NTNU
Norwegian University of
Science and Technology

Assessing interoperability in Internet of Things ecosystems

Lars Bendik Dølvik

Applied Computer Science
Submission date: June 2017
Supervisor: Rune Hjelsvold, IDI

Norwegian University of Science and Technology
Department of Computer Science

Preface

This is a Master's thesis conducted at NTNU. This thesis was completed during the autumn semester of 2017. The idea and motivation behind this thesis were discovered through the "Advanced Course in Web Technology" class. This course first introduced the Internet of Things to the author. The thought of a unified Internet of Things ecosystem, compatible with all connected devices and services was very interesting. Therefore, it was chosen to pursue this achievement and its challenges.

This thesis targets readers with knowledge in information technology as well as stakeholders in the Internet of Things Industry.

01-06-2017

Acknowledgment

I would like to thank everyone who has supported and helped me, not only through this thesis but also through the length of my studies. First, I would like to thank my supervisor Prof. Rune Hjelsvold for sharing his experiences and giving valuable input and guidance throughout the whole thesis. I would also like to thank all of my current and former classmate for all the support and help during my studies.

At last, I want to thank my friends and family for all their unconditional support.

L.D.

Abstract

The Internet of Things has been a familiar term for quite some time. Connecting devices to track, monitor, enhance, augment and interact with the real world have opened doors for new services which previously was not possible. This has lead the industry stakeholders to see the current and future potential of the industry. This has resulted in a rapid growth of connected devices and application services. These devices are often connected to an Internet of Things ecosystem. An ecosystem allows devices to interact and exist within the same environment, being controlled by an application. The service providers often either choose to provide their own ecosystem, to ensure that all their products can interoperate within the environment, or support an existing ecosystem. This has resulted in an industry with devices and applications only being compatible with specific ecosystems. Because of this, the Internet of Things has an interoperability problem.

This thesis has assessed the interoperability of three ecosystems provided by industry leading corporations and foundations. The goal of this assessment was too determined whether or not the ecosystems provided end-to-end interoperability, to compare strengths and weaknesses, and how the technological and architectural foundation of the ecosystem affect its stakeholders. The assessment has been conducted according to the Criteria-based Assessment Model. With this model, the ecosystems have been assessed and compared in detail. The information and data assessed was obtained through both a qualitative study of the ecosystems documentation and development experiments.

The results show that the all ecosystems provides end-to-end interoperability, and that they do it in a similar way. Even though they are built with similar architectures they support different operating systems and transports. The technological and architectural differences have proved to restrict integration of devices and services, denying potential interoperability. These differences also affect the stakeholders and restrict them to only provide devices and applications according to the ecosystems supported technologies. This further shows that the ecosystems give the stakeholders a lot of responsibility when building an interoperable system.

Contents

List of Figures

List of Tables

1 Introduction

The Internet of Things, shortened IoT, is a very popular term in the technology industry. As the technology has progressed, the advantages of being able to connect digital and physical objects to the Internet has shown an amazing potential. These inter-networks of connected smart devices can be anything from small sensors or actuators to smartphones, vehicles, buildings and other objects with network connectivity. However, simply defining the term IoT can be a very difficult task as it can be used in many situations. This means that the Internet of Things definition can be formed based on the perspective taken. However, the general definition by Gubbi et al. it defines it as:

> "Interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications. This is achieved by seamless ubiquitous sensing, data analytics and information representation with Cloud computing as the unifying framework."[1]

The goal of these objects is to collect information about its surroundings, often together with other objects to reach a common goal. The Internet of Things has proven its relevance through a wide range of applications, services and frameworks, making it possible to track, monitor, enhance, augment and interact with the real world.[2]

An example of an IoT system working together towards a common goal is what it is known as a "Smart Environment". A good example of a Smart Environment is what is known as a "Smart Home". A Smart Home typically consist of multiple subsystems of objects to simplify or solve needs of the consumer. The objects or things works as either a sensor or an actuator. A sensor will track, monitor and obtain information in the context of its functionality. These systems can then store and process the data before displaying it for the user. Actuators acts on appliances in the home, it is through the actuators the system and the user is able to interact with the home. The goal of these systems is often simply to improve the quality of life of the inhabitants.[3] On Figure 1, it is illustrated a typical setup of a smart environment.

However, these systems can be taken advantages of in other important sections of the industry such as health, medicine, education and Wireless Body Area Networks. Because of this, in addition to quality of life improvement, the development and improvement of the IoT is crucial[4].

1.1 Problem Area

As the progression of the IoT has kept developing, the stakeholders has begun to discover its true potential. A result of this is the rapid growth of connected devices. These devices are often connected to an IoT ecosystem. While some device manufacturers and service providers provide their own ecosystem to ensure full interoperability among their products, others chose to integrate their devices into existing systems. As a consequence of this, the systems are built on different platforms and technologies, with devices communicating over different protocols and standards, making it almost impossible for these systems to cooperate. Because of this common silo system design, a Smart Environment would need to implement multiple different IoT systems and services to meet a user's many needs.

This is in fact very similar to what happened to the Internet when it first came. There were no single universal application layer protocol for networks to interoperate, so they could not exchange information. And instead of building a new protocol it was decided to use one that already existed, the Web. By using the Web as a platform it was possible to make versatile, scalable and interactive applications with well-known and well-developed web tools, techniques and languages.[5] The Web saved the Internet in many ways, and now the IoT needs that same type of interoperable solution to make it interconnected and save it.

All though building universal IoT ecosystem can solve a lot of the challenges faced with the homogeneous architecture there are still a lot of challenges when building an interoperable and dynamic IoT ecosystem. Building an IoT ecosystem means that there will be a lot of architectural and technological decision that will affect the ecosystem and the system stakeholders. Alongside the architectural and technical challenges, these systems need features and mechanisms to handle devices, data and services, among others. Further, as the goal is to make these systems interoperable among both devices and services and to deal with discovery of devices, how these systems are developed, in terms of specifications and supported technologies, are very crucial for their existence.

Therefore, the main focus of this thesis is to assess how these IoT ecosystems are built. Discover and present differences and weaknesses in terms of architectural and technological choices affecting interoperability. Then, by assessing the systems internally through research and experiments it will be possible to understand how these choices will affect the ecosystem itself, the developers and stakeholders of the systems.

In this thesis, the ecosystems that have been assessed is Open Connection Consortiums IoTivity, Googles Weave and Apple's HomeKit. These are recognized as the leading IoT ecosystems to date, allowing the results to be as contributing and industry relevant as possible[6]. These ecosystems have been assessed according to

the software evaluation model known as Criteria-based assessment Model, where the criteria assessed is interoperability. This assessment shows how these ecosystems deal with interoperability and allow for a layer by layer comparison of the ecosystems.

1.2 Internet of Things ecosystems

As illustrated in Figure 1 it is important to recognize the data flow in the system and how it all runs through the gateway. At the bottom level we have the objects, appliances, sensors and actuators. These object uses different connectivity protocols like Bluetooth and Zigbee to communicate with the gateway. The gateway, also know as Smart Gateway, is the brains of the system. A gateway is typically a hub at the location of the system allowing users to remotely interact with their ecosystem. The gateway is the middleware in the systems, allowing devices to communicate over different protocols and can perform actions on the data received from the devices before passing it further to a cloud and an end-user application. This allows constrained devices that only support light weight transports such as Bluetooth Low Energy and ZigBee also to be remotely controlled and be a part of the same ecosystem as devices with network connection through WiFi or Ethernet. These gateways normally run a lightweight Web Server with an API which is what allows the consumer to interact with devices through an application. Even if most of these ecosystems have the same goal they are built with different technologies and architectures. However, their most common features are handling device discovery and management, data transmission and management, and at last security.[7]

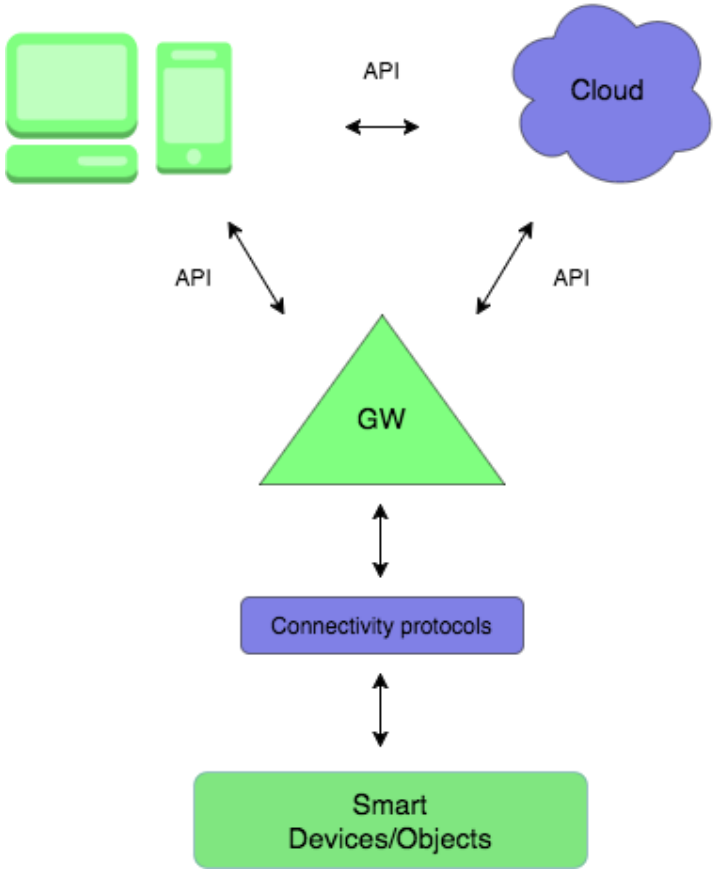


Figure 1: A typical smart environment setup, where one can see the data flowing from the devices to the gateway, shortened GW, through connectivity protocols and further to the client or a cloud through an API, such as a RESTful API.

1.3 Stakeholders

There are multiple stakeholders involved in IoT ecosystems. First, there are the devices manufacturers. Their main goal is to integrate their devices into ecosystems either alongside other of their products or together with devices from other manufacturers. Further, there is application developers and service providers. Their goal is to provide third-party applications and services to interact with the ecosystem and the devices in it. At last, there is the consumer or the end-user. The main goal of the end-user is to install a smart environment compatible with the devices providing the needed functionality and a fitting application to control it.

1.4 Research Questions

1.4.1 1. Research Question

How does acknowledged IoT ecosystems provide and ensure interoperability?

Because of the many different standards and specifications in the IoT, wide, end-to-end, interoperability can be a tall order. Therefore, this thesis will be assessing IoT ecosystems trying to solve this challenge to enlighten their current state, compare them, and in what grade they achieve their goal.

1.4.2 2. Research Question

In what way does connectivity affect the interoperability in IoT ecosystems?

Connectivity is a big part of the IoT ecosystems. Devices and objects today are integrated with many different technologies and support for different transport protocols. Which could affect their interoperability depending on the IoT ecosystems transport protocol support.

1.4.3 3. Research Question

In what way does the architectural and technological foundation of an IoT ecosystem affect its stakeholders?

Ecosystems involve many stakeholders. Which means that how these are built, their features and what technology they support, affect device and appliance manufacturers, services providers, application developers and the end-user. Therefore it is important to evaluate what consequences the ecosystems characteristics has for the respective stakeholders.

1.5 Project motivation

This thesis will present these ecosystems underlying architecture and technologies alongside their main features. Further, the goal will be to assess them and determine in what grade they offer end-to-end interoperability or if it would be possible to achieve it. As a result of this, it will be easier for stakeholders to choose one or more ecosystems they would like to support and/or interact with, depending on their perspective. Interoperability, together with security and connectivity, has been recognized as the three biggest concerns by the IoT industry as of early 2017[8]. Therefore, this thesis will be useful for current and potential stakeholders in the IoT industry.

1.6 Keywords

Internet of Things - Things - Middleware - Sensor - Actuator - Devices - Services - Application - Discovery - Interoperability - Ecosystems

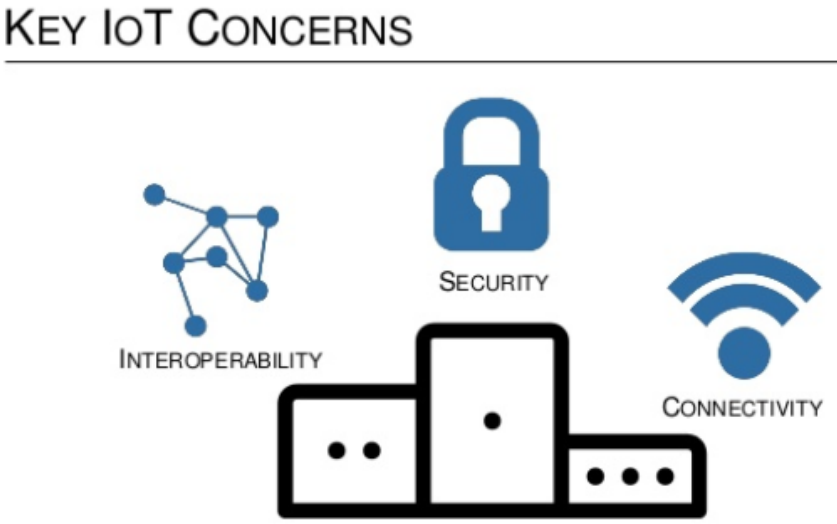


Figure 2: Top 3 key concerns in IoT[8]

1.7 Ethical and legal considerations

During this project, there has not been any specific focus towards security and privacy. This stand for both technical and data-related security and privacy. Although both security and privacy are recognized as important parts of these systems they are not within the scope of the thesis and will therefore not be discussed. As for research literature, specifications, APIs and other documentation, there will be a clear explanation between this thesis' contribution and previous work, previous work will be credited accordingly. The sensor data generated as a part of the development process was generated by simulated sensors. This means that there is no personalized or sensitive information used as a part of the experiments.

1.8 Thesis structure

This thesis is structured accordingly:

Chapter 1 - Introduction The first chapter is the Introduction 1. The Introduction first introduces the field of the thesis before presenting the scope of the research, problem area and the research questions.

Chapter 2 - Background and related work In the Background and related work chapter 2 all relevant information is presented in a structured fashion. This includes more specified information to better understand the research questions as well as information about the ecosystems presented.

Chapter 3 - Methodology The Methodology chapter 3 provides an overview of the assessment model and characteristics used to evaluate the ecosystems. As well as a justification of the sample size and the research and development design. The methodology also includes a data analysis plan.

Chapter 4 - Experiments The Experiments chapter 4 provides an overview of the experiments conducted. Each experiment is explained in detail, in terms of both the implementation and the goal of the experiment.

Chapter 5 - Results The Results chapter 5 present the results of the assessment in a structured manner. The Results chapter also provides a summary table to provide an overview of the results.

Chapter 6 - Discussion The Discussion chapter 6 is split into four section. A discussion and interpretation of the results, validation of the research design, the thesis' limitations and further work.

Chapter 7 - Conclusion The Conclusion chapter 7 contains defendable conclusions addressing the research questions based on the results obtained from the research, development, and assessment.

