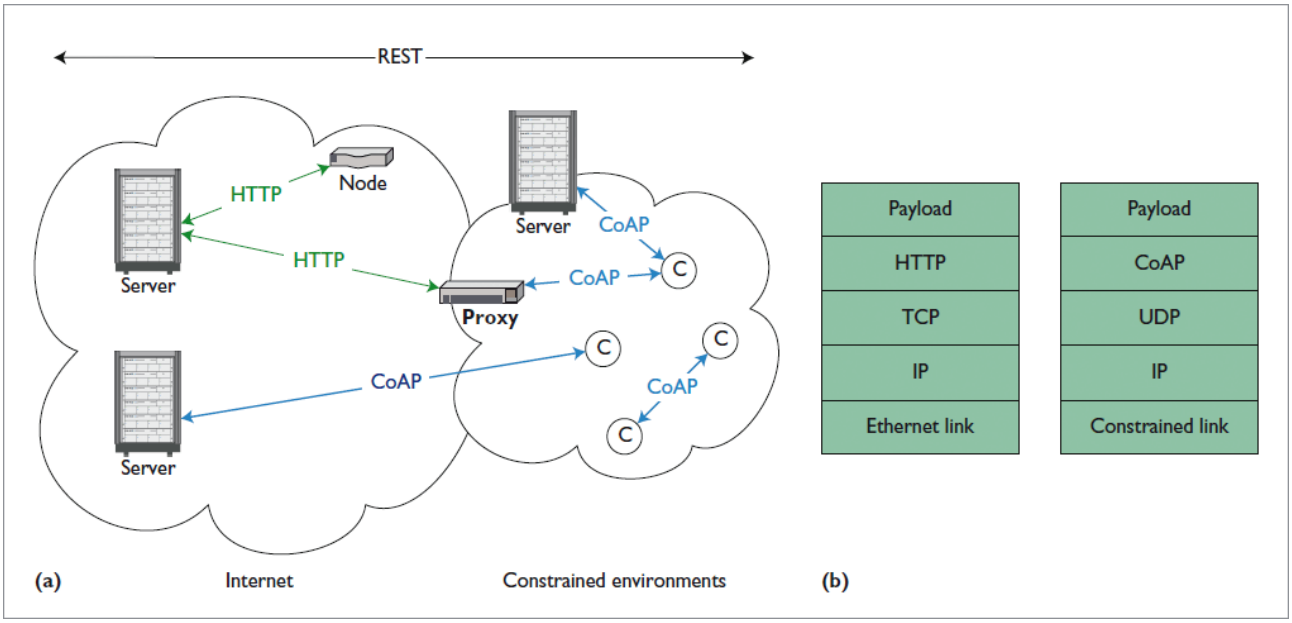

Figure 11 - Internet vs constrained environment[16]

CoAP is also simple enough to be implemented from scratch for a simple application, but generic

[15] https://www.survivingwithandroid.com/coap-protocol-tutorial/

[16] https://www.computer.org/csdl/magazine/ic/2012/02/mic2012020062/13rRUx0xPxs

implementations are also available. It is noteworthy to mention at least **libcoap** [17], which is a C implementation of CoAP that can be used on constrained devices and on larger POSIX systems, and **Eclipse Californium[18],** which is a full CoAP framework targeting back-end services communicating with smaller IoT devices.

2.7. Philips Hue

2.7.1. Introduction

Philips Hue is a line of, color changing, LED lamps[19] and white bulbs, which can be controlled wirelessly. It was introduced in October 2012 and was updated in 2015 and 2016. The lamps are currently created and manufactured by Signify N.V. Philips Hue is based on ZigBee, a low-power and reliable technology to control the lights. New features and improvements are continuously added to the system

2.7.2. Philips Bridge

The Bridge is the heart of the Philips Hue system that connects the smart device to the Philips Hue lights. Up to 50 Philips Hue lights and accessories can be added to one Bridge. Linked to Wi-Fi via router, it, also, connects the system to the wider world via the Internet for out-of-home control and other smart features. Used to enable the smart bulbs to communicate with each other and the Portal via the Internet. The main set of smart lighting APIs are those offered by the bridge. These allow control of all the settings of the lights in the system. These smart lighting APIs require direct access to the bridge so they will only be accessible when the app and bridge are on the same local network.

2.7.3. Remote Connection

This is a way to gain access to the bridge without being on the local network. It can be used for control commands or for checking the status.

[17] https://libcoap.net/

[18] https://www.eclipse.org/californium/

[19] https://en.wikipedia.org/wiki/LED_lamp

3. Integration with HbbTV Companion Screen Application

3.1. Overview

Figure 12 Interoperability between the EasyTV Companion Screen application and philips hue via philips hub.

The EasyTV HbbTV Companion Screen application can integrate with all Philips hue devices that are connected to a Phlips Bridge. The Companion Screen application and the bridge must be connected in the same network. A working Internet connection is not necessary. The companion screen application communicates with the Philips bridge though a REST API that is hosted on the bridge. This API offers endpoints that return a list of available devices in the home network and also offers endpoints to change the status (on/off, brightness) of a specific device.