2 Background and related work

2.1 Background

Even though the term Internet of Things has been gaining its popularity mostly in last decade. The action of connecting devices and objects to the Internet has been popular for quite some time already. The first application of this kind was born back in 1989, and popularly known as the Trojan Room coffee pot. This is where Dr. Quentin Stafford-Fraser and Paul Jardetzky invented the world's first Web camera. This application streamed a Web camera with the image of the offices coffee pot to the office network to save employees the disappointment if the coffee pot were to be empty. The first interaction on physical devices over the Internet was done in 1990 by John Romkey. In his project he managed to both turn a toaster on and off over the Internet.[9]

The term the Internet of Things were still not coined before a man with the name Kevin Ashton did so in 1999. When he defined it through uniquely connecting objects with radio-frequency identification (RFID) and make them identifiable and interoperable.[2] The first traces of Web of Things and the use of Web in the world of IoT was in 2002. Tim Kindberg was able to allow users to retrieve Web pages by scanning bar codes and interact with infrared interfaces.[10]

Since then IoT has grown at a rapid pace and now, in 2017, IoT has proven why it was worth all the excitement through its early stages. The continuous improvements of the Internet of Things are very important for the future. It is common to associate the Internet of Things with Home Automation, but it is so much more. As mentioned in Introduction 1 the development is very important to allow multiple fields to take advantage of the technology. Therefore it is very significant that we continue to work with the current drawbacks in the field, allowing it to keep progressing.

The Eclipse IoT Working Group, IEEE IoT, AGILE IoT and IoT Council co-sponsored an online survey in February and March 2017 to better understand how developers are building IoT solutions. The survey had a total of 713 individual participants where the goal was to highlight trends, concerns, and preferred choices in the IoT industry. This survey determined that the top 3 concerns for IoT were interoperability, together with security and connectivity, as shown in Figure 3.[8] This proves to us that interoperability is still a major challenge in Smart Homes[11] and Smart Cities[12] among other directions in the field. As a matter of facts, Cisco has pre-

dicted that by 2020 there will be more that 50 billion connected devices which mean the interoperability challenges needs to be improved[13].

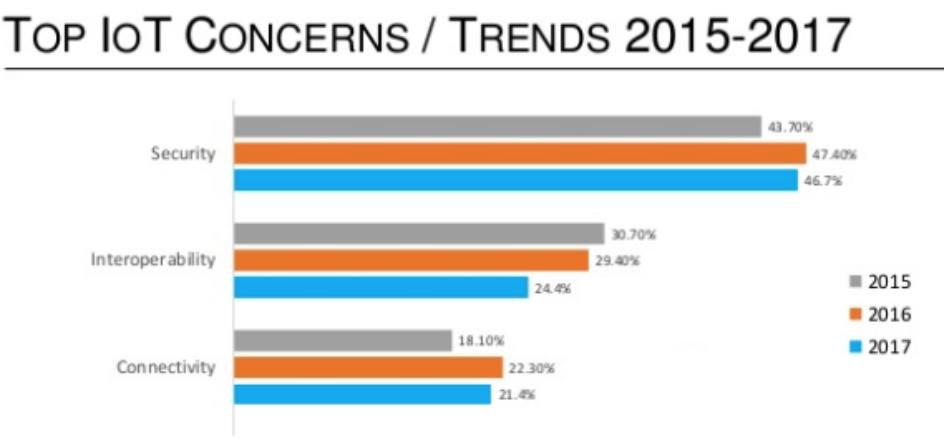


Figure 3: Diagram of the top 3 current concerns in the IoT industry[8].

2.2 Interoperability

The IEEE defines interoperability as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged"[14]. What one can read from this definitions is that interoperability and connectivity are very depended on each other. Meaning that two systems or objects can be interoperable, however, they need connectivity to exchange information. Therefore, there has been found important to focus on connectivity in these ecosystems and how it affect the interoperability.

It is common to consider interoperability only as allowing devices to seamlessly integrate into an existing IoT solution. However, interoperability should be considered across all the layers of the hardware and software stack to provide end-to-end interoperability.

- **The devices layer**
- **The network layer**
- **The middleware layer**
- **The application service layer**
- **The data and semantics layer**

The device layer:

The device layer of an IoT ecosystem is the layer containing all the devices. This means anything from a sensor, light switch to a thermostat. The device layer is the system's connection to the physical world, this layer allows the system to augment,

enhance and interact with the world. These devices are developed by manufacturers, and vendors, and in most cases the manufacturers can choose what IoT ecosystem they would like to support. In an interoperable solution, the device layer may consist of different device types from different device vendors and manufacturers.

The network layer:

The network layer in IoT ecosystems could potentially affect the interoperability of the system. There are many parts of the system that is very dependent on connectivity in terms of protocols and transports. The network layer is responsible for providing a connection between devices, services, and clients in the ecosystem, allowing them to communicate and exchange information. The network is also responsible for both local and remote interaction with the system.

The middleware layer:

The middleware is responsible for applying an interface to the ecosystem. It allows clients and applications to perform actions on devices over different transports using the ecosystems API. This middleware can be a physical hub, known as a gateway, or be accessed in the cloud through Internet connection.

The application service layer:

The application services layer is responsible for the client application. These client applications help the end-user control and interact with the ecosystem. To provide end-to-end interoperability in an ecosystem third-party application should be able to connect to the ecosystem and use its API to interact with the devices in it.

The data and semantics layer:

The data and semantics layer is crucial for providing interoperability in the IoT ecosystems. This layer covers the data formats and data structure in the ecosystems. The data formats and schemas are the structure and architecture of the information sent between the devices and other clients of the systems. This allows for data semantics which means devices and applications in an ecosystem understand the meaning of the information. This could allow devices and applications to act on data with and without human interaction.

2.3 The Open Connectivity Foundation and IoTivity

The Open Connectivity Foundation (OCF), previously knows as the Open Interconnect Consortium (OIC) is an industry group that was created July 2014 by Intel, Broadcom, and Samsung Electronics. Their stated mission is to develop standards and certifications for IoT devices based on the Constrained Applications Protocol (CoAP).[15],[16] In September 2015 they released their first candidate of the specifications known was the OIC specification 1.0 which has four main parts, the core

framework, security, smart home device and resource type[17]. By defining this open specification they wish to provide a secure and reliable device discovery and connectivity across multiple operating systems (OS) and platforms. The OCF is an umbrella organization which currently has more than 300 member companies, including Cisco Systems, General Electric, Intel, LG, Canon, and Samsung. Last year, February 2016, the OCF added Microsoft, Qualcomm, and Electrolux to their member base when Qualcomm signed over The AllSeen Alliance, which is a workgroup similar to OCF, to the Linux Foundation which later merged with the OCF.[7]

The OCF is currently sponsoring an open source project named IoTivity. IoTivity is hosted by the Linux Foundation and is an implementation of the open OCF specification. IoTivity is designed to enable application developers and device manufacturers to offer interoperable devices and services over multiple platforms like Andriod, iOS, Windows, Linux and Tizen, among others. The IoTivity framework is written in C/C++ and comes with an extensive API (Application Programming Interface) documentation making it easy to develop your own solutions as well as contributing to the open source project.[18]

2.3.1 Technology stack

Before going deeper into the framework architecture and the OCF specification there is quite a few important technologies that are put to use which are very central for allowing this framework and this specification to reach its goals.

REST (Representational state transfer):

REST is currently one of the most, if not the most, popular architecture of the web. REST provides a loosely coupled application layer architecture. The key concept of REST is resources which are hosted on a server and identified using a Uniform Resource Identifier (URI). A client can then interact with these resources using the URI and the HTTP methods (GET, PUT, POST and DELETE).[19] An example of its usage would be getting a list of pants from a clothing store by sending a GET request to the pants URI, *http://clothingstore.com/api/cloths/pants*. The server would then respond by returning the list of pants in either JavaScript Object Notation (JSON) or Extensible Markup Language (XML) which is the two most popular data models for transporting and storing data.[20]

Constrained Application Protocol (CoAP):

The Constrained Application Protocol, also known as CoAP, is an optimized version of HTTP (Hypertext Transfer Protocol). HTTP over TCP (Transmission Control Protocol) is not optimal for IoT solutions. HTTP can be very battery and CPU heavy for constrained devices as well as taken up a lot of memory. As CoAP over UDP (User Datagram Protocol) is much simpler, lightweight and has been designed to use minimal resources. It has proven its relevance as a very useful protocol in the

IoT industry. Even with its lightweight design CoAP still, support the four HTTP methods making it very suitable for solutions using the popular web architecture REST (Representational state transfer). However, CoAP and HTTP compatibility exists by using intermediaries that can translate between the two protocols[19]

Discovery:

CoAP also comes with support for multicast. Multicast in computer networking means group communication in terms of one-to-many or many-to-many. This is what is known in the web as resource discovery and it is similar to when a human enters a servers default resource, like *index.html* which often will return links to its related resources and servers. Machines can also perform this type of multicast resource discovery asking for all available and connected servers or devices to introduce themselves.[19]

Observe:

When using HTTP, transactions are also initiated by the client through a GET operation. And the only way to keep up-to-date with the status of a resource would be *polling* which means sending a request, again and again, updating the status. This, as one might imagine, will quickly become very expensive in a resource-constrained environment. CoAP, however, uses a more asynchronous approach for pushing information between a server and a client which is called "Observing". In an initial GET request, the client asks the server to allow it to observe it. If it allows it, the client will receive asynchronous notification message if the server notify changes, returning a message in the same structure as the initial GET request.[19]

The Concise Binary Object Representation (CBOR):

CBOR is defined in the Internet Standards Document, RFC 7049. CBOR is based on the very popular data format, JSON. CBOR uses numbers, strings, arrays and maps, known as objects in JSON, alongside values. CBOR embraces a binary encoding which saves bulk and allows for faster processing. Because of it design CBOR offers the possibility for extremely small code and message sizes, as well as a more conservative CPU-usage. CBOR also defines tags as a mechanism to identify data that is applied beyond the basic data model, this allows specifications, applications, and services that include CBOR to stay extensible. CBOR is also very much compatible with JSON and support all JSON data types and JSON documents.[21]

2.3.2 The OCF standardization and specification

The goal of OCF is to allow the billions of devices out there to communicate. However, this is not an easy task at all. Devices are created by different manufacturers, running different operating systems, with different chipsets and support different transports. By reaching their goal all these different devices will be able to interop-

erate in a single environment.[15]

Their solution is standardization. As mentioned, OCF has created an open specification that will allow for interoperability between enabled devices. Further, by hosting IoTivity as an open source project they allow anyone from big industry partners to home automation consumers to either participate in the development or create their own solutions and compatible devices.[15] It is important to mention that this standardization has two parts. The first part is defining the device through JSON and CBOR data models and how to communicate with them using the RESTful API Modeling Language (RAML). These models can be found at `http://oneiota.org`. This site also allows for crowd-sourcing of data models for new devices and appliances.[22] These resource schemas for all OCF resources can also be found in the Resource Type section of the OCF specification.

The second part of the standardization is the specification itself. The specifications architecture allows for interaction among IoT artifacts, physical devices or applications. The specifications are built according to industry standards, technologies, and provides solutions for connectivity and information flow. The OCF core specification is currently in version 1.1.2.[17]

OCF core specification architecture:

The architecture is built with devices, resources, and operations. The resources are abstracting the entities, where the entity is an element in the real world which is exposed through a device. As seen in Figure 4 the devices are split into two roles. A device can be either a client or a server and a platform can contain multiple devices. Any device can act as a client and act on the device with a server role using the RESTful operations. At the same time, any devices exposing an entity as a resource acts as a server. This means that devices can have the role of both a client and a server. Operations are actions performed on a resource. These RESTful operations are generic create, retrieve, update, delete and notify (CRUDN) operations defined using the RESTful paradigm to model the interactions with a resource. An example of operations on a resource could be client device sending a CoAP or HTTP GET request to a resource with the URI */a/temperature*, see Figure 5 for a graphical overview. The server holding the resource would then return the proper data.[17]

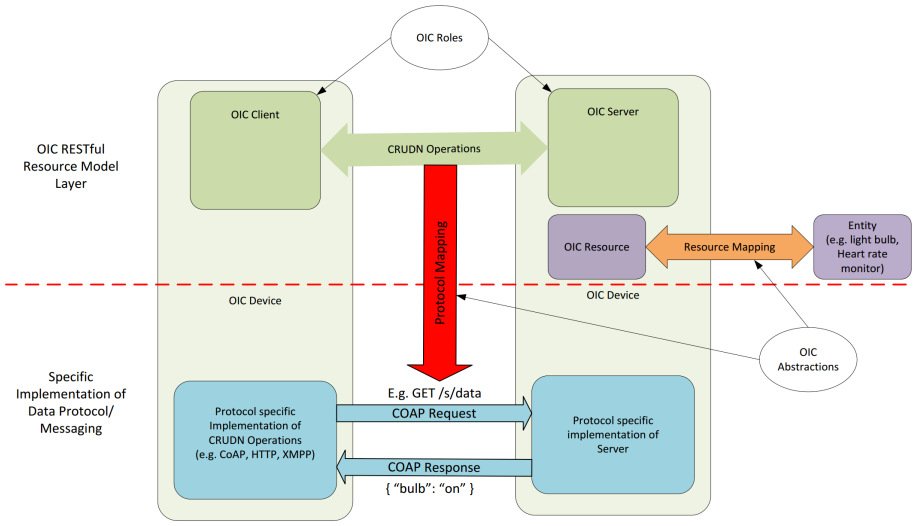


Figure 4: The OCF core specifications conceptual architecture[17].

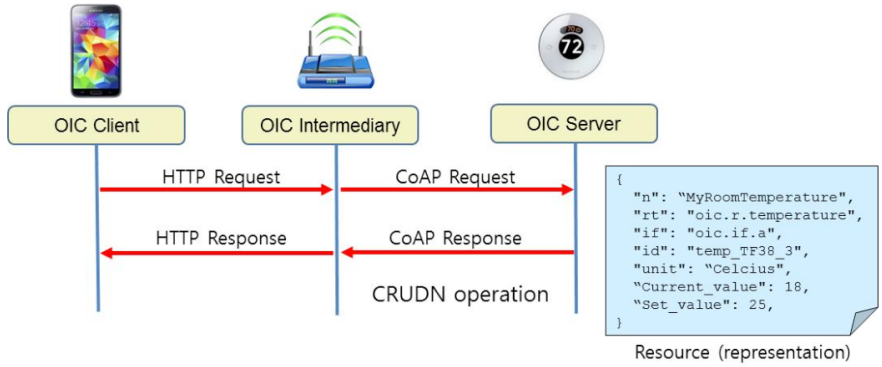


Figure 5: An example of a GET request from a client through a gateway on a resource with the correct response[17].

2.3.3 The resource model

The resource model is a very important concept in the OCF ecosystem. It is because of a strict resource model the OCF ecosystem allows for interoperability between device from a wide variety of manufacturers. The resource model describe how resources are presented in the OCF framework.[17] An example of the resource model in use can be seen in Figure 5.

Resource type:

Through the resource type, it is identified what type of the resource is being handled. This means that if the resource represents, for example, a temperature sensor it would say in the resource type.

Resource interface:

The resource interface declares the interfaces supported by the resource. The resource interface declares if the resource is either a sensor or an actuator.

Resource name:

This represents a human-readable name to allow humans to easy identify the entity exposed as a resource.

Resource identity:

The resource identity, ID, is a unique identification of the resource allowing the framework to interact with a specific resource through the resource identity.

Resource property:

The resource property contains two parts, the *Property Name* and the *Property value*. The resource property defines all the properties that apply to the resource. The property is expressed as a key-value pair where the key is the Property Name and the value is the Property Value. An example of this would be a resource with Property Name "temperature" and Property Value of "14°C". In JSON, this property would be represented as a "key": value.

2.3.4 Discovery

According to the OCF specification discovery is completed by two parts, Device Onboarding, and Resource discovery. These two parts contain mechanics which according to the OCF core specification must be implemented to recognize these devices as IoTivity devices and allow them to join the OCF network.[17]

Onboarding:

Onboarding is the process of integrating a new OCF device into an existing OCF network. During an Onboarding process, the device acquires detailed information and required parameter values, like Service Set Identifier (SSID) for WiFi and authentication credentials, to be able to connect to the network. A successful connection to the network will end the Onboarding. With a specification aiming for high interoperability it would be able to Onboard a wide range of different devices. Therefore, there is no specified process for Onboarding, but rather a specified Devices state is needed to complete the Onboarding. This allows the manufacturers of the devices to handle the Onboarding according to their own preferences and their devices capability. However, a popular solution for connecting devices to a WiFi

network is by the device setting up a temporary Access Point (AP). Then, connect to the AP using another device, such as a smartphone, and enter the local WiFi credential to connect the device to the network.

Resource discovery:

The OCF core specification promotes two main methods for discovery. These two methods are *Direct Discovery* and *Indirect Discovery*. In the OCF core specification, it is specified that all devices shall support direct discovery. The core resources that must be implemented to support discovery are shown in Table 1.

Direct discovery:

In direct discovery, the device providing the information must host and publish the information to enable discovery. A client can then issue a retrieve request either specifically to a resource or perform a multicast using CoAP. Depending on the retrieve request either the specific resource or all resource will respond a CBOR payload, presenting itself to the client making the request.

Indirect discovery:

Indirect discovery allows other devices to host and publish information on behalf of resource constrained devices through what is called a *Resource Directory*. This means that information about the resource to be discovered is hosted on a server that is not holding the actual resource. Indirect discovery makes it possible to discover sleepy nodes and other devices that might not be able to respond to a discovery request without assistance.

Direct Discovery			
URI	**Resource Name**	**Resource Type**	**Response**
/oic/res	Default	oic.wk.res	Returns all available resources in the ecosystem
/oic/p	Platform	oic.wk.p	Returns all available platforms in the ecosystem
/oic/d	Device	oic.wk.d	Returns all available devices in the ecosystem

Table 1: Mandatory implementation of direct discovery according to the OCF core specification.

2.3.5 The IoTivity framework

The IoTivity framework is an implementation of the OCF specification which is an open source project sponsored by OCF, and hosted by the Linux Foundation.

The framework provides a middleware framework functioning on top of different operating system together with different connectivity protocols. IoTivity is also a generic framework, meaning that it is not developed with a specific solution in mind, such as health or education, but rather a solution that can be applied in a wide range of different fields.[7]

The framework architecture:

The framework consists of four main components, also shown in Figure 6.[16]

1. **Discovery:** The discovery component is split into Endpoint discovery and Resource discovery. CoAP based Endpoint discovery is done by multicasting to discovery all OCF devices on the network, this allow two OCF Endpoints to discover each other. Resource based discovery is when you ask either a devices to return its resource or multicast to all devices, asking them to return their resources.

2. **Data transmission:** In the Data transmission component one will find the use of message protocols for RESTful operations between server and client using the CRUDN methods and the resource model.

3. **Data management:** The Data Management component is responsible for the collection, storage and analysis of data obtained from the different resources.

4. **Device management:** Onboarding of new OCF devices alongside device monitoring and diagnostics is done in the Device management component. It also allows for firmware updating devices.

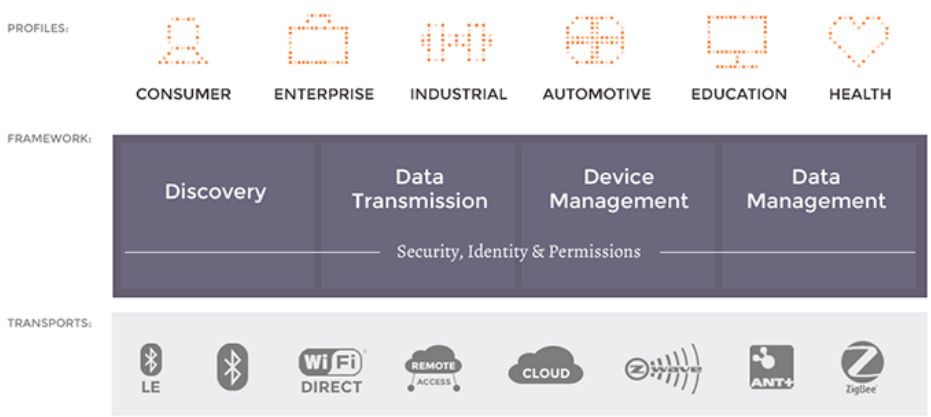


Figure 6: The IoTivity architecture[15].

The IoTivity API and documentation:

The main API is split into core and high level. Where the core API is written C and high level is written in C++. The API functionality is divided into client role and server role, but as mentioned, a device can have the role of both client and server. There also exists API bindings based on the main API available in Java and Javascript (Node.js). Both the API and the framework documentation is very well documented.[23]

Low-level abstraction:

The beauty of this API is that it abstract the lower levels of the system. This means that the developers do not need to figure out how to apply CoAP over a wide range of connectivity protocol as this is already abstracted in the API framework. An example of this could be doing CoAP discovery using multicast. The developers would then use the proper class found in the API documentation to discover all devices on the network. The API will abstract the low-level technologies applying CoAP on top of all supported connectivity protocols, which is called connectivity abstraction, and will return the proper payload to the client, also specified in the API. This also allows non-IP technologies to be discovered as a part of the OCF network. In Figure 6 one can see some of the supported connectivity protocols.

2.4 Google Weave and Android Things

Google is a very important company in terms of the IoT industry. The reason for this is their incredibly big user base and resources. Google is directing the Android software which, according to Gartner, a leading information technology research company, had a smartphone operating system market share of 86.2%, as of the second quarter of 2016[24]. Google also have a strong infrastructure and the needed resources to run big data models and offer cloud services. Google offers two products to the field of IoT, Google Weave, and Android Things. Android Things is a revised version of their old IoT OS, Brillo OS. Google Weave is an applications layer protocol working as a communication platform in a Weave ecosystem. Android Things is and Android-based operating system targeting IoT devices. Android Things offers a Weave compatible OS, designed to run on low amounts of storage, memory, and power, making the OS ideal for IoT devices. These technologies are compatible, which means that they can interoperate without being dependent on each other. As a result of this IoT device does not need to run the Android Things OS to be able to support Google Weave.[7]

Google Weave is a competitor to OCFs IoTivity framework, because the try to achieve the same goal. Weave is a communications platform that includes the open source Weave Devices SDK library and the Weave Server, which allows device man-

ufacturers to connect their devices to the Google Cloud Services. Both the Device SDK and Weave services have an open API documentation guiding device manufacturers and application developers using the platform. Google provides all required work on the back-end to manage the cloud and to allow the devices to interoperate with Googles services, such as Google Assistant.[25]

2.4.1 Weave Device

An IoT device that supports the Weave protocol integrated with the Weave Device SDK is per definition a Weave Device. The Weave device allows support for the Weave protocol by providing a description of the IoT devices features using specific device schemas and traits. To provide maximum portability, this open source library (libiota[26]) is written in the C language and implements the Device API. At this point, Weave supports WiFi as it only transport and all devices must have an internet connection to use the libiota library. However, Google has announced that support for Bluetooth Low Energy is under development. The Weave devices SDK allow devices to become a part of the Weave ecosystem and communicate with the Weave servers. Google promises that the devices support will be expanded in the future, but as of now the Weave device SDK only supports devices of the following kind:

- HVAC (heating, ventilation and air condition) controllers
- Lights
- Outlets
- Televisions
- Wall switches

2.4.2 Weave Server

The Weave server serves multiple important purposes in the Weave ecosystem. First of all, it provides an easy and secure devices registration, state storage and integration to the rest of the Google cloud services. Weave server also allows for remote interaction to all registered Weave devices through and REST API. Google provides for all the needed back-end resources for devices to live in a Weave ecosystem integrating with existing Google services.

2.4.3 Weave Client

A Weave client is a device that can communicate remotely or locally with a Weave Device. Libraries needed for allowing mobile devices to speak the Weave protocol is available for devices running either the iOS or Android OS. This library is also available for the Web, making it possible to develop web-based weave applications.

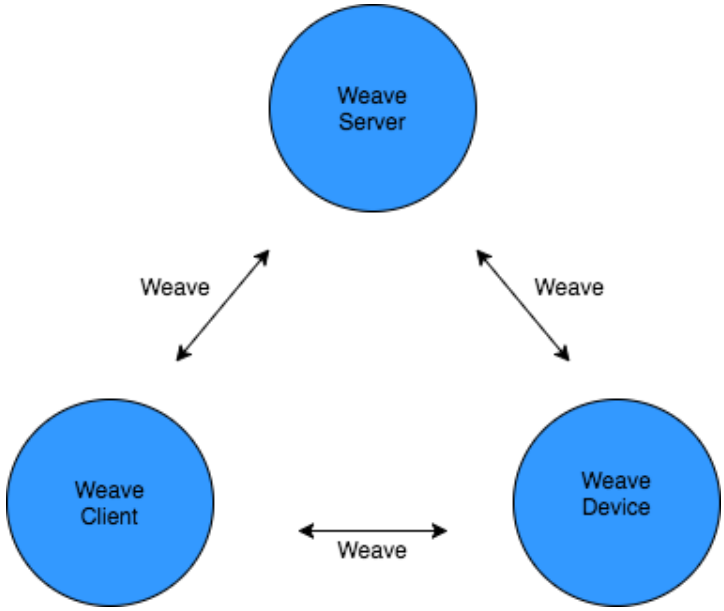


Figure 7: Overview of the Weave communication platform.

2.4.4 Discovery and registration

Seamless mechanics for discovery and registration of new Weave devices to integrate them easily into an IoT ecosystem is very important. Allowing consumers and user to take a device and "out-of-the-box" connect it to their existing ecosystem making it interoperate with the already connected devices and clients. The process of integrating these Weave devices consists of three parts.[25]

- Discovery
- Secure channel
- Registration

Discovery:

To discover a Weave device it needs to be connected to the local network. Google clearly specify that it is the manufacturer of the Weave enabled devices' responsibility to provide a mechanism to provision the network connection. After they are connected to the WiFi or local network the Weave devices can be discovery using either information encoded in a prescribed SSID format for WiFi or through mDNS, a multicast DNS service discovery, on a local network. This devices discovery can happen over one or more transports, it all depends on the technology and hardware supported.

Secure channel:

When the device is discovered it will start setting up a secure channel to the registered client. This channel is kept secure by sending small amounts of information "out-of-band" between the device and the registration client.

Registration:

After the devices are discovered and there has been established a secure channel with the cloud the application responsible for device registration can transport relevant information to the device. This includes cloud registration information. After the device is registered with the Google Weave services it can start receiving Weave commands from clients and applications. This registration must be complete by the user through the provisioning applications provided by the device vendor. When the device is connected to the network and has been registered in the cloud the device integration process is fully completed.

2.4.5 Device schema and traits

The Weave device data model is standardized using schemas. The resources in the Weave communication platform use the JSON data format. A schema is a data model specifying the device type and the device types relevant data structure and properties. As mentioned, the Weave protocol currently only support a few device types. This means there only exists devices schemas for these specific device types. A device schema is built with Components and Traits. Components describe the devices functionality as well as the relation between the traits. Traits define the current state of a component. An example of this would be a light resource with a component "power_switch" and trait "OnOff" The component describe the function and the traits describe its state. For overview of a schema for a light resource see Table 2[27]. To allow for new and innovative functionality and device types in the future these schemas are extensible.[28]

Light Schema			
Component name	**Trait**	**Mandatory**	**Comments**
power_switch	OnOFf	Required	The main power switch for this light
dimmer	Brightness	Optional	An optional dimmer control for this light
color_xy	ColorXy	Optional	An optional color setting using the XY color space
color_temp	ColorTemp	Optional	An optional color setting in degrees Kelvin
color_mode	ColorMode	Optional	If this light resource support both ColorXY and ColorTemp then this Trait will reflect the resources current mode

Table 2: Overview of the standardized schema for a Weave light resource.

2.4.6 Device manufacturers

The manufacturers of Weave devices has to develop their devices according to strict standards and schemas. It is very important that these standards and schemas are followed to allow interoperability among device types from different device manufacturers. This will also create a predictable surface for application developers, allowing them to easy know how to interact with the Weave devices using the API and documentation. To assure this all manufacturers must be certified through "The Weave Certification Program" before they are allowed to commercialize their Weave devices.[25]

Device provision:

Google clearly specify in their documentation that the device manufacturers are responsible for providing mechanics to both connect the device to the network and to the Weave server. There are multiple ways to connect devices to the network. To connect the device to the network there are two popular solutions. Either the device manufacturers provide a mobile application that provisions the devices through Bluetooth Low Energy or other transports. The other solution is that the devices set up a temporary access point when started that the consumer can connect to and provide WiFi credentials allowing the devices to connect to the network. The

manufacturers must also provide a way to both registered and deregistered device on the Weave server to allow the device to receive Weave commands both locally and remotely.

2.4.7 API and documentation

The Weave platform is based on the RESTful architecture communication using a REST API. The API is split into two sections. The Device API for provision and register devices and the Companion REST API providing an API for developers to interact with resources and the Weave services. The API also handles abstractions. These means that the client developers do not have to consider what transport the communicate is exchanged over. Because the API maps Weave on top of all the supported, and future supported transports.[28]

Device API:

A Device API that describes a REST API to communicate with the devices rather then the resources. The API is an implementation of the open source device library libiota. This API works together with the Weave device SDK providing a way for device manufacturers to integrate Weave devices in the ecosystem as well as communication with the Weave server. This means that this API is responsible for registration and deregistration of devices on the Weave server. This API also handles mechanisms when a device has been rebooted or need a firmware update. This API currently contain three methods, CLAIM, HELLO, and PATCHSTATE. CLAIM handles the provision and registration of a device. HELLO handles initial devices states and rebooting of the device. PATCHSTATE provides mechanisms for patching, device uptime, and device updates.[28]

Companion REST API:

The Weave Companion REST API that describes the REST API to develop application and clients to interact with devices and resources. This REST API provides a communication channel between the Weave services and the client. According to Googles Weave documentation the Companion REST API, at this time, support four Weave methods: GET, DELETE, LIST and PROVISION.[28]

GET:

The GET method executes an HTTP GET requests to Weave services, taking a device name as a parameter. The response would be a JSON payload containing all the information related to the devices, such as name, device description, serial number, firmware version as well as all its traits, the device type, and the device connection status. An example of this would be a GET request to an URL (Uniform Resource Locator), such as,
*https://weavecompanion.googleapis.com/v1/name=devices/** responding with the

JSON payload as shown in Figure 8.