3.2. Philips setup

After the Philips bridge and the companion screen is connected on the same network, the user can complete the integration through the companion screen user interface. The first step is to open the devices view from companion screen application and then the user will see a view like in **Figure 13**. When the companion screen application is opened for the first time, an authentication with the bridge is required. This can be done by pressing the button of hub and then the “create user” button.

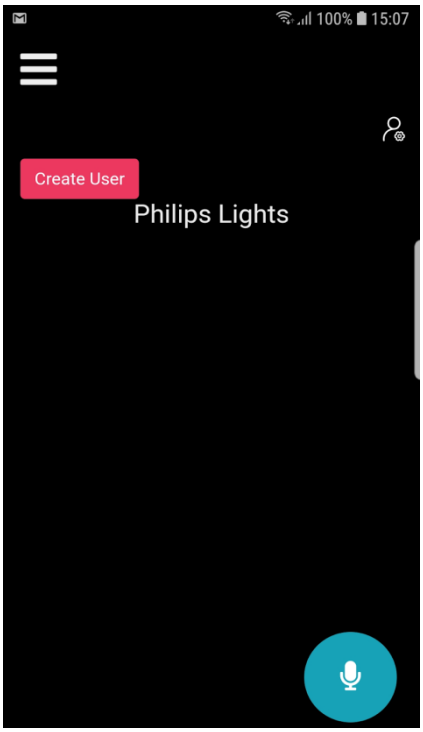

Figure 13 Device View

After this step is finished, the companion screen is ready to discover lights in the user's home. This can be done with the "Discover lights" button. It is an action that will fetch all the smart light bulbs that are compatible and connected to the Philips bridge. After this action is invoked, the devices that were found will be presented as in figure 14.

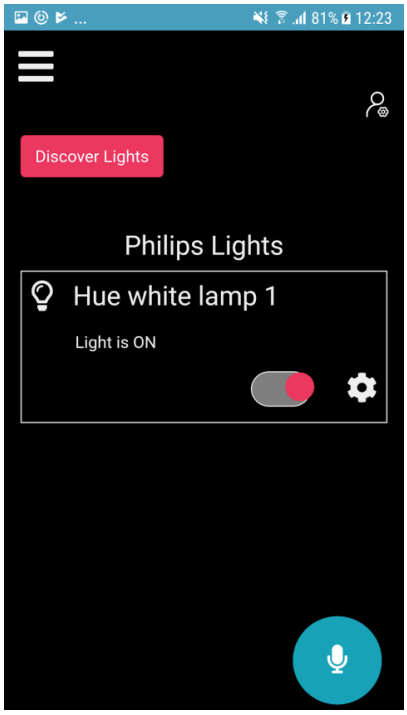

Figure 14 Discovered lights

To change the stage of the light, the user can click on the ON/OFF toggle. Moreover, he can change the brightness of the light bulb. These actions can also be performed with voice commands.

4. Conclusion

The Internet of Things is growing everyday and the interaction with smart home devices via network is very useful. A device can be handled even when not at home e.g. if someone forgets to turn off the lights, he/she can close them remotely. As the technology for smart homes continues to evolve, the range of capabilities is only going to grow. However, various factors like security, privacy and data storage, also, need to be considered.

5. References

[1] Bluetooth SIG, *Bluetooth Core Specification v5.1*., 2019.

[2] Bluetooth SIG. [Online]. https://www.bluetooth.com/bluetooth-technology/topology-options/le-mesh/mesh-faq/

[3] IEEE. [Online]. https://standards.ieee.org/standard/802_15_4-2003.html

[4] ZigBee Alliance. [Online]. https://zigbee.org/

[5] ZigBee Alliance. [Online]. https://zigbee.org/zigbee-for-developers/greenpower/

[6] Adam Gschwender and Ata Elahi, ZigBee Wireless Sensor and Control Network.: Prentice Hall, 2009.

[7] ZigBee Alliance, ZigBee Cluster Library Specification., 2016.

[8] MQTT FAQ. [Online]. http://mqtt.org/faq

[9] OASIS. MQTT Version 5.0. [Online]. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

[10] OASIS. MQTT Version 3.1.1. [Online]. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[11] ISO. MQTT Version 3.1.1. [Online]. https://www.iso.org/standard/69466.html

[12] ISO. ISO/IEC 19464:2014. [Online]. https://www.iso.org/standard/64955.html

[13] OASIS. AMQP Core Messaging v1.0. [Online]. http://docs.oasis-open.org/amqp/core/v1.0/amqp-core-messaging-v1.0.html

[14] OASIS. AMQP Core Complete v1.0. [Online]. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf

[15] RabbitMQ. AMQP 0-9-1 specs. [Online]. https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf

[16] IETF. (2014) RFC 7252. [Online]. https://tools.ietf.org/html/rfc7252