LIST:

The LIST method is also an HTTP GET requests that are similar to the GET method. However, the LIST method does not require a parameter as it returns a list of objects where each object represents a device. Each of this device objects has the same JSON payload structure as seen in Figure 8. This allows applications to discovery all available devices either in the Weave services or the local network.

DELETE:

The DELETE methods is an HTTP DELETE request taking a device name as a parameter. This method provide a method for deleting devices. If successful, there will be an empty response to this request.

PROVISION:

PROVISION handles the Weave services side of the device registrations. The provision happens by the application providing the device a device ID using a POST request. The device can then use its CLAIM method to claim the device ID and obtain a resource.

```
{
  "name": string,
  "displayName": string,
  "description": string,
  "nicknames": [
    string
  ],
  "serialNumber": string,
  "modelManifestId": string,
  "interfaceVersion": string,
  "firmwareVersion": string,
  "oemName": string,
  "modelName": string,
  "components": {
    object
  },
  "traits": {
    object
  },
  "deviceKind": enum(DeviceKind),
  "state": {
    object
  },
  "connectionStatus": enum(ConnectionStatus),
  "createTime": string,
  "lastUpdateTime": string,
  "lastUseTime": string,
  "lastSeenTime": string,
}
```

Figure 8: The structure of a JSON payload from a Weave GET request[28].

Documentation:

Google Weave comes with an open documentation for the Device API, Device Libraries, Devices Schemas, and Companion REST API.[28]

2.4.8 Android Things

The Android Things operating system is based on Android and optimized for IoT devices. Android Things is a part of the Android Open Source Project (AOSP) and is designed to support devices, sensors, display controllers and networking as well as using a low amount of CPU, storage, memory and power. The Android Things OS already has the Weave Library integration meaning any devices implementing the OS supports the Weave protocol instantly.[7] Android Things also utilized the existing Android SDKs, tools, APIs and resource allowing experienced Android developers to immediately implement the OS. It also provide a low-level I/O (input/output) libraries for IoT components such as temperature sensors and display controllers allowing developers to develop their own Android IoT devices.[29]

2.5 Apple Homekit

Apple's HomeKit was first introduced back in 2014 at the Apple WWDC (World-Wide Developers Conference). The HomeKit ecosystem was created to provide a trusted and consistent user experience with the focus on the smart home environment. This HomeKit ecosystem is accessible for all users owning iOS devices. And according to Credit Suisse, a multinational financial service holding company, Apple had a total of 588 million users as of the second quarter of 2016[30]. Which means their HomeKit ecosystem potentially has an incredibly large user base.[31]

Apple's HomeKit contains two key parts. First, a HomeKit framework made by a set of public iOS APIs for developing smart home applications and interact with the accessories in the ecosystem. An accessory is an IoT devices, however, Apple prefers to call them accessories. The second key part is the HomeKit Accessory Protocol (HAP). HAP is Apple's application protocol, this protocol allows iOS clients to interact with HomeKit supported accessories in the ecosystem. Apple also has their own home automation application called "Home" which recently updated iOS device already have installed.[31]

2.5.1 The HomeKit Framework

Apple's HomeKit framework is built on top of sets of public iOS APIs. The main goal of the framework is to help developers develop HomeKit enabled applications. By utilizing the framework developers can create applications to interact with and control accessories supporting the HomeKit Accessory Protocol. HomeKit also allows for seamless integration of all iOS devices. However, all HomeKit compatible application must be developed using Apple own development platform, Xcode. This

platform is only available on computers with macOS, and to get access to the API framework in Xcode you have to gain membership in the iOS Developer Program. This membership is available through paying an annual fee.[32]

Data containment hierarchy:

The HomeKit data containment hierarchy consists of Homes, Rooms, Accessories and Services. This allows a user to use one simple HomeKit application to control multiple homes. Within a home, there can be associated multiple rooms. Room are optional and do not have any physical characteristics, just a name, such as "bedroom". A room can contain multiple accessories, however, if the home does not contain any rooms, the accessory will be added to a default room for the home. An accessory can contain multiple services. The services represent the actual services that the accessories provide. An accessory can contain two types of services. Either user-controllable services, such as a light or temperature sensor, or an accessory services, like a firmware update service. An accessory may have multiple user-controllable services, such as services for both a light bulb and temperature sensor in the same accessory. A service can also have multiple characteristics, an example of this is a light bulb service having a characteristic for both turning the light on and off, and dimming the brightness. A graphical overview of the hierarchy has been providing in Figure 9.[32]

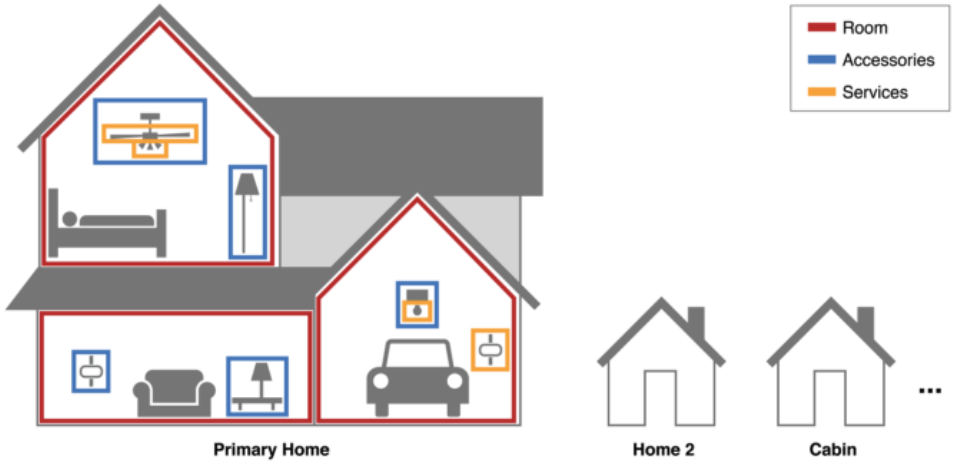


Figure 9: Overview of Apple HomeKits data containment hierarchy[33].

Database:

HomeKit accessory objects are stored in a database within the iOS devices running the application. The database can be synchronized using iCloud allowing other

iOS devices to obtain the accessory objects. This database also contains all the information about the user's home or homes, and rooms. This database provides a structure relation between all the HomeKit applications and accessories.[32]

The HomeKit API:

The HomeKit API enables the possibility for one single application to control and interact with a wide range of accessories from many different accessory vendors. The HomeKit API provide mechanisms to handle three major functions, discovery, database configuration and accessory interaction. The HomeKit API also provide low-level of abstraction. This means that developers utilizing the HomeKit API to do discovery, as an example, will not have to adapt their application according to transports. The API will map the HomeKit framework on top of it.[33]

Discovery:

Discovery is done using an object found in the API called *HMAccessoryBrowser* which will run Apples Bonjour, an mDNS browser. This object will search for accessories and use *HMAccessoryBrowserDelegate* and alarm the application if it succeeds in finding new accessories. Then it can add the accessory to a home using a discovery completion handler. HomeKit supports discovery over two transports, IP and Bluetooth Low Energy. The discovery object will only discovery accessories not yet associated with a home.

Database configuration:

This part of the API allows third-party application to display, edit, analyze and do actions on the data stored in the application database. Among many functions, this allows the application to fetch objects from the database. Such as, all homes, all rooms in a specific home, all accessories in either a room or a home. This run objects according to the data containment hierarchy depending on the request.

Accessory interaction:

Accessory interaction makes it possible for an application to communicate directly with accessories. This API provide mechanisms to change the state of services by changing its characteristics, such as turn a light accessory on and off. It is also possible to remotely interact with the accessories. However, this is only possible if there is a 3rd generation, or later, Apple TV connected to the network. It is important that the Apple TV is logged in to the same iCloud user as the iOS device using the HomeKit application.

Framework and API documentation:

The Apple HomeKit framework and APIs come with an extensive and easy to understand documentation for discovery, database configuration and accessory interaction, among other features.

2.5.2 The HomeKit Accessory Protocol

The HomeKit Accessory Protocol (HAP) is a proprietary and closed-source application protocol. Therefore vendor needs to grant an MFi (Made For iOS) license to get access to the specified resource and allowing the hardware manufacturers to release and create hardware with the HomeKit technology. As this protocol is closed-source it is hard to obtain information about the specification. However, when Apple released the HomeKit framework back at the WWDC they held a public presentation revealing some information about the protocol.[34]

The HomeKIt Accessory Protocol is an application protocol that allows accessories to communicate and connect to an HomeKit application. HAP support two different transports, Bluetooth Low Energy and IP, allowing an iOS device either to interact with an accessory directly using Bluetooth or through a network using IP. In Figure 10 the HAP stack is shown both for Bluetooth Low Energy and IP. HomeKit and HAP are on the top of the stack forming and a common language that communicate with all HomeKit devices and accessories. Further down in the stack is GATT (Generic Attribute Profile) and JSON. This layer is responsible for serializing the services and characteristics, before the deeper layer represented by ATT (Attribute Protocol) and HTTP package the data before its transmitted in the next layer using L2CAP (The Logical Link Control and Adaption Protocol) and TCP. To add, the IP stack of HAP implements a REST API using URLs to choose what type of accessory, services or characteristic to be requested.

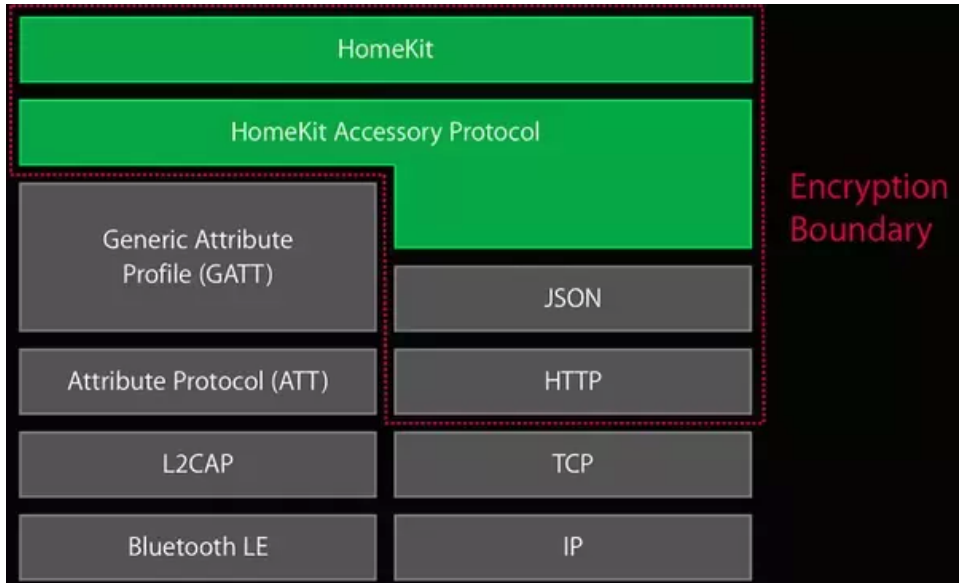


Figure 10: Overview of the HomeKit Accessory Protocol stack[35].

At last, Apple has promised protocol extensibility. This allows for services and characteristics for new accessories or new services within an accessory. Both Apple and accessory manufacturers can define new services and characteristics as long as it does not intervene with the original functionality.[34]

2.5.3 MFi license

The Made For iOS license is a program for all vendors wanting to release hardware utilizing the HomeKit technology such as the HAP specification and accessory HomeKit library. This strict license ensures interoperability between accessories from different vendors in the HomeKit ecosystem. As the bar for releasing HomeKit compatible accessories is high it might improve the general quality of the accessories. To get the license a manufacturer must pass a test conducted by Apple. The manufacturers must send their accessory prototypes to Apple for testing, if it passes the tests, Apple approves the manufacturers.[36]

3 Methodology

Whenever you define a project you should create a foundation for your work. It is important when starting that it is clear what the project are trying to accomplish, why you are doing it and how you are going to do it. By having a very defined plan you will have a better chance of reach the project's goal. It is also very important to be critical, this means being critical of others work, your own work and resources affecting the project. Being critical to everything that becomes a part of the project will higher the quality of its content because the project's solution will be built with validity. Also being aware of the strength's and weaknesses of the project will provide a better perspective and prevent pitfalls during the working period.

This thesis has a strict and defined methodology and the main goal of the methodology is to form a plan to address the thesis' research questions. The methodology explains each step towards this goal both to assure the reader that the results are valid, but also to provide an understanding as of how the results were acquired. A specific and well-defined methodology will also allow other to recreate the study and experiments and evaluation and end up with the same results.

3.1 Evaluating and assessing interoperability

Interoperability is one of the biggest challenges currently in the IoT field. Today there is many different ecosystems based on different technologies, specifications, and standards. Because of this, they speak different languages, meaning they do not understand each other, they can not interoperate. This has lead the Internet of Things dream scenario, being the Internet of Everything, where everything in the world can be connected to a scenario reminding more of Isle of Something. Meaning we have multiple ecosystems being able to interoperate with a set amount of compatible devices and services.

3.1.1 Sample size justification

After identifying a problem it is impossible to solve the problem before you know exactly what is causing it. Therefore this project has been assessing interoperability in three well-known IoT ecosystems, Open Connection Foundations IoTivity, Googles Weave, and Apples Homekit. As well as being identified in the IoT industry as very resourceful ecosystems they are built differently in terms of openness, architecture, and technology. Even if they are built with differences they have many of the same goals, one among them being to provide an interoperable IoT ecosys-

tem for the stakeholders. Therefore, it was determined that these solutions would present a good sample size of ecosystems for assessing interoperability in IoT.

3.1.2 Evaluation model

To structure an evaluation of these IoT ecosystems the evaluation has been completed according to an evaluation model. In this thesis, the evaluations model is based on Criteria-based Assessment model[37]. This model is originally created for software evaluation by assessing the software according to criteria and sub-criteria. In this thesis, there has been identified one criteria which is interoperability. However, to evaluate the end-to-end interoperability in these ecosystems there has been identified five sub-criteria. These five sub-critera are based on the hardware and software stack of IoT systems. All of the five identified layers has been chosen because they are are the building blocks in an end-to-end solution. As a result of this, they either alone or together with other layers affect the interoperability in an IoT solution. These layers are:

- **The devices layer**
- **The network layer**
- **The middleware layer**
- **The application service layer**
- **The data and semantics layer**

Each of the sub-criteria has characteristics helped assess the layers in a detailed manner. These characteristics have been identified during the study and experiments as key parts of the layer. The characteristics of each layer were then evaluated to obtain the three most important characteristics for each layer, in terms of interoperability.

The device layer:

- Device certification
- Onboarding
- Bridging

The network layer:

- Transports
- Protocols
- Remote communication

The middleware layer:

- Device management
- API abstraction
- Operating system support

The application service layer:

- Third-party application support
- Documentation
- Development Tools

The data and semantics layer:

- Data format
- Schema definition
- Schema extensibility

3.1.3 Research design

This research design will provide the needed information about the ecosystems to be able to conduct an assessment with contributing results. As well as the results from the assessment and comparison, by utilizing a strict and defined research design it will be possible for others to assess ecosystem in a similar fashion by following the evaluation model and thereby compare this thesis' results with their own.

The study:

This thesis has been based on a qualitative study and experiments to obtain the needed information to assess and evaluate the ecosystems. The focus of the study has been gathering information in the most reliable and valid fashion possible, to enhance the quality of the thesis' content and the result. Then the main resource during the study in addition to the experiments has been the specifications and documentations of the ecosystems. This has been done to provide a comparison and assessment of the ecosystems to be as consistent as possible across all three solutions. As well as providing a consistent comparison the information is gathered directly from the original source, resulting in a comparison based on reliable and valid information. These ecosystems documentation mention standards, technologies, and protocols which they support in their solutions, without explaining them. Therefore, the study has also involved research on these specific areas to provide the knowledge needed to conduct the comparison and evaluation. As a second source, to the original source, information included in the thesis has been obtained through published research papers. However, information gathered for other than its original source has been critically evaluated before integrated into the thesis.

The experiments:

In addition to the study, there were conducted experiments. The main goal of these experiments was to experience the ecosystems at first hand. By conducting these experiments there were possible to obtain information and experience about the different layers of the assessment model which were not included in the ecosystem

documentation. These experiments were set up using simulated sensors together with the ecosystems middleware and an end-user application. As well as obtaining contributing information the experiments allowed for testing of the development tools affecting stakeholders. These tools allow the stakeholders to experiment and test their products to ensure their interoperability.

3.1.4 Data analysis plan

The data from the qualitative study and experiments was assessed according to the evaluation model. This means that the ecosystems were evaluated, layer by layer. This assessment model allowed for comparison of the ecosystems, not only as a whole but layer by layer. The analysis will provide information as of how these ecosystems deal with interoperability in the different layers as well as their technical and architectural differences. These results will make it possible to address potential strengths and weaknesses in the ecosystems. This information will be relevant for stakeholders, no matter what perspective they have.

4 Experiments

To assist the qualitative study of the ecosystem's resources there were performed experiments with the ecosystems. Both to further investigate the ecosystems and get a more practical view of the stakeholders perspective, but also test their development tools. The goal of the experiments was to set up simulated devices, discovery them and perform actions on them using a client application.

4.1 OCF IoTivity

The IoTivity experiment consisted of three parts, a Raspberry Pi 3 implementing the IoTivity framework, the IoTivity tool for simulating devices and an Angular 2, single page application, as a client. The main goal of this experiment was to implement the IoTivity framework on a middleware and use it to discover the devices simulated with the IoTivity development tool controlled by a web application. Because of the developer's past experiences, this experiment utilized an IoTivity binding. This binding provides a JavaScript API using the IoTivity implementation as a backend. This allows the developer to interact with framework using JavaScript instead of the original C or C++. This nodeJS binding is sponsored by Intel and is open source[38].

First, the IoTivity framework built on the Raspberry Pi 3, a small single-board computer, utilized as a middleware or gateway. The Raspberry Pi was running the Raspbian Jessie operating system, which is a Debian-based OS built on the Linux Kernel. Building the framework on the Raspberry Pi 3 proved to be a very time consuming and challenging task. To build the IoTivity framework required a lot of external libraries and a complex building environment. Even with a setup guide provided in the IoTivty documentation, there were a lot of obstacles as the framework turned out to be very version sensitive. After much effort the framework was build and the middleware was connected to the network using an Ethernet cable.

The IoTivity Simulator tool is an Eclipse Plug-In which provides two perspectives. The Service Provider perspective, used in this experiment, can simulate OCF resources by using Resource model definition (RAML) files. An overview of this tool is provided in Figure 11. This perspective lets users create and delete devices as well as changing their attributes manually and automatically. This tool was run on the Raspberry Pi separate from the IoTivity framework.

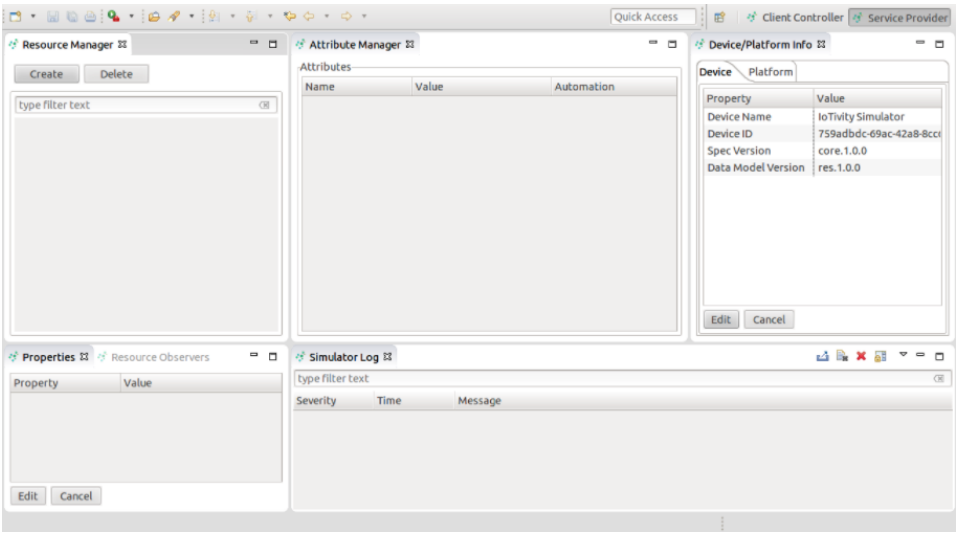


Figure 11: Overview of the service provider perspective.

On top of the IoTivity framework, there were implemented a REST API server. This is an open source nodeJS server which can be integrated into the IoTivity nodeJS binding[39]. This REST API server listens on port 8000 for any GET request on the local network. This server implements the API for the discovery of resources, platforms, and devices as shown in Table 1. If there is performed a GET request on any of the API endpoints it will execute the proper function through the IoTivity nodeJS binding and receive a response similar to the one in Figure 12.

Figure 12: The nodeJS servers response on an IoTivity multicast.

The client application was developed as a web application in the Angular 2 framework. This was a very simple application making a GET request on the resource endpoint of the API, displayed in Figure 12. The REST API server would the ask the IoTivity framework to discover all IoTivity resources on the local network and return it to the application. This specific REST API server returned the payload in the JSON format, instead of CBOR which the IoTivity framework nor-

mally uses. Just to show that the resources had been discovery and provided to the application, the application simply just outputs the JSON payload in a structured fashion. As illustrated in Figure 13, the application can discovery resource through the "update" button. This will trigger the GET request. To display the results of the request the application loop through the results and automatically put them in a unordered list. For each of the devices their API URI, resource type, and interface type are listed to identify the resources. On the right side, in the developer console, the actually JSON payload is displayed.

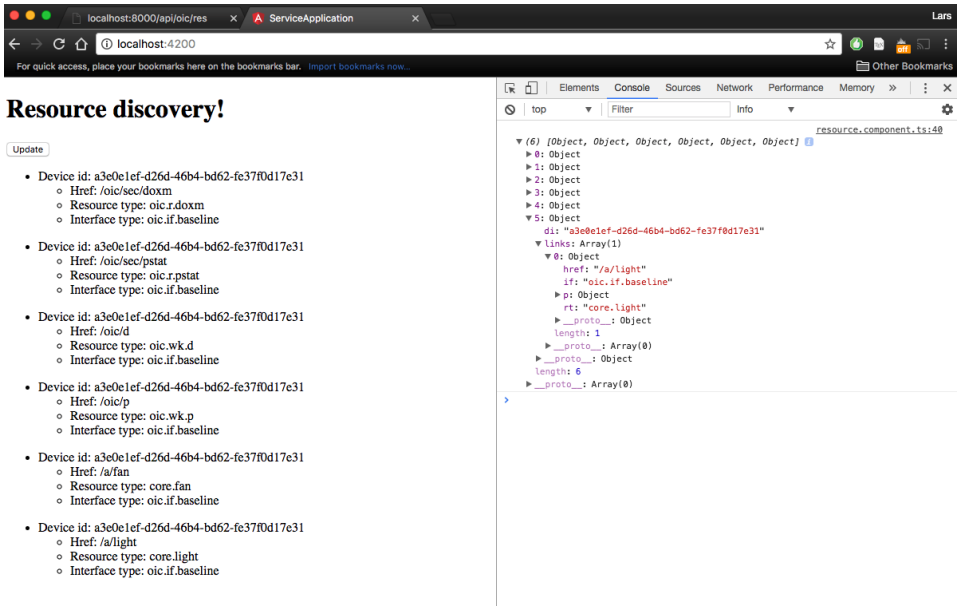


Figure 13: An Angular 2 Web application initiating and displaying the results of resource discovery as well as the JSON payload.

Because there was chosen to do the IoTiivity experiment with an external application to test it the integration of third-party application the client controller perspective of the IoTivity tool were not used. However, in Figure 14 it is displayed. In this figure, one can see the possibility for discovery of resources using the Resource Manager and performing actions on the resource using the Attribute Manager. Even though this tool was not a part of this experiment, it is still a very useful tool for devices vendors to check the compatibility and interoperability of their devices. It can also be used by application developers to further understand how to interact with the resources in the ecosystem.

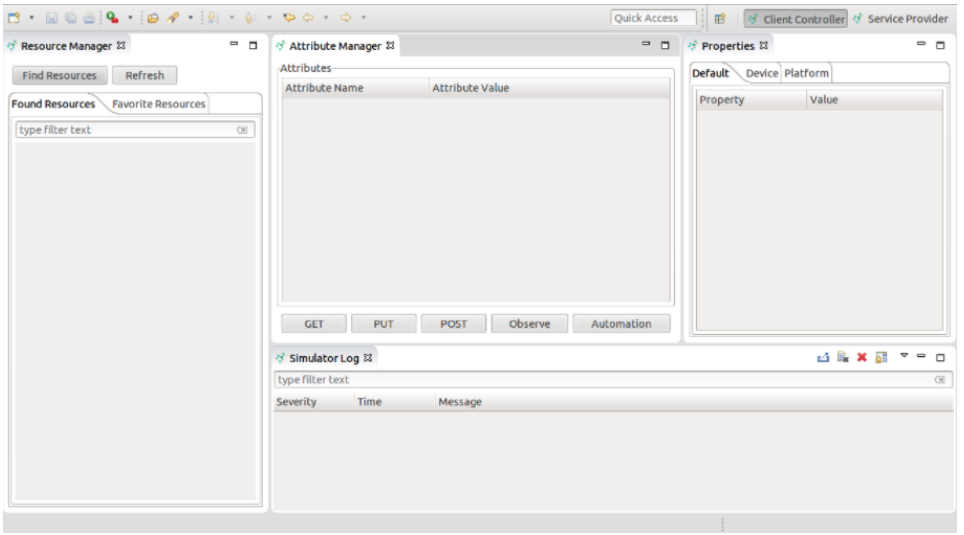


Figure 14: Overview of the client controller perspective.

4.2 Google Weave

The Google Weave platform experiment included the Weave Developer Application and IoT Developer Console. The goal of this experiment was to investigate the IoT Developer Console by setting up a simulated Weave device. Then try to discover it, and interact with it using the Weave Developer Application.

The IoT Developer Console works both as a device manager and a device simulator. The developer console allows device vendors to in a structured and organized way set up and manage their products, it also allows vendors to upload important interface information about the device. It also lets application developer which wishes to their applications to set up simulated Weave devices. In Figure 15 there is displayed and overview of the IoT Developer Console simulator. Through this overview, application developers can easily add virtual Weave devices, get an overview of their devices and what devices have been interacted with in the last 7 days. When creating a new simulated Weave devices it is only possible to created devices with a standardized schema definition.

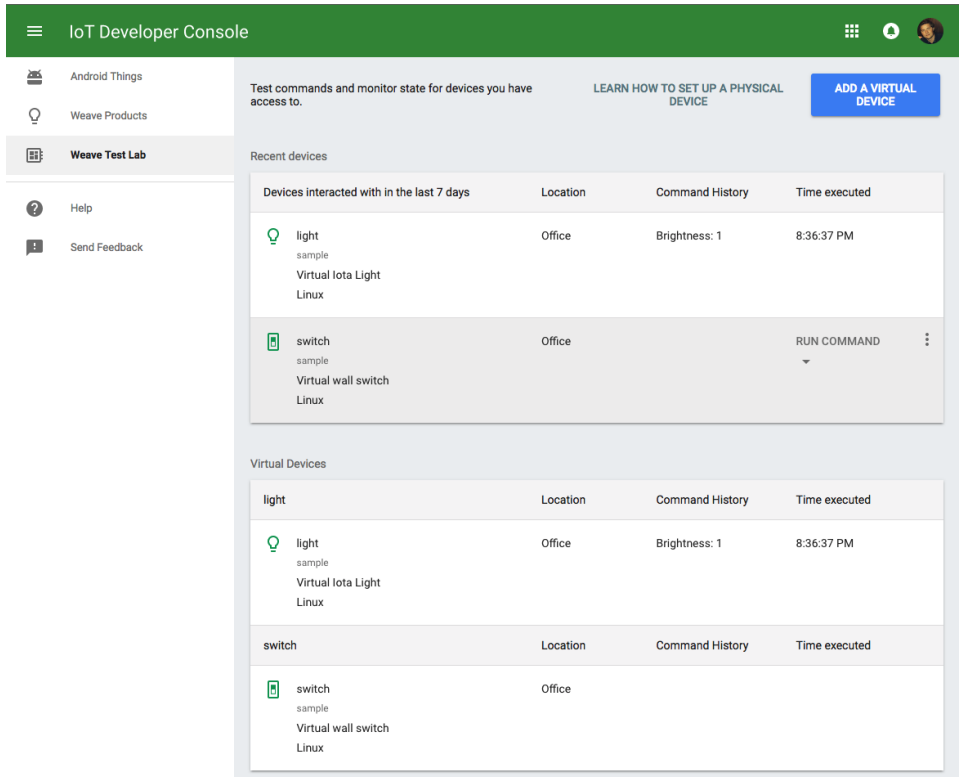


Figure 15: Overview of the IoT Developer Console simulator.

In this experiment, there was created a light device. When you have created a device there is possible open a user interface for it. This user interface provides an overview of the components of the devices, as this is a light devices it has the same structure as displayed in Table 2. At the top of this user interface, as shown in Figure 16, it is possible to see the id, nickname, and status of the devices as well as the "Show device resource json" button. This button opens a view of the devices full JSON object. This JSON document contains every detail about the device such as kind, id, uptime, name, description, location, owner, and connection status. It also shows the full structure of the devices schema definition with its components and traits. This makes it very easy for developers to get a good overview of the structure of the completed device object containing information about the device itself and its functionality.

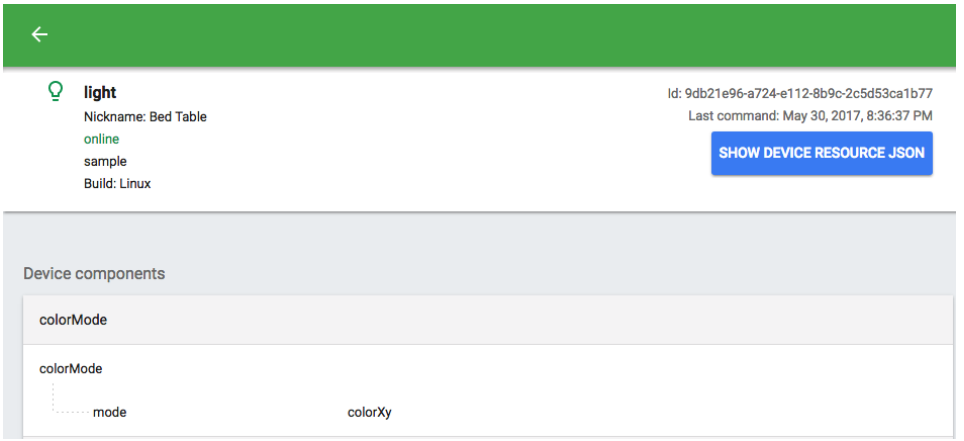


Figure 16: The light device's user interface.

This user interface also allow for direct interaction with the components of the devices. In Figure 17 there is displayed a JSON request updating the dimmer component by changing the value of the brightness trait.

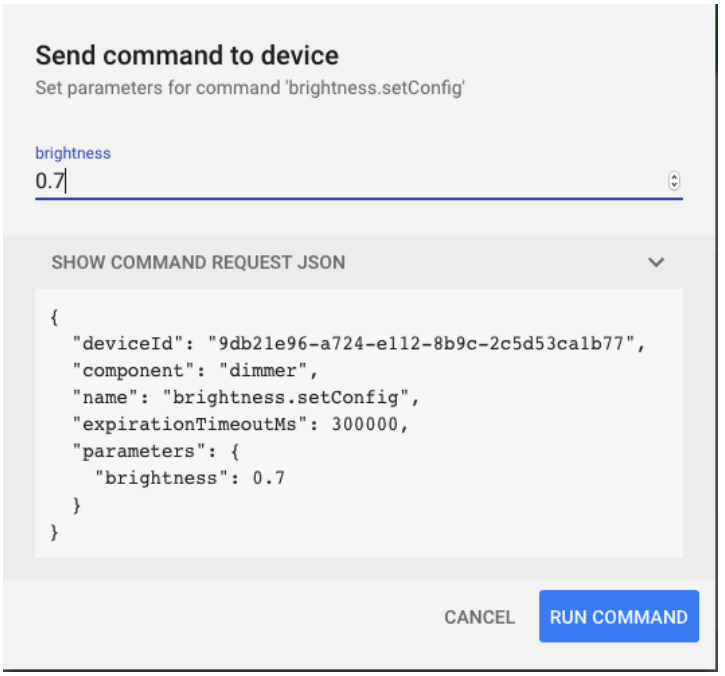
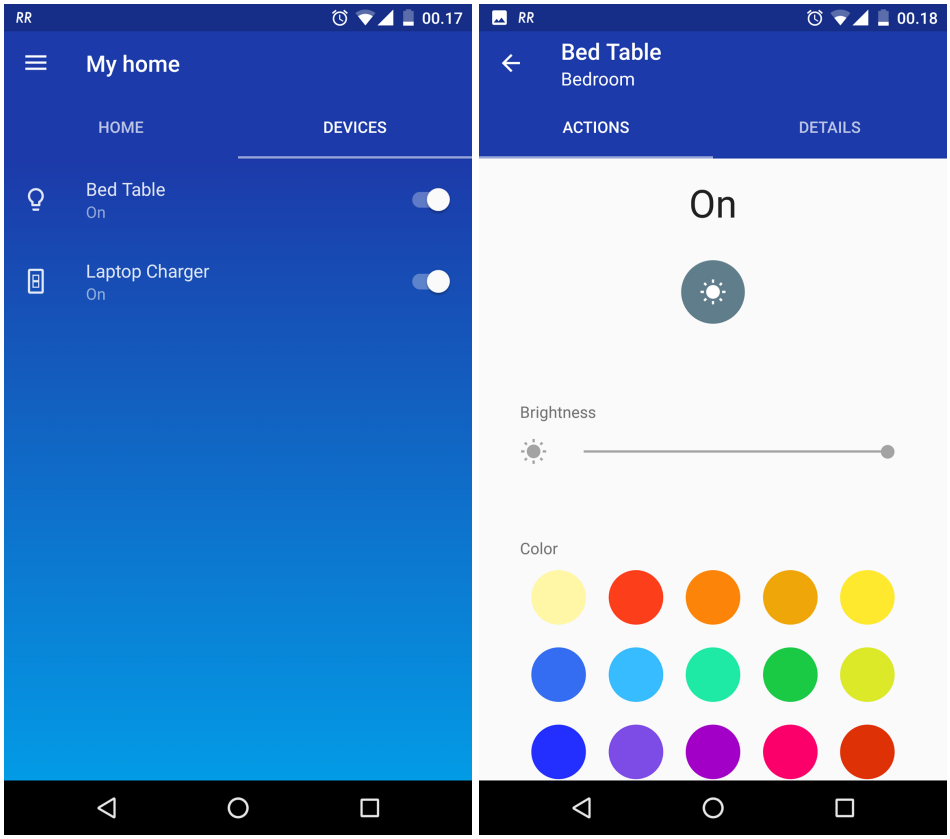


Figure 17: Controlling the lights brightness by changing the state of the dimmer components brightness trait.

To discover and interact with the simulated Weave devices the Weave Developer Application was tested. This application is a useful tool for devices vendor implementing the Weave library into their devices, to test their device compatibility. The Weave Developer Application uses the Google account of the smartphone to connect to the Weave server and then check for registered devices. And as shown in Figure 15 there has been created two Weave devices on the account which also the smartphone is connected to. Both of these devices are automatically discovered by the application, illustrated in Figure 18a. By clicking on one of the devices the user can interact with the components by changing the state of its traits through the user interface, displayed in Figure 18b.



(a) Overview of discovered Weave devices (b) Interacting with the light device

Figure 18: Overview of discovered Weave devices and interaction with a light device

4.3 Apple HomeKit

Because of resource restriction, there were not conducted any experiments on the HomeKit framework. To get access to the HomeKit services, developers are required to install the Xcode platform which is Apple's development platform. On this and only this, is it possible to develop HomeKit application for iOS devices. This is because the API framework is only available through the development platform. Still, with the development platform installed only developers with iOS developer membership will be able to utilize the framework services. To obtain this developer membership Apple requires developers to pay an annual fee.

Apple also provides an HomeKit Accessory Simulator to allow iOS developers to test their applications without having to set up real physical accessories. In Figure 19 there is illustrated an overview of the device manager of a simulated Light accessory using the HomeKit Accessory Simulator. When creating a new device it is possible to give it a name, manufacturer name and it is given a generated Serial Number. It is also possible to add other characteristics to the Accessory Information related to firmware and hardware depending on the what is needed. A single accessory can contain multiple services. In Figure 19 a Light accessory is given a Lightbulb service. There is only possible to add services according to the HomeKits schema definitions. The Lightbulb services come with preset characteristics as seen in the figure, but there is also possible to add more characteristics. However, there is only possible to add characteristics compatible with the chosen service. These accessories are easily activated and deactivated.

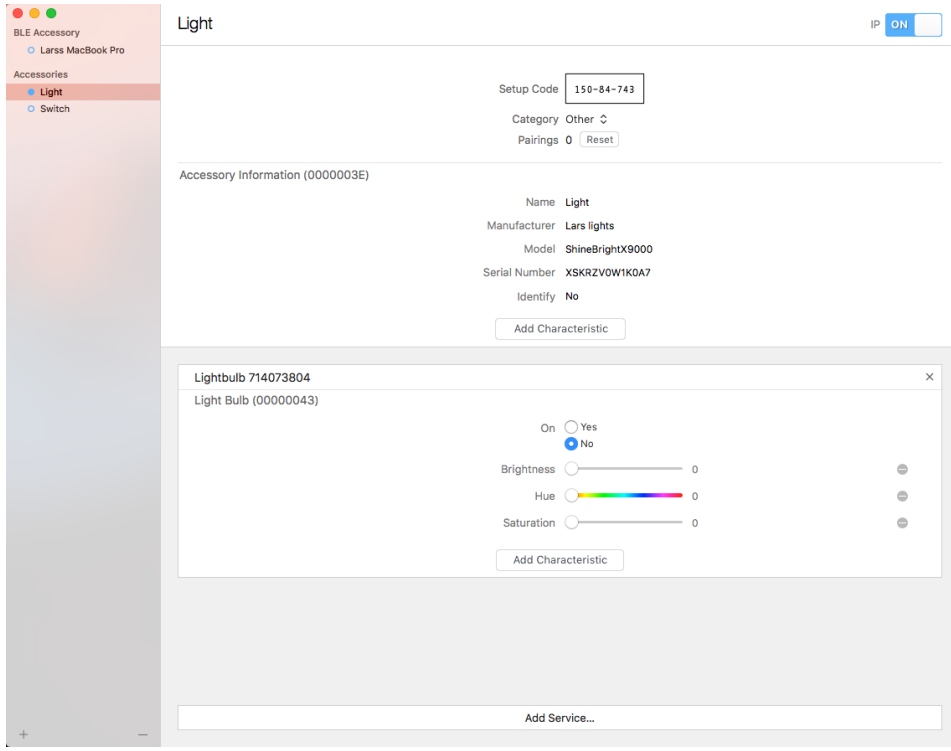


Figure 19: Overview of accessory management in the HomeKit Accessory Simulator.

5 Results

The representation of the results is split into two section. The first section is a layer by layer presentation of the results obtained from the study and experiments. Each of the layers has a summary of the characteristics and its respective results. In the second part there is is a structured summary utilizing a table to provide a precise overview of the results regarding the ecosystems, the layers, and the characteristics.

5.1 The device layer

- Device certification
- Onboarding
- Bridging

Device certification:

All of the three ecosystems accept new devices into their ecosystem. But, in order to commercialize products as a part of the respective ecosystems, device manufacturers must acquire either a certification or a license, depending on the ecosystem requirements. These certifications and licenses ensure that certified devices are fully ecosystem compatible to provide maximum device interoperability.

IoTivity

The OCF certification process contains two steps. First, the manufacturers need to become an OCF member by paying the applicable annual membership fee of 2,000 US dollars. As members vendors can submit their devices and contact information to an OCF authorized test laboratory. If the device passes the vendor receives a certification and authorization to release products as IoTivity compatible. If the device fails the vendors can alter the device and retest it.

Google Weave

To ensure full interoperability between the Weave device and the Weave server, Google services and the ecosystem, Google verifies device through the Weave device certification program. The first part is run through the IoT Developer Console provide by Google. This console uses a set of automated tests to test the compatibility of the devices. Completing this test will provide vendors a limited prototype license. Further, to be able to distribute their products as Weave devices the vendors has to sign a Weave Program Agreement and then pass a compatibility test performed by Google developers.

Apple HomeKit

Apple's MFi program provide a license which all vendors must obtain before developing HomeKit accessories. In contrast to IoTivity and Weave, Apple's HomeKit technology and resources are closed-sourced. Accessory manufacturers first need to join the MFi program by registering and ensure Apple of their existence, then purchase the identity verification. After joining the program Apple will release the needed resources, libraries, technologies and tools needed to develop HomeKit compatible devices. Before releasing their accessories to the marked, vendors need to send their devices to Apple to pass their internal testing

Onboarding:

Onboarding is a term used for making the devices available in the ecosystems. All the three ecosystems need their devices to onboard the ecosystem before they can be discovered and interacted with.

IoTivity

In the OCF specification, it is stated that there is no set way of onboarding devices into the ecosystem. The reason for this is that IoTivity supports many communication and connectivity protocols that there is no "one size fits all" solution. There is stated that it is the device manufacturers responsibility that the devices are able to onboard.

Google Weave

Google clearly specify in the Weave documentation that it is the device manufacturers responsibility to provide functionality to connect the Weave device to the network allowing it network connection. With network connection the devices can access the ecosystem by registered with the Weave server, making them available for the user and other Google services.

Apple HomeKit

As mentioned in the device certification characteristic, HomeKit accessory information is closed-source. As a result of this manufacturers needs to obtain the MFi license to know the specifics of the HomeKit onboarding process. Through their released resource it is confirmed that is an onboarding process and that requires specialized Apple hardware and libraries in order to enter the ecosystem.

Bridging:

All three ecosystems have a certification program needed for vendors to commercialized their products as ecosystem compatible. Bridging a device into the ecosystem would mean allowing support for a device that is not commercialized compatible as well as devices that also support other frameworks.

IoTivity

There is possible to bridge non-IoTivity devices into the IoTivity framework. This is possible by developing and translator (resource container), translating the non-IoTivity device into an IoT compatible device. This would register the devices as IoTivity devices and then translate relevant services and actions, allowing IoTivty to fully map its technology on top of the bridge.

Google Weave

As of now, there are no resources claiming that it is possible to bridge non-Weave device into a Weave ecosystem.

Apple HomeKit

HomeKit allows bridging of specific devices, to protect against security issues, into their ecosystem. Homebridge is an open-source tool running a lightweight nodeJS server on the local network that emulates the iOS HomeKit API through user-contributed plugins. This emulator will map the non-HomeKit accessories as HomeKit compatible accessories.

5.2 The network layer

- Transports
- Protocols
- Remote communication

Transports:

An important part of the ecosystem is their range of supported physical transports. A wider range of supported transports will allow for a wider range of compatible devices.

IoTivity

IoTivity supports an exceptional range of wireless transports. IoTitivty support communication over WiFi, Bluetooth Low Energy, Bluetooth, ANT+, Zigbee, and Z-wave. All very popular wireless technologies in the IoT world because of their different advantages.

Google Weave

As of this day, the Google Weave ecosystem only support devices with WiFi connection. However, Google has announced that support for Bluetooth Low Energy is under development.

Apple HomeKit

According to Apple's resources, the Apple HomeKit support two types of transports in their ecosystem. The accessories must be able to communicate over either WiFi

or Bluetooth Low Energy.

Protocols:

Each of the three ecosystems utilizes one specific application layer protocol for communicating with their devices.

IoTivity

IoTivity uses CoAP, a constrained application protocol, which is similar to HTTP, only more lightweight.

Google Weave

The Google Weave ecosystem is based on their own application layer protocol, Weave.

Apple HomeKit

Apple has also built their own application layer protocol for HomeKit, HAP, HomeKit Accessory Protocol.

Remote communication:

The ecosystem all provide the possibility for controlling the environment remotely. Allowing the consumer to interact with the ecosystem even when not in the actual area of the system.

IoTivity

IoTivity allows remote interact with devices using XMPP (Extensible Messaging and Presence Protocol) to access the server remotely over the internet. The external connection is handled by the middleware, a gateway, providing the interaction between the device and the internet.

Google Weave

Google Weave provide remote interaction through the Weave server in the cloud. A Weave client can either communicate directly to Weave device over the local network or use the Weave Server as middleware. As all Weave device are connected to the Weave server, the client and act on a device in the cloud and the cloud server will forward the interaction to the actual device in the ecosystem.

Apple HomeKit

To be able to remotely interact with accessories in a HomeKit ecosystem the system needs to implement an Apple TV of the 3rd generation or later. An Apple TV will function as a middleware between the iOS client and the accessories

5.3 The middleware layer

The main purpose of the middleware is to provide an interface for the ecosystem by hosting an API framework on top of the ecosystem. In IoT systems, the middleware

is often solved by a hub or gateway, but it is also possible to utilize other clients like a smartphone or the cloud as a middleware.

- Device management
- API abstraction
- Operating system support

Device management:

Device management is a crucial feature of the ecosystems. Device management is mechanisms for dealing with devices provided by the API. These mechanisms are the API specifics that allow the ecosystem to discover, register, delete, and interact with the devices.

IoTivity

IoTivity utilizes the RESTful API Modeling Language (RAML) to deal with the lower levels of device management. CoAPs multicast can discovery new endpoints on the network and register them in the IoTivity ecosystem as web resources. Then allows interaction with the resource using their URI and the CoAP methods.

Google Weave

Google Weave also provide an API based on the RESTful architecture. This REST API interacts with the device through the Weave methods, the network, and the Weave server. The API provides mechanisms for discovery and registration of Weave devices as well as other interactions. These Weave methods are based on the HTTP verbs.

Apple HomeKit

HomeKit provides device management through their HomeKit API framework for mechanisms such as discovery, registering and interactions with the accessories in the iOS client's database. This API is also based on the REST architecture. However, the low-level implementation of the API is proprietary which means that only vendors with the MFi license have the underlying resource available.

API abstraction:

All of the three ecosystems provide API abstraction. This means that their framework map on top underlying technology to allow applications to communicate with the ecosystem using a unified interface. This allows the ecosystem providers to add addition support in the underlying technology without affecting the functionality of existing application using the framework API. This is what allows IoTivity to map CoAP, and HomeKit to map HAP on top of non-IP connectivity protocols such as Bluetooth Low Energy.

Operating system support:

These ecosystems has differences in terms of target groups and platforms, and therefore support different operating systems.

IoTivity

The IoTivity platform supports a wide range of operating systems. Currently, IoTivity supports Linux, Tizen, Android, Arduino, and Windows. However, IoTivity shows no OS restrictions towards third-party applications controlling the platform or towards devices integrating the OCF specification.

Google Weave

As the Google Weave ecosystem is based on a protocol and libraries it does not need to be implemented on top any hardware with a specification OS. However, the Google Weave platform supports applications running on either iOS, Android or the Web. All though Google promotes Android Things as the chosen OS for Weave devices, there are no restrictions for devices implementing the Weave technology.

Apple HomeKit

Apple's HomeKit is very restricted in terms of operating systems. The HomeKit API only supports iOS device which means that only devices with the iOS operating system will be able to utilize the HomeKit Accessory Protocol and interact with the accessories. As the accessory resource are closed-source there is not released other information than that accessories must be built on Apple-approved hardware.

5.4 The application service layer

- Third-party application support
- Documentation
- Development Tools

Third-party application support:

All the three ecosystems support the integration of third-party applications. IoTivity shows not restrictions related to Third-party applications. Google Weave only allows integration of application running on either the iOS or Android operating system in addition to the Web. Apple HomeKit only supports application run on iOS devices.

Documentation:

All of the three ecosystems provides proper documentation about their specifications, ecosystem architecture and the framework API. Apple HomeKit is the only one of the systems that acquire a license before releasing accessory documentation. However, documentation needed to develop client applications are fully public.

Development Tools:

Development tools are very useful for both application and device developers. These tools let them simulate parts of the ecosystem that they want to create interoperable products towards and build a testing ground for their products

IoTivity

IoTivity provides an Eclipse plugin tool called IoTivity Simulator. The tool allows for simulation of both Service Provider perspective and a Client Controller perspective. The Service Provider part of the tool all for simulation of OCF resources by using RAML files. This allows developers and service providers to simulate device with different resources to test their application interoperability with the IoTivity devices. The other part, The Client Controller, simulate and OCF compatible client. This tool uses the IoTivity framework API and creates and user interface that allows users to perform resource discovery as well as performing other CoAP methods on the resources.

Google Weave

Google provides what they call Weave Developer Tools consisting of a Weave Developer Application, an IoT Developer Console, and Command-line tools. The Weave Developer Application is a mobile application available for Android and iOS. This application lets users control their registered Weave devices. Developers can use this application to test device state management, commands, and registration support. The IoT Developer Console is a Web application for device developers and applications developers. The developer console provides a simple and structured way to manage products and to upload interface information about the devices. This developer console also lets application developers set up simulated Weave device and inspect their states, schemas, and to test their application compatibility. The Command-line tool, weave_client, is a tool for performing Weave-related tasks, such as registering and view devices, send commands and track device states.

Apple HomeKit

Because of Apple's closed-source approach towards accessories, it can be hard for third-party application developers to test how their application interoperate with the HomeKit Accessories. Therefore Apple provides an HomeKit Accessory Simulator. This allows application developers to simulate accessories where the user can control the services and characteristics of the accessories. To use this tool the developers must either run their application in the iOS Simulator or run it on an iOS device using Apple's development platform, Xcode.

5.5 The data and semantics layer:

- Data format

- Schema definition
- Schema extensibility

Data format

A data format allows machines to interchange, read and parse data. These data formats are used to structure requests, responses, and payloads in systems that transmits data. A common data format together with schema definitions allows the devices of the system the possibility to understand their own data and often makes the data human readable.

IoTivity

To provide a data format with as little constraint on the devices as possible IoTivity utilize the binary format, Concise Binary Object Representation (CBOR), as their data exchange format. CBOR is compatible with JSON which allow IoTitiy to also use JSON documents.

Google Weave

The Google Weave platform uses the JSON data format as their only data model.

Apple HomeKit

Apple HomeKit utilizes two data formats depending on the transport used. Over Bluetooth Low Energy the payload used the Generic Attribute Profile (GATT) and over WiFi it uses JSON.

Schema definition

Schema definitions are extremely important in these ecosystems. It is the schema definitions that is responsible for the device interoperability. A schema definition ensures that a device type has a standardized schema, providing two devices from different vendors to have the exact same data properties and data structure, allowing the ecosystem to utilize both devices in the same way, even though they are from different vendors. All the ecosystems schema definitions has been explained in the chapter Background and related work 2.

IoTivity

IoTivity uses the OCF specified resource model containing a resource type, the resource interface, the resource identity and the resource property. OCF allows for crowd-sourcing of resource models for new devices which provides an exceptional wide range of resource models, which have resulted in a wide ranges of supported devices types

Google Weave

IoTivity uses the OCF specified resource model containing a resource type, the resource interface, the resource identity and the resource property. OCF allows for

crowd-sourcing of resource models for new devices which provide an exceptionally wide range of resource models, which have resulted in a wide range of supported devices types

Apple HomeKit

The Apple HomeKit accessories are described using services and characteristics. Both Apple and accessory vendors are allowed to define new services and characteristics as long as it does not intervene with original functionality

Schema extensibility

The data format of the three ecosystems is built with extensibility in mind. As much as a set standardization of schemas allow for optimal interoperability, not allowing any extensibility will deny any form of device innovation. However, they all state that the extensibility, if implemented, must not affect the functionality and definitions already implemented.

5.6 Summary

This is a summary of the results from the Criteria-based assessment of the three ecosystems. Each of the layers has been divided into separate tables structured with the ecosystems and the characteristics of the layer.

The device layer			
Characteristics	**IoTivity**	**Google Weave**	**Apple HomeKit**
Device certification	Yes, through gaining membership, pay annual fee and pass an OCF authorized device test	Yes, by signing the Weave Program Agreement and pass a device compatibility test	Yes, by register as vendor, purchase identity verification, and pass the internal device test
Onboarding	Requires the device manufacturers to provide onboarding mechanisms to connect to the ecosystem	Requires the device manufacturers to provide onboarding mechanisms to connect to the ecosystem	Requires the device manufacturers to provide onboarding mechanisms to connect to the ecosystem
Bridging	Yes, but only of specific devices	No resources claim support for device bridging	Yes, through resource containers. No restrictions documented.

The network layer			
Characteristics	**IoTivity**	**Google Weave**	**Apple HomeKit**
Transports	WiFi, Bluetooth Low Energy, Bluetooth, ANT+, Zigbee, Z-wave	WiFi	WiFi, Bluetooth Low Energy
Protocols	CoAP	Weave	HAP
Remote communication	Yes, through XMPP and the middleware	Yes, through an Apple TV, 3rd generation or later	Yes, through the Weave server located in the cloud

The middleware layer			
Characteristics	**IoTivity**	**Google Weave**	**Apple HomeKit**
Device management	Yes, through the API framework	Yes, through the API framework	Yes, through the API framework
API abstraction	Yes, provided by the ecosystems middeware	Yes, provided by the ecosystems middeware	Yes, provided by the ecosystems middeware
Operating system support	Linux, Tizen, Android, Arduino and Windows	Android, iOS, Web	iOS

The application service layer			
Characteristics	**IoTivity**	**Google Weave**	**Apple HomeKit**
Third-party application support	Yes	Yes	Yes, but must pay annaul fee to use the API framework
Documentation	Yes	Yes	Yes, except for HAP and accessories
Development tools	Eclipse plugin to simulate both devices and clients	Web application to manage and simulate devices and an Android and iOS application to test from client side	HomeKit Accessory Simulator to simulate accessories, only available on Apple devices.

The data and semantics layer			
Characteristics	**IoTivity**	**Google Weave**	**Apple HomeKit**
Data format	CBOR, JSON	JSON	GATT, JSON
Schema definition	OCF resource model	Components, traits	Serices, characteristics
Schema extensibility	Yes, but must not interfere with current functionality	Yes, but must not interfere with current functionality	Yes, but must not interfere with current functionality

6 Discussion

This discussion chapter is split into four sections. The first section a discussion on the results and the interpretations of these results. The second section will be a discussion regarding the validity of the chosen research design. In the third section, there will be a paragraph concerning the limitations of the work. At last, there will be recommendations for further research.

6.1 Device certification and bridging

All of the three ecosystems require vendors to obtain a device certification to allow them to commercialize their products as ecosystem compatible. This procedure will ensure that all certified devices will have full interoperability with the ecosystem. Apple's HomeKit is the only one of the ecosystem with proprietary device information, as a consequence of this, the device innovation and experimentation will be more limited than with the other ecosystems. This could potentially reduce the amount of integrated devices compared to the other ecosystems with a fully open device resources.

Onboarding is important as it is responsible for connecting the device to a network, making it possible for the ecosystem to discover and interact with them. None of the three ecosystems specify any specific process that this onboarding has to follow. What is stated by all the three ecosystems is that it is the device vendors responsibility to provide mechanisms for onboarding of the devices into the ecosystem. It is reasonable to believe that the onboarding mechanisms will be tested in the device certification process.

However, there is not mentioned any restriction related to a vendor having multiple ecosystem certifications. At the same time, two of the three ecosystems allow for bridging of devices that also supports other ecosystems, into their own. This theoretically opens a door of possible cross-ecosystem compatible devices. This will allow device manufacturers to release devices that can provide interoperability within more than one ecosystem. This is the vendor's decision to make as this will be a complicated process because it will require onboarding into the different ecosystems as well as bridging technology. Some of the devices could also be to constrained to run the HomeKit API emulator needed to bridge into the HomeKit ecosystem. A possible solution could be a vendor providing a provisioning application that supports onboarding multiple ecosystems. When choosing the specific ecosystem to board, it would be possible for the vendor application to request

the appropriate bridge needed to connect the device. As for resource constrained devices, the vendors could utilize a hub as a resource container or middleware between the devices and the ecosystem, handling the bridging and ecosystem interaction on behalf of the constrained device..

There is not yet confirmed if the device certification programs will pass devices build for supporting multiple ecosystems. One reason for denying these devices could be the potential security issues related to the bridging of technologies. A possible denial will not discard the solution, but the device can not be commercialized as ecosystem compatible.

During the study and experiments, there were not found any information about device bridging for Google Weave device. On the other hand, there were not confirmed that it is not possible. But as a consequence of lacking information, it was determined to assume that there Google Weave does not support device bridging, as of now. But considering that Weave devices only implements and SDK to connect to the Weave server it could potentially be possible in the future. It is also important to mention that Apples HomeKit does not support bridging of device that can compromise the security of the ecosystem, such as door locks.

6.2 Transport restrictions and framework abstraction

Google Weave, as of now, only supports devices with WiFi support. By only supporting one specific transport vendors of Weave devices are very transport restricted.WiFi is also very resource heavy, which will result in resource constrained devices not being compatible with the ecosystem. Apple's HomeKit is also very transport restricted as it only supports Bluetooth Low Energy in addition to WiFi. However, this provides an option for vendors to allow constrained devices to connect. IoTivity, on the other hand, has an incredibly wide range of supported connectivity protocols. By supporting what some might consider all important transports in the IoT industry, IoTivity provides a very desirable platform for device vendors. IoTivitys approach makes their ecosystem transport independent in contrast to Google Weave and Apples HomeKit, allowing for an ecosystem consisting of many interoperable devices from different vendors, independent of their integrated technologies.

All of these ecosystems is built with an API framework on top of their ecosystem, providing an abstraction of low-level technologies and architecture. This abstraction allows the ecosystem providers to implement support for different transports in the future without interfering with existing implementations of the API. It would also be possible for the same middleware providing a potential device bridge and also act as a hotspot or bridge for non-supported transports and thereby connect them to the network.

6.3 Schema standardization

Both Google Weave and Apple HomeKit uses JSON as their data format for exchanging data over WiFi. In addition to that Apple HomeKit utilizes GATT when passing payloads over Bluetooth Low Energy. IoTivity has adopted the binary, but JSON compatible, CBOR format. Even though CBOR offers smaller code and message sizes as well as providing lower CPU-usage then JSON its binary format makes it much less readable for humans. JSON provides a human readable format which makes it much more convenient for industry stakeholders to interpret the payload.

These data formats contain one of the most, if not the most, important aspects of an interoperable IoT ecosystem, and that is their standardized schema definition. All the three ecosystems have their own standardized data models to in a unified way define the properties and functionality of the devices. Even though their data models are not they same, they have the same goal and intention. And this is to allow devices from of and kind and vendor to communicate in the ecosystem to provide full interoperability. However, their approach to standardizing these schemas differ drastically. IoTivity allows for crowd-sourcing of these schemes. Even though these crowd-sourced schemes has to be officially approved and release by OCF before they can be utilized by device and application developers there has already been integrated an incredibly amount of data models to support different types of devices and devices functionality. On the other side, we have Google Weave which as of now only offer schema definitions for HVAC (heating, ventilation and air condition), lights, outlets, televisions and wall outlets, and has shown no signs of allowing for any crowd-sourcing of schemas. Apple HomeKit is somewhere in the middle they provide their own schema definition, but they also allow device manufacturers to apply for new schema definitions as long as it does not interfere with existing functionality. The amount of different defined schemas also defines a number of possible device types that can interoperate within the ecosystem. This could be a big factor for device vendors looking for an ecosystem to support. It is reasonable to believe that a device vendor would like to make their devices to be compatible with a system that allows for a wide range of devices, as this will also attract consumers and end-users that want one system to control everything.

6.4 Affected stakeholders

There are many stakeholders in an IoT ecosystem. First, there is the device vendor or manufacturer which, by all these three ecosystems, are given a lot of the responsibility for devices interoperability. Choosing an ecosystem to support is a big decision as the vendor has to obtain a certification, provide onboarding and implementation the data model, which without a bridge locks that device to the specific ecosystem. On top of this, the ecosystems support different transports which mean

that the devices must support these specific transports to onboard the system. And as both Apple HomeKit and Google Weave has proven to be very transport restricted, it does not leave manufacturers with many choices.

The consumer or end-user share some decisions with the vendors. They have to choose an ecosystem to adopt into their smart environment. And as mentioned, there is reasonable to believe that an end-user would like an ecosystem that supports a wide range of device types, to allow the user to control everything while only interaction with one single application. Another very important factor for the consumer as well as the application developers are operating system support. As Apple HomeKit only allow their devices to be controlled by iOS clients which limit their potential user base to only users of iOS products. IoTivity does not have any client application OS restrictions, and even though it need a platform, such as a gateway to run on top off it still offers support for a wide range of gateway operating systems. While Google Weave supports both iOS and Android it also supports Web application. Web applications can, in theory, run on any OS with a web browser, which opens up a lot of doors for developers, as well as potential users.

Application developers and service providers can develop third-party application for all of the three ecosystems as they all support it. However, they have to consider the operating system support. Either they could choose an ecosystem with support for all types of clients operating systems and then either choose to develop an application supporting that ecosystem in one or multiple operating systems. However, maintaining and updating an application for multiple operating systems can be both time and money requiring. Therefore, it would also make sense to choose an ecosystem like Apples HomeKit which would result in only providing the application for a single operating system. Using the Web as a universal application platform would also provide the opportunity for developing only a single Web application. All the three ecosystem provides developers with tools to aid the the development. Both Google Weave and IoTivity provide both client and device simulation tools. These are a very nice edition to both devices manufacturers and application developers wanting to test their product interoperability during the development. Apple only provides a simulation tool for their accessories, which is also the most important tool to provide of client and device in the HomeKit ecosystem, as Apple do not share technologies behind the accessories for anyone without the MFi license.

6.5 The research design

As mentioned in the Methodology 3 this thesis has been based on a qualitative study of the ecosystems specifications, standards, and technologies, as well as experiments, to be able to assess and evaluate their current end-to-end interoperabil-

ity. To provide this assessment with a solid and valid foundation the information and data collected in the qualitative study have been, as far as possible, obtained from their original source. As a consequence of this, the results are based on the quality and details of the ecosystems documentations. A poorly written documentation could be affecting the results of the evaluation. However, it is reasonable to believe that all these ecosystems, being as acknowledged as they are, have a complete, updated and correct documentation.

The experiments conducted provide practical experience and information about the ecosystems layers that the documentation alone could not provide. By conducting the experiments and set up the ecosystem it was possible to further see the systems from a stakeholders perspective. The experiments allowed for addition information about ecosystem set up, APIs and framework abstraction, testing integration of third-party applications, development tools and, data formats and schema definitions.

To assess the data and information from the study and experiments, this thesis built its assessment on a Criteria-Based Assessment model. This model is used for evaluating a specific software according to multiple criteria and sub-criteria. In this assessment, there were only one criteria, interoperability, but there was multiple software. This should not affect the validity of the assessment as theoretically the ecosystems were assessed one by one using the interoperability criteria. By using this research model and ecosystem assessment it will allow stakeholders to get an overview of how the ecosystems compare in terms of interoperability. As well as providing a structured assessment model for assessing interoperability in ecosystems, this will allow stakeholders to compare other ecosystems to the ones in the thesis.

6.6 Limitations

As a result of both time and resource constraints, the experimental implementations were only completed for the IoTivity and Google Weave ecosystems. Both these implementations utilized the device simulating tools. This removed the need for approval and uncertainties related to sensing data, as well as it allowed for testing of the simulation tool. As for client side testing, there were developed both an Angular 2, single-page-application, for IoTivity to test third-party application support. For Google Weave only the Weave Developer Application was tested for client side testing. This Weave Developer Application is only available for accepted Weave tester. Even though the IoTivity Simulators Client Controller were not implemented as a part of the experiment, it was still represented, allowing stakeholders to understand its features.

There were not conducted any experiments on the HomeKit ecosystem. How-

ever, as with the IoTivity Client Controller, Apple's HomeKit Accessory Simulator was still tested and presented to in the thesis to illustrate its use to stakeholders. Even though there were not conducted an experiment on the HomeKit ecosystem, it should not affect the validity of the results. It was determined that the documentation of Apple's HomeKit was information enough to include it as a part of the interoperability assessment.

The characteristics found in each layer of the assessment model has been identified and evaluated during the study and experiments. This evaluation was done with the goal of identifying the three most important characteristics according to how the ecosystems provide and ensure interoperability. But also to highlight the differences and challenges from the all of the stakeholder perspectives.

6.7 Further work

During the duration of this thesis, there has surfaced new possible research questions related to the scope of this thesis, both as a result of the findings of the study and the experiments.

A continuation of this thesis could be to further research the possibility around bridging. Allowing a single hub or middleware to bridge a device making it compatible with a wide range of protocols, transports, and data models. This will allow the devices with and without constrains to exists side by side in a very transport and protocol restricted ecosystem as well as making them interoperable with data models in multiple ecosystems.

IoTivity is as mentioned an open-source project based on the OCF specification with a goal of making all connected devices in the world interoperable. IoTivity had a competitor, AllJoyn by the Allseen Alliance, which had the same exact goal. Even with the same goal AllJoyn and IoTivity are built on very different technologies and with every different architectures. As of February 2016, it was decided to merge OCF and the Allseen Alliance. It would be very interesting to research the differences, compare strengths and weaknesses and implement a "best-of-bread" ecosystem.

7 Conclusion

Ever since the Internet of Things (IoT) caught fire and the industry began to see its true potential, the number of connected devices have grown incredibly. As a result of this rapid growth, it is has become challenging to provide one ecosystem for all these devices to exists in and communicate. Some service providers have developed their own systems, while others have decided to support external ecosystems. This has resulted in IoT devices only being compatible with specific ecosystems. A consequence of this is that a potential end-user would have to install multiple ecosystems in the same environment to cover all of the user's needs. Because of this, the IoT industry has an interoperability problem. During this thesis three ecosystems sponsored by acknowledged corporations and foundations has been assessed. As a result of thorough research, and development experiments, the assessment allows IoT industry stakeholders to be enlightened on the end-to-end interoperability of the systems, as well as their technological and architectural building blocks. Through the assessment model, it will be possible for stakeholders to use this thesis as a template for further assessment of ecosystems and to compare them, layer by layer.

For the outside looking in these ecosystems might look completely different. Apple's HomeKit present an ecosystem with more proprietary and closed-source resources, Google Weave is a communication platform based on a single protocol and OCFs IoTivity is an open-source implementation of an open specification. However, they are not as different after all, as this thesis' assessment also shows. Assessing them layer by layer has allowed both for an end-to-end evaluation of their interoperability and a comparison of their technical and architectural similarities and differences. This assessment has proven that all the three ecosystems provide end-to-end interoperability and that they achieve it using a very similar architecture. They all ensure device interoperability through device certification and schema definitions. As well as allowing integration of third-party application through documented API frameworks, which abstract their lower software stack and map their chosen protocol on top of it. Even though they all provide end-to-end interoperability, their level of possible interoperability differs because of their differences in supported technologies.

This thesis has also shown the importance of connectivity in IoT ecosystems to enhance the possibilities for interoperability. As of now Google Weave only supports WiFi making the platform very transport restricted, denying the possibility

for interoperability and integration of constrained devices or devices without WiFi support. As Apple's HomeKit also support Bluetooth Low Energy, in addition to WiFi, it provides an options for devices manufacturers wanting to integrate their constrained devices. Still, by only supporting Bluetooth Low Energy in addition to WiFi, it creates a very restricted ecosystem in terms of connectivity. By only supporting these two transports the ecosystems deny integration of many devices that support other popular lightweight wireless technologies. IoTivity, on the other side, does not have this restriction as it supports many of the popular transports. As a result of this, it provides transports for many more devices than the other two ecosystems, potentially allowing devices restricted to different transport to interoperate in the system.

These ecosystem provides the technologies, structure and architecture needed to allow devices to interoperate, and the possibility for interaction with third-party application through documented API frameworks and development tools. However, the stakeholders has much of the responsibility to ensure for this interoperability as well. To integrate with a ecosystem, the device vendors has to obtain a device certification, support of one of the integrate transports of the ecosystem, provide mechanism for onboarding and implement the schema definitions standardized by the ecosystem. A consequence of this is a device locked to a specific ecosystem, which greatly restrict the vendor's device.

Application developers are also restricted in terms of application support. Both Apple's HomeKit and Google Weave only support applications running on specific operating systems, which restricts the developers to develop application only for these specific operating systems. At the same time, Google Weave and IoTivity supports applications running on multiple operating systems. Therefore, developers must choose to either support all of the operating systems or, exclude potential users by only support specific operating systems.

At last, we have the consumer which must consider both. The end-user must choose and adopt an ecosystem which provides the device interoperability needed to cover all the smart environments needs. The end-user must also consider the ecosystems supported operating systems, to be able to control the ecosystem with an application compatible with the user's preferred client device.

Bibliography

[1] Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. 2013. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7), 1645–1660.

[2] Li, S., Da Xu, L., & Zhao, S. 2015. The internet of things: a survey. *Information Systems Frontiers*, 17(2), 243–259.

[3] Soliman, M., Abiodun, T., Hamouda, T., Zhou, J., & Lung, C.-H. 2013. Smart home: Integrating internet of things with web services and cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 2, 317–320. IEEE.

[4] Movassaghi, S., Abolhasan, M., Lipman, J., Smith, D., & Jamalipour, A. 2014. Wireless body area networks: A survey. *IEEE Communications Surveys & Tutorials*, 16(3), 1658–1686.

[5] Guinard, D., Trifa, V. M., & Wilde, E. 2010. *Architecting a mashable open world wide web of things*. ETH, Department of Computer Science.

[6] Kizza, J. M. 2017. Internet of things (iot): Growth, challenges, and security. In *Guide to Computer Network Security*, 517–531. Springer.

[7] Amiri-Kordestani, M. & Bourdoucen, H. 2017. A survey on embedded open source system software for the internet of things.

[8] Eclipse IoT, IEEE, I. C. 2017. Iot developer trends 2017 edition. https://ianskerrett.wordpress.com/2017/04/19/iot-developer-trends-2017-edition/. (Accessed on 05/02/2017).

[9] Suresh, P., Daniel, J. V., Parthasarathy, V., & Aswathy, R. 2014. A state of the art review on the internet of things (iot) history, technology and fields of deployment. In *Science Engineering and Management Research (ICSEMR), 2014 International Conference on*, 1–8. IEEE.

[10] Guinard, D., Trifa, V., Mattern, F., & Wilde, E. 2011. From the internet of things to the web of things: Resource-oriented architecture and best practices. In *Architecting the Internet of things*, 97–129. Springer.

[11] Stojkoska, B. L. R. & Trivodaliev, K. V. 2017. A review of internet of things for smart home: Challenges and solutions. *Journal of Cleaner Production*, 140, 1454–1464.

[12] Kelaidonis, D., Vlacheas, P., Stavroulaki, V., Georgoulas, S., Moessner, K., Hashi, Y., Hashimoto, K., Miyake, Y., Yamada, K., & Demestichas, P. 2017. Cloud internet of things framework for enabling services in smart cities. In *Designing, Developing, and Facilitating Smart Cities*, 163–191. Springer.

[13] Kamruzzaman, M., Sarkar, N. I., Gutierrez, J., & Ray, S. K. 2017. A study of iot-based post-disaster management. In *Information Networking (ICOIN), 2017 International Conference on*, 406–410. IEEE.

[14] Elkhodr, M., Shahrestani, S., & Cheung, H. 2016. The internet of things: new interoperability, management and security challenges. *arXiv preprint arXiv:1604.04824*.

[15] Ocf - about. https://openconnectivity.org/about. (Accessed on 04/12/2017).

[16] Pătru, I.-I., Carabaş, M., Bărbulescu, M., & Gheorghe, L. 2016. Smart home iot system. In *RoEduNet Conference: Networking in Education and Research, 2016 15th*, 1–6. IEEE.

[17] Ocf - specifications. https://openconnectivity.org/resources/specifications. (Accessed on 04/12/2017).

[18] Iotivity - overview. https://wiki.iotivity.org/. (Accessed on 04/13/2017).

[19] Bormann, C., Castellani, A. P., & Shelby, Z. 2012. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(2), 62–67.

[20] Villaverde, B. C., Pesch, D., Alberola, R. D. P., Fedor, S., & Boubekeur, M. 2012. Constrained application protocol for low power embedded networks: A survey. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, 702–707. IEEE.

[21] Cbor — concise binary object representation | overview. http://cbor.io/. (Accessed on 04/11/2017).

[22] Ocf - oneiota data model tool. https://openconnectivity.org/resources/oneiota-data-model-tool. (Accessed on 04/20/2017).

[23] Documentation | iotivity. https://www.iotivity.org/documentation. (Accessed on 04/14/2017).

[24] Gartner says five of top 10 worldwide mobile phone vendors increased sales in second quarter of 2016. http://www.gartner.com/newsroom/id/3415117. (Accessed on 04/29/2017).

[25] Documentation - weave - google developers. https://developers.google.com/weave/guides/overview/what-is-weave. (Accessed on 05/06/2017).

[26] weave - libiota. https://weave.googlesource.com/weave/libiota/. (Accessed on 05/05/2017).

[27] Light interface - weave - google developers. https://developers.google.com/weave/reference/schemas/lightbulb. (Accessed on 05/10/2017).

[28] Weave reference - weave - google developers. https://developers.google.com/weave/reference/. (Accessed on 05/10/2017).

[29] Developer | android things. https://developer.android.com/things/index.html. (Accessed on 05/11/2017).

[30] Credit suisse estimates 588 million apple users. http://www.businessinsider.com/credit-suisse-estimates-588-million-apple-users-2016-4?r=DE&IR=T&IR=T. (Accessed on 05/15/2017).

[31] Introducing homekit - apple developer. https://developer.apple.com/videos/play/wwdc2014/213. (Accessed on 05/16/2017).

[32] Homekit | apple developer documentation. https://developer.apple.com/reference/homekit. (Accessed on 05/19/2017).

[33] Homekit developer guide. https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/HomeKitDeveloperGuide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40015050. (Accessed on 05/19/2017).

[34] Designing accessories for ios and os x- apple developer. https://developer.apple.com/videos/play/wwdc2014/701/. (Accessed on 05/19/2017).

[35] What protocol does homekit use to communicate with its devices? - quora. https://www.quora.com/What-protocol-does-HomeKit-use-to-communicate-with-its-devices. (Accessed on 05/20/2017).

[36] Mfi program enrollment. https://mfi.apple.com/MFiWeb/getFAQ.action. (Accessed on 05/22/2017).

[37] Mike Jackson, Steve Crouch, R. B. Software evaluation service. https://www.software.ac.uk/sites/default/files/SSI-SoftwareEvaluationCriteria.pdf. (Accessed on 05/03/2017).

[38] otcshare/iotivity-node: Node.js bindings for iotivity. https://github.com/otcshare/iotivity-node. (Accessed on 05/12/2017).

[39] 01org/iot-rest-api-server: iot-rest-api-server. https://github.com/01org/iot-rest-api-server. (Accessed on 05/12/2017